Package: microViz (via r-universe)

August 29, 2024

Title Microbiome Data Analysis and Visualization

Version 0.12.4

URL https://david-barnett.github.io/microViz,
https://github.com/david-barnett/microViz

Description Microbiome data visualization and statistics tools built upon phyloseq.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Depends R (>= 3.5.0)

Imports phyloseq, vegan, microbiome, ggiraph, ggplot2, dplyr (>= 1.0.0), rlang (>= 1.0.0), shiny (>= 1.5.0), ComplexHeatmap (>= 2.0.0), circlize, seriation, colorspace, scales, purrr, tibble, tidyr, broom, cowplot

Suggests DT, ggraph (>= 2.0.0), tidygraph, corncob (>= 0.2.1), ggrepel, GUniFrac, patchwork (>= 1.0.0), ggtext, viridisLite, stringr, forcats, future, future.apply, ggside, skimr, devtools, testthat, vdiffr, knitr, methods, grDevices, rmarkdown, shinytest2

Repository https://david-barnett.r-universe.dev

RemoteUrl https://github.com/david-barnett/microViz

RemoteRef HEAD

RemoteSha 739f3881b6e841f80ef96794331f798dc5f469e2

Contents

`add_paths` *Add paths connecting points on a ggplot scatterplot*

Description

Useful for tracing a few select individuals over time on an ordination plot. Samples in phyloseq must be arranged in order of timepoint for the path connections to be drawn in the correct order! You can arrange the samples in timepoint order with ps_arrange.

Usage

```
add_paths(
  ggplot,
  id_var,
  id_values,
  mapping = NULL,
  arrow = grid::arrow(length = grid::unit(2, units = "mm")),
  ...
)
```

Arguments

`ggplot`	ggplot scatterplot such as the output of ord_plot
`id_var`	name of variable used to identify grouping of points
`id_values`	values of id_var variable used to identify groups of points to draw
`mapping`	ggplot aesthetics created by aes(), e.g. aes(colour = ?) - group is already set to id_var internally!
`arrow`	arrowhead to add to path, NULL for none
`...`	additional arguments passed to geom_path

Value

ggplot with added layer with geom_path

Examples

```
library(ggplot2)
data("dietswap", package = "microbiome")

# arrange by timepoint first (or whatever your own time variable is)
dietswap %>%
  ps_arrange(timepoint) %>%
  tax_fix() %>%
  tax_transform("clr", rank = "Genus") %>%
  ord_calc(method = "PCA") %>%
  ord_plot(colour = "timepoint", alpha = 0.5, size = 2) %>%
  add_paths(
    id_var = "subject", id_values = c("azl", "byn"),
    mapping = aes(colour = timepoint), linewidth = 1.5
    # size = 1.5 # size instead of linewidth in older ggplot2 versions
  )

# paths do NOT connect points in the correct order without arranging first
dietswap %>%
  tax_fix() %>%
  tax_transform("clr", rank = "Genus") %>%
  ord_calc(method = "PCA") %>%
  ord_plot(colour = "timepoint", alpha = 0.5) %>%
  add_paths(
    id_var = "subject", id_values = c("azl", "byn"),
    mapping = aes(colour = timepoint), linewidth = 1.5
    # size = 1.5 # size instead of linewidth in older ggplot2 versions
  ) +
  ggtitle("WRONG PATH ORDER", "use ps_arrange first!")
```

`adjacent_side` *Simple heatmap helper to get a default adjacent side for another annotation*

Description

Simple heatmap helper to get a default adjacent side for another annotation

Usage

```
adjacent_side(side = c("top", "right", "bottom", "left"))
```

Arguments

`side`	one of "top", "right", "bottom", or "left"

Value

character

Examples

```
adjacent_side("top")
```

`anno_cat` *Create colored rectangle annotations for categorical data*

Description

Similar to anno_simple but with individual boxes!

Usage

```
anno_cat(
  x,
  which,
  renamer = identity,
  col = distinct_palette(),
  width = NULL,
  height = NULL,
  box_col = "white",
  box_lwd = 0.5,
  border_col = NA,
  border_lwd = 1,
  legend = TRUE,
  legend_title = ""
)
```

Arguments

`x`	data vector, treated as categorical
`which`	Whether it is a column annotation or a row annotation?
`renamer`	function renaming variable values for legend
`col`	colors vector, at least as long as unique(x), optionally named by x levels
`width`	grid unit object or NULL
`height`	grid unit object or NULL
`box_col`	colour of boxes around individual cells
`box_lwd`	line width of boxes around individual cells
`border_col`	colour of border around all cells
`border_lwd`	line width of border around all cells
`legend`	generate legend for this annotation (attached as attribute of heatmap, and not automatically included in plot)
`legend_title`	title for legend, if drawn

Value

AnnotationFunction

Examples

```
library(ComplexHeatmap)
# draw the annotation without a heatmap, you will never normally do this!
vp <- viewport(width = 0.75, height = 0.75)

grid::grid.newpage()
pushViewport(vp)
cats <- letters[1:4]
draw(anno_cat(cats, which = "row"))

grid::grid.newpage()
pushViewport(vp)
draw(
  anno_cat(
    x = cats, col = structure(names = cats, 1:4), which = "column",
    box_col = "black", box_lwd = 5
  )
)
```

```
# developer note #
# list of annotations can be split and ordered (adding NULL makes a list)
# https://jokergoo.github.io/ComplexHeatmap-reference/book/a-list-of-heatmaps.html
# (section #4.8 concatenate-only-the-annotations)
grid::grid.newpage()
pushViewport(vp)
annoList <- rowAnnotation(
  hi = anno_cat(cats, which = "row", border_col = "black")
) +
  NULL
draw(object = annoList, row_split = c(1, 1:3), row_order = 4:1)
pushViewport(viewport(x = 0.6))
draw(anno_cat(cats, "row", legend_title = "abcd") %>% attr("Legend"))
```

`anno_cat_legend` *Convenience function for generating a legend for anno_cat annotations.*

Description

Convenience function for generating a legend for anno_cat annotations.

Usage

```
anno_cat_legend(col, x = NULL, renamer = identity, title = "", ...)
```

Arguments

`col`	vector of colors, named by all levels of data (e.g. x) or not named
`x`	optional: vector of data to pair with unnamed col or check against named col
`renamer`	function applied to generate labels: from names(col) or levels of x
`title`	title of legend
`...`	Arguments passed on to `ComplexHeatmap::Legend`

- `labels` Labels corresponding to `at`. If it is not specified, the values of `at` are taken as labels.
- `nrow` For legend which is represented as grids, `nrow` controls number of rows of the grids if the grids are arranged into multiple rows.
- `ncol` Similar as `nrow`, `ncol` controls number of columns of the grids if the grids are arranged into multiple columns. Note at a same time only one of `nrow` and `ncol` can be specified.
- `by_row` Are the legend grids arranged by rows or by columns?
- `grid_height` The height of legend grid. It can also control the height of the continuous legend if it is horizontal.
- `grid_width` The width of legend grid. It can also control the width of the continuous legend if it is vertical.

`gap` If legend grids are put into multiple rows or columns, this controls the gap between neighbouring rows or columns, measured as a `unit` object.

`labels_gp` Graphic parameters for labels.

`labels_rot` Text rotation for labels. It should only be used for horizontal continuous legend.

`border` Color of legend grid borders. It also works for the ticks in the continuous legend.

`type` Type of legends. The value can be one of `grid`, `points`, `lines` and `boxplot`.

`direction` Direction of the legend, vertical or horizontal?

`title_position` Position of title relative to the legend. `topleft`, `topcenter`, `leftcenter-rot` and `lefttop-rot` are only for vertical legend and `leftcenter`, `lefttop` are only for horizontal legend.

`title_gap` Gap between title and the legend body.

Value

a ComplexHeatmap Legend class object

Examples

```
grid::grid.newpage()
ComplexHeatmap::draw(
  anno_cat_legend(
    col = c("ibd" = "blue", "nonibd" = "grey90"),
    renamer = toupper, title = "Hi there, I'm a title"
  )
)
```

`anno_sample` *Helper to specify simple comp_heatmap annotation for other sample data*

Description

Use this as an argument to sampleAnnotation(), which itself is used by comp_heatmap() as sample_anno argument.

This creates a vector, which sampleAnnotation() interprets as a simple annotation, so then you set colours and legend parameters for each simple annotation as further arguments in sampleAnnotation.

Usage

```
anno_sample(var, fun = identity, data = NULL, samples = NULL)
```

Arguments

`var`	name of variable to use for annotation data
`fun`	function to transform variable `var`
`data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`samples`	OPTIONAL selection vector of sample names, only set this if providing data argument to override default

Value

vector of values

See Also

`sampleAnnotation()`

Examples

```
# see `?sampleAnnotation()`
```

`anno_sample_cat` *Helper to specify comp_heatmap annotation for categorical sample data*

Description

Use this as an argument to sampleAnnotation(), which itself is used by comp_heatmap() as sample_anno argument.

Usage

```
anno_sample_cat(
  var,
  col = distinct_palette(),
  renamer = identity,
  size = grid::unit(5, "mm"),
  legend = TRUE,
  legend_title = "",
  box_col = "white",
  box_lwd = 0.5,
  border_col = NA,
  border_lwd = 1,
  data = NULL,
  samples = NULL,
  which = NULL,
  ...
)
```

Arguments

`var`	name of variable to use for annotation data
`col`	colors vector, at least as long as unique(x), optionally named by x levels
`renamer`	function to rename levels of variable `var`
`size`	width or height as a grid unit object
`legend`	generate legend for this annotation (attached as attribute of heatmap, and not automatically included in plot)
`legend_title`	title for legend, if drawn
`box_col`	colour of boxes around individual cells
`box_lwd`	line width of boxes around individual cells
`border_col`	colour of border around all cells
`border_lwd`	line width of border around all cells
`data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`samples`	OPTIONAL selection vector of sample names, only set this if providing data argument to override default
`which`	OPTIONAL indicating if it is a 'column' or a 'row' annotation, only set this if providing data argument to override default
`...`	Arguments passed on to `anno_cat` `x` data vector, treated as categorical `width` grid unit object or NULL `height` grid unit object or NULL

Value

vector of values

Examples

```
library("ComplexHeatmap")
data("ibd", package = "microViz")
psq <- ibd
samples <- phyloseq::sample_names(psq)

# makes a function that takes data, taxa and which (at minimum)
fun <- anno_sample_cat(var = "ibd")

# manually specify the prevalence barplot function by giving it data etc.
heatmapAnnoFunction <- fun(data = psq, which = "row", samples = samples)

# draw the barplot without a heatmap, you will never normally do this!
vp <- viewport(width = 0.75, height = 0.75)

grid::grid.newpage()
pushViewport(vp)
```

```
draw(heatmapAnnoFunction)
# A legend is attached by default to anno_cat() output, let's plot that.
pushViewport(viewport(x = 0.75))
draw(attr(heatmapAnnoFunction, "Legend"))

# change some options and specify the data up front
grid::grid.newpage()
pushViewport(vp)
anno_sample_cat(
  data = psq, var = "DiseaseState", samples = samples, which = "column",
  size = grid::unit(5, "cm"), col = distinct_palette(pal = "kelly")
) %>%
  draw()
```

anno_tax_box *Helper to specify heatmap annotation for showing taxa abundance on boxplot*

Description

Use this as an argument to taxAnnotation(), which itself is used by cor_heatmap and comp_heatmap as tax_anno argument.

Usage

```
anno_tax_box(
  undetected = 0,
  only_detected = TRUE,
  trans = "compositional",
  zero_replace = 0,
  use_counts = TRUE,
  size = grid::unit(30, "mm"),
  border = TRUE,
  gp = grid::gpar(fill = "#CCCCCC"),
  ylim = NULL,
  extend = 0.05,
  outline = TRUE,
  box_width = 0.6,
  pch = 1,
  pointsize = grid::unit(0.5, "mm"),
  axis = TRUE,
  ...,
  data = NULL,
  taxa = NULL,
  which = NULL
)
```

Arguments

`undetected`	the value above which taxa are classed as detected/present in a sample
`only_detected`	only plot values for samples where the taxon abundance is > undetected
`trans`	name of transformation suitable for tax_transform, or a function calling tax_transform, and/or tax_scale, (a function must take a phyloseq or psExtra, and return one)
`zero_replace`	zero_replace value for for tax_transform, ignored if trans is a function
`use_counts`	try to retrieve counts from data object?
`size`	width or height as a grid unit object
`border`	Wether draw borders of the annotation region?
`gp`	Graphic parameters for the boxes. The length of the graphic parameters should be one or the number of observations.
`ylim`	Data ranges.
`extend`	The extension to both side of `ylim`. The value is a percent value corresponding to `ylim[2] - ylim[1]`.
`outline`	Whether draw outline of boxplots?
`box_width`	Relative width of boxes. The value should be smaller than one.
`pch`	Point style.
`pointsize`	size of outlier points, as grid::unit() object
`axis`	Whether to add axis?
`...`	Arguments passed on to `ComplexHeatmap::anno_boxplot` `axis_param` parameters for controlling axis. See `default_axis_param` for all possible settings and default parameters.
`data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`taxa`	OPTIONAL selection vector of taxa (names, numbers or logical), only set this if providing data argument to override default
`which`	OPTIONAL indicating if it is a 'column' or a 'row' annotation, only set this if providing data argument to override default

Value

function or ComplexHeatmap AnnotationFunction object

Examples

```
library("ComplexHeatmap")
data("ibd", package = "microViz")
psq <- tax_filter(ibd, min_prevalence = 5)
psq <- tax_mutate(psq, Species = NULL)
psq <- tax_fix(psq)
psq <- tax_agg(psq, rank = "Family")
taxa <- tax_top(psq, n = 15, rank = "Family")
# makes a function that takes data, taxa and which (at minimum)
```

```
fun <- anno_tax_box()
# manually specify the prevalence barplot function by giving it data etc.
heatmapAnnoFunction <- fun(data = psq, which = "column", taxa = taxa)
# draw the barplot without a heatmap, you will never normally do this!
vp <- viewport(width = 0.75, height = 0.75)
grid.newpage()
pushViewport(vp)
draw(heatmapAnnoFunction)

# let's change some style options and specify the data up front
grid::grid.newpage()
pushViewport(vp)
draw(anno_tax_box(
  data = psq, taxa = taxa, which = "row", pointsize = grid::unit(1, "mm"),
  gp = grid::gpar(fill = "red"), border = FALSE, box_width = 0.2
))

# clear drawings
grid::grid.newpage()
```

---

anno_tax_density — *Helper to specify heatmap annotation for showing taxa abundance density plot*

Description

Use this as an argument to taxAnnotation(), which itself is used by cor_heatmap and comp_heatmap as tax_anno argument.

Usage

```
anno_tax_density(
  undetected = 0,
  only_detected = TRUE,
  trans = "log10p",
  zero_replace = 0,
  use_counts = TRUE,
  size = grid::unit(30, "mm"),
  type = c("lines", "violin", "heatmap"),
  xlim = NULL,
  heatmap_colors = c("white", "forestgreen"),
  joyplot_scale = 1.5,
  border = TRUE,
  gp = grid::gpar(fill = "lightgrey"),
  axis = TRUE,
  ...,
  data = NULL,
  taxa = NULL,
  which = NULL
)
```

Arguments

`undetected`	the value above which taxa are classed as detected/present in a sample
`only_detected`	only plot values for samples where the taxon abundance is > undetected
`trans`	name of transformation suitable for tax_transform, or a function calling tax_transform, and/or tax_scale, (a function must take a phyloseq or psExtra, and return one)
`zero_replace`	zero_replace value for for tax_transform, ignored if trans is a function
`use_counts`	try to retrieve counts from data object?
`size`	width or height as a grid unit object
`type`	Type of graphics to represent density distribution. "lines" for normal density plot; "violine" for violin plot and "heatmap" for heatmap visualization of density distribution.
`xlim`	Range on x-axis.
`heatmap_colors`	A vector of colors for interpolating density values.
`joyplot_scale`	Relative height of density distribution. A value higher than 1 increases the height of the density distribution and the plot will represented as so-called "joyplot".
`border`	Wether draw borders of the annotation region?
`gp`	Graphic parameters for the boxes. The length of the graphic parameters should be one or the number of observations.
`axis`	Whether to add axis?
`...`	Arguments passed on to `ComplexHeatmap::anno_density` `axis_param` parameters for controlling axis. See `default_axis_param` for all possible settings and default parameters.
`data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`taxa`	OPTIONAL selection vector of taxa (names, numbers or logical), only set this if providing data argument to override default
`which`	OPTIONAL indicating if it is a 'column' or a 'row' annotation, only set this if providing data argument to override default

Value

function or ComplexHeatmap AnnotationFunction object

Examples

```
library("ComplexHeatmap")
data("ibd", package = "microViz")
psq <- tax_filter(ibd, min_prevalence = 5)
psq <- tax_mutate(psq, Species = NULL)
psq <- tax_fix(psq)
psq <- tax_agg(psq, rank = "Family")
taxa <- tax_top(psq, n = 15, rank = "Family")
# makes a function that takes data, taxa and which (at minimum)
fun <- anno_tax_density()
```

```
# manually specify the density plot function by giving it data etc.
heatmapAnnoFunction <- fun(data = psq, which = "column", taxa = taxa)

# draw the density plot without a heatmap, you will never normally do this!
vp <- viewport(width = 0.75, height = 0.75)
grid.newpage()
pushViewport(vp)
draw(heatmapAnnoFunction)

# let's change some style options and specify the data up front
grid.newpage()
pushViewport(vp)
draw(anno_tax_density(
  data = psq, taxa = taxa, which = "row",
  gp = grid::gpar(fill = "red"), border = FALSE
))

# heatmap type, with alternative transformation and axis_param
grid.newpage()
pushViewport(vp)
draw(anno_tax_density(
  data = psq, taxa = taxa, which = "row", type = "heatmap",
  trans = "log2", zero_replace = "halfmin", axis_param = list(labels_rot = 0)
))

grid.newpage()
```

anno_tax_prev — *Helper to specify heatmap annotation for showing taxa prevalence as barplot*

Description

Use this as an argument to taxAnnotation(), which itself is used by cor_heatmap and comp_heatmap as tax_anno argument.

Usage

```
anno_tax_prev(
  undetected = 0,
  use_counts = TRUE,
  size = grid::unit(20, "mm"),
  baseline = 0,
  border = TRUE,
  bar_width = 0.6,
  gp = grid::gpar(fill = "#CCCCCC"),
  ylim = NULL,
  extend = 0.05,
  axis = TRUE,
  ...,
  data = NULL,
  taxa = NULL,
  which = NULL
)
```

Arguments

undetected	the value above which taxa are classed as detected/present in a sample
use_counts	try to retrieve counts from data object?
size	width or height as a grid unit object
baseline	baseline of bars. The value should be "min" or "max", or a numeric value. It is enforced to be zero for stacked barplots.
border	Wether draw borders of the annotation region?
bar_width	Relative width of the bars. The value should be smaller than one.
gp	Graphic parameters for bars. The length of each graphic parameter can be 1, length of `x` if `x` is a vector, or number of columns of `x` is `x` is a matrix.
ylim	Data ranges. By default it is `range(x)` if `x` is a vector, or `range(rowSums(x))` if `x` is a matrix.
extend	The extension to both side of `ylim`. The value is a percent value corresponding to `ylim[2] - ylim[1]`.
axis	Whether to add axis?
...	Arguments passed on to `ComplexHeatmap::anno_barplot` `axis_param` parameters for controlling axis. See `default_axis_param` for all possible settings and default parameters.
data	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
taxa	OPTIONAL selection vector of taxa (names, numbers or logical), only set this if providing data argument to override default
which	OPTIONAL indicating if it is a 'column' or a 'row' annotation, only set this if providing data argument to override default

Value

function or ComplexHeatmap AnnotationFunction object

Examples

```
library("ComplexHeatmap")
data("ibd", package = "microViz")
psq <- tax_filter(ibd, min_prevalence = 5)
psq <- tax_mutate(psq, Species = NULL)
psq <- tax_fix(psq)
psq <- tax_agg(psq, rank = "Family")
taxa <- tax_top(psq, n = 15, rank = "Family")
```

```
# makes a function that takes data, taxa and which (at minimum)
fun <- anno_tax_prev()

# manually specify the prevalence barplot function by giving it data etc.
heatmapAnnoFunction <- fun(data = psq, which = "row", taxa = taxa)

# draw the barplot without a heatmap, you will never normally do this!
vp <- viewport(width = 0.75, height = 0.75)

grid::grid.newpage()
pushViewport(vp)
draw(heatmapAnnoFunction)

# let's change some style options and specify the data up front
grid::grid.newpage()
pushViewport(vp)
anno_tax_prev(
  data = psq, taxa = taxa, which = "column",
  gp = grid::gpar(fill = "red", lwd = 3, alpha = 0.5),
  border = FALSE, bar_width = 1
) %>%
  draw()

# clear drawings
grid::grid.newpage()
```

anno_var_box *Helper to specify heatmap annotation for variable distribution boxplots*

Description

Use this as an argument to varAnnotation(), which itself is used by cor_heatmap as var_anno() argument.

Usage

```
anno_var_box(
  fun = identity,
  size = grid::unit(30, "mm"),
  border = TRUE,
  gp = grid::gpar(fill = "#CCCCCC"),
  ylim = NULL,
  extend = 0.05,
  outline = TRUE,
  box_width = 0.6,
  pch = 1,
  pointsize = grid::unit(0.5, "mm"),
  axis = TRUE,
  ...,
  data = NULL,
  vars = NULL,
  which = NULL
)
```

Arguments

`fun`	function applied to all variables, with apply()
`size`	width or height as a grid unit object
`border`	Wether draw borders of the annotation region?
`gp`	Graphic parameters for the boxes. The length of the graphic parameters should be one or the number of observations.
`ylim`	Data ranges.
`extend`	The extension to both side of `ylim`. The value is a percent value corresponding to `ylim[2] - ylim[1]`.
`outline`	Whether draw outline of boxplots?
`box_width`	Relative width of boxes. The value should be smaller than one.
`pch`	Point style.
`pointsize`	size of outlier points, as grid::unit() object
`axis`	Whether to add axis?
`...`	Arguments passed on to `ComplexHeatmap::anno_boxplot` `axis_param` parameters for controlling axis. See `default_axis_param` for all possible settings and default parameters.
`data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`vars`	OPTIONAL selection vector of variable names, only set this if providing data argument to override default
`which`	OPTIONAL indicating if it is a 'column' or a 'row' annotation, only set this if providing data argument to override default

Value

function or ComplexHeatmap AnnotationFunction object

Examples

```
library(ComplexHeatmap)
set.seed(123)
fakeData <- as.data.frame.matrix(matrix(rnorm(500, 10, 3), ncol = 10))
names(fakeData) <- paste0("var_", 1:10)

# draw the boxplot without a heatmap, you will never normally do this!
vp <- viewport(width = 0.75, height = 0.75)
```

```
grid.newpage()
pushViewport(vp)
draw(
  anno_var_box(data = fakeData, vars = names(fakeData), which = "column")
)

grid.newpage()
pushViewport(vp)
draw(
  anno_var_box(
    data = fakeData, fun = function(x) log(x + 1),
    vars = rev(names(fakeData)),
    which = "row"
  )
)
```

`anno_var_density` *Helper to specify heatmap annotation for variable distribution density plot*

Description

Use this as an argument to varAnnotation(), which itself is used by cor_heatmap var_anno argument.

Usage

```
anno_var_density(
  fun = identity,
  size = grid::unit(30, "mm"),
  type = c("lines", "violin", "heatmap"),
  xlim = NULL,
  heatmap_colors = c("white", "forestgreen"),
  joyplot_scale = 1.5,
  border = TRUE,
  gp = grid::gpar(fill = "lightgrey"),
  axis = TRUE,
  ...,
  data = NULL,
  vars = NULL,
  which = NULL
)
```

Arguments

`fun`	function applied to all variables, with apply()
`size`	width or height as a grid unit object

`type`	Type of graphics to represent density distribution. "lines" for normal density plot; "violine" for violin plot and "heatmap" for heatmap visualization of density distribution.
`xlim`	Range on x-axis.
`heatmap_colors`	A vector of colors for interpolating density values.
`joyplot_scale`	Relative height of density distribution. A value higher than 1 increases the height of the density distribution and the plot will represented as so-called "joyplot".
`border`	Wether draw borders of the annotation region?
`gp`	Graphic parameters for the boxes. The length of the graphic parameters should be one or the number of observations.
`axis`	Whether to add axis?
`...`	Arguments passed on to `ComplexHeatmap::anno_density` `axis_param` parameters for controlling axis. See `default_axis_param` for all possible settings and default parameters.
`data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`vars`	OPTIONAL selection vector of variable names, only set this if providing data argument to override default
`which`	OPTIONAL indicating if it is a 'column' or a 'row' annotation, only set this if providing data argument to override default

Value

function or ComplexHeatmap AnnotationFunction object

Examples

```
library(ComplexHeatmap)
set.seed(123)
fakeData <- as.data.frame.matrix(matrix(rnorm(500, 10, 3), ncol = 10))
names(fakeData) <- paste0("var_", 1:10)

# draw the plots without a heatmap, you will never normally do this!
vp <- viewport(width = 0.75, height = 0.75)
grid.newpage()
pushViewport(vp)
draw(
  anno_var_density(data = fakeData, vars = names(fakeData), which = "row")
)

grid.newpage()
pushViewport(vp)
draw(
  anno_var_density(
    data = fakeData, fun = function(x) log(x + 1),
    vars = rev(names(fakeData)), type = "heatmap",
    which = "column"
  )
)
```

anno_var_hist *Helper to specify heatmap annotation for variable distribution histograms*

Description

Use this as an argument to varAnnotation(), which itself is used by cor_heatmap var_anno argument.

Usage

```
anno_var_hist(
  fun = identity,
  size = grid::unit(30, "mm"),
  n_breaks = 11,
  border = FALSE,
  gp = grid::gpar(fill = "#CCCCCC"),
  axis = TRUE,
  ...,
  data = NULL,
  vars = NULL,
  which = NULL
)
```

Arguments

`fun`	function applied to all variables, with apply()
`size`	width or height as a grid unit object
`n_breaks`	number of breaks
`border`	Wether draw borders of the annotation region?
`gp`	Graphic parameters for the boxes. The length of the graphic parameters should be one or the number of observations.
`axis`	Whether to add axis?
`...`	Arguments passed on to `ComplexHeatmap::anno_density` `axis_param` parameters for controlling axis. See `default_axis_param` for all possible settings and default parameters.
`data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`vars`	OPTIONAL selection vector of variable names, only set this if providing data argument to override default
`which`	OPTIONAL indicating if it is a 'column' or a 'row' annotation, only set this if providing data argument to override default

Value

function or ComplexHeatmap AnnotationFunction object

Examples

```
library(ComplexHeatmap)
set.seed(123)
fakeData <- as.data.frame.matrix(matrix(rnorm(500, 10, 3), ncol = 10))
names(fakeData) <- paste0("var_", 1:10)

# draw the histograms without a heatmap, you will never normally do this!
vp <- viewport(width = 0.75, height = 0.75)
grid.newpage()
pushViewport(vp)
draw(
  anno_var_hist(data = fakeData, vars = names(fakeData), which = "row")
)

grid.newpage()
pushViewport(vp)
draw(
  anno_var_hist(
    data = fakeData, fun = sqrt,
    vars = rev(names(fakeData)), n_breaks = 5,
    which = "column", gp = grid::gpar(fill = 2:6, lwd = c(0.9, 2.5))
  )
)
```

`comp_barplot` *Plot (grouped and ordered) compositional barplots*

Description

Stacked barplots showing composition of phyloseq samples for a specified number of coloured taxa. Normally your phyloseq object should contain counts data, as by default `comp_barplot()` performs the "compositional" taxa transformation for you, and requires count input for some sample_order methods!

Usage

```
comp_barplot(
  ps,
  tax_level,
  n_taxa = 8,
  tax_order = sum,
  merge_other = TRUE,
  taxon_renamer = function(x) identity(x),
  sample_order = "bray",
  order_with_all_taxa = FALSE,
  label = "SAMPLE",
  group_by = NA,
  facet_by = NA,
  bar_width = 1,
  bar_outline_colour = "grey5",
  bar_outline_width = 0.1,
  palette = distinct_palette(n_taxa),
  tax_transform_for_ordering = "identity",
  tax_transform_for_plot = "compositional",
  seriate_method = "OLO_ward",
  keep_all_vars = TRUE,
  interactive = FALSE,
  max_taxa = 10000,
  other_name = "Other",
  x = "SAMPLE",
  ...
)
```

Arguments

`ps`	phyloseq object
`tax_level`	taxonomic aggregation level (from rank_names(ps))
`n_taxa`	how many taxa to show distinct colours for (all others grouped into "Other")
`tax_order`	order of taxa within the bars, either a function for tax_sort (e.g. sum), or a vector of (all) taxa names at tax_level to set order manually
`merge_other`	if FALSE, taxa coloured/filled as "other" remain distinct, and so can have bar outlines drawn around them
`taxon_renamer`	function that takes taxon names and returns modified names for legend
`sample_order`	vector of sample names; or any distance measure in dist_calc that doesn't require phylogenetic tree; or "asis" for the current order as is returned by phyloseq::sample_names(ps)
`order_with_all_taxa`	if TRUE, this will always use all taxa (not just the top n_taxa) to calculate any distances needed for sample ordering
`label`	name of variable to use for labelling samples, or "SAMPLE" for sample names
`group_by`	splits dataset by this variable (must be categorical) • resulting in a list of plots, one for each level of the group_by variable.
`facet_by`	facets plots by this variable (must be categorical). If group_by is also set the faceting will occur separately in the plot for each group.
`bar_width`	default 1 avoids random gapping otherwise seen with many samples (set to less than 1 to introduce gaps between samples)
`bar_outline_colour`	line colour separating taxa and samples (use NA for no outlines)
`bar_outline_width`	width of line separating taxa and samples (for no outlines set bar_outline_colour = NA)
`palette`	palette for taxa fill colours

`tax_transform_for_ordering`	transformation of taxa values used before ordering samples by similarity
`tax_transform_for_plot`	default "compositional" draws proportions of total counts per sample, but you could reasonably use another transformation, e.g. "identity", if you have truly quantitative microbiome profiling data
`seriate_method`	name of any ordering method suitable for distance matrices (see ?seriation::seriate)
`keep_all_vars`	FALSE may speed up internal melting with ps_melt for large phyloseq objects but TRUE is required for some post-hoc plot customisation
`interactive`	creates plot suitable for use with ggiraph
`max_taxa`	maximum distinct taxa groups to show (only really useful for limiting complexity of interactive plots e.g. within ord_explore)
`other_name`	name for other taxa after N
`x`	name of variable to use as x aesthetic: it probably only makes sense to change this when also using facets (check only one sample is represented per bar!)
`...`	extra arguments passed to facet_wrap() (if facet_by is not NA)

Details

- sample_order: Either specify a list of sample names to order manually, or the bars/samples can/will be sorted by similarity, according to a specified distance measure (default 'bray'-curtis),
- seriate_method specifies a seriation/ordering algorithm (default Ward hierarchical clustering with optimal leaf ordering, see seriation::list_seriation_methods())
- group_by: You can group the samples on distinct plots by levels of a variable in the phyloseq object. The list of ggplots produced can be arranged flexibly with the patchwork package functions. If you want to group by several variables you can create an interaction variable with interaction(var1, var2) in the phyloseq sample_data BEFORE using comp_barplot.
- facet_by can allow faceting of your plot(s) by a grouping variable. Using this approach is less flexible than using group_by but means you don't have to arrange a list of plots yourself like with the group_by argument. Using facet_by is equivalent to adding a call to facet_wrap(facets = facet_by, scales = "free") to your plot(s). Calling facet_wrap() yourself is itself a more flexible option as you can add other arguments like the number of rows etc. However you must use keep_all_vars = TRUE if you will add faceting manually.
- bar_width: No gaps between bars, unless you want them (decrease width argument to add gaps between bars).
- bar_outline_colour: Bar outlines default to "grey5" for almost black outlines. Use NA if you don't want outlines.
- merge_other: controls whether bar outlines can be drawn around individual (lower abundance) taxa that are grouped in "other" category. If you want to see the diversity of taxa in "other" use merge_taxa = FALSE, or use TRUE if you prefer the cleaner merged look
- palette: Default colouring is consistent across multiple plots if created with the group_by argument, and the defaults scheme retains the colouring of the most abundant taxa irrespective of n_taxa

Value

ggplot or list of harmonised ggplots

Examples

```
library(ggplot2)
data(dietswap, package = "microbiome")

# illustrative simple customised example
dietswap %>%
  ps_filter(timepoint == 1) %>%
  comp_barplot(
    tax_level = "Family", n_taxa = 8,
    bar_outline_colour = NA,
    sample_order = "bray",
    bar_width = 0.7,
    taxon_renamer = toupper
  ) + coord_flip()

# change colour palette with the distinct_palette() function
# remember to set the number of colours to the same as n_taxa argument!
dietswap %>%
  ps_filter(timepoint == 1) %>%
  comp_barplot(
    tax_level = "Family", n_taxa = 8,
    bar_outline_colour = NA,
    sample_order = "bray",
    bar_width = 0.7,
    palette = distinct_palette(8, pal = "kelly"),
    taxon_renamer = toupper
  ) + coord_flip()

# Order samples by the value of one of more sample_data variables.
# Use ps_arrange and set sample_order = "default" in comp_barplot.
# ps_mutate is also used here to create an informative variable for axis labelling
dietswap %>%
  ps_mutate(subject_timepoint = interaction(subject, timepoint)) %>%
  ps_filter(nationality == "AAM", group == "DI", sex == "female") %>%
  ps_arrange(desc(subject), desc(timepoint)) %>%
  comp_barplot(
    tax_level = "Genus", n_taxa = 12,
    sample_order = "default",
    bar_width = 0.7,
    bar_outline_colour = "black",
    order_with_all_taxa = TRUE,
    label = "subject_timepoint"
  ) + coord_flip()

# Order taxa differently:
# By default, taxa are ordered by total sum across all samples
# You can set a different function for calculating the order, e.g. median
dietswap %>%
  ps_filter(timepoint == 1) %>%
  comp_barplot(tax_level = "Genus", tax_order = median) +
  coord_flip()

# Or you can set the taxa order up front, with tax_sort() and use it as is
dietswap %>%
  ps_filter(timepoint == 1) %>%
  tax_sort(at = "Genus", by = sum) %>%
  comp_barplot(tax_level = "Genus", tax_order = "asis") +
  coord_flip()

# how many taxa are in those light grey "other" bars?
# set merge_other to find out (& remember to set a bar_outline_colour)
dietswap %>%
  ps_filter(timepoint == 1) %>%
  comp_barplot(
   tax_level = "Genus", n_taxa = 12, merge_other = FALSE, bar_outline_colour = "grey50",
  ) +
  coord_flip()


# Often to compare groups, average compositions are presented
p1 <- phyloseq::merge_samples(dietswap, group = "group") %>%
  comp_barplot(
    tax_level = "Genus", n_taxa = 12,
    sample_order = c("ED", "HE", "DI"),
    bar_width = 0.8
  ) +
  coord_flip() + labs(x = NULL, y = NULL)
p1

# However that "group-averaging" approach hides a lot of within-group variation
p2 <- comp_barplot(dietswap,
  tax_level = "Genus", n_taxa = 12, group_by = "group",
  sample_order = "euclidean", bar_outline_colour = NA
) %>%
  patchwork::wrap_plots(nrow = 3, guides = "collect") &
  coord_flip() & labs(x = NULL, y = NULL) &
  theme(axis.text.y = element_blank(), axis.ticks.y = element_blank())
p2

# Only from p2 you can see that the apparently higher average relative abundance
# of Oscillospira in group DI is probably driven largely by a subgroup
# of DI samples with relatively high Oscillospira.

# make a list of 2 harmonised composition plots (grouped by sex)
p <- comp_barplot(dietswap,
  n_taxa = 15, tax_level = "Genus",
  bar_outline_colour = "black", merge_other = TRUE,
  sample_order = "aitchison", group_by = "sex"
)

# plot them side by side with patchwork package
```

```
patch <- patchwork::wrap_plots(p, ncol = 2, guides = "collect")
patch & coord_flip() # make bars in all plots horizontal (note: use & instead of +)

# beautifying tweak #
# modify one plot in place (flip the order of the samples in the 2nd plot)
# notice that the scaling is for the x-axis
# (that's because coord_flip is used afterwards when displaying the plots
patch[[2]] <- patch[[2]] + scale_x_discrete(limits = rev)
# Explainer: rev() function takes current limits and reverses them.
# You could also pass a completely arbitrary order, naming all samples

# you can theme all plots with the & operator
patch & coord_flip() &
  theme(axis.text.y = element_text(size = 5), legend.text = element_text(size = 6))
# See https://patchwork.data-imaginist.com/index.html
```

comp_heatmap *Draw heatmap of microbiome composition across samples*

Description

Heatmap made with `ComplexHeatmap::Heatmap()`, with optional annotation of taxa prevalence/abundance, and/or other sample data.

Transform your data with `tax_transform()` prior to plotting (and/or scale with `tax_scale()`).

See the heatmaps vignette for more examples of use.

Plotting "compositional" data can give an idea of the dominant taxa in each sample. Plotting some form of log or clr transformed (or scaled) microbial features can highlight other patterns.

The data will be ordered via your selected seriation methods and distances on either the transformed data (default) or the original count data (or with any other transformation).

Any cell numbers printed can be transformed independently of the colour scheme, and do not affect ordering.

Usage

```
comp_heatmap(
  data,
  taxa = NA,
  taxa_side = "right",
  tax_anno = NULL,
  taxon_renamer = identity,
  samples = NA,
  sample_side = adjacent_side(taxa_side),
  sample_anno = NULL,
  sample_names_show = FALSE,
  colors = heat_palette(palette = "Rocket", rev = TRUE),
  numbers = NULL,
  sample_seriation = "OLO_ward",
  sample_ser_dist = "euclidean",
 sample_ser_counts = !sample_ser_dist %in% c("euclidean", "maximum", "manhattan",
    "canberra", "binary"),
  sample_ser_trans = NULL,
  tax_seriation = "OLO_ward",
  tax_ser_dist = "euclidean",
  tax_ser_counts = FALSE,
  tax_ser_trans = NULL,
  numbers_trans = NULL,
  numbers_zero_replace = 0,
  numbers_use_counts = TRUE,
  grid_col = "white",
  grid_lwd = 0.5,
  name = "Abd.",
  anno_tax = NULL,
  ...
)
```

Arguments

`data`	phyloseq or phyloseq extra
`taxa`	list of taxa to include, or NA for all
`taxa_side`	"top"/"right"/"bottom"/"left": controls heatmap orientation and where any annotations specified in tax_anno are placed
`tax_anno`	NULL or annotation function for taxa: taxAnnotation() output.
`taxon_renamer`	function to rename taxa before plotting
`samples`	list of samples to include on plot
`sample_side`	which side to show any sample annotation on, must be adjacent to taxa_side
`sample_anno`	NULL or annotation function for samples: sampleAnnotation() output.
`sample_names_show`	show sample names? (you can control side and rotation of names with other ComplexHeatmap::Heatmap arguments)
`colors`	output of heat_palette() to set heatmap fill color scheme
`numbers`	output of heat_numbers() to draw numbers on heatmap cells
`sample_seriation`	name of method used to order the samples (from seriation::seriate)
`sample_ser_dist`	name of distance to use with sample_seriation method (if needed)
`sample_ser_counts`	insist on using count data for sample seriation?
`sample_ser_trans`	function for transformation of data used for sample seriation (such as a call to `tax_transform()`)
`tax_seriation`	name of method used to order the taxa (from seriation::seriate)

`tax_ser_dist`	name of distance to use with tax_seriation method (if needed)
`tax_ser_counts`	insist on using count data for taxa seriation?
`tax_ser_trans`	function for transformation of data used for taxa seriation (such as a call to `tax_transform()`)
`numbers_trans`	name of tax_transform transformation, or a function for transformation of data used for drawing numbers on cells
`numbers_zero_replace`	zero replacement method used if named transformation given to number_trans
`numbers_use_counts`	insist on using count data for number drawing? (if TRUE, any numbers_trans transformation would be applied to count data)
`grid_col`	colour of gridlines, or NA for none
`grid_lwd`	width of gridlines
`name`	used as legend title (colourbar)
`anno_tax`	DEPRECATED: optional annotation of taxa distributions: tax_anno() list output, or a pre-made ComplexHeatmap HeatmapAnnotation
`...`	Arguments passed on to `ComplexHeatmap::Heatmap`

`row_dend_side` Should the row dendrogram be put on the left or right of the heatmap?

`row_dend_width` Width of the row dendrogram, should be a `unit` object.

`show_row_dend` Whether show row dendrogram?

`row_dend_gp` Graphic parameters for the dendrogram segments. If users already provide a `dendrogram` object with edges rendered, this argument will be ignored.

`show_row_names` Whether show row names.

`row_names_gp` Graphic parameters for row names.

`row_names_rot` Rotation of row names.

`row_names_centered` Should row names put centered?

See Also

`cor_heatmap()`

Examples

```
library(dplyr)
data("dietswap", package = "microbiome")
# create a couple of numerical variables to use
psq <- dietswap %>%
  ps_mutate(
    weight = recode(bmi_group, obese = 3, overweight = 2, lean = 1),
    female = if_else(sex == "female", true = 1, false = 0),
    african = if_else(nationality == "AFR", true = 1, false = 0)
  )
psq <- tax_filter(psq, min_prevalence = 1 / 10, min_sample_abundance = 1 / 10)
psq <- tax_agg(psq, "Genus")
```

```
# randomly select 20 taxa from the 40 top taxa, and 40 random samples

set.seed(123)
taxa <- sample(tax_top(psq, n = 40), size = 20)
samples <- sample(1:122, size = 40)

comp_heatmap(data = psq, taxa = taxa, samples = samples)

# transforming taxon abundances #

# NOTE: if you plan on transforming taxa (e.g. to compositional data or clr)
# but only want to plot a subset of the taxa (e.g. most abundant)
# you should NOT subset the original phyloseq before transformation!
# Instead, choose the subset of taxa plotted with:

# Note 2, choose a symmetrical palette for clr-transformed data
psq %>%
  tax_transform("clr", zero_replace = "halfmin") %>%
  comp_heatmap(
    taxa = taxa, samples = samples, colors = heat_palette(sym = TRUE)
  )

# Almost all the taxa have high values (>> 0) because they are a highly
# abundant subset taken after clr transformation was calculated on all taxa

# See how just taking the first 30 taxa from the dataset gives more balance
psq %>%
  tax_transform("clr", zero_replace = "halfmin") %>%
  comp_heatmap(
    taxa = 1:30, samples = samples, colors = heat_palette(sym = TRUE)
  )

# annotating taxa #

# Notes:
# - Unlike cor_heatmap, taxa are not annotated by default
# - Detection threshold set to 50 as HITchip example data seems to have background noise

comp_heatmap(
  data = psq, taxa = taxa, samples = samples,
  tax_anno = taxAnnotation(Prev = anno_tax_prev(undetected = 50))
)

# annotating samples #

htmp <- psq %>%
  tax_transform("clr", zero_replace = "halfmin") %>%
  comp_heatmap(
    taxa = taxa, samples = samples, colors = heat_palette(sym = TRUE),
    sample_anno = sampleAnnotation(
      Nation. = anno_sample_cat("nationality", legend_title = "Nation.")
    )
  )
htmp

# legends from `anno_sample_cat()` are stored as an attribute of the Heatmap
ComplexHeatmap::draw(
  object = htmp,
  annotation_legend_list = attr(htmp, "AnnoLegends"), merge_legends = TRUE
)
```

---

cor_heatmap *Microbe-to-sample-data correlation heatmap*

Description

Plot correlations between (transformed) microbial abundances and (selected) numeric-like sample_data variables from a phyloseq object.

Lots of customisation options available through the listed arguments, and you can pass any other argument from `ComplexHeatmap::Heatmap()` too.

Usage

```
cor_heatmap(
  data,
  taxa = NA,
  tax_anno = taxAnnotation(Prev. = anno_tax_prev(), Abun. = anno_tax_box()),
  taxon_renamer = identity,
  vars = NA,
  var_anno = NULL,
  cor = c("pearson", "kendall", "spearman"),
  cor_use = "everything",
  colors = heat_palette(palette = "Blue-Red 2", sym = TRUE),
  numbers = heat_numbers(decimals = 1, col = "black", fontface = "plain"),
  taxa_side = "right",
  vars_side = adjacent_side(taxa_side),
  seriation_method = "OLO_ward",
  seriation_dist = "euclidean",
  seriation_method_col = seriation_method,
  seriation_dist_col = seriation_dist,
  var_fun = "identity",
  grid_col = "white",
  grid_lwd = 0.5,
  anno_tax = NULL,
  anno_vars = NULL,
  ...
)
```

Arguments

`data`	phyloseq or phyloseq extra
`taxa`	list of taxa to include, or NA for all
`tax_anno`	NULL or annotation function for taxa: taxAnnotation() output.
`taxon_renamer`	function to rename taxa before plotting
`vars`	selection of variable names from sample_data
`var_anno`	NULL or annotation function for variables: varAnnotation() output.
`cor`	correlation coefficient. pearson/kendall/spearman, can be abbreviated (used as legend title)
`cor_use`	passed to cor(use = cor_use)
`colors`	output of heat_palette() to set heatmap fill color scheme
`numbers`	output of heat_numbers() to draw numbers on heatmap cells
`taxa_side`	"top"/"right"/"bottom"/"left": controls heatmap orientation and where any annotations specified in tax_anno are placed
`vars_side`	which side to place any variable annotations specified in var_anno, must be an adjacent side to taxa_side
`seriation_method`	method to order the rows (in seriation::seriate)
`seriation_dist`	distance to use in seriation_method (if needed)
`seriation_method_col`	method to order the columns (in seriation::seriate)
`seriation_dist_col`	distance to use in seriation_method_col (if needed)
`var_fun`	a function (or name of) to be applied to columns of a matrix of vars before correlating (but not used in any variable annotations)
`grid_col`	colour of gridlines, or NA for none
`grid_lwd`	width of gridlines
`anno_tax`	DEPRECATED: optional annotation of taxa distributions: tax_anno() list output, or a pre-made ComplexHeatmap HeatmapAnnotation
`anno_vars`	DEPRECATED: use var_anno argument instead. Optional annotation of variable distributions: var_anno() list output, or a pre-made ComplexHeatmap HeatmapAnnotation
`...`	Arguments passed on to `ComplexHeatmap::Heatmap`

- `row_dend_width` Width of the row dendrogram, should be a `unit` object.
- `show_row_dend` Whether show row dendrogram?
- `row_dend_gp` Graphic parameters for the dendrogram segments. If users already provide a `dendrogram` object with edges rendered, this argument will be ignored.
- `show_row_names` Whether show row names.
- `row_names_gp` Graphic parameters for row names.
- `row_names_rot` Rotation of row names.
- `row_names_centered` Should row names put centered?
- `show_heatmap_legend` Whether show heatmap legend?

Details

Using a data.frame for the data argument is also possible, in which case the (selected) numeric-like variables will be correlated with each other, and all arguments relating to taxa will be ignored.

See Also

`taxAnnotation()` `varAnnotation()`

`comp_heatmap()`

`ComplexHeatmap::Heatmap()`

Examples

```
library(dplyr)
data("dietswap", package = "microbiome")

# create a couple of numerical variables to use
psq <- dietswap %>%
  ps_mutate(
    weight = recode(bmi_group, obese = 3, overweight = 2, lean = 1),
    female = if_else(sex == "female", true = 1, false = 0),
    african = if_else(nationality == "AFR", true = 1, false = 0)
  )
psq <- tax_filter(psq, min_prevalence = 1 / 10, min_sample_abundance = 1 / 10)
psq <- tax_agg(psq, "Genus")

# randomly select 20 taxa from the 50 most abundant taxa
set.seed(123)
taxa <- sample(tax_top(psq, n = 50), size = 20)

# NOTE: detection threshold set to 50 as HITchip example data seems to have background noise
ud <- 50

# make simple correlation heatmap with all numeric-like variables
cor_heatmap(
  data = psq, taxa = taxa,
  tax_anno = taxAnnotation(
    Prv. = anno_tax_prev(undetected = ud),
    Abd. = anno_tax_box(undetected = ud)
  )
)

# You can create an annotation object separately in advance
taxAnno <- taxAnnotation(
  Prv. = anno_tax_prev(undetected = ud), Abd. = anno_tax_box(undetected = ud)
)
class(taxAnno) # "function"

# You can select which numeric-like variables to correlate taxa with
cor_heatmap(
  psq, taxa,
  vars = c("african", "female", "weight"), tax_anno = taxAnno
)

# Also you can choose alternative correlation measures
cor_heatmap(psq, taxa, cor = "spearman", tax_anno = taxAnno)

# Annotating variables is possible, and easy with varAnnotation()
cor_heatmap(
  data = psq, taxa = taxa, tax_anno = taxAnno,
  var_anno = varAnnotation(Val. = anno_var_box(size = grid::unit(2, "cm")))
)

# you can transform the variables before correlating by var_fun
# notice this does not affect the data used for annotations
cor_heatmap(
  data = psq, taxa = taxa, tax_anno = taxAnno, var_fun = "exp",
  var_anno = varAnnotation(Val. = anno_var_box(size = grid::unit(2, "cm")))
)

# other and multiple annotations
cor_heatmap(
  data = psq, taxa = taxa[1:10], vars = c("african", "weight", "female"),
  tax_anno = taxAnno,
  var_anno = varAnnotation(
    value = anno_var_hist(size = grid::unit(15, "mm")),
    log10p = anno_var_box(function(x) log10(x + 1))
  )
)

# make the same heatmap, but rotated
cor_heatmap(
  data = psq, taxa = taxa[1:10], vars = c("african", "weight", "female"),
  tax_anno = taxAnno, taxa_side = "top",
  var_anno = varAnnotation(
    value = anno_var_hist(size = grid::unit(15, "mm")),
    log10p = anno_var_box(function(x) log10(x + 1))
  )
)

# You can change the colour scheme used, using heat_palette()
cor_heatmap(
  data = psq, taxa = taxa, tax_anno = taxAnno,
  colors = heat_palette("Green-Orange", rev = TRUE, sym = TRUE)
)

# You can hide or change the style of the numbers with heat_numbers()
cor_heatmap(data = psq, taxa = taxa, tax_anno = taxAnno, numbers = NULL)
cor_heatmap(
  data = psq, taxa = taxa, tax_anno = taxAnno,
  colors = heat_palette("Berlin", rev = TRUE, sym = TRUE),
  numbers = heat_numbers(decimals = 2, col = "white", fontface = "bold")
)

# You can hide or change the style of the grid lines with grid_col & grid_lwd
```

```
cor_heatmap(psq, taxa = taxa, tax_anno = taxAnno, grid_col = NA) # hidden
cor_heatmap(psq, taxa = taxa, tax_anno = taxAnno, grid_lwd = 3) # bigger

# You can pass any other argument from `ComplexHeatmap::Heatmap()` to `...`

# e.g. You can set the absolute width and height of the heatmap body
cor_heatmap(
  data = psq, taxa = taxa, tax_anno = taxAnno,
  width = grid::unit(40, "mm"), height = grid::unit(10, "cm")
)

# e.g. You can suppress the legend
cor_heatmap(
  data = psq, taxa = taxa, tax_anno = taxAnno, show_heatmap_legend = FALSE,
  width = grid::unit(40, "mm"), height = grid::unit(10, "cm")
)
```

cor_test *Simple wrapper around cor.test for y ~ x style formula input*

Description

Intended for use within the function tax_model

Usage

```
cor_test(formula, data, ...)
```

Arguments

formula	a formula in form y ~ x
data	dataframe
...	passed to cor.test

Examples

```
data("shao19")
ps <- shao19 %>%
  ps_filter(family_role == "mother") %>%
  tax_filter(min_prevalence = 20) %>%
  tax_agg("family")

cors <- ps %>% tax_model(
  rank = "family", variables = list("age", "number_reads"), type = cor_test
)

tax_models_get(cors)
```

deprecated-heatmap-annotations

DEPRECATED Heatmap annotations helpers

Description

Functions to easily define ComplexHeatmap annotations for taxa and/or variables

- tax_anno creates list describing taxa annotation (for cor_heatmap or comp_heatmap)
- var_anno creates list describing variable annotation (for cor_heatmap)

Usage

```
tax_anno(
  undetected = 0,
  which = NA,
  prev = 1,
  abund = 2,
  size = 30,
  gap = 2,
  rel_sizes = NA,
  args = NULL,
  ...
)

anno_prev(
  data,
  taxa,
  undetected = 0,
  which = "row",
  size = 15,
  bar_width = 0.6,
  gp = grid::gpar(fill = "grey85"),
  ...
)

anno_abund(
  data,
  taxa,
  undetected = 0,
  which = "row",
  size = 15,
  point_size = 0.75,
  box_width = 0.6,
  gp = grid::gpar(fill = "grey85"),
  ...
)
```

```
var_anno(
  annos = "var_box",
  funs = "identity",
  names = NA,
  which = "column",
  size = 15 * length(annos),
  gap = 2,
  rel_sizes = NA,
  args = NULL,
  ...
)

old_anno_var_hist(data, vars = NA, which = "column", size = 15, ...)

old_anno_var_box(data, vars = NA, which = "column", size = 15, ...)
```

Arguments

`undetected`	value above which taxa are considered present/detected in a sample
`which`	"row" or "column" annnotation
`prev`	order in which prevalence annotation shown (number, or NA to not show)
`abund`	order in which abundance annotation shown (number, or NA to not show)
`size`	total size (mm) of annotations (width/height depending on which)
`gap`	gap in mm between annotations
`rel_sizes`	relative sizes of annotations (NA for equal sizes, or same length as annos)
`args`	extra args passed to each annotation: give as list of lists (one inner list per arg, named, e.g. list(prev = list(whatever = whatever))
`...`	further named args to be passed on (to list)
`data`	phyloseq or ps-extra (or a data.frame or matrix for anno_var_* functions)
`taxa`	names of taxa to plot
`bar_width`	relative width of barchart bars
`gp`	a grid::gpar() object for graphics parameter settings like fill or lwd
`point_size`	size of outlier points in mm
`box_width`	relative width of boxplot boxes
`annos`	name(s) of annotation(s) to show, in order (e.g. 'var_box', 'var_hist')
`funs`	function(s) to transform matrix of variable values before plotting (length must be 1 or same length as annos)
`names`	names to use for each annotation in annos
`vars`	names of variables to plot

distinct_palette *Colour palettes suitable for 20+ categories*

Description

Available palettes (max colors) are "brewerPlus" (41), "kelly" (20) and "greenArmytage" (25).

- "brewerPlus" is an arbitrary expansion of the "Paired" and "Dark2" colorbrewer palettes. The philosophy behind this expansion was to ensure that similar colours are far apart, and the earlier colours are attractive.
- "kelly" is based on the 22-colour palette developed by Kenneth Kelly but with white and black starting colours removed. This palette is ordered such that the first colours are most distinct.
- "greenArmytage" is based on a 26-colour palette proposed by Paul Green-Armytage, with black removed. This palette is not ordered by maximum contrast.

Usage

```
distinct_palette(n = NA, pal = "brewerPlus", add = "lightgrey")
```

Arguments

n	number of colours to return
pal	palette name, one of "brewerPlus", "kelly", "greenArmytage"
add	colour to append to end of palette, as colour n+1, lightgrey by default for the use as "other" taxa in comp_barplot, or NA for no additional colour.

Details

Hex color codes for 'kelly' and 'greenArmytage' palettes are copied and slightly modified from the Polychrome R package: i.e. Polychrome::kelly.colors() and Polychrome::green.armytage.colors()

Please consider also citing Coombes 2019 doi:10.18637/jss.v090.c01 if you use either of these palettes.

See the Polychrome reference manual for more information: https://CRAN.R-project.org/package=Polychrome

Value

vector of colours

Examples

```
brewerPlus <- distinct_palette()
scales::show_col(brewerPlus)

kelly <- distinct_palette(pal = "kelly")
scales::show_col(kelly)
```

```
greenArmytage <- distinct_palette(pal = "greenArmytage")
scales::show_col(greenArmytage)
```

dist_bdisp *Wrapper for vegan::betadisper()*

Description

Takes the output of dist_calc function. Or use with the result of the permanova function to ensure the results correspond to exactly the same input data. Runs betadisper for all categorical variables in variables argument. See help('betadisper', package = 'vegan').

Usage

```
dist_bdisp(
  data,
  variables,
  method = c("centroid", "median")[[1]],
  complete_cases = TRUE,
  verbose = TRUE
)
```

Arguments

`data`	psExtra output from dist_calc
`variables`	list of variables to use as group
`method`	centroid or median
`complete_cases`	drop samples with NAs in any of the variables listed
`verbose`	sends messages about progress if true

Value

psExtra containing betadisper results

Examples

```
library(phyloseq)
library(vegan)
data("dietswap", package = "microbiome")

# add some missings to demonstrate automated removal
sample_data(dietswap)$sex[3:6] <- NA
# create a numeric variable to show it will be skipped with a warning
dietswap <- ps_mutate(dietswap, timepoint = as.numeric(timepoint))

# straight to the betadisp
```

```
bd1 <- dietswap %>%
  tax_agg("Genus") %>%
  dist_calc("aitchison") %>%
  dist_bdisp(variables = c("sex", "bmi_group", "timepoint")) %>%
  bdisp_get()
bd1$sex
# quick vegan plotting methods
plot(bd1$sex$model, label.cex = 0.5)
boxplot(bd1$sex$model)

# compute distance and use for both permanova and dist_bdisp
testDist <- dietswap %>%
  tax_agg("Genus") %>%
  dist_calc("bray")

PERM <- testDist %>%
  dist_permanova(
    variables = c("sex", "bmi_group"),
    n_processes = 1, n_perms = 99
  )
str(PERM, max.level = 1)

bd <- PERM %>% dist_bdisp(variables = c("sex", "bmi_group"))
bd
```

---

dist_calc — *Calculate distances between pairs of samples in phyloseq object*

Description

Can compute various sample-sample distances using the microbiota composition of your samples:

- Bray Curtis ('bray') or any other ecological distance from phyloseq::distance() / vegan::vegdist()
- UniFrac distances (using the GUniFrac package)
  - generalised: 'gunifrac' (optionally set weighting alpha in gunifrac alpha)
  - unweighted: 'unifrac'
  - weighted: 'wunifrac'
- Aitchison distance (Euclidean distance after centered log ratio transform clr, see details)
- Euclidean distance

Use dist_calc with psExtra output of tax_transform (or tax_agg). It returns a psExtra object containing the phyloseq and the name of the distance used in addition to the distance matrix itself. The resulting object is intended to be piped into ord_calc or dist_permanova functions. Alternatively you can directly access the distance matrix with dist_get().

Usage

```
dist_calc(data, dist = "bray", gunifrac_alpha = 0.5, ...)
```

Arguments

`data`	psExtra object, e.g. output from tax_transform()
`dist`	name of distance to calculate between pairs of samples
`gunifrac_alpha`	setting alpha value only relevant if gunifrac distance used
`...`	optional distance-specific named arguments passed to phyloseq::distance()

Value

psExtra object including distance matrix and name of distance used

Aitchison distance note

You should EITHER:

1. skip the dist_calc function and call ord_calc(method = "PCA") directly on an object with taxa transformed with tax_transform(trans = "clr")
2. pass an object with untransformed (or 'identity' transformed) taxa to the data argument of dist_calc() and specify dist = "aitchison".

If ordination plots with taxon loading vectors are desired, users require option 1. If the distance matrix is required for permanova, users require option 2.

Binary Jaccard distance note

Jaccard distance can be computed on abundances, but often in microbiome research it is the Binary Jaccard distance that is desired. So remember to first perform a "binary" transformation with `tax_transform("binary")`, OR pass an additional argument to `dist_calc("jaccard", binary = TRUE)`

See Also

`tax_transform` for the function to use before dist_calc

`ord_calc`

`ord_plot`

`dist_permanova`

`phyloseq::distance`

`vegan::vegdist`

Examples

```
# bray curtis distance on genera-level features
data("dietswap", package = "microbiome")
bc <- dietswap %>%
  tax_agg("Genus") %>%
  dist_calc("bray")
bc
class(bc)
```

```
# gunifrac distance using phyloseq input
data("esophagus", package = "phyloseq")
gunifrac <- esophagus %>%
  dist_calc("gunifrac") %>%
  dist_get()
class(gunifrac)
```

`dist_calc_seq` *Calculate distances between sequential samples in ps_extra/phyloseq object*

Description

Calculate distances between sequential samples in ps_extra/phyloseq object

Usage

```
dist_calc_seq(
  data,
  dist,
  group,
  seq,
  unequal = "warn",
  start_value = NaN,
  return = "data",
  var_name = paste0(dist, "_DistFromLast")
)
```

Arguments

`data`	psExtra object, e.g. output from tax_transform()
`dist`	name of distance to calculate between pairs of sequential samples
`group`	name of variable in phyloseq sample_data used to define groups of samples
`seq`	name of variable in phyloseq sample_data used to define order of samples within groups
`unequal`	"error" or "warn" or "ignore" if groups of samples, defined by group argument, are of unequal size
`start_value`	value returned for the first sample in each group, which has no preceding sample in the group's sequence, and so has no obvious value
`return`	format of return object: "data" returns psExtra with sorted samples and additional variable. "vector" returns only named vector of sequential distances.
`var_name`	name of variable created in psExtra if return arg = "data"

Value

psExtra object sorted and with new sequential distance variable or a named vector of that variable

See Also

dist_calc

Examples

```
library(ggplot2)
library(dplyr)
data("dietswap", package = "microbiome")

pseq <- dietswap %>%
  tax_transform("identity", rank = "Genus") %>%
  dist_calc_seq(
    dist = "aitchison", group = "subject", seq = "timepoint",
    # group sizes are unequal because some subjects are missing a timepoint
    unequal = "ignore"
  )

pseq %>%
  samdat_tbl() %>%
  dplyr::select(1, subject, timepoint, dplyr::last_col())

# ggplot heatmap - unsorted
pseq %>%
  samdat_tbl() %>%
  filter(timepoint != 1) %>%
  ggplot(aes(x = timepoint, y = subject)) +
  geom_tile(aes(fill = aitchison_DistFromLast)) +
  scale_fill_viridis_c(na.value = NA, name = "dist") +
  theme_minimal(base_line_size = NA) +
  scale_y_discrete(limits = rev(levels(samdat_tbl(pseq)$subject)))

# ComplexHeatmap plotting with clustering #
library(tidyr)
library(ComplexHeatmap)

# make data matrix
heatmat <- pseq %>%
  samdat_tbl() %>%
  filter(timepoint != 1) %>%
  pivot_wider(
    id_cols = subject,
    names_from = timepoint, names_prefix = "t",
    values_from = aitchison_DistFromLast
  ) %>%
  tibble::column_to_rownames("subject")

heatmat <- as.matrix.data.frame(heatmat)

heatmap <- Heatmap(
  name = "dist",
  matrix = heatmat, col = viridisLite::viridis(12), na_col = "white",
  cluster_columns = FALSE,
  cluster_rows = hclust(dist(heatmat), method = "ward.D"),
  width = unit(1.5, "in"), rect_gp = gpar(col = "black"),
  row_names_side = "left", row_names_gp = gpar(fontsize = 8)
)
heatmap

# comparison with subject tracking on PCA
pca <- pseq %>%
  # already sorted data
  dist_calc("aitchison") %>%
  ord_calc("PCoA") %>%
  ord_plot(alpha = 0.1, shape = "nationality", size = 2) %>%
  add_paths(
    mapping = aes(colour = subject, alpha = timepoint, size = timepoint),
    id_var = "subject", id_values = c(
      "eve", "hsf", # low variation
      "vem", # medium
      "ufm", # high variation
      "pku" # starts high
    )
  ) +
  scale_alpha_continuous(range = c(0.3, 1), breaks = c(2, 4, 6)) +
  scale_size_continuous(range = c(1, 2), breaks = c(2, 4, 6))

heatmap
pca
```

`dist_permanova` *Calculate PERMANOVA after dist_calc()*

Description

`dist_permanova` runs a Permutational Multivariate ANOVA (aka Non-parametric MANOVA). This is a way to test for the statistical significance of (independent) associations between variables in your phyloseq::sample_data(), and a microbiota distance matrix you have already calculated with dist_calc().

This function is a wrapper around vegan's `adonis2()` function. See `?vegan::adonis2()` for more insight.

You can also read this excellent book chapter on PERMANOVA by Marti Anderson: doi:10.1002/9781118445112.stat07841

Or this NPMANOVA page on GUSTA ME: `https://sites.google.com/site/mb3gustame/hypothesis-tests/manova/npmanova`

Usage

```
dist_permanova(
  data,
  variables = NULL,
  interactions = NULL,
  complete_cases = TRUE,
  n_processes = 1,
  n_perms = 999,
  seed = NULL,
  by = "margin",
  verbose = TRUE,
  ...
)
```

Arguments

`data`	psExtra output from dist_calc()
`variables`	character vector of variables to include in model or character representation of the right-hand side of a formula, e.g "varA + varB + varA:varB"
`interactions`	optional argument to define any interactions between variables, written in the style of e.g. "var_a * var_b"
`complete_cases`	if TRUE, drops observations if they contain missing values (otherwise stops if missings are detected)
`n_processes`	how many parallel processes to use? (on windows this uses parallel::makePSOCKcluster())
`n_perms`	how many permutations? e.g. 9999. Less is faster but more is better!
`seed`	set a random number generator seed to ensure you get the same results each run
`by`	passed to vegan::adonis2() by argument: what type of sums of squares to calculate? "margin" or "terms"
`verbose`	sends messages about progress if TRUE
`...`	additional arguments are passed directly to vegan::adonis2() (e.g. strata, add, sqrt.dist etc.)

Details

The variables argument will be collapsed into one string (if length > 1) by pasting together, separated by "+". Any interaction terms described in the interactions argument will be pasted onto the end of the pasted variables argument. Alternatively, you can supply the complete right hand side of the formula yourself e.g variables = "varA + varB + varC*varD"

Watch out, if any of your variable names contain characters that would normally separate variables in a formula then you should rename the offending variable (e.g. avoid any of "+" "*" "|" or ":") otherwise permanova will split that variable into pieces.

Value

psExtra list containing permanova results and (filtered) input objects

See Also

`dist_calc` for calculating the required distance matrix input

`ord_plot` with constraints as a way to visualise the microbial associations of significant predictors

`vegan::adonis2`

Examples

```
data("dietswap", package = "microbiome")

# add some missings to demonstrate automated removal
phyloseq::sample_data(dietswap)$sex[3:6] <- NA

# compute distance
testDist <- dietswap %>%
  tax_agg("Genus") %>%
  tax_transform("identity") %>%
  dist_calc("bray")

PERM <- testDist %>%
  dist_permanova(
    seed = 1,
    variables = c("sex", "bmi_group"),
    n_processes = 1,
    n_perms = 99 # only 99 perms used in examples for speed (use 9999+!)
  )
PERM
str(PERM, max.level = 1)

# try permanova with interaction terms
PERM2 <- testDist %>%
  dist_permanova(
    seed = 1,
    variables = "nationality + sex * bmi_group",
    n_processes = 1, n_perms = 99
  )
perm_get(PERM2)

# specify the same model in alternative way
PERM3 <- testDist %>%
  dist_permanova(
    seed = 1,
    variables = c("nationality", "sex", "bmi_group"),
    interactions = "sex * bmi_group",
    n_processes = 1, n_perms = 99
  )
perm_get(PERM3)

identical(PERM3, PERM2) # TRUE

# take same distance matrix used for the permanova and plot an ordination
PERM2 %>%
  ord_calc(method = "PCoA") %>%
  ord_plot(color = "bmi_group")
# this trick ensures any samples dropped from the permanova
# for having missing values in the covariates are NOT included
# in the corresponding ordination plot
```

`heat_grid` *set options for drawing gridlines on heatmaps*

Description

set options for drawing gridlines on heatmaps

Usage

```
heat_grid(
  col = "white",
  alpha = 1,
  lty = 1,
  lwd = 0.5,
  lex = 1,
  lineend = "round",
  linejoin = "round"
)
```

Arguments

`col`	Colour for lines and borders.
`alpha`	Alpha channel for transparency
`lty`	Line type
`lwd`	Line width
`lex`	Multiplier applied to line width
`lineend`	Line end style (round, butt, square)
`linejoin`	Line join style (round, mitre, bevel)

`heat_numbers` *Aesthetic settings for drawing numbers on heatmap tiles*

Description

Works with comp_heatmap() and cor_heatmap(). See the help for those functions.

Usage

```
heat_numbers(
  decimals = 0,
  fontsize = 7,
  col = "darkgrey",
  fontface = "bold",
  fmt = NULL,
  ...
)
```

Arguments

`decimals`	number of decimal places to print
`fontsize`	fontsize specification,
`col`	colour of font
`fontface`	plain, bold, italic
`fmt`	NULL or number print format, see ?sprintf, overrides decimals arg if set
`...`	passed to grid::gpar() for grid.text

Value

list

`heat_palette` *Easy palettes for ComplexHeatmap*

Description

Pass a named colorspace hcl palette to circlize::colorRamp2.

- If you do not specify a range this function returns a function and the heatmap color palette will use the range of the data automatically
- If you do specify a range, this returns a colour palette with that range

Usage

```
heat_palette(
  palette = ifelse(sym, "Blue-Red 3", "Rocket"),
  breaks = "auto",
  range = NA,
  sym = FALSE,
  rev = FALSE
)
```

Arguments

`palette`	named palette from colorspace::hcl_palettes() diverging/sequential or a vector of colour names/hexcodes
`breaks`	number of breaks, "auto" is 11 for a named palette, or uses palette length
`range`	NA to return palette generating function that takes range or numeric vector indicating the range, to return a palette
`sym`	makes palette range symmetrical around 0 if TRUE
`rev`	reverse the palette?

Value

circlize::colorRamp2 palette if range = NA, or function returning a palette when given a range

---

`ibd` *IBD study data in phyloseq object.*

Description

A phyloseq object with an OTU table and sample data from an IBD microbiome study. Originally released as ibd example data in the corncob package.

Usage

```
ibd
```

Format

A phyloseq-class experiment-level object with an OTU table and sample data.

References

Papa, E., Docktor, M., Smillie, C., Weber, S., Preheim, S. P., Gevers, D., Giannoukos, G., Ciulla, D., Tabbaa, D., Ingram, J., Schauer, D. B., Ward, D. V., Korzenik, J. R., Xavier, R. J., Bousvaros, A., Alm, E. J. & Schauer, D. B. (2012). *Non-invasive mapping of the gastrointestinal microbiota identifies children with inflammatory bowel disease*. PloS One, 7(6), e39242. <doi.org/10.1371/journal.pone.0039242>.

Duvallet, C., Gibbons, S., Gurry, T., Irizarry, R., & Alm, E. (2017). *MicrobiomeHD: the human gut microbiome in health and disease [Data set]*. Zenodo. <doi.org/10.5281/zenodo.1146764>.

`microViz` *microViz: microbiome data analysis and visualization*

Description

microViz provides functions for statistics and visualization of microbiome sequencing data. microViz wraps, extends and complements popular microbial ecology packages like phyloseq, vegan, and microbiome.

Check out the website for tutorials and illustrated help pages.

`https://david-barnett.github.io/microViz/`

Author(s)

David Barnett (ORCID) (GitHub)

See Also

Useful links:

- `https://david-barnett.github.io/microViz`
- `https://github.com/david-barnett/microViz`

`Ordination-arrows` *Create ordination plot vector styling lists*

Description

Used by ord_plot, see examples there.

Usage

```
vec_constraint(
  linewidth = 1,
  alpha = 0.8,
  colour = "brown",
 arrow = grid::arrow(length = grid::unit(0.005, units = "npc"), type = "closed", angle =
    30),
  lineend = "round",
  linejoin = "mitre",
  ...
)

vec_tax_sel(
  linewidth = 0.5,
  alpha = 1,
  colour = "black",
 arrow = grid::arrow(length = grid::unit(0.005, units = "npc"), type = "closed", angle =
    30),
  lineend = "round",
  linejoin = "mitre",
  ...
)

vec_tax_all(linewidth = 0.5, alpha = 0.25, arrow = NULL, ...)
```

Arguments

`linewidth`	width of vector
`alpha`	opacity of vector
`colour`	colour of vector
`arrow`	arrow style specified with grid::arrow() or NULL for no arrow
`lineend`	Line end style (round, butt, square).
`linejoin`	Line join style (round, mitre, bevel).
`...`	further arguments passed to geom_segment

Value

list

`Ordination-labels` *Create list for ord_plot() *_lab_style arguments*

Description

Customise taxa and constraint labels on your ordination plots. Choose 'text' or 'label' type, rotate and/or justify the text/labels and set aesthetic appearances using `tax_lab_style()` or `constraint_lab_style()`.

Usage

```
tax_lab_style(
  type = "label",
  max_angle = 0,
  perpendicular = FALSE,
  aspect_ratio = 1,
  justify = "auto",
  size = 2,
  alpha = 1,
  colour = "black",
  ...
)

constraint_lab_style(
  type = "label",
  max_angle = 0,
  perpendicular = FALSE,
  aspect_ratio = 1,
  justify = "auto",
  size = 2.5,
  alpha = 1,
  colour = "brown",
  ...
)
```

Arguments

`type`	'label', 'text' or 'richtext' ('richtext' also used if 'label' type are rotated, when max_angle > 0)
`max_angle`	maximum angle of rotation to allow to match vector angle (requires ggtext package to rotate "label" type)
`perpendicular`	if TRUE, sets rotated labels perpendicular to desired angle, not parallel
`aspect_ratio`	aspect ratio of plot (y/x) must also be used in coord_fixed() ratio argument (must be set when rotated labels are used, to ensure match to arrow angles)
`justify`	"center", "side", or "auto"? Should the text/label align with the arrows at the text center or text sides (uses hjust, if 'auto', picks based on whether max_angle is greater than 0)

`size`	fixed size of text or label
`alpha`	fixed alpha of text or label
`colour`	fixed colour of text or label
`...`	further named arguments passed to geom_text, geom_label or geom_richtext

Value

named list

Examples

```
# These examples show styling of taxa labels with tax_lab_style().
# The same options are available for constraint labels in constrained
# ordinations. constraint_lab_style() just has different default settings.

library(ggplot2)

# get example inflammatory bowel disease stool dataset from corncob package
data("ibd", package = "microViz")

# filter out rare taxa and clean up names etc
ibd <- ibd %>%
  tax_filter(min_prevalence = 3) %>%
  tax_fix() %>%
  phyloseq_validate()

# calculate a centered-log-ratio transformed PCA ordination
ibd_ord <- ibd %>%
  tax_transform("clr", rank = "Genus") %>%
  ord_calc("PCA")

# basic plot with default label style
ibd_ord %>% ord_plot(color = "ibd", plot_taxa = 1:10)

# Rotating labels: requires the ggtext package #
# A fixed coordinate ratio must be set to ensure label rotation
# matches the vectors. It is also helpful to set the vector and label length
# multipliers manually for a good look. Rotated labels are justified to the
# 'sides' automatically by tax_lab_style() with justify = 'auto'
ibd_ord %>%
  ord_plot(
    color = "ibd", plot_taxa = 1:7,
    tax_vec_length = 1.3, tax_lab_length = 1.3,
    tax_lab_style = tax_lab_style(max_angle = 90)
  ) +
  coord_fixed(ratio = 1, clip = "off", xlim = c(-3.5, 3.5))

# You can use text instead of labels
# - a bold fontface helps text to stand out
# - see ?ggplot2::geom_text for all settings available
ibd_ord %>%
  ord_plot(
    color = "ibd", plot_taxa = 1:7,
    tax_vec_length = 1.3, tax_lab_length = 1.4,
    tax_lab_style = tax_lab_style(
      type = "text", max_angle = 90, size = 2.5, fontface = "bold.italic"
    )
  ) +
  coord_fixed(ratio = 1, clip = "off", xlim = c(-3.5, 3.5))

# With text you can prevent overlaps with check_overlap = TRUE
ibd_ord %>%
  ord_plot(
    color = "ibd", plot_taxa = 1:12,
    tax_vec_length = 1.3, tax_lab_length = 1.4,
    tax_lab_style = tax_lab_style(
      type = "text", max_angle = 90, size = 3, fontface = "bold.italic",
      check_overlap = TRUE
    )
  ) +
  coord_fixed(ratio = 1, clip = "off", xlim = c(-3.5, 3.5))

# With labels, you can reduce the padding and line weight to free space
# but check_overlap is not available
# see ?ggtext::geom_richtext for more possibilities
ibd_ord %>%
  ord_plot(
    color = "ibd", plot_taxa = 1:7,
    tax_vec_length = 1.3, tax_lab_length = 1.35,
    tax_lab_style = tax_lab_style(
      max_angle = 90, fontface = "italic", size = 2.5, fill = "grey95",
      label.size = 0.1, # width outline
      label.padding = unit(0.1, "lines"),
      label.r = unit(0, "lines") # reduces rounding of corners to radius 0
    )
  ) +
  coord_fixed(ratio = 1, clip = "off", xlim = c(-3.5, 3.5))

# Perpendicular angled labels/text are possible
ibd_ord %>%
  ord_plot(
    color = "ibd", plot_taxa = 1:12,
    tax_lab_style = tax_lab_style(
      type = "text", max_angle = 90, perpendicular = TRUE, size = 3,
      check_overlap = TRUE
    )
  ) +
  coord_fixed(ratio = 1, clip = "off", xlim = c(-3.5, 3.5))


# You can limit and/or attenuate the angle of rotation by:
#  - setting a lower max_angle
#  - decreasing the aspect_ratio in the tax_lab_style call
ibd_ord %>%
  ord_plot(
    shape = "circle", color = "ibd", plot_taxa = 1:7,
    tax_vec_length = 1.3, tax_lab_length = 1.3,
    tax_lab_style = tax_lab_style(
      max_angle = 10, size = 2, label.size = 0.1,
      label.padding = unit(0.1, "lines"), label.r = unit(0, "lines")
    )
  ) +
  coord_fixed(ratio = 1, clip = "off", xlim = c(-3.5, 3.5))

ibd_ord %>%
  ord_plot(
    shape = "circle", color = "ibd", plot_taxa = 1:7,
    tax_vec_length = 1.3, tax_lab_length = 1.3,
    tax_lab_style = tax_lab_style(
      max_angle = 90, size = 2, label.size = 0.1, aspect_ratio = 0.5,
      label.padding = unit(0.1, "lines"), label.r = unit(0, "lines")
    )
  ) +
  coord_fixed(ratio = 1, clip = "off", xlim = c(-3.5, 3.5))

# another example with some extras #
ibd_ord %>%
  ord_plot(
    shape = "circle filled", fill = "ibd",
    plot_taxa = 1:10,
    taxon_renamer = function(x) stringr::str_replace_all(x, "_", " "),
    tax_vec_length = 2, tax_lab_length = 2.1,
    tax_lab_style = tax_lab_style(
      type = "text", max_angle = 90, size = 2.5,
      fontface = "bold.italic", check_overlap = TRUE
    )
  ) +
  coord_fixed(1, clip = "off", xlim = c(-5, 5)) +
  theme(legend.position = c(0.8, 0.2), legend.background = element_rect()) +
  stat_chull(mapping = aes(colour = ibd, fill = ibd), alpha = 0.1)
```

`ord_calc` *Ordinate samples (arrange by similarity in multiple dimensions)*

Description

Used before plotting with ord_plot() or explorating interactively with ord_explore(). Use method = "auto" to automatically pick an appropriate method from:

- "PCA" (Principal Components Analysis) combines taxa abundances into new dimensions. The first axes display the greatest variation in your microbial data.
- "RDA" (Redundancy Analysis) is constrained PCA, roughly speaking. It finds variation in your data that can be explained both by the constraints variables, and the microbial data.

- "PCoA" (Principal Coordinates Analysis) finds a coordinate system that best preserves the original distances between samples.
- "CAP" (Constrained Analysis of Principal Coordinates) is also known as distance-based Redundancy Analysis.

Alternatively to leaving method = "auto", you can explicitly specify any of the above methods, or choose one of the following:

- "CCA" (Canonical Correspondence Analysis) - NOT canonical correlation analysis!
- "NMDS" (Non-metric Multidimensional Scaling)

You are strongly recommended to check out this useful website for introductory explanations of these methods the "GUide to STatistical Analysis in Microbial Ecology": https://sites.google.com/site/mb3gustame/

Usage

```
ord_calc(
  data,
  method = "auto",
  constraints = NULL,
  conditions = NULL,
  scale_cc = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`data`	psExtra object: output from dist_calc(), or tax_transform() if no distance calculation required for method e.g. for RDA
`method`	which ordination method to use? "auto" means automatically determined from psExtra and other args. If you really know what you want: manually set one of ’PCoA’, ’PCA’, ’CCA’, ’CAP’ or ’RDA’
`constraints`	(a vector of) valid sample_data name(s) to constrain analyses, or leave as NULL for unconstrained ordination. Non-NULL values are compatible with method = "auto"/"RDA"/"CAP"
`conditions`	(a vector of) valid sample_data name(s) to partial these out of analyses with Condition(), or leave as NULL
`scale_cc`	If TRUE (default) ensures any constraints and conditions variables are scaled before use, to ensure their effects are comparable. If set to FALSE you must ensure you have already set the variables on a similar scale yourself! If there are no constraints or conditions, scale_cc does nothing.
`verbose`	If TRUE or "max", show any warnings and messages about constraint and conditions scaling and missings etc. FALSE suppresses warnings!
`...`	optional arguments passed on to phyloseq::ordinate()

Details

Extends functionality of phyloseq::ordinate(). Results can be used directly in ord_plot(). You can extract the ordination object for other analyses with ord_get()

Value

psExtra object

See Also

`dist_calc` for distance matrix calculation

`ord_plot` and `ord_explore`

`phyloseq` `ordinate`

Examples

```
library(phyloseq)
library(vegan)
data("dietswap", package = "microbiome")

# create a couple of numerical variables to use as constraints
dietswap <- ps_mutate(
  dietswap,
  female = dplyr::if_else(sex == "female", true = 1, false = 0),
  weight = dplyr::recode(bmi_group, obese = 3, overweight = 2, lean = 1)
)

# add a couple of missing values to demo automated dropping of observations with missings
sample_data(dietswap)$female[c(3, 4)] <- NA

# compute ordination
test <- dietswap %>%
  tax_agg("Genus") %>%
  dist_calc("bray") %>%
  ord_calc(constraints = c("weight", "female"))

# familiarise yourself with the structure of the returned psExtra object
test
str(test, max.level = 1)

# compute RDA with centre-log-ratio transformed taxa
test2 <- dietswap %>%
  tax_agg("Genus") %>%
  tax_transform("clr") %>%
  ord_calc(constraints = c("weight", "female"))

# plot with vegan package graphics to show it returns a standard ordination object
ord_get(test2) %>% vegan::ordiplot()

# This is equivalent to CAP with "aitchison" distance
ord_plot(test2, plot_taxa = 8:1)
```

```
# but the latter (below) doesn't allow plotting taxa loadings with ord_plot
dietswap %>%
  tax_agg("Genus") %>%
  dist_calc("aitchison") %>%
  ord_calc(constraints = c("weight", "female")) %>%
  ord_plot()
```

`ord_explore` *Interactively explore microbial compositions of ordinated samples*

Description

A Shiny app used to create and explore an interactive version of `ord_plot()`. You can select samples on an ordination plot to view their composition with stacked barplots.

The `ord_explore()` data argument takes either:

- the output of `ord_calc()` (i.e. a psExtra with an ordination)
- a plain phyloseq object: `ord_explore()` will help you build an ordination

Once the app is running (in your browser), you can:

1. Create/edit the ordination if required
    - look at the R console error messages if your chosen options don't build
2. Style the ordination plot (e.g. choose dimensions; set colour and size; ...)
    - Taxa loading arrows can be added only to PCA, RDA and CCA plots
    - Convex hulls or ellipses can only be drawn if Colour is set to a variable
    - To track individuals over time with the path plotter, your data MUST already be sorted by time (e.g. with ps_arrange)!
3. Click on or use the lasso tool to select 1 or more samples to view their compositions
    - By default samples can be selected individually
    - Set the "Select" option to another variable to select by level of that variable
4. Style the taxonomic compositions barplot
    - The samples are ordered using the seriate_method argument and the same transformation and distance as used in the ordination plot
    - The app may lag if you select 100s of samples and ungroup the "Other" category
    - To avoid this lag: either reduce the number of taxa or samples, or deselect "Interactive" barplot
5. Stop the app by clicking the red stop button in the R console
    - Closing the web browser window doesn't stop the app, (you can find the app again at the local http address shown in the R console)
    - Don't forget to copy the ordination plot code before you close the app

See the Details section for some known limitations of the app. Please report any other app problems on the microViz GitHub issues page.

Usage

```
ord_explore(
  data,
  sample_id = NULL,
  seriate_method = "OLO_ward",
  app_options = list(launch.browser = TRUE),
  plot_widths = c(7, 9),
  modal_fade = TRUE,
  notification_durations = list(2, 20),
  ...
)
```

Arguments

`data`	a phyloseq, or the psExtra output of ord_calc
`sample_id`	name of sample ID variable to use as default for selecting samples
`seriate_method`	seriation method to order phyloseq samples by similarity
`app_options`	passed to shinyApp() options argument
`plot_widths`	widths of plots in inches, including any legends (first number is ordination, second is composition barplot)
`modal_fade`	should the popover menus (modals) have a fade animation?
`notification_durations`	length 2 list giving duration in seconds of short and long notifications or NULL for notifications that do not disappear automatically
`...`	additional arguments passed to ord_plot

Details

Limitations:

- If a "Select:" grouping variable is NA for some samples, then that grouping variable cannot be used to select those samples
- "Shape:" can only be mapped to variables with maximum 5 distinct levels, not including NAs. NAs in the shape variable are shown as hollow circles.

On some web browsers, e.g. older versions of Firefox, the numeric inputs' buttons are sometimes hard to click. As a workaround, click the box and type a number or use the arrow keys. This problem occurs in all Shiny apps, not just microViz.

Value

nothing, opens default browser

Examples

```
# example code only runs in interactive R session
if (interactive()) {
  library(phyloseq)
  library(dplyr)

  # example of quickstart approach with interactive ordination calculation #
  microViz::ibd %>%
    # filtering makes subsequent calculations faster
    tax_filter(min_prevalence = 2) %>%
    tax_fix() %>%
    ord_explore()

  # simple example with precalculated ordination #
  data("enterotype")
  taxa_names(enterotype)[1] <- "unclassified" # replaces the "-1" taxon name
  ps <- tax_fix(enterotype) # remove NA taxa
  ord1 <- ps %>%
    tax_transform("identity", rank = "Genus") %>%
    dist_calc("bray") %>%
    ord_calc(method = "PCoA")

  ord_explore(data = ord1, auto_caption = 6)

  # constrained ordination example #
  data("dietswap", package = "microbiome")

  # create a couple of numerical variables to use as constraints
  dietswap <- dietswap %>%
    ps_mutate(
      weight = recode(bmi_group, obese = 3, overweight = 2, lean = 1),
      female = if_else(sex == "female", true = 1, false = 0)
    ) %>%
    tax_agg("Genus")

  constrained_aitchison_rda <- dietswap %>%
    tax_transform("clr") %>%
    ord_calc(constraints = c("weight", "female"))

  # label style arguments can be passed to ord_explore
  constrained_aitchison_rda %>%
    ord_explore(
      tax_lab_style = list(size = 3),
      constraint_lab_style = list(size = 4), auto_caption = 6
    )
  # Try changing the point colour to bmi_group or similar
  # Style points interactively!
  # (setting colour/shape/etc as arguments doesn't work)

  # dietswap is actually a longitudinal dataset, with multiple samples per
  # subject. If we arrange by timepoint first (!!!), we can use the "paths"
  # additional plot layer from the ord_explore "Add:" menu to track
```

```
    # individual subjects over time.
    dietswap %>%
      ps_arrange(timepoint) %>%
      tax_fix() %>%
      ord_explore()


    # Another dataset, where "size" variable drives gradient on PC1
    # Try setting size and/or alpha to correspond to "size"!
    # Then edit the ordination to use "size" as a condition, see what happens
    # hmp2 <- microbiomeutilities::hmp2
    hmp2 %>%
      tax_fix() %>%
      tax_transform(rank = "Genus", "identity") %>%
      dist_calc("aitchison") %>%
      ord_calc() %>%
      ord_explore()

    # another dataset
    data("soilrep", package = "phyloseq")
    # test auto creation of SAMPLE var
    ps <- soilrep %>% ps_select(-Sample)
    # The barplot is actually quite useless with the 16000+ anonymous OTUs
    # in this dataset, but the 1000s of unmerged "Other" categories do render
    phyloseq_validate(ps) %>%
      tax_fix() %>%
      dist_calc("aitchison") %>%
      ord_calc() %>%
      ord_explore()
  }
```

ord_plot — *Customisable ggplot2 ordination plot*

Description

Draw ordination plot. Utilises psExtra object produced by of `ord_calc`.

- For an extensive tutorial see the ordination vignette.
- For interpretation see the the relevant pages on PCA, PCoA, RDA, or CCA on the GUide to STatistical Analysis in Microbial Ecology (GUSTA ME) website: `https://sites.google.com/site/mb3gustame/`

Usage

```
ord_plot(
  data,
  axes = 1:2,
  plot_taxa = FALSE,
  tax_vec_length = NA,
  tax_vec_style_all = vec_tax_all(),
  tax_vec_style_sel = vec_tax_sel(),
  tax_lab_length = tax_vec_length * 1.1,
  tax_lab_style = list(),
  taxon_renamer = function(x) identity(x),
  constraint_vec_length = NA,
  constraint_vec_style = vec_constraint(),
  constraint_lab_length = constraint_vec_length * 1.1,
  constraint_lab_style = list(),
  var_renamer = function(x) identity(x),
  plot_samples = TRUE,
  scaling = 2,
  auto_caption = 8,
  center = FALSE,
  clip = "off",
  expand = !center,
  interactive = FALSE,
  ...
)
```

Arguments

`data`	psExtra object with ordination attached, i.e. output from ord_calc
`axes`	which axes to plot: numerical vector of length 2, e.g. 1:2 or c(3,5)
`plot_taxa`	if ord_calc method was "PCA/RDA" draw the taxa loading vectors (see details)
`tax_vec_length`	taxon arrow vector scale multiplier. NA = auto-scaling, or provide a numeric multiplier yourself.
`tax_vec_style_all`	list of named aesthetic attributes for all (background) taxon vectors
`tax_vec_style_sel`	list of named aesthetic attributes for taxon vectors for the taxa selected by plot_taxa
`tax_lab_length`	scale multiplier for label distance/position for any selected taxa
`tax_lab_style`	list of style options for the taxon labels, see tax_lab_style() function.
`taxon_renamer`	function that takes any plotted taxon names and returns modified names for labels
`constraint_vec_length`	constraint arrow vector scale multiplier. NA = auto-scaling, or provide a numeric multiplier yourself.
`constraint_vec_style`	list of aesthetics/arguments (colour, alpha etc) for the constraint vectors
`constraint_lab_length`	label distance/position for any constraints (relative to default position which is proportional to correlations with each axis)
`constraint_lab_style`	list of aesthetics/arguments (colour, size etc) for the constraint labels

`var_renamer`	function to rename constraining variables for plotting their labels
`plot_samples`	if TRUE, plot sample points with geom_point
`scaling`	Type 2, or type 1 scaling. For more info, see `https://sites.google.com/site/mb3gustame/constrained-analyses/redundancy-analysis`. Either "species" or "site" scores are scaled by (proportional) eigenvalues, and the other set of scores is left unscaled (from ?vegan::scores.cca)
`auto_caption`	size of caption with info about the ordination, NA for none
`center`	expand plot limits to center around origin point (0,0)
`clip`	clipping of labels that extend outside plot limits?
`expand`	expand plot limits a little bit further than data range?
`interactive`	creates plot suitable for use with ggiraph (used in ord_explore)
`...`	pass aesthetics arguments for sample points, drawn with geom_point using aes_string

Details

How to specify the plot_taxa argument (when using PCA, CCA or RDA):

- FALSE –> plot no taxa vectors or labels
- integer vector e.g. 1:3 –> plot labels for top 3 taxa (by longest line length)
- single numeric value e.g. 0.75 –> plot labels for taxa with line length > 0.75
- character vector e.g. c('g__Bacteroides', 'g__Veillonella') –> plot labels for the exactly named taxa

Value

ggplot

See Also

`tax_lab_style` / `tax_lab_style` for styling labels

`ord_explore` for interactive ordination plots

`ord_calc` for calculating an ordination to plot with ord_plot

Examples

```
library(ggplot2)
data("dietswap", package = "microbiome")

# create a couple of numerical variables to use as constraints or conditions
dietswap <- dietswap %>%
  ps_mutate(
    weight = dplyr::recode(bmi_group, obese = 3, overweight = 2, lean = 1),
    female = dplyr::if_else(sex == "female", true = 1, false = 0)
  )

# unconstrained PCA ordination
unconstrained_aitchison_pca <- dietswap %>%
  tax_transform("clr", rank = "Genus") %>%
  ord_calc() # method = "auto" --> picks PCA as no constraints or distances

unconstrained_aitchison_pca %>%
  ord_plot(colour = "bmi_group", plot_taxa = 1:5) +
  stat_ellipse(aes(linetype = bmi_group, colour = bmi_group))

# you can generate an interactive version of the plot by specifying
# interactive = TRUE, and passing a variable name to another argument
# called `data_id` which is required for interactive point selection
interactive_plot <- unconstrained_aitchison_pca %>%
  ord_plot(
    colour = "bmi_group", plot_taxa = 1:5,
    interactive = TRUE, data_id = "sample"
  )

# to start the html viewer, and allow selecting points, we must use a
# ggiraph function called girafe and set some options and css
ggiraph::girafe(
  ggobj = interactive_plot,
  options = list(
    ggiraph::opts_selection(
      css = ggiraph::girafe_css(
        css = "fill:orange;stroke:black;",
        point = "stroke-width:1.5px"
      ),
      type = "multiple", # this activates lasso selection (click top-right)
      only_shiny = FALSE # allows interactive plot outside of shiny app
    )
  )
)


# remove effect of weight with conditions arg
# scaling weight with scale_cc is not necessary as only 1 condition is used
dietswap %>%
  tax_transform("clr", rank = "Genus") %>%
  ord_calc(conditions = "weight", scale_cc = FALSE) %>%
  ord_plot(colour = "bmi_group") +
  stat_ellipse(aes(linetype = bmi_group, colour = bmi_group))

# alternatively, constrain variation on weight and female
constrained_aitchison_rda <- dietswap %>%
  tax_transform("clr", rank = "Genus") %>%
  ord_calc(constraints = c("weight", "female")) # constraints --> RDA

constrained_aitchison_rda %>%
  ord_plot(colour = "bmi_group", constraint_vec_length = 2) +
  stat_ellipse(aes(linetype = bmi_group, colour = bmi_group))

# ggplot allows additional customisation of the resulting plot
p <- constrained_aitchison_rda %>%
  ord_plot(colour = "bmi_group", plot_taxa = 1:3) +
  lims(x = c(-5, 6), y = c(-5, 5)) +
  scale_colour_brewer(palette = "Set1")

p + stat_ellipse(aes(linetype = bmi_group, colour = bmi_group))
p + stat_density2d(aes(colour = bmi_group))

# you can rename the taxa on the labels with any function that
# takes and modifies a character vector
constrained_aitchison_rda %>%
  ord_plot(
    colour = "bmi_group",
    plot_taxa = 1:3,
    taxon_renamer = function(x) stringr::str_extract(x, "^.")
  ) +
  lims(x = c(-5, 6), y = c(-5, 5)) +
  scale_colour_brewer(palette = "Set1")

# You can plot PCoA and constrained PCoA plots too.
# You don't typically need/want to use transformed taxa variables for PCoA
# But it is good practice to call tax_transform("identity") so that
# the automatic caption can record that no transformation was applied
dietswap %>%
  tax_agg("Genus") %>%
  tax_transform("identity") %>%
  # so caption can record (lack of) transform
  dist_calc("bray") %>%
  # bray curtis
  ord_calc() %>%
  # guesses you want an unconstrained PCoA
  ord_plot(colour = "bmi_group")

# it is possible to facet these plots
# (although I'm not sure it makes sense to)
# but only unconstrained ordination plots and with plot_taxa = FALSE
unconstrained_aitchison_pca %>%
  ord_plot(color = "sex", auto_caption = NA) +
  facet_wrap("sex") +
  theme(line = element_blank()) +
  stat_density2d(aes(colour = sex)) +
  guides(colour = "none")

unconstrained_aitchison_pca %>%
  ord_plot(color = "bmi_group", plot_samples = FALSE, auto_caption = NA) +
  facet_wrap("sex") +
  theme(line = element_blank(), axis.text = element_blank()) +
  stat_density2d_filled(show.legend = FALSE) +
  geom_point(size = 1, shape = 21, colour = "black", fill = "white")
```

ord_plot_iris *Circular compositional barplot sorted by ordination angle*

Description

Use with `ord_calc` output as data argument. Order of samples extracted from ordination axes in data. Best paired with ordination plot made from same `ord_calc` output.

Usage

```
ord_plot_iris(
  data,
  tax_level,
  axes = 1:2,
  n_taxa = 10,
  ord_plot = "none",
  taxon_renamer = function(x) identity(x),
  palette = distinct_palette(n_taxa),
  anno_colour = NULL,
  anno_colour_style = list(),
  anno_binary = NULL,
  anno_binary_style = list(),
  keep_all_vars = FALSE,
  scaling = 2,
  count_warn = TRUE,
  ...
)
```

Arguments

`data`	psExtra output of ord_calc
`tax_level`	taxonomic aggregation level (from rank_names(ps))
`axes`	which 2 axes of ordination to use for ordering bars
`n_taxa`	how many taxa to colour show distinct colours for (all other taxa grouped into "other").
`ord_plot`	add a matching ordination plot to your iris plot ('list' returns separate plots in a list, 'above'/'below' uses patchwork to pair plots together into one)
`taxon_renamer`	function to rename taxa in the legend
`palette`	colour palette
`anno_colour`	name of sample_data variable to use for colouring geom_segment annotation ring
`anno_colour_style`	list of further arguments passed to geom_segment e.g. size
`anno_binary`	name(s) of binary sample_data variable(s) (levels T/F or 1/0) to use for filtered geom_point annotation ring(s) (annotates at TRUE values)
`anno_binary_style`	list of further arguments passed to geom_point e.g. colour, size, y, etc.
`keep_all_vars`	slows down processing but is required for any post-hoc plot customisation options

`scaling`	Type 2, or type 1 scaling. For more info, see https://sites.google.com/site/mb3gustame/constrained-analyses/redundancy-analysis. Either "species" or "site" scores are scaled by (proportional) eigenvalues, and the other set of scores is left unscaled (from ?vegan::scores.cca)
`count_warn`	warn if count data are not available? i.e. phyloseq otu_table is not positive integers and psExtra counts slot is NULL
`...`	Arguments passed on to `comp_barplot`
	`merge_other` if FALSE, taxa coloured/filled as "other" remain distinct, and so can have bar outlines drawn around them
	`bar_width` default 1 avoids random gapping otherwise seen with many samples (set to less than 1 to introduce gaps between samples)
	`bar_outline_colour` line colour separating taxa and samples (use NA for no outlines)
	`bar_outline_width` width of line separating taxa and samples (for no outlines set bar_outline_colour = NA)
	`tax_transform_for_plot` default "compositional" draws proportions of total counts per sample, but you could reasonably use another transformation, e.g. "identity", if you have truly quantitative microbiome profiling data
	`interactive` creates plot suitable for use with ggiraph
	`max_taxa` maximum distinct taxa groups to show (only really useful for limiting complexity of interactive plots e.g. within ord_explore)
	`other_name` name for other taxa after N

Details

data must also contain counts table if taxa were transformed (e.g. for clr PCA ordination) (i.e. you must have used `tax_transform` with keep_counts = TRUE, if transformation was not "identity")

You cannot set a variable fill aesthetic (only fixed) for the annotation points, as the fill is used for the taxonomic composition bars

Value

ggplot

Examples

```
library(dplyr)
library(ggplot2)
data("dietswap", package = "microbiome")

# although these iris plots are great for 100s of samples
# we'll take a subset of the data (for speed in this example)
ps <- dietswap %>%
  ps_filter(timepoint %in% c(1, 2)) %>%
  # copy an otu to the sample data
  ps_otu2samdat("Prevotella melaninogenica et rel.") %>%
  # create a couple of useful variables
  ps_mutate(
    female = sex == "female",
    african = nationality == "AFR",
    log_P.melaninogenica = log10(`Prevotella melaninogenica et rel.` + 1)
  )

# define a function for taking the end off the long genus names in this dataset
tax_renamer <- function(tax) {
  stringr::str_remove(tax, " [ae]t rel.")
}

ord <- ps %>%
  tax_agg("Genus") %>%
  dist_calc("aitchison") %>%
  ord_calc(method = "PCoA")

# ordination plot for comparison
ord %>% ord_plot(color = "log_P.melaninogenica", size = 3)

ord_plot_iris(
  data = ord,
  tax_level = "Genus",
  n_taxa = 10,
  anno_colour = "nationality",
  anno_colour_style = list(size = 3),
  anno_binary = "female",
  anno_binary_style = list(shape = "F", size = 2.5),
  taxon_renamer = tax_renamer
) +
  scale_colour_brewer(palette = "Dark2")

# It is also possible to use comp_barplot customisation arguments
# like bar_width and bar_outline_colour, and to make interactive iris plots
# using ggiraph:

if (interactive()) {
  hover_over_me <- ord_plot_iris(
    data = ord,
    tax_level = "Genus",
    n_taxa = 10,
    anno_colour = "nationality",
    anno_colour_style = list(size = 3),
    anno_binary = "female",
    anno_binary_style = list(shape = "F", size = 2.5),
    taxon_renamer = tax_renamer,
    interactive = TRUE,
    bar_width = 0.8, bar_outline_colour = "black"
  ) +
    scale_colour_brewer(palette = "Dark2")

  ggiraph::girafe(ggobj = hover_over_me)
}

# Using PCA for ordination after transformations (e.g. clr) means the untransformed taxonomic
# data are only available for plotting as compositions if you transformed with
# tax_transform(keep_counts = TRUE) and your original data were in fact counts.
# Compositional data will also work, and you can set count_warn to FALSE to avoid the warning

clr_pca <- ps %>%
  tax_agg("Genus") %>%
  tax_transform("clr") %>%
  ord_calc(method = "PCA")

# you can generate a simple paired layout of ord_plot and iris plot
# or separately create and pair the plots yourself, for more control

# simple pairing
ord_plot_iris(
  data = clr_pca, n_taxa = 12,
  tax_level = "Genus",
  taxon_renamer = tax_renamer,
  ord_plot = "below",
  bar_width = 0.8, bar_outline_colour = "black",
  anno_binary = "african",
  anno_binary_style = list(
    y = 1.08, colour = "gray50", shape = "circle open", size = 1, stroke = 1.5
  )
)

# manual pairing
plot1 <- clr_pca %>% ord_plot(
  plot_taxa = 6:1, tax_vec_length = 0.6,
  colour = "gray50", shape = "nationality",
  taxon_renamer = tax_renamer,
  auto_caption = NA, center = TRUE,
) +
  scale_shape_manual(values = c(AFR = "circle", AAM = "circle open"))

iris <- ord_plot_iris(
  data = clr_pca, n_taxa = 15,
  tax_level = "Genus",
  taxon_renamer = tax_renamer,
  anno_binary = "african",
  anno_binary_style = list(y = 1.05, colour = "gray50", shape = "circle", size = 1)
) +
  # shrink legend text size
  theme(legend.text = element_text(size = 7))

cowplot::plot_grid(plot1, iris, nrow = 1, align = "h", axis = "b", rel_widths = 3:4)

# you can add multiple rings of binary annotations
ord_plot_iris(
  data = clr_pca, n_taxa = 15,
  tax_level = "Genus",
  taxon_renamer = tax_renamer,
  anno_binary = c("african", "female"),
  anno_binary_style = list(
    colour = c("gray50", "coral"),
    shape = c("circle", "F"), size = c(0.5, 2)
  )
) +
  theme(legend.text = element_text(size = 7))
```

phyloseq_validate   *Check for (and fix) common problems with phyloseq objects*

Description

- It checks for, and messages about, common uninformative entries in the tax_table, which often cause unwanted results
- If there is no sample_data, it creates a sample_data dataframe with the sample_names (as "SAMPLE" variable)
- If there is no tax_table, it creates a 1-column tax_table matrix with the taxa_names, and calls the rank "unique"
- If remove_undetected = TRUE, it removes taxa where `phyloseq::taxa_sums()` is equal to zero, with a warning

Usage

```
phyloseq_validate(
  ps,
  remove_undetected = FALSE,
  min_tax_length = 4,
  verbose = TRUE
)
```

Arguments

`ps`	phyloseq object
`remove_undetected`	if TRUE, removes taxa that sum to zero across all samples
`min_tax_length`	minimum number of characters to not consider a tax_table entry suspiciously short
`verbose`	print informative messages if true

Value

possibly modified phyloseq object

Examples

```
data(dietswap, package = "microbiome")

# expect warning about taxa summing to zero
phyloseq_validate(dietswap, remove_undetected = TRUE, verbose = TRUE)

# verbose = FALSE will suppress messages and warnings but still:
# replace NULL sample_data and remove taxa that sum to 0 across all samples
# (if remove_undetected = TRUE)
phyloseq_validate(dietswap, verbose = FALSE)

# Sometimes you might have a phyloseq with no sample_data
# This isn't compatible with some microViz functions, like comp_barplot
# So some functions internally use phyloseq_validate to fix this
dietswap@sam_data <- NULL
phyloseq_validate(dietswap)

# Sometimes you might have a phyloseq with no tax_table
# This isn't compatible with some microViz functions, like tax_top,
# so this is another reason to start your analyses with phyloseq_validate!
data("soilrep", package = "phyloseq")
soilrep # has NULL tax_table
phyloseq_validate(soilrep)

# If no messages or warnings are emitted,
# this means no problems were detected, and nothing was changed
# (but only if verbose = TRUE)
```

---

prev — *Calculate prevalence from numeric vector*

Description

Useful as helper for taxon prevalence calculation

Usage

```
prev(x, undetected = 0)
```

Arguments

x	numeric vector (of taxon counts or proportions)
undetected	value above which a taxon is considered present or detected

Value

numeric value

Examples

```
prev(c(0, 0, 1, 2, 4))
prev(c(0, 0, 1, 2, 4), undetected = 1.5)
```

`print.psExtraInfo` *Print method for psExtraInfo object*

Description

Print method for psExtraInfo object

Usage

```
## S3 method for class 'psExtraInfo'
print(
  x,
  ...,
  which = c("tax_agg", "tax_trans", "tax_scale", "dist_method", "ord_info")
)
```

Arguments

`x`	psExtraInfo object
`...`	ignored
`which`	which elements of psExtraInfo list to print

`psExtra-accessors` *Extract elements from psExtra class*

Description

- `ps_get` returns phyloseq
- `info_get` returns psExtraInfo object
- `dist_get` returns distance matrix (or NULL)
- `ord_get` returns ordination object (or NULL)
- `perm_get` returns adonis2() permanova model (or NULL)
- `bdisp_get` returns results of betadisper() (or NULL)
- `otu_get` returns phyloseq otu_table matrix with taxa as columns
- `tt_get` returns phyloseq tax_table
- `tax_models_get` returns list generated by tax_model or NULL
- `tax_stats_get` returns dataframe generated by tax_models2stats or NULL
- `taxatree_models_get` returns list generated by taxatree_models or NULL
- `taxatree_stats_get` returns dataframe generated by taxatree_models2stats or NULL
- `samdat_tbl` returns phyloseq sample_data as a tibble with sample_names as new first column called .sample_name

Usage

```
ps_get(psExtra, ps_extra, counts = FALSE, warn = TRUE)

dist_get(psExtra, ps_extra)

ord_get(psExtra, ps_extra)

info_get(psExtra, ps_extra)

perm_get(psExtra, ps_extra)

bdisp_get(psExtra, ps_extra)

tax_models_get(psExtra)

tax_stats_get(psExtra)

taxatree_models_get(psExtra)

taxatree_stats_get(psExtra)

otu_get(data, taxa = NA, samples = NA, counts = FALSE, warn = TRUE)

tt_get(data)

samdat_tbl(data, sample_names_col = ".sample_name")
```

Arguments

`psExtra`	psExtra S4 class object
`ps_extra`	deprecated! don't use this
`counts`	should ps_get or otu_get attempt to return counts? if present in object
`warn`	if counts = TRUE, should a warning be emitted if counts are not available? set warn = "error" to stop if counts are not available
`data`	phyloseq or psExtra
`taxa`	subset of taxa to return, NA for all (default)
`samples`	subset of samples to return, NA for all (default)
`sample_names_col`	name of column where sample_names are put. if NA, return data.frame with rownames (sample_names)

Value

element(s) from psExtra object (or NULL)

Examples

```
data("dietswap", package = "microbiome")

psx <- tax_transform(dietswap, "compositional", rank = "Genus")

psx

ps_get(psx)

ps_get(psx, counts = TRUE)

info_get(psx)

dist_get(psx) # this psExtra has no dist_calc result

ord_get(psx) # this psExtra has no ord_calc result

perm_get(psx) # this psExtra has no dist_permanova result

bdisp_get(psx) # this psExtra has no dist_bdisp result

# these can be returned from phyloseq objects too
otu_get(psx, taxa = 6:9, samples = c("Sample-9", "Sample-1", "Sample-6"))

otu_get(psx, taxa = 6:9, samples = c(9, 1, 6), counts = TRUE)

tt_get(psx) %>% head()

samdat_tbl(psx)

samdat_tbl(psx, sample_names_col = "SAMPLENAME")
```

`psExtra-class` *Define psExtra class S4 object*

Description

Define psExtra class S4 object

Slots

`info` list.

`counts` otu_table.

`dist` dist.

`ord` ANY.

`permanova` ANY.

`bdisp` ANY.

`taxatree_models`	list.
`taxatree_stats`	data.frame.
`tax_models`	list.
`tax_stats`	data.frame.

Examples

```
library(phyloseq)
data("shao19")

ps <- shao19 %>% ps_filter(infant_age == 12)
ps %>% tax_agg("genus")
```

`ps_arrange` *Arrange samples in phyloseq by sample_data variables or taxon abundance*

Description

Uses information in the sample_data or tax_table of phyloseq object to set the order of the samples (sample_data or tax_table specified by .target arg)

Give this function arguments in the same way you would use dplyr::arrange()

Usage

```
ps_arrange(ps, ..., .target = "sample_data")
```

Arguments

`ps`	phyloseq object
`...`	dots passed directly to dplyr::arrange()
`.target`	arrange samples by "sample_data" variables or "otu_table" taxa abundances

Value

phyloseq

See Also

`arrange`

Examples

```
data("dietswap", package = "microbiome")

dietswap %>%
  ps_arrange(subject, timepoint) %>%
  phyloseq::sample_data() %>%
  head(8)

ps <- dietswap %>% ps_arrange(subject, desc(timepoint))
phyloseq::sample_data(ps) %>% head(8)
phyloseq::otu_table(ps)[1:8, 1:8]

# you can also arrange samples by the abundances of taxa in the otu tables
pst <- dietswap %>% ps_arrange(desc(Akkermansia), .target = "otu_table")
phyloseq::otu_table(pst)[1:8, 1:8]
phyloseq::sample_data(pst) %>% head(8)
```

---

ps_calc_diversity *Calculate diversity index and add to phyloseq sample data*

Description

Wrapper around microbiome::diversity() function. Takes and returns a phyloseq object. Calculates an alpha diversity index at a given taxonomic rank. Returns phyloseq unaggregated, with an additional variable. Variable name is by default created by pasting the index and rank.

Usage

```
ps_calc_diversity(
  ps,
  rank,
  index = "shannon",
  exp = FALSE,
  varname = paste0(ifelse(exp, "exp_", ""), paste0(index, "_", rank))
)
```

Arguments

`ps`	phyloseq
`rank`	taxonomic rank name, or "unique"
`index`	name of diversity index from microbiome::diversity(). One of: "inverse_simpson", "gini_simpson", "shannon", "fisher", "coverage"
`exp`	exponentiate the result? (i.e. return e^index) - see details
`varname`	name of the variable to be added to phyloseq sample data

Details

Don't filter taxa before calculating alpha diversity.

See the following resources for a discussion of exponentiated diversity indices http://www.loujost.com/Statistics%20and%20I
http://www.loujost.com/Statistics%20and%20Physics/Diversity%20and%20Similarity/EffectiveNumberOfSpecies.htm

Value

phyloseq

Examples

```
data(ibd, package = "microViz")
ibd %>%
  ps_filter(abx == "abx") %>%
  tax_fix() %>%
  ps_calc_diversity("Genus", index = "shannon", exp = TRUE) %>%
  ps_calc_diversity("Family", index = "inverse_simpson") %>%
  tax_transform(rank = "Genus", transform = "clr") %>%
  ord_calc("PCA") %>%
  ord_plot(
    colour = "exp_shannon_Genus", size = "inverse_simpson_Family"
  ) +
  ggplot2::scale_colour_viridis_c()
```

`ps_calc_dominant` *Calculate dominant taxon in each phyloseq sample*

Description

Which taxon is most abundant in each sample of your phyloseq object? This function adds this information as a new variable in your phyloseq sample_data.

- If the most abundant taxon is below the proportional abundance threshold, the dominant taxon will be "none" for that sample
- If there are more than n_max dominant taxa across all samples (not including "none") the dominant taxon will be "other" for those samples

Usage

```
ps_calc_dominant(
  ps,
  rank,
  threshold = 0.3,
  n_max = 6,
  var = paste("dominant", rank, sep = "_"),
  none = "none",
  other = "other"
)
```

Arguments

`ps`	phyloseq object
`rank`	taxonomic rank to calculate dominance at
`threshold`	minimum proportion at which to consider a sample dominated by a taxon
`n_max`	maximum number of taxa that can be listed as dominant taxa
`var`	name of variable to add to phyloseq object sample data
`none`	character value to use when no taxon reaches threshold
`other`	character value to use when another taxon (>n_max) dominates

Details

Thanks to Vitor Heidrich for the idea and a draft implementation

Value

phyloseq object

Examples

```
library(ggplot2)
ps <- microViz::ibd %>%
  tax_filter(min_prevalence = 3) %>%
  tax_fix() %>%
  phyloseq_validate()

ps %>%
  ps_filter(DiseaseState == "CD") %>%
  ps_calc_dominant(rank = "Genus") %>%
  comp_barplot(tax_level = "Genus", label = "dominant_Genus", n_taxa = 12) +
  coord_flip()

ps %>%
  ps_calc_dominant(rank = "Genus") %>%
  tax_transform(rank = "Genus", trans = "clr") %>%
  ord_calc("PCA") %>%
  ord_plot(colour = "dominant_Genus", size = 3, alpha = 0.6) +
  scale_colour_brewer(palette = "Dark2")

# customise function options
ps %>%
  ps_calc_dominant(
    rank = "Family", other = "Other", none = "Not dominated",
    threshold = 0.4, n_max = 3
  ) %>%
  tax_transform(rank = "Genus", trans = "clr") %>%
  ord_calc("PCA") %>%
  ord_plot(colour = "dominant_Family", size = 3, alpha = 0.6) +
  scale_colour_manual(values = c(
    Bacteroidaceae = "forestgreen", Lachnospiraceae = "darkblue",
    Ruminococcaceae = "darkorange", Other = "red", "Not dominated" = "grey"
  ))
```

ps_calc_richness *Calculate richness estimate and add to phyloseq sample data*

Description

Wrapper around microbiome::richness() function. Takes and returns a phyloseq object. Calculates a richness estimate at a given taxonomic rank. Returns phyloseq unaggregated, with an additional variable. Variable name is by default created by pasting the index and rank.

Usage

```
ps_calc_richness(
  ps,
  rank,
  index = "observed",
  detection = 0,
  varname = paste0(index, "_", rank)
)
```

Arguments

`ps`	phyloseq
`rank`	taxonomic rank name, or "unique"
`index`	"observed" or "chao1" - name of richness estimate from microbiome::richness()
`detection`	Detection threshold. Used for the "observed" index.
`varname`	name of the variable to be added to phyloseq sample data

Details

Don't filter taxa before calculating richness.

These richness indices are estimates. For a discussion of the uncertainty and bias of these estimates see e.g. work by Amy Willis https://doi.org/10.3389/fmicb.2019.02407

Value

phyloseq

See Also

`ps_calc_diversity`

`microbiome::richness`

Examples

```
data(ibd, package = "microViz")
ibd %>%
  ps_filter(abx == "abx") %>%
  tax_fix() %>%
  ps_calc_richness("Genus", index = "observed") %>%
  ps_calc_richness("Family", index = "chao1") %>%
  tax_transform(rank = "Genus", transform = "clr") %>%
  ord_calc("PCA") %>%
  ord_plot(
    colour = "observed_Genus", size = "chao1_Family"
  ) +
  ggplot2::scale_colour_viridis_c()
```

ps_dedupe *De-duplicate phyloseq samples*

Description

Use one or more variables in the sample_data to identify and remove duplicate samples (leaving one sample per group).

methods:

- method = "readcount" keeps the one sample in each duplicate group with the highest total number of reads (phyloseq::sample_sums)
- method = "first" keeps the first sample in each duplicate group encountered in the row order of the sample_data
- method = "last" keeps the last sample in each duplicate group encountered in the row order of the sample_data
- method = "random" keeps a random sample from each duplicate group (set.seed for reproducibility)

More than one "duplicate" sample can be kept per group by setting `n` samples > 1.

Usage

```
ps_dedupe(
  ps,
  vars,
  method = "readcount",
  verbose = TRUE,
  n = 1,
  .keep_group_var = FALSE,
  .keep_readcount = FALSE,
  .message_IDs = FALSE,
  .label_only = FALSE,
  .keep_all_taxa = FALSE
)
```

Arguments

`ps`	phyloseq object
`vars`	names of variables, whose (combined) levels identify groups from which only 1 sample is desired
`method`	keep sample with max "readcount" or the "first" or "last" or "random" samples encountered in given sample_data order for each duplicate group
`verbose`	message about number of groups, and number of samples dropped?
`n`	number of 'duplicates' to keep per group, defaults to 1
`.keep_group_var`	keep grouping variable .GROUP. in phyloseq object?
`.keep_readcount`	keep readcount variable .READCOUNT. in phyloseq object?
`.message_IDs`	message sample names of dropped variables?
`.label_only`	if TRUE, the samples will NOT be filtered, just labelled with a new logical variable .KEEP_SAMPLE.
`.keep_all_taxa`	keep all taxa after removing duplicates? If FALSE, the default, taxa are removed if they never occur in any of the retained samples

Details

What happens when duplicated samples have exactly equal readcounts in method = "readcount"? The first encountered maximum is kept (in sample_data row order, like method = "first")

Value

phyloseq object

See Also

`ps_filter` for filtering samples by sample_data variables

Examples

```
data("dietswap", package = "microbiome")

dietswap
# let's pretend the dietswap data contains technical replicates from each subject
# we want to keep only one of them
ps_dedupe(dietswap, vars = "subject", method = "readcount", verbose = TRUE)

# contrived example to show identifying "duplicates" via the interaction of multiple columns
ps1 <- ps_dedupe(
  ps = dietswap, method = "readcount", verbose = TRUE,
  vars = c("timepoint", "group", "bmi_group")
)
phyloseq::sample_data(ps1)
```

```
ps2 <- ps_dedupe(
  ps = dietswap, method = "first", verbose = TRUE,
  vars = c("timepoint", "group", "bmi_group")
)
phyloseq::sample_data(ps2)
```

---

ps_drop_incomplete — *Deselect phyloseq samples with sample_data missings*

Description

Check phyloseq object sample_data for missing values (NAs)

- specify which variables to check with vars argument, or check all
- drop samples with any missings

Usage

```
ps_drop_incomplete(ps, vars = NA, verbose = FALSE)
```

Arguments

`ps`	phyloseq with sample_data
`vars`	vector of variable names to check for missings (or NA, which uses all variables in sample data)
`verbose`	message about number of samples dropped if verbose not FALSE, (and only if > 0 samples dropped) and message about number of missing per variable in vars if verbose = "max" (and message even if 0 samples dropped)

Details

This is a wrapper for `stats::complete.cases` function.

Value

phyloseq

See Also

`ps_filter`

Examples

```
library(phyloseq)
data("enterotype", package = "phyloseq")

enterotype
ps_drop_incomplete(enterotype)
ps_drop_incomplete(enterotype, vars = "Enterotype", verbose = TRUE)
ps_drop_incomplete(enterotype, vars = "Sample_ID", verbose = TRUE)
ps_drop_incomplete(enterotype, vars = c("Enterotype", "Sample_ID"))
ps_drop_incomplete(enterotype, verbose = "max")
```

`ps_filter` *Filter phyloseq samples by sample_data variables*

Description

Keep only samples with sample_data matching one or more conditions. By default this function also removes taxa which never appear in any of the remaining samples, by running tax_filter(min_prevalence = 1). You can prevent this taxa filtering with .keep_all_taxa = TRUE.

Usage

```
ps_filter(ps, ..., .target = "sample_data", .keep_all_taxa = FALSE)
```

Arguments

`ps`	phyloseq object
`...`	passed directly to dplyr::filter (see examples and ?dplyr::filter)
`.target`	which slot of phyloseq to use for filtering by, currently only "sample_data" supported
`.keep_all_taxa`	if FALSE (the default), remove taxa which are no longer present in the dataset after filtering

Details

Use ps_filter as you would use use dplyr::filter(), but with a phyloseq object!

Value

phyloseq object (with filtered sample_data)

See Also

`filter` explains better how to give arguments to this function

`tax_filter` for filtering taxa (not samples)

Examples

```
library(phyloseq)
library(dplyr)

data("enterotype", package = "phyloseq")
enterotype
sample_data(enterotype)[1:10, 1:5]

# keep only samples with seqtech not equal to sanger
ps1 <- ps_filter(enterotype, SeqTech != "Sanger")
ps1
sample_data(ps1)[1:10, 1:5]

# keep only samples with no NAs in any variables
ps2 <- enterotype %>% ps_filter(!if_any(everything(), is.na))
ps2
sample_data(ps2)[1:8, 1:8]

# ps2 is equivalent to dropping samples with incomplete sample_variables and tax_filtering 0s
ps3 <- enterotype %>%
  ps_drop_incomplete() %>%
  tax_filter(undetected = 0, use_counts = FALSE)
# we needed to set a low detection threshold because this example data is proportions
identical(ps2, ps3) # TRUE

# function will give warning if some of the otu_values are negative
# (which may happen when filtering data that has e.g. clr-transformed taxa abundances)
# as it attempts to discard any taxa that become always absent/0 after filtering (by default)
# set .keep_all_taxa = TRUE to avoid this filtering behaviour, which is unwanted in this case
enterotype %>%
  tax_transform("clr") %>%
  ps_get() %>%
  ps_filter(SeqTech == "Sanger", .keep_all_taxa = TRUE)
```

`ps_join` *Join a dataframe to phyloseq sample data*

Description

You can use most types of join from the dplyr::*_join function family, including e.g. "inner", "left", "semi", "anti" (see details below). Defaults to type = "left" which calls left_join(), this supports x as a phyloseq and y as a dataframe. Most of the time you'll want "left" (adds variables with no sample filtering), or "inner" (adds variables and filters samples). This function simply:

1. extracts the sample_data from the phyloseq as a dataframe
2. performs the chosen type of join (with the given arguments)
3. filters the phyloseq if type = inner, semi or anti
4. reattaches the modified sample_data to the phyloseq and returns the phyloseq

Usage

```
ps_join(
  x,
  y,
  by = NULL,
  match_sample_names = NULL,
  keep_sample_name_col = TRUE,
  sample_name_natural_join = FALSE,
  type = "left",
  .keep_all_taxa = FALSE
)
```

Arguments

`x`	phyloseq (or dataframe)
`y`	dataframe (or phyloseq for e.g. type = "right")
`by`	A character vector of variables to join by (col must be present in both x and y or paired via a named vector like c("xname" = "yname", etc.))
`match_sample_names`	match against the phyloseq sample_names by naming a variable in the additional dataframe (this is in addition to any variables named in by)
`keep_sample_name_col`	should the column named in match_sample_names be kept in the returned phyloseq's sample_data? (only relevant if match_sample_names is not NULL)
`sample_name_natural_join`	if TRUE, use sample_name AND all shared colnames to match rows (only relevant if match_sample_names is not NULL, this arg takes precedence over anything also entered in `by` arg)
`type`	name of type of join e.g. "left", "right", "inner", "semi" (see dplyr help pages)
`.keep_all_taxa`	if FALSE (the default), remove taxa which are no longer present in the dataset after filtering

Details

Mutating joins, which will add columns from a dataframe to phyloseq sample data, matching rows based on the key columns named in the `by` argument:

- "inner": includes all rows in present in both x and y.
- "left": includes all rows in x. (so x must be the phyloseq)
- "right": includes all rows in y. (so y must be the phyloseq)
- "full": includes all rows present in x or y. (will likely NOT work, as additional rows cannot be added to sample_data!)

If a row in x matches multiple rows in y (based on variables named in the `by` argument), all the rows in y will be added once for each matching row in x. This will cause this function to fail, as additional rows cannot be added to the phyloseq sample_data!

Filtering joins filter rows from x based on the presence or absence of matches in y:

- "semi": return all rows from x with a match in y.
- "anti": return all rows from x without a match in y.

Value

phyloseq with modified sample_data (and possibly filtered)

See Also

`ps_mutate` for computing new variables from existing sample data

`ps_select` for selecting only some sample_data variables

https://www.garrickadenbuie.com/project/tidyexplain/ for an animated introduction to joining dataframes

Examples

```
library(phyloseq)
data("enterotype", package = "phyloseq")

x <- enterotype
y <- data.frame(
  ID_var = sample_names(enterotype)[c(1:50, 101:150)],
  SeqTech = sample_data(enterotype)[c(1:50, 101:150), "SeqTech"],
  arbitrary_info = rep(c("A", "B"), 50)
)

# simply match the new data to samples that exist in x, as default is a left_join
# where some sample names of x are expected to match variable ID_var in dataframe y
out1A <- ps_join(x = x, y = y, match_sample_names = "ID_var")
out1A
sample_data(out1A)[1:6, ]


# use sample_name and all shared variables to join
# (a natural join is not a type of join per se,
# but it indicates that all shared variables should be used for matching)
out1B <- ps_join(
  x = x, y = y, match_sample_names = "ID_var",
  sample_name_natural_join = TRUE, keep_sample_name_col = FALSE
)
out1B
sample_data(out1B)[1:6, ]

# if you only want to keep phyloseq samples that exist in the new data, try an inner join
# this will add the new variables AND filter the phyloseq
# this example matches sample names to ID_var and by matching the shared SeqTech variable
out1C <- ps_join(x = x, y = y, type = "inner", by = "SeqTech", match_sample_names = "ID_var")
out1C
sample_data(out1C)[1:6, ]

# the id variable is named Sample_ID in x and ID_var in y
# semi_join is only a filtering join (doesn't add new variables but just filters samples in x)
out2A <- ps_join(x = x, y = y, by = c("Sample_ID" = "ID_var"), type = "semi")
out2A
sample_data(out2A)[1:6, ]

# anti_join is another type of filtering join
out2B <- ps_join(x = x, y = y, by = c("Sample_ID" = "ID_var"), type = "anti")
out2B
sample_data(out2B)[1:6, ]

# semi and anti joins keep opposite sets of samples
intersect(sample_names(out2A), sample_names(out2B))

# you can mix and match named and unnamed values in the `by` vector
# inner is like a combination of left join and semi join
out3 <- ps_join(x = x, y = y, by = c("Sample_ID" = "ID_var", "SeqTech"), type = "inner")
out3
sample_data(out3)[1:6, ]
```

ps_melt — *Melt phyloseq data object into large data.frame (tibble)*

Description

The ps_melt function is a specialized melt function for melting phyloseq objects (instances of the phyloseq class), usually for producing graphics with `ggplot2`. The naming conventions used in downstream phyloseq graphics functions have reserved the following variable names that should not be used as the names of `sample_variables` or taxonomic `rank_names`. These reserved names are `c("Sample", "Abundance", "OTU")`. Also, you should not have identical names for sample variables and taxonomic ranks. That is, the intersection of the output of the following two functions `sample_variables`, `rank_names` should be an empty vector (e.g. `intersect(sample_variables(ps), rank_names(ps))`). All of these potential name collisions are checked-for and renamed automatically with a warning. However, if you (re)name your variables accordingly ahead of time, it will reduce confusion and eliminate the warnings.

NOTE: Code and documentation copied (and very slightly modified) from an old version of speedyseq by Michael McLaren. speedyseq reimplements `psmelt` from `phyloseq` to use functions from the `tidyr` and `dplyr` packages. The name in microViz is changed to `ps_melt` for consistency with other functions.

Usage

```
ps_melt(ps)
```

Arguments

`ps`	(Required). An `otu_table-class` or `phyloseq-class`. Function most useful for phyloseq-class.

Details

Note that "melted" phyloseq data is stored much less efficiently, and so RAM storage issues could arise with a smaller dataset (smaller number of samples/OTUs/variables) than one might otherwise expect. For common sizes of graphics-ready datasets, however, this should not be a problem. Because the number of OTU entries has a large effect on the RAM requirement, methods to reduce the number of separate OTU entries – for instance by agglomerating OTUs based on phylogenetic distance using `tip_glom` – can help alleviate RAM usage problems. This function is made user-accessible for flexibility, but is also used extensively by plot functions in phyloseq.

Value

A `tibble` class data frame.

See Also

`psmelt`

Examples

```
library(ggplot2)
library(phyloseq)
data("GlobalPatterns")
gp_ch <- subset_taxa(GlobalPatterns, Phylum == "Chlamydiae")
mdf <- ps_melt(gp_ch)
mdf2 <- psmelt(gp_ch) # slower
# same dataframe, except with somewhat different row orders
dplyr::all_equal(tibble::as_tibble(mdf), mdf2, convert = TRUE) # TRUE
nrow(mdf2)
ncol(mdf)
colnames(mdf)
head(rownames(mdf))
p <- ggplot(mdf, aes(x = SampleType, y = Abundance, fill = Genus))
p <- p + geom_bar(color = "black", stat = "identity", position = "stack")
# This example plot doesn't make any sense
print(p + coord_flip())
# TODO replace this...
```

`ps_mutate` *Modify or compute new sample_data in phyloseq object*

Description

Add or compute new phyloseq sample_data variables. Uses `dplyr::mutate()` syntax.

Usage

```
ps_mutate(ps, ...)
```

Arguments

`ps`	phyloseq object with sample data
`...`	passed straight to dplyr::mutate (see examples and dplyr::mutate help)

Value

phyloseq object with modified sample_data

See Also

`tax_mutate` for manipulation of tax_table variables

`mutate`

Examples

```
library(phyloseq)
library(dplyr)
data("enterotype")

sample_data(enterotype)[1:10, ]

months_in_year <- 12
ps <- enterotype %>%
  ps_mutate(
    age_months = Age * months_in_year,
    IDs_match = as.character(Sample_ID) == as.character(SampleID),
    placeholder = "Word"
  )

sample_data(ps)[1:10, ]

# Using the dplyr::across functionality is also possible
ps <- ps %>%
  ps_mutate(
    dplyr::across(where(is.factor), toupper),
    another_var = TRUE,
    SeqTech = NULL # deletes SeqTech variable
  )

head(sample_data(ps))
```

`ps_otu2samdat` *Copy phyloseq otu_table data to sample_data*

Description

Copy phyloseq otu_table data to sample_data

Usage

```
ps_otu2samdat(ps, taxa = NULL)
```

Arguments

`ps`	phyloseq with sample_data
`taxa`	list of taxa_names to copy to sample_data, or NULL (which selects all with `phyloseq::taxa_names()`)

Value

phyloseq with augmented sample_data

Examples

```
library(phyloseq)
data("dietswap", package = "microbiome")

ps <- dietswap %>% ps_otu2samdat("Akkermansia")
sample_variables(ps)

# or if you do not specify any taxa, all are copied
ps_all <- dietswap %>% ps_otu2samdat()
sample_variables(ps_all)[1:15]

# this could be useful for colouring ordination plots, for example
ps %>%
  ps_mutate(log_akkermansia = log(Akkermansia)) %>%
  dist_calc("bray") %>%
  ord_calc(method = "PCoA") %>%
  ord_plot(
    colour = "log_akkermansia",
    size = 3, shape = "nationality"
  )
```

`ps_reorder` *Set order of samples in phyloseq object*

Description

Manually set order of samples by specifying samples names in desired order.

Usage

```
ps_reorder(ps, sample_order)
```

Arguments

`ps`	phyloseq
`sample_order`	names or current numerical indices of samples in desired order

Details

Ordering of samples in a phyloseq is controlled from the otu_table slot!

Value

phyloseq

See Also

`ps_arrange` for arranging samples by sample_data variables (or otu_table)

`ps_seriate` for arranging samples by microbiome similarity

`ps_filter` for keeping only some samples, based on sample_data

Examples

```
library(phyloseq)
data("dietswap", package = "microbiome")

dietswap %>%
  sample_data() %>%
  head(8)

new_order <- rev(sample_names(dietswap))
dietswap %>%
  ps_reorder(new_order) %>%
  sample_data() %>%
  head(8)

# random ordering with numbers
set.seed(1000)
random_order <- sample(1:nsamples(dietswap))
dietswap %>%
  ps_reorder(random_order) %>%
  sample_data() %>%
  head(8)
```

`ps_select` *Select phyloseq sample_data using dplyr::select syntax*

Description

Simple selection of phyloseq sample_data variables, might be useful for printing reduced sample_data, or inside other functions

Usage

```
ps_select(ps, ...)
```

Arguments

`ps`	phyloseq with sample_data
`...`	passed straight to dplyr::select

Value

phyloseq object

Examples

```
library(phyloseq)
library(dplyr)
data("enterotype", package = "phyloseq")

head(sample_data(enterotype))

enterotype %>%
  ps_select(!contains("Sample")) %>%
  sample_data() %>%
  head()
```

`ps_seriate` *Arrange samples in a phyloseq by microbiome similarity*

Description

Uses seriation methods from seriation::seriate and often dist_calc (depending on if seriation method requires a distance matrix)

Usage

```
ps_seriate(
  ps,
  method = "OLO_ward",
  dist = "bray",
  tax_transform = "identity",
  add_variable = FALSE,
  rank = NA
)
```

Arguments

`ps`	phyloseq object
`method`	seriation method for ordering samples, from seriation::seriate
`dist`	distance method for dist_calc (only used if required for particular seriation method!)
`tax_transform`	transformation to apply before seriation or any distance calculation
`add_variable`	add a variable to the sample data indicating seriation order
`rank`	taxonomic rank to aggregate at, before seriation, NA for no aggregation

Value

phyloseq

See Also

`ps_arrange` `ps_reorder`

Examples

```
library(phyloseq)
data("dietswap", package = "microbiome")

dietswap %>%
  sample_data() %>%
  head(8)

dietswap %>%
  tax_agg("Genus") %>%
  ps_get() %>%
  ps_seriate(method = "OLO_ward", dist = "bray") %>%
  sample_data() %>%
  head(8)
```

`ps_sort_ord` *Sort phyloseq samples by ordination axes scores*

Description

`ps_sort_ord` reorders samples in a phyloseq object based on their relative position on 1 or 2 ordination axes.

`ord_order_samples` gets the sample_names in order from the ordination contained in a psExtra. This is used internally by `ps_sort_ord`

If 2 axes given, the samples are sorted by anticlockwise rotation around the selected ordination axes, starting on first axis given, upper right quadrant. (This is used by ord_plot_iris.)

If 1 axis is given, samples are sorted by increasing value order along this axis. This could be used to arrange samples on a rectangular barplot in order of appearance along a parallel axis of a paired ordination plot.

Usage

```
ps_sort_ord(ps, ord, axes = 1:2, scaling = 2)

ord_order_samples(ord, axes = 1:2, scaling = 2)
```

Arguments

`ps`	phyloseq object to be sorted
`ord`	psExtra with ordination object
`axes`	which axes to use for sorting? numerical vector of length 1 or 2
`scaling`	Type 2, or type 1 scaling. For more info, see `https://sites.google.com/site/mb3gustame/constrained-analyses/redundancy-analysis`. Either "species" or "site" scores are scaled by (proportional) eigenvalues, and the other set of scores is left unscaled (from ?vegan::scores.cca)

Value

`ps_sort_ord` returns a phyloseq

`ord_order_samples` returns a character vector

See Also

- These functions were created to support ordering of samples on `ord_plot_iris`
- `tax_sort_ord` for ordering taxa in phyloseq by ordination

Examples

```
# attach other necessary packages
library(ggplot2)

# example data
ibd <- microViz::ibd %>%
  tax_filter(min_prevalence = 2) %>%
  tax_fix() %>%
  phyloseq_validate()

# create numeric variables for constrained ordination
ibd <- ibd %>%
  ps_mutate(
    ibd = as.numeric(ibd == "ibd"),
    steroids = as.numeric(steroids == "steroids"),
    abx = as.numeric(abx == "abx"),
    female = as.numeric(gender == "female"),
    # and make a shorter ID variable
    id = stringr::str_remove_all(sample, "^[0]{1,2}|-[A-Z]+")
  )

# create an ordination
ordi <- ibd %>%
  tax_transform("clr", rank = "Genus") %>%
  ord_calc()

ord_order_samples(ordi, axes = 1) %>% head(8)
ps_sort_ord(ibd, ordi, axes = 1) %>%
  phyloseq::sample_names() %>%
  head(8)

p1 <- ord_plot(ordi, colour = "grey90", plot_taxa = 1:8, tax_vec_length = 1) +
  geom_text(aes(label = id), size = 2.5, colour = "red")

b1 <- ibd %>%
  ps_sort_ord(ord = ordi, axes = 1) %>%
  comp_barplot(
    tax_level = "Genus", n_taxa = 12, label = "id",
    order_taxa = ord_order_taxa(ordi, axes = 1),
    sample_order = "asis"
  ) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

patchwork::wrap_plots(p1, b1, ncol = 1)

# constrained ordination example (and match vertical axis) #
cordi <- ibd %>%
  tax_transform("clr", rank = "Genus") %>%
  ord_calc(
    constraints = c("steroids", "abx", "ibd"), conditions = "female",
    scale_cc = FALSE
  )

cordi %>% ord_plot(plot_taxa = 1:6, axes = 2:1)
```

sampleAnnotation *Helper to specify a HeatmapAnnotation for samples in comp_heatmap*

Description

Helper to specify a HeatmapAnnotation for samples in comp_heatmap

Usage

```
sampleAnnotation(
  ...,
  name,
  annotation_legend_param = list(),
  show_legend = TRUE,
  gp = grid::gpar(col = NA),
  border = FALSE,
  gap = grid::unit(2, "mm"),
  show_annotation_name = TRUE,
  annotation_label = NULL,
  annotation_name_gp = grid::gpar(),
  annotation_name_offset = NULL,
  annotation_name_rot = NULL,
  annotation_name_align = FALSE,
  annotation_name_side = "auto",
  .data = NULL,
  .samples = NULL,
  .side = NULL
)
```

Arguments

`...`	Name-value pairs where the names correspond to annotation names and values are the output of sample annotation functions such as anno_sample(), or manually specified AnnotationFunction objects
`name`	Name of the heatmap annotation, optional.
`annotation_legend_param`	A list which contains parameters for annotation legends. See `color_mapping_legend,ColorMapping-me` for all possible options.
`show_legend`	Whether show annotation legends. The value can be one single value or a vector.
`gp`	Graphic parameters for simple annotations (with `fill` parameter ignored).
`border`	border of single annotations.
`gap`	Gap between annotations. It can be a single value or a vector of `unit` objects.
`show_annotation_name`	Whether show annotation names? For column annotation, annotation names are drawn either on the left or the right, and for row annotations, names are draw either on top or at the bottom. The value can be a vector.
`annotation_label`	Labels for the annotations. By default it is the same as individual annotation names.
`annotation_name_gp`	Graphic parameters for annotation names. Graphic parameters can be vectors.
`annotation_name_offset`	Offset to the annotation names, a `unit` object. The value can be a vector.
`annotation_name_rot`	Rotation of the annotation names. The value can be a vector.
`annotation_name_align`	Whether to align the annotation names.
`annotation_name_side`	Side of the annotation names.
`.data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`.samples`	OPTIONAL selection vector of sample names, only set this if providing .data argument to override default

`.side`	OPTIONAL string, indicating the side for the variable annotations: only set this to override default

Value

HeatmapAnnotation object

See Also

`taxAnnotation()`

Examples

```
library("ComplexHeatmap")
data("ibd", package = "microViz")
psq <- tax_filter(ibd, min_prevalence = 5)
psq <- tax_mutate(psq, Species = NULL)
psq <- tax_fix(psq)
psq <- tax_agg(psq, rank = "Family")
taxa <- tax_top(psq, n = 15, rank = "Family")
samples <- phyloseq::sample_names(psq)

set.seed(42) # random colours used in first example
# sampleAnnotation returns a function that takes data, samples, and which
fun <- sampleAnnotation(
  gap = grid::unit(2.5, "mm"),
  Dis1 = anno_sample(var = "DiseaseState"),
  IBD = anno_sample_cat(var = "ibd"),
  Dis2 = anno_sample_cat(var = "DiseaseState", col = 1:4)
)

# manually specify the sample annotation function by giving it data etc.
heatmapAnnoFunction <- fun(.data = psq, .side = "top", .samples = samples)

# draw the annotation without a heatmap, you will never normally do this!
grid.newpage()
vp <- viewport(width = 0.65, height = 0.75)
pushViewport(vp)
draw(heatmapAnnoFunction)
pushViewport(viewport(x = 0.7, y = 0.6))
draw(attr(heatmapAnnoFunction, "Legends"))
```

`scale_shape_girafe_filled`

Filled shapes for ggiraph interactive plots

Description

Generates a custom ggplot2 shape scale, as used in ord_explore's ordination. Uses filled shapes, therefore fill aesthetic must be set, in addition to colour, to have filled shapes. Points with NA values for the shape variable are shown as hollow circles.

Usage

```
scale_shape_girafe_filled()
```

Details

Composite shapes e.g. number 7 "square cross" cause ggiraph interactive plots to fail when a variable shape and tooltip is set.

Shapes used are, in order: "circle filled", "triangle filled", "square filled", "diamond filled", and "triangle down filled"

Value

ggplot2 Scale object

Examples

```
microViz::ibd %>%
  tax_fix() %>%
  phyloseq_validate() %>%
  tax_transform(rank = "Genus", trans = "clr") %>%
  ord_calc(
    method = "PCA"
  ) %>%
  ord_plot(
    axes = c(1, 2),
    plot_taxa = 1:6,
    colour = "DiseaseState", fill = "DiseaseState",
    shape = "circle", alpha = 0.5,
    size = 3
  ) +
  scale_shape_girafe_filled()
```

`shao19` *Gut microbiota relative abundance data from Shao et al. 2019*

Description

A phyloseq dataset containing 1644 gut microbiome samples. Fecal samples were collected from 596 infants. 175 mothers also provided fecal samples.

Usage

```
shao19
```

Format

large phyloseq object

Data acquisition

The processed (metaphlan3) relative abundance data were obtained using curatedMetagenomicData package version 3.4.2. https://waldronlab.io/curatedMetagenomicData/articles/available-studies.html

A small amount of data cleaning was performed after download:

1. to make sample metadata more user friendly
2. to fill gaps in the taxonomy table.

Source

Stunted microbiota and opportunistic pathogen colonization in caesarean-section birth. doi:10.1038/s4158601915601

`stat_chull` *Draw convex hull for a set of points on a ggplot*

Description

Draws a (convex) polygon around the outermost points of a set of points. Useful as a visual aid for identifying groups of points on a scatterplot, such as an ordination plot.

Usage

```
stat_chull(
  mapping = NULL,
  data = NULL,
  geom = "polygonHollow",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

	A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`geom`	The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following: • A `Geom` ggproto subclass, for example `GeomPoint`. • A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`. • For more information and other ways to specify the geom, see the layer geom documentation.
`position`	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following: • The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`. • For more information and other ways to specify the position, see the layer position documentation.
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.
`...`	Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.

- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Details

This is a ggplot2 extension - slightly modified from the original code found here:

https://CRAN.r-project.org/package=ggplot2/vignettes/extending-ggplot2.html

See Also

ggplot2::stat_ellipse

ord_plot

Examples

```
library(ggplot2)
microViz::ibd %>%
  tax_fix() %>%
  tax_transform(rank = "Genus", trans = "clr") %>%
  ord_calc(method = "PCA") %>%
  ord_plot(colour = "DiseaseState", shape = "DiseaseState", alpha = 0.5) +
  stat_chull(aes(colour = DiseaseState))

microViz::ibd %>%
  tax_fix() %>%
  tax_transform(rank = "Genus", trans = "clr") %>%
  ord_calc(method = "PCA") %>%
  ord_plot(colour = "DiseaseState", shape = "DiseaseState", alpha = 0.5) +
  stat_chull(aes(colour = DiseaseState, fill = DiseaseState), alpha = 0.1)
```

`taxAnnotation` *Helper to specify a HeatmapAnnotation for taxa*

Description

Helper to specify a HeatmapAnnotation for taxa

Usage

```
taxAnnotation(
  ...,
  name,
  annotation_legend_param = list(),
  show_legend = TRUE,
  gp = grid::gpar(col = NA),
  border = FALSE,
  gap = grid::unit(2, "mm"),
  show_annotation_name = TRUE,
  annotation_label = NULL,
  annotation_name_gp = grid::gpar(),
  annotation_name_offset = NULL,
  annotation_name_rot = NULL,
  annotation_name_align = TRUE,
  annotation_name_side = "auto",
  .data = NULL,
  .taxa = NULL,
  .side = NULL
)
```

Arguments

`...`	Name-value pairs where the names correspond to annotation names and values are the output of taxon annotation functions such as anno_tax_prev() or manually specified AnnotationFunction objects
`name`	Name of the heatmap annotation, optional.
`annotation_legend_param`	A list which contains parameters for annotation legends. See `color_mapping_legend,ColorMapping-me` for all possible options.
`show_legend`	Whether show annotation legends. The value can be one single value or a vector.
`gp`	Graphic parameters for simple annotations (with `fill` parameter ignored).
`border`	border of single annotations.
`gap`	Gap between annotations. It can be a single value or a vector of `unit` objects.
`show_annotation_name`	Whether show annotation names? For column annotation, annotation names are drawn either on the left or the right, and for row annotations, names are draw either on top or at the bottom. The value can be a vector.
`annotation_label`	Labels for the annotations. By default it is the same as individual annotation names.
`annotation_name_gp`	Graphic parameters for annotation names. Graphic parameters can be vectors.
`annotation_name_offset`	Offset to the annotation names, a `unit` object. The value can be a vector.
`annotation_name_rot`	Rotation of the annotation names. The value can be a vector.
`annotation_name_align`	Whether to align the annotation names.
`annotation_name_side`	Side of the annotation names.

`.data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`.taxa`	OPTIONAL selection vector of taxa (names, numbers or logical), only set this if providing .data argument to override default
`.side`	OPTIONAL string, indicating the side the taxa annotation should be placed: only set this to override default

Value

HeatmapAnnotation object

Examples

```
library("ComplexHeatmap")
data("ibd", package = "microViz")
psq <- tax_filter(ibd, min_prevalence = 5)
psq <- tax_mutate(psq, Species = NULL)
psq <- tax_fix(psq)
psq <- tax_agg(psq, rank = "Family")
taxa <- tax_top(psq, n = 15, rank = "Family")

customAxis <- list(labels_rot = 0, at = c(0, 0.5, 1))

# makes a function that takes data, taxa and which (at minimum)
fun <- taxAnnotation(
  gap = grid::unit(2.5, "mm"),
  Prev. = anno_tax_prev(axis_param = customAxis, ylim = c(0, 1), extend = 0),
  `Prop. Abd.` = anno_tax_box(size = unit(40, "mm"), axis_param = customAxis),
  `Log10p Abd.` = anno_tax_density(type = "heatmap")
)

# manually specify the prevalence barplot function by giving it data etc.
heatmapAnnoFunction <- fun(.data = psq, .side = "top", .taxa = taxa)

# draw the annotation without a heatmap, you will never normally do this!
grid.newpage()
vp <- viewport(width = 0.65, height = 0.75)
pushViewport(vp)
draw(heatmapAnnoFunction)

# try again as a row annotation
grid.newpage()
pushViewport(vp)
draw(fun(.data = psq, .side = "right", .taxa = rev(taxa)))
```

taxatree_funs *Create node and edge dataframes for taxatree_plots*

Description

Mostly you will not have to use these functions directly: instead call `taxatree_plots` with the output of `taxatree_stats`

- `taxatree_nodes` creates taxon nodes and calculates a summary statistic about each taxon (given by `fun`). Takes a psExtra or phyloseq object.
- `taxatree_edges` uses the output of `taxatree_nodes` to create a dataframe of edges.

Usage

```
taxatree_nodes(
  ps,
  fun = list(sum = sum),
  ranks = "all",
  .sort = NULL,
  .use_counts = TRUE
)

taxatree_edges(nodes_df)
```

Arguments

`ps`	phyloseq object or psExtra
`fun`	function to calculate for each taxon/node
`ranks`	selection of taxonomic ranks to make nodes for ("all", or names)
`.sort`	sort nodes by "increasing" or "decreasing" values of fun function result
`.use_counts`	use count data if available (instead of transformed data)
`nodes_df`	dataframe output from taxatree_nodes

Details

`taxatree_nodes` makes nodes for taxa at all ranks or for a list of consecutive ranks (plus a root rank if tree is not rooted).

`taxatree_label` *Add logical label column to taxatree_stats dataframe*

Description

`taxatree_label` is used internally by `taxatree_plotkey`, but can also be used prior to `taxatree_plots` to label those plots directly

`...` arguments are passed to `dplyr::filter()`, so all `filter` syntax can be used.

Usage

```
taxatree_label(
  data,
  ...,
  .label_var = "label",
  .node_fun = list(prevalence = prev)
)
```

Arguments

`data`	psExtra (or phyloseq)
`...`	REQUIRED logical conditions for labelling e.g. rank == "Phylum", p.value < 0.1 | taxon %in% listOfTaxa
`.label_var`	name of label indicator variable to be created. If you change this, beware that taxatree_plotkey will not work, you will need to called taxatree_plot_label with
`.node_fun`	named list of length 1 providing `taxatree_nodes` `fun` arg. (name of list iterm is available for use in ...)

Details

If taxatree_stats missing (or if data is a phyloseq) it will create a plain taxatree_stats dataframe using only taxatree_nodes

`node_fun` can also be a precalculated dataframe (output of taxatree_nodes) but you should probably not use this option. This is used internally for efficiency inside `taxatree_plotkey()`

Value

psExtra with (modified) taxatree_stats dataframe

Examples

```
# simple example with plain phyloseq input
data("dietswap", package = "microbiome")
labelled <- dietswap %>%
  tax_prepend_ranks() %>%
  taxatree_label(rank == "Phylum", prevalence > 0.1)

# Note that "prevalence" column was available in data
# because it is created by `taxatree_nodes()` using the named function
# provided to the `node_fun` argument

# psExtra is returned
labelled

# notice how both conditions must be met for label column to be TRUE
labelled %>% taxatree_stats_get()
```

`taxatree_models` *Statistical modelling for individual taxa across multiple ranks*

Description

`taxatree_models` runs `tax_model` on every taxon at multiple taxonomic ranks (you choose which ranks with the plural `ranks` argument). It returns the results as a named nested list of models attached to a psExtra object. One list per rank, one model per taxon at each rank.

The result can then be used with `taxatree_models2stats` to extract a dataframe of statistics for use with `taxatree_plots`.

Usage

```
taxatree_models(
  ps,
  ranks = NULL,
  type = "lm",
  variables = NULL,
  formula = NULL,
  use_future = FALSE,
  checkVars = TRUE,
  checkNA = "warning",
  verbose = "rank",
  trans = "identity",
  trans_args = list(),
  ...
)
```

Arguments

`ps`	phyloseq object or psExtra
`ranks`	vector of rank names at which to aggregate taxa for modelling
`type`	name of regression modelling function, or the function itself
`variables`	vector of variable names, to be used as model formula right hand side. If variables is a list, not a vector, a model is fit for each entry in list.
`formula`	Right hand side of a formula, as a formula object or character string. Or a list of these. (alternative to variables argument, do not provide both)
`use_future`	if TRUE parallel processing with future is possible, see details of ?tax_model.
`checkVars`	check variance of variables?
`checkNA`	check variables for NAs?
`verbose`	message about progress: "rank" only notifies which rank is being processed; TRUE notifies you about each taxon being processed; FALSE for no messages.
`trans`	name of tax_transform transformation to apply to aggregated taxa before fitting statistical models

`trans_args`	named list of any additional arguments to tax_transform e.g. list(zero_replace = "halfmin")
`...`	extra arguments are passed directly to modelling function

See Also

`tax_model` for more details and examples

`taxatree_plots` for how to plot the output of `taxatree_models`

`taxatree_models2stats` *Extract statistics from taxatree_models or tax_model output*

Description

Runs a function e.g. `broom::tidy` on a list of models, i.e. the output of `taxatree_models` or `tax_model`

Usage

```
taxatree_models2stats(data, fun = "auto", ..., .keep_models = FALSE)

tax_models2stats(data, rank = NULL, fun = "auto", ..., .keep_models = FALSE)
```

Arguments

`data`	psExtra with attached tax_models or taxatree_models list, or just the list of models
`fun`	function to assist extraction of stats dataframe from models, or "auto"
`...`	extra arguments passed to fun
`.keep_models`	should the models list be kept in the psExtra output?
`rank`	string naming rank at which tax_model was run (needed if data is a list)

Value

data.frame, attached to psExtra

Functions

- `taxatree_models2stats()`: Extract stats from list or psExtra output of taxatree_models
- `tax_models2stats()`: Extract stats from list or psExtra output of tax_model

Examples

```
# This example is an abbreviated excerpt from article on taxon modelling on
# the microViz documentation website

library(dplyr)
data("ibd", package = "microViz")

# We'll keep only the Ulcerative Colitis and Healthy Control samples, to
# simplify the analyses for this example. We'll also remove the Species
# rank information, as most OTUs in this dataset are not assigned to a
# species. We'll also use `tax_fix` to fill any gaps where the Genus is
# unknown, with the family name or whatever higher rank classification is
# known.

phylo <- ibd %>%
  ps_filter(DiseaseState %in% c("UC", "nonIBD")) %>%
  tax_mutate(Species = NULL) %>%
  tax_fix()

# Let's make some sample data variables that are easier to use and compare
# in the statistical modelling ahead. We will convert dichotomous
# categorical variables into similar binary variables (values: 1 for true,
# or 0 for false). We will also scale and center the numeric variable for
# age.

phylo <- phylo %>%
  ps_mutate(
    UC = ifelse(DiseaseState == "UC", yes = 1, no = 0),
    female = ifelse(gender == "female", yes = 1, no = 0),
    antibiotics = ifelse(abx == "abx", yes = 1, no = 0),
    steroids = ifelse(steroids == "steroids", yes = 1, no = 0),
    age_scaled = scale(age, center = TRUE, scale = TRUE)
  )

lm_models <- phylo %>%
  tax_fix() %>%
  tax_prepend_ranks() %>%
  tax_transform("compositional", rank = "Genus") %>%
  tax_filter(min_prevalence = 0.1, use_counts = TRUE) %>%
  taxatree_models(
    type = lm,
    trans = "log2", trans_args = list(zero_replace = "halfmin"),
    ranks = c("Phylum", "Class", "Genus"),
    variables = c("UC", "female", "antibiotics", "steroids", "age_scaled")
  )

lm_stats <- lm_models %>% taxatree_models2stats()

# inspect the psExtra returned, now with taxatree_stats dataframe added
lm_stats

# inspect the dataframe itself
```

```
lm_stats %>% taxatree_stats_get()

# keep the models on the psExtra object
lm_models %>% taxatree_models2stats(.keep_models = TRUE)

# you can adjust the p values with taxatree_stats_p_adjust

# you can plot the results with taxatree_plots
```

`taxatree_plotkey` *Draw labelled key to accompany taxatree_plots*

Description

Draw labelled key to accompany taxatree_plots

Usage

```
taxatree_plotkey(
  data,
  ...,
  size_stat = list(prevalence = prev),
  node_size_range = c(1.5, 5),
  edge_width_range = node_size_range * 0.8,
  size_guide = "none",
  size_trans = "identity",
  colour = "lightgrey",
  edge_alpha = 0.7,
  title = "Key",
  title_size = 14,
  taxon_renamer = identity,
  .combine_label = any,
  .draw_label = TRUE,
  .calc_label = TRUE,
  layout = "tree",
  layout_seed = NA,
  circular = identical(layout, "tree"),
  node_sort = NULL,
  add_circles = isTRUE(circular),
  drop_ranks = TRUE
)
```

Arguments

`data`	psExtra (or phyloseq)
`...`	logical conditions for labelling e.g. rank == "Phylum", p.value < 0.1 | taxon %in% listOfTaxa

`size_stat`	named list of length 1, giving function calculated for each taxon, to determine the size of nodes (and edges). Name used as size legend title.
`node_size_range`	min and max node sizes, decrease to avoid node overlap
`edge_width_range`	min and max edge widths
`size_guide`	guide for node sizes, try "none", "legend" or ggplot2::guide_legend()
`size_trans`	transformation for size scale you can use (the name of) any transformer from the scales package, such as "identity", "log1p", or "sqrt"
`colour`	fixed colour and fill of nodes and edges
`edge_alpha`	fixed alpha value for edges
`title`	title of plot (NULL for none)
`title_size`	font size of title
`taxon_renamer`	function that takes taxon names and returns modified names for labels
`.combine_label`	all or any: function to combine multiple logical "label" values for a taxon (relevant if taxatree_stats already present in data)
`.draw_label`	should labels be drawn, or just the bare tree, set this to FALSE if you want to draw customised labels with taxatree_plot_labels afterwards
`.calc_label`	if you already set labels with taxatree_label: set this to FALSE to use / avoid overwriting that data (ignores `...` if FALSE)
`layout`	any ggraph layout, default is "tree"
`layout_seed`	any numeric, required if a stochastic igraph layout is named
`circular`	should the layout be circular?
`node_sort`	sort nodes by "increasing" or "decreasing" size? NULL for no sorting. Use `tax_sort()` before `taxatree_plots()` for finer control.
`add_circles`	add faint concentric circles to plot, behind each rank?
`drop_ranks`	drop ranks from tree if not included in stats dataframe

Value

ggplot

Examples

```
library(ggplot2)
# Normally you make a key to accompany taxatree_plots showing stats results
# So see ?taxatree_plots examples too!
#
# You can also use the taxatree key visualization just to help understand your
# tax_table contents and hierarchical structure
shao19 %>%
  ps_filter(family_id %in% 1:5) %>%
  tax_filter(min_prevalence = 7) %>%
  tax_fix() %>%
  tax_agg("genus") %>%
  tax_prepend_ranks() %>%
  taxatree_plotkey(rank %in% c("phylum", "genus"))


# # Let's look at some of the most prevalent Actinobacteria
actinoOnlyPhylo <- shao19 %>%
  tax_select("Actinobacteria", ranks_searched = "phylum") %>%
  tax_filter(min_prevalence = 100)

actinoOnlyPhylo %>%
  tax_mutate(order = NULL) %>%
  tax_sort(by = "prev", at = "species", tree_warn = FALSE) %>%
  taxatree_plotkey(
    circular = FALSE, rank != "genus",
    .draw_label = FALSE, # suppress drawing now so we can make custom labels after
    size_guide = "legend", colour = "skyblue", edge_alpha = 0.2
  ) %>%
  taxatree_plot_labels(
    circular = FALSE, # be sure you match the plotkey layout
    fun = geom_text, fontface = "bold.italic", size = 2.5, hjust = 0,
    nudge_y = 0.1 # nudge y if you want to nudge x (due to coord_flip!)
  ) +
  coord_flip(clip = "off") +
  scale_y_reverse(expand = expansion(add = c(0.2, 2))) +
  theme(legend.position = c(0.15, 0.25), legend.background = element_rect())


# You can even choose other tree layouts from igraph (e.g. kk)
# and configure multiple styles of custom labels on one plot
set.seed(1) # set seed for reproducibility of ggrepel label positions
actinoOnlyPhylo %>%
  tax_mutate(order = NULL) %>%
  taxatree_label(rank == "family", .label_var = "family_lab") %>%
  taxatree_label(rank == "species", .label_var = "species_lab") %>%
  taxatree_plotkey(
    circular = FALSE, rank != "genus", layout = "kk",
    .draw_label = FALSE, # important not to draw the default labels
    colour = "skyblue", edge_alpha = 0.2
  ) %>%
  taxatree_plot_labels(
    circular = FALSE, label_var = "family_lab", fun = geom_label,
    fontface = "bold.italic", colour = "orange", fill = "black", size = 3
  ) %>%
  taxatree_plot_labels(
    circular = FALSE, label_var = "species_lab", fun = ggrepel::geom_text_repel,
    fontface = "bold.italic", size = 2, force = 20, max.time = 0.1
  )
```

`taxatree_plots` *Plot statistical model results for all taxa on a taxonomic tree*

Description

- Uses a psExtra object to make a tree graph structure from the taxonomic table.
- Then adds statistical results stored in "taxatree_stats" of psExtra data
- You must use `taxatree_models()` first to generate statistical model results.
- You can adjust p-values with `taxatree_stats_p_adjust()`

Usage

```
taxatree_plots(
  data,
  colour_stat = "estimate",
  palette = "Green-Brown",
  reverse_palette = FALSE,
  colour_lims = NULL,
  colour_oob = scales::oob_squish,
  colour_trans = "abs_sqrt",
  size_stat = list(prevalence = prev),
  node_size_range = c(1, 4),
  edge_width_range = node_size_range * 0.8,
  size_guide = "legend",
  size_trans = "identity",
  sig_stat = "p.value",
  sig_threshold = 0.05,
  sig_shape = "circle filled",
  sig_size = 0.75,
  sig_stroke = 0.75,
  sig_colour = "white",
  edge_alpha = 0.7,
  vars = "term",
  var_renamer = identity,
  title_size = 10,
  layout = "tree",
  layout_seed = NA,
  circular = identical(layout, "tree"),
  node_sort = NULL,
  add_circles = isTRUE(circular),
  drop_ranks = TRUE,
  l1 = if (palette == "Green-Brown") 10 else NULL,
  l2 = if (palette == "Green-Brown") 85 else NULL,
  colour_na = "grey35"
)
```

Arguments

`data`	psExtra with taxatree_stats, e.g. output of `taxatree_models2stats()`
`colour_stat`	name of variable to scale colour/fill of nodes and edges
`palette`	diverging hcl colour palette name from `colorspace::hcl_palettes("diverging")`

`reverse_palette`	reverse direction of colour palette?
`colour_lims`	limits of colour and fill scale, NULL will infer lims from range of all data
`colour_oob`	scales function to handle colour_stat values outside of colour_lims (default simply squishes "out of bounds" values into the given range)
`colour_trans`	name of transformation for colour scale: default is "abs_sqrt", the square-root of absolute values, but you can use the name of any transformer from the `scales` package, such as "identity" or "exp"
`size_stat`	named list of length 1, giving function calculated for each taxon, to determine the size of nodes (and edges). Name used as size legend title.
`node_size_range`	min and max node sizes, decrease to avoid node overlap
`edge_width_range`	min and max edge widths
`size_guide`	guide for node sizes, try "none", "legend" or ggplot2::guide_legend()
`size_trans`	transformation for size scale you can use (the name of) any transformer from the scales package, such as "identity", "log1p", or "sqrt"
`sig_stat`	name of variable indicating statistical significance
`sig_threshold`	value of sig_stat variable indicating statistical significance (below this)
`sig_shape`	fixed shape for significance marker
`sig_size`	fixed size for significance marker
`sig_stroke`	fixed stroke width for significance marker
`sig_colour`	fixed colour for significance marker (used as fill for filled shapes)
`edge_alpha`	fixed alpha value for edges
`vars`	name of column indicating terms in models (one plot made per term)
`var_renamer`	function to rename variables for plot titles
`title_size`	font size of title
`layout`	any ggraph layout, default is "tree"
`layout_seed`	any numeric, required if a stochastic igraph layout is named
`circular`	should the layout be circular?
`node_sort`	sort nodes by "increasing" or "decreasing" size? NULL for no sorting. Use `tax_sort()` before `taxatree_plots()` for finer control.
`add_circles`	add faint concentric circles to plot, behind each rank?
`drop_ranks`	drop ranks from tree if not included in stats dataframe
`l1`	Luminance value at the scale endpoints, NULL for palette's default
`l2`	Luminance value at the scale midpoint, NULL for palette's default
`colour_na`	colour for NA values in tree. (if unused ranks are not dropped, they will have NA values for colour_stat)

Details

`taxatree_plotkey` plots same layout as `taxatree_plots`, but in a fixed colour

See website article for more examples of use: https://david-barnett.github.io/microViz/articles/web-only/modelling-taxa.html

Uses ggraph, see help for main underlying graphing function with `?ggraph::ggraph`

It is possible to provide multiple significance markers for multiple thresholds, by passing vectors to the sig_shape, sig_threshold, etc. arguments. It is critically important that the thresholds are provided in decreasing order of severity, e.g. sig_threshold = c(0.001, 0.01, 0.1) and you must provide a shape value for each of them.

Value

list of ggraph ggplots

See Also

`taxatree_models()` to calculate statistical models for each taxon

`taxatree_plotkey()` to plot the corresponding labelled key

`taxatree_plot_labels()` and `taxatree_label()` to add labels

`taxatree_stats_p_adjust()` to adjust p-values

Examples

```
# Limited examples, see website article for more

library(dplyr)
library(ggplot2)

data(dietswap, package = "microbiome")
ps <- dietswap

# create some binary variables for easy visualisation
ps <- ps %>% ps_mutate(
  female = if_else(sex == "female", 1, 0, NaN),
  african = if_else(nationality == "AFR", 1, 0, NaN)
)

# This example dataset has some taxa with the same name for phylum and family...
# We can fix problems like this with the tax_prepend_ranks function
# (This will always happen with Actinobacteria!)
ps <- tax_prepend_ranks(ps)

# filter out rare taxa
ps <- ps %>% tax_filter(
  min_prevalence = 0.5, prev_detection_threshold = 100
)

# delete the Family rank as we will not use it for this small example
# this is necessary as taxatree_plots can only plot consecutive ranks
```

```
ps <- ps %>% tax_mutate(Family = NULL)

# specify variables used for modelling
models <- taxatree_models(
  ps = ps, type = corncob::bbdml, ranks = c("Phylum", "Genus"),
  formula = ~ female + african, verbose = TRUE
)
# models list stored as attachment in psExtra
models

# get stats from models
stats <- taxatree_models2stats(models, param = "mu")
stats

plots <- taxatree_plots(
  data = stats, colour_trans = "identity",
  size_stat = list(mean = mean),
  size_guide = "legend", node_size_range = c(1, 6)
)

# if you change the size_stat for the plots, do the same for the key!!
key <- taxatree_plotkey(
  data = stats,
  rank == "Phylum" | p.value < 0.05, # labelling criteria
  .combine_label = all, # label only taxa where criteria met for both plots
  size_stat = list(mean = mean),
  node_size_range = c(2, 7), size_guide = "none",
  taxon_renamer = function(x) {
    stringr::str_remove_all(x, "[PG]: | [ae]t rel.")
  }
)

# cowplot is powerful for arranging trees and key and colourbar legend
legend <- cowplot::get_legend(plots[[1]])
plot_col <- cowplot::plot_grid(
  plots[[1]] + theme(legend.position = "none"),
  plots[[2]] + theme(legend.position = "none"),
  ncol = 1
)
cowplot::plot_grid(key, plot_col, legend, nrow = 1, rel_widths = c(4, 2, 1))
```

`taxatree_plot_labels` *Add labels to taxatree plots/key*

Description

Finer control over label drawing for `taxatree_plotkey` (with .draw_label = FALSE), and label drawing for `taxatree_plots` output too.

Usage

```
taxatree_plot_labels(
  p,
  circular = TRUE,
  taxon_renamer = identity,
  fun = ggrepel::geom_text_repel,
  label_var = "label",
  x_nudge = 0.1,
  y_nudge = 0.025,
  rotate = 0,
  fontface = "bold",
  size = 2.5,
  colour = "grey15",
  max.overlaps = Inf,
  min.segment.length = 0,
  segment.size = 0.15,
  segment.color = "grey15",
  point.padding = 0.05,
  box.padding = 0.1,
  seed = NA,
  ...
)
```

Arguments

`p`	taxatree_plotkey or taxatree_plots output plot
`circular`	is the plot layout circular? labels are drawn differently for circular trees
`taxon_renamer`	function that takes taxon names and returns modified names for labels
`fun`	ggrepel labelling function: geom_text_repel or geom_label_repel
`label_var`	name of variable in taxatree_stats that indicates which taxa to label
`x_nudge`	absolute amount by which the initial position of taxon labels is nudged (relevant only for circular layouts, use nudge_x for other layouts)
`y_nudge`	absolute amount by which the initial position of taxon labels is nudged (relevant only for circular layouts, use nudge_y for other layouts)
`rotate`	angle to rotate labels' outer edges away from horizontal (relevant only for circular layouts, use angle for other layouts)
`fontface`	fontface of label text
`size`	size of labels
`colour`	colour of label outlines and text
`max.overlaps`	max number of overlapping labels tolerated
`min.segment.length`	min length of label line segment to bother drawing
`segment.size`	thickness of line segment
`segment.color`	colour of line segment

`point.padding`	padding around node points (for label positioning)
`box.padding`	padding around labels/text (for label positioning)
`seed`	set this for reproducible label positions
`...`	Arguments passed on to `ggrepel::geom_text_repel`

- `arrow` specification for arrow heads, as created by `arrow`
- `force` Force of repulsion between overlapping text labels. Defaults to 1.
- `force_pull` Force of attraction between a text label and its corresponding data point. Defaults to 1.
- `max.time` Maximum number of seconds to try to resolve overlaps. Defaults to 0.5.
- `max.iter` Maximum number of iterations to try to resolve overlaps. Defaults to 10000.
- `xlim,ylim` Limits for the x and y axes. Text labels will be constrained to these limits. By default, text labels are constrained to the entire plot area.
- `direction` "both", "x", or "y" – direction in which to adjust position of labels
- `verbose` If `TRUE`, some diagnostics of the repel algorithm are printed

---

`taxatree_stats_p_adjust`

Adjust p values in taxatree_stats dataframe

Description

Apply a p value adjustment method from `stats::p.adjust.methods`, such as false-discovery rate adjustment with "BH", or more conservative family-wise error rate controlling methods such as "holm" or "bonferroni".

Usage

```
taxatree_stats_p_adjust(
  data,
  method,
  grouping = "rank",
  p = "p.value",
  new_var = NULL
)
```

Arguments

`data`	psExtra with taxatree_stats dataframe, or just the dataframe
`method`	any method from `stats::p.adjust.methods`
`grouping`	defines grouping of p-values into families for adjustment, see details.
`p`	name of variable containing p values for adjustment
`new_var`	name of new variable created for adjusted p values (automatically inferred by default)

Details

Define how to group the p values for adjustment with the `grouping` argument. The default is to adjust the p values in groups at each taxonomic rank, but you could also adjust per "taxon" or per "term". Or even group by a combination of rank and term with c("rank", "term"). You should specify the name of the new variable containing the adjusted p values in the new_var argument. If left as NULL the new variable name will be created by pasting together p.adj, the method, and the grouping variable(s) separated by ".".

Value

psExtra with dataframe of statistics, or just the data.frame

See Also

`taxatree_models2stats`

`taxatree_models`

`stats::p.adjust`

Examples

```
# This example is an abbreviated excerpt from article on taxon modelling on
# the microViz documentation website

library(corncob)
library(dplyr)
data("ibd", package = "microViz")

# We'll keep only the Ulcerative Colitis and Healthy Control samples, to
# simplify the analyses for this example. We'll also remove the Species
# rank information, as most OTUs in this dataset are not assigned to a
# species. We'll also use `tax_fix` to fill any gaps where the Genus is
# unknown, with the family name or whatever higher rank classification is
# known.

phylo <- ibd %>%
  ps_filter(DiseaseState %in% c("UC", "nonIBD")) %>%
  tax_mutate(Species = NULL) %>%
  tax_fix()

# Let's make some sample data variables that are easier to use and compare
# in the statistical modelling ahead. We will convert dichotomous
# categorical variables into similar binary variables (values: 1 for true,
# or 0 for false). We will also scale and center the numeric variable for
# age.

phylo <- phylo %>%
  ps_mutate(
    UC = ifelse(DiseaseState == "UC", yes = 1, no = 0),
    female = ifelse(gender == "female", yes = 1, no = 0),
    antibiotics = ifelse(abx == "abx", yes = 1, no = 0),
    steroids = ifelse(steroids == "steroids", yes = 1, no = 0),
    age_scaled = scale(age, center = TRUE, scale = TRUE)
  )

bb_models <- phylo %>%
  tax_fix() %>%
  tax_prepend_ranks() %>%
  tax_filter(min_prevalence = 0.3) %>%
  taxatree_models(
    type = corncob::bbdml,
    ranks = c("Phylum", "Class", "Order"),
    variables = c("UC", "female", "antibiotics", "steroids", "age_scaled")
  )

bb_stats <- bb_models %>%
  taxatree_models2stats(param = "mu") %>%
  taxatree_stats_p_adjust(method = "BH", grouping = "rank")

bb_stats

bb_stats %>% taxatree_stats_get()

# you can also directly modify the dataframe,
# and choose a different variable name
bb_stats %>%
  taxatree_stats_get() %>%
  taxatree_stats_p_adjust(
    method = "holm", grouping = "taxon", new_var = "p_adj_holm"
  )

# see all available adjustment methods
stats::p.adjust.methods
```

tax_agg *Aggregate taxa and track aggregation in psExtra*

Description

tax_agg sums the abundances of the phyloseq taxa at the given rank. It records the tax_agg rank argument in the info of the psExtra object output. This psExtra object tracks aggregation, and any further transformations and scaling, to help you keep track of what you have done with your phyloseq object and automatically caption ordination plots.

Instead of tax_agg, consider using tax_transform() with a rank argument instead, to both aggregate and transform the taxa. This is also useful when you want to aggregate but not transform the taxa, and yet still log the "identity" transformation in psExtra for captioning your ordination plots. e.g. tax_transform(rank = "Genus", trans = "identity")

tax_agg allows you to pass NA or "unique" to the rank argument which will NOT aggregate the taxa. If you use rank = "unique" or add_unique = TRUE, it will add a new rank called unique, identical to the taxa_names (after any aggregation)

Be aware: you should not use the top_N argument yourself without good reason. top_N provides a feature inspired by the deprecated microbiome function aggregate_top_taxa which is primarily useful for decluttering compositional barplots. microViz comp_barplot (and ord_plot_iris) already run tax_agg with a top_N argument for you, so you should not. The tax_table produced when using top_N is otherwise INVALID FOR MOST OTHER ANALYSES.

Usage

```
tax_agg(
  ps,
  rank = NA,
  sort_by = NA,
  top_N = NA,
  force = FALSE,
  add_unique = FALSE
)
```

Arguments

`ps`	phyloseq object
`rank`	NA (for tax_names level) or name of valid taxonomic rank (try phyloseq::rank_names(ps)) or "unique"
`sort_by`	if not NA, how should the taxa be sorted, uses tax_sort(), takes same options as by arg
`top_N`	NA does nothing, but if top_N is a number, it creates an extra tax_table column called top, which is the same as the unique column for the first top_N number of taxa, and "other" otherwise.
`force`	If TRUE, this forces aggregation at chosen rank to occur regardless of if the output will be sensible! This avoids the "Taxa not unique at rank: ..." error, but may allow very inappropriate aggregation to occur. Do not use force = TRUE unless you know why you are doing this, and what the result will be. If you are getting an error with force = FALSE, it is almost certainly better to examine the tax_table and fix the problem. force = TRUE is similar to microbiome::aggregate_taxa, which also does not check that the taxa are uniquely defined by only the aggregation level.
`add_unique`	if TRUE, adds a rank named unique, identical to the rownames after aggregation

Details

This function is inspired by `microbiome::aggregate_taxa`. However if `microbiome::aggregate_taxa` is used, microViz cannot track this aggregation.

Comparing aggregate_taxa and tax_agg:

Except for the ordering of taxa, and the addition of a "unique" rank being optional (in tax_agg), the resulting phyloseq objects are identical for aggregating a phyloseq with no ambiguous taxa. Taxa are ambiguous when the tax_table converges at a lower rank after branching, such as if two different genera share the same species (e.g. "s__"). `microbiome::aggregate_taxa` handles ambiguous taxa by creating a "unique" rank with all of the taxonomic rank info pasted together into one, often very long, name. `tax_agg` throws an error, and directs the user to `tax_fix()` to fix the ambiguous taxa before aggregation, which should then result in (much) shorter unique names at the aggregation rank.

Value

psExtra object including phyloseq and tax_agg rank info

See Also

tax_fix

tax_fix_interactive

tax_transform

Examples

```
library(phyloseq)
library(dplyr)
data("dietswap", package = "microbiome")

tax_agg(ps = dietswap, "Phylum") %>%
  ps_get() %>%
  tax_table()
tax_agg(ps = dietswap, "Family") %>%
  ps_get() %>%
  tax_table()

# create some missing values
tax_table(dietswap)[3:7, "Genus"] <- "g__"

# this will produce an error, instructing the user to use tax_fix
# tax_agg(ps = dietswap, "Genus")

# this will then work:
dietswap %>%
  tax_fix() %>%
  tax_agg("Genus")

# you can replace unknown values with `tax_fix()`
# which will fix most problems, like the common "g__" and "s__"
# but default tax_fix settings won't catch this long unknown
tax_table(dietswap)[13:17, "Family"] <- "some_unknown_family"
dietswap %>%
  tax_fix(unknowns = "some_unknown_family") %>%
  tax_agg("Family")

# try tax_fix_interactive() to help you find and fix all the uninformative
# and converging values in your taxonomy table.

# the code below won't aggregate taxa,
# but just adds a new rank called unique, equal to taxa_names
```

```
tax_agg(ps = dietswap, rank = NA, add_unique = TRUE)
identical(tax_agg(dietswap, NA, add_unique = TRUE), tax_agg(dietswap, "unique")) # TRUE
```

tax_filter *Filter rare and/or low abundance taxa from a phyloseq object*

Description

Removes taxa (from all samples) that do not meet a given criterion or combination of criteria. If a value for min_prevalence, min_total_abundance or min_sample_abundance is 1 or greater, then it is treated as an absolute minimum number of samples/reads. If <1, it is treated as proportion of all samples/reads. This function is designed to work with counts. otu_table must contain counts particularly if you want to set a non-zero value for min_total_abundance.

Usage

```
tax_filter(
  ps,
  min_prevalence = 1,
  prev_detection_threshold = 1,
  min_total_abundance = 0,
  min_sample_abundance = 0,
  tax_level = NA,
  names_only = FALSE,
  use_counts = TRUE,
  undetected = NULL,
  verbose = TRUE
)
```

Arguments

`ps`	phyloseq or psExtra (ideally with count data available)
`min_prevalence`	number or proportion of samples that a taxon must be present in
`prev_detection_threshold`	min required counts (or value) for a taxon to be considered present in that sample (or set undetected arg)
`min_total_abundance`	minimum total readcount of a taxon, summed across all samples (can be proportion of all counts)
`min_sample_abundance`	taxa must have at least this many reads in one or more samples
`tax_level`	if given, aggregates data at named taxonomic rank before filtering, but returns phyloseq at the ORIGINAL level of aggregation!
`names_only`	if names_only is true return only names of taxa, not the phyloseq
`use_counts`	expect count data in phyloseq otu_table? default is TRUE

`undetected`	e.g. 0, value at (or below) which a taxon is considered not present in that sample. If set, this overrides prev_detection_threshold.
`verbose`	message about proportional prevalence calculations?

Value

filtered phyloseq object AT ORIGINAL LEVEL OF AGGREGATION (not at the level in tax_level)

Examples

```
data("dietswap", package = "microbiome")
# Dropping rare and low abundance taxa #
# Filter at unique taxa level, keeping only those with a prevalence of 10% or more
# and at least 10 thousand reads when summed across all samples.
# Then aggregate to Family level taxonomy.
dietswap %>%
  tax_filter(min_prevalence = 0.1, min_total_abundance = 10000) %>%
  tax_agg("Family")

# Keeping ubiquitous families #
# keep only families that have at least 1000 counts present in 90% of samples
# then aggregate the remaining taxa at 'Genus' level
dietswap %>%
  tax_filter(
    tax_level = "Family", min_prevalence = 0.90,
    prev_detection_threshold = 1000
  ) %>%
  tax_agg("Genus")
```

`tax_fix` *Replace unknown, NA, or short tax_table values*

Description

Identifies phyloseq tax_table values as unknown or uninformative and replaces them with the first informative value from a higher taxonomic rank.

- Short values in phyloseq tax_table are typically empty strings or " ", or "g__" etc. so it is helpful to replace them. (If this is unwanted: set `min_length` = 0 to avoid filtering on length.)
- Values in unknowns are also removed, even if longer than `min_length`. It is up to the user to specify sensible values in unknowns if their dataset has other unwanted values.
- NA values are also replaced.

See this article for an extended discussion of tax_table fixing. https://david-barnett.github.io/microViz/articles/web-only/tax-fixing.html

Usage

```
tax_fix(
  ps,
  min_length = 4,
  unknowns = NA,
  suffix_rank = "classified",
  sep = " ",
  anon_unique = TRUE,
  verbose = TRUE
)
```

Arguments

`ps`	phyloseq or tax_table (taxonomyTable)
`min_length`	replace strings shorter than this
`unknowns`	also replace strings matching any in this vector, NA default vector shown in details!
`suffix_rank`	"classified" (default) or "current", when replacing an entry, should the suffix be taken from the lowest classified rank for that taxon, "classified", or the "current" unclassified rank?
`sep`	character(s) separating new name and taxonomic rank level suffix (see suffix_rank)
`anon_unique`	make anonymous taxa unique by replacing unknowns with taxa_name? otherwise they are replaced with paste("unknown", first_rank_name), which is therefore the same for every anonymous taxon, meaning they will be merged if tax_agg is used. (anonymous taxa are taxa with all unknown values in their tax_table row, i.e. cannot be classified even at highest rank available)
`verbose`	emit warnings when cannot replace with informative name?

Details

By default (unknowns = NA), unknowns is set to a vector containing:

’s__’ ’g__’ ’f__’ ’o__’ ’c__’ ’p__’ ’k__’ ’S__’ ’G__’ ’F__’ ’O__’ ’C__’ ’P__’ ’K__’ ’NA’ ’NaN’ ’ ’ ” ’unknown’ ’Unknown’ ’s__unknown’ ’s__Unknown’ ’s__NA’ ’g__unknown’ ’g__Unknown’ ’g__NA’ ’f__unknown’ ’f__Unknown’ ’f__NA’ ’o__unknown’ ’o__Unknown’ ’o__NA’ ’c__unknown’ ’c__Unknown’ ’c__NA’ ’p__unknown’ ’p__Unknown’ ’p__NA’ ’k__unknown’ ’k__Unknown’ ’k__NA’ ’S__unknown’ ’S__Unknown’ ’S__NA’ ’G__unknown’ ’G__Unknown’ ’G__NA’ ’F__unknown’ ’F__Unknown’ ’F__NA’ ’O__unknown’ ’O__Unknown’ ’O__NA’ ’C__unknown’ ’C__Unknown’ ’C__NA’ ’P__unknown’ ’P__Unknown’ ’P__NA’ ’K__unknown’ ’K__Unknown’ ’K__NA’

Value

object same class as ps

See Also

`tax_fix_interactive` for interactive tax_fix help

Examples

```
library(dplyr)
library(phyloseq)

data(dietswap, package = "microbiome")
ps <- dietswap

# create unknowns to test filling
tt <- tax_table(ps)
ntax <- ntaxa(ps)
set.seed(123)
g <- sample(1:ntax, 30)
f <- sample(g, 10)
p <- sample(f, 3)
tt[g, 3] <- "g__"
tt[f, 2] <- "f__"
tt[p, 1] <- "p__"
tt[sample(1:ntax, 10), 3] <- "unknown"
# create a row with only NAs
tt[1, ] <- NA

tax_table(ps) <- tax_table(tt)

ps
# tax_fix with defaults should solve most problems
tax_table(ps) %>% head(50)

# this will replace "unknown"s as well as short values including "g__" and "f__"
tax_fix(ps) %>%
  tax_table() %>%
  head(50)

# This will only replace short entries, and so won't replace literal "unknown" values
ps %>%
  tax_fix(unknowns = NULL) %>%
  tax_table() %>%
  head(50)

# Change rank suffix and separator settings
tax_fix(ps, suffix_rank = "current", sep = " - ") %>%
  tax_table() %>%
  head(50)

# by default, completely unclassified (anonymous) taxa are named by their
# taxa_names / rownames at all ranks.
# This makes anonymous taxa distinct from each other,
# and so they won't be merged on aggregation with tax_agg.
# If you think your anonymous taxa should merge on tax_agg,
# or you just want them to be named the all same for another reason,
# set anon_unique = FALSE (compare the warning messages)
tax_fix(ps, anon_unique = FALSE)
tax_fix(ps, anon_unique = TRUE)
```

```
# here's a larger example tax_table shows its still fast with 1000s rows,
# from microbiomeutilities package
# library(microbiomeutilities)
# data("hmp2")
# system.time(tax_fix(hmp2, min_length = 1))
```

`tax_fix_interactive` *Shiny app to help you use tax_fix*

Description

Try this app if you get errors with `tax_fix()` that are tricky to work past, or suggestions to use `tax_fix()` that you don't understand.

The app shows you the tax_table of your data (searchable) with unknown values highlighted.

It allows you to interactively modify minimum allowed length and to select further values to be defined as unknown.

It will show you the correct `tax_fix()` code to copy paste into your script to reproduce the interactive filtering.

Usage

```
tax_fix_interactive(data, app_options = list(launch.browser = TRUE))
```

Arguments

`data`	a phyloseq object
`app_options`	options list passed to shinyApp()

Value

nothing

See Also

`tax_fix` for the non-interactive function to use in your scripts

Examples

```
library(dplyr)
library(phyloseq)

# create some problem-filled example tax_table data
data(dietswap, package = "microbiome")
ps <- dietswap
# create unknowns to test filling
tt <- tax_table(ps)
ntax <- ntaxa(ps)
```

```
set.seed(123)
g <- sample(1:ntax, 30)
f <- sample(g, 10)
p <- sample(f, 3)
tt[g, 3] <- "g__"
tt[f, 2] <- "f__"
tt[p, 1] <- "p__"
tt[sample(1:ntax, 10), 3] <- "unknown"
# create a row with only NAs
tt[1, ] <- NA
tax_table(ps) <- tax_table(tt)

# function takes a phyloseq and shows code for how to fix the tax_table
# tax_fix_interactive(data = ps)
```

tax_model *Statistical modelling for individual taxa in a phyloseq*

Description

tax_model provides a simple framework to statistically model the abundance of individual taxa in your data. You can choose which type of statistical model you want to fit, and you can choose at which rank and (optionally) which specific taxa to fit statistical models for. tax_model takes a phyloseq and returns a list of statistical models, one model for each taxon. The same independent variables are used for all models, as specified in variables or formula argument (latter takes precedence).

taxatree_models runs tax_model on every taxon at multiple taxonomic ranks (you choose which ranks with the plural ranks argument), and returns the results as a named nested list designed for use with taxatree_plots. One list per rank, one model per taxon at each rank.

type = "bbdml" will run beta binomial regression model(s) using the corncob package. For bbdml the same formula/variables is/are used for modelling both the abundance and dispersion parameters.

Usage

```
tax_model(
  ps,
  rank,
  type = "lm",
  variables = NULL,
  formula = NULL,
  taxa = NULL,
  use_future = FALSE,
  return_psx = TRUE,
  checkVars = TRUE,
  checkNA = "warning",
  verbose = TRUE,
  trans = "identity",
  trans_args = list(),
  ...
)
```

Arguments

`ps`	phyloseq object
`rank`	name of taxonomic rank to aggregate to and model taxa at
`type`	name of modelling function to use, or the function itself
`variables`	vector of variable names, to be used as model formula right hand side. If variables is a list, not a vector, a model is fit for each entry in list.
`formula`	Right hand side of a formula, as a formula object or character string. Or a list of these. (alternative to variables argument, do not provide both)
`taxa`	taxa to model (named, numbered, logical selection, or defaulting to all if NULL)
`use_future`	if TRUE parallel processing with future is possible, see details.
`return_psx`	if TRUE, list of results will be returned attached to psExtra object
`checkVars`	should the predictor variables be checked for zero variance?
`checkNA`	One of "stop", "warning", "message", or "allow", which indicates whether to check predictor variables for NAs, and how to report any NAs if present?
`verbose`	message about progress and any taxa name modifications
`trans`	name of tax_transform transformation to apply to aggregated taxa before fitting statistical models
`trans_args`	named list of any additional arguments to tax_transform e.g. list(zero_replace = "halfmin")
`...`	extra args passed directly to modelling function

Details

`tax_model` and `taxatree_models` can use parallel processing with the `future` package. This can speed up analysis if you have many taxa to model. Set use_future = TRUE and run a line like this before doing your modelling: `future::plan(future::multisession, workers = 3)` (This requires the future and future.apply packages to be installed.)

Value

psExtra with named list of model objects attached. Or a list of lists of models (if multiple models per taxon). Or just a list (of lists), if return_psx is FALSE.

See Also

`tax_models_get` for the accessor to retrieve model results from psExtra

`taxatree_models` for more details on the underlying approach

Examples

```
library(corncob)
library(dplyr)

data(dietswap, package = "microbiome")
ps <- dietswap

# create some binary variables for easy visualization
ps <- ps %>% ps_mutate(
  female = if_else(sex == "female", 1, 0, NaN),
  overweight = if_else(bmi_group == "overweight", 1, 0, NaN),
  obese = if_else(bmi_group == "obese", 1, 0, NaN)
)

# This example HITChip dataset has some taxa with the same name for phylum and family...
# We can fix problems like this with the tax_prepend_ranks function
ps <- tax_prepend_ranks(ps)

# filter out rare taxa (it is often difficult to fit multivariable models to rare taxa)
ps <- ps %>% tax_filter(min_prevalence = 0.1, min_total_abundance = 10000)

# specify variables used for modelling
VARS <- c("female", "overweight", "obese")

# Model first 3 genera using all VARS as predictors (just for a quick test)
models <- tax_model(ps, type = "bbdml", rank = "Genus", taxa = 1:3, variables = VARS)

# Alternative method using formula arg instead of variables to produce identical results
models2 <- tax_model(
  ps = ps, rank = "Genus", type = "bbdml",
  taxa = 1:3, formula = ~ female + overweight + obese, return_psx = FALSE
)
all.equal(models, models2) # should be TRUE

# Model only one genus, NOTE the modified name,
# which was returned by tax_prepend_ranks defaults
models3 <- ps %>%
  tax_model(
    rank = "Genus", type = "bbdml",
    taxa = "G: Bacteroides fragilis et rel.", variables = VARS
  )

# Model all taxa at multiple taxonomic ranks (ranks 1 and 2)
# using only female variable as predictor
models4 <- taxatree_models(
  ps = ps, type = "bbdml", ranks = 1:2, formula = ~female, verbose = FALSE
)

# modelling proportions with simple linear regression is also possible via type = lm
# and transforming the taxa to compositional first
models_lm <- ps %>%
  tax_transform("compositional") %>%
  tax_model(rank = "Genus", taxa = 1:3, variables = VARS, type = "lm")
```

tax_mutate *Modify or compute new taxonomic ranks in phyloseq*

Description

Add or overwrite tax_table ranks. Use dplyr::mutate() syntax.

Usage

```
tax_mutate(ps, ...)
```

Arguments

ps	phyloseq object with a tax_table, or just a tax_table
...	passed straight to dplyr::mutate (see examples and dplyr::mutate help)

Value

phyloseq object with modified tax_table

See Also

mutate

ps_mutate

Examples

```
library(phyloseq)
library(dplyr)
data("dietswap", package = "microbiome")

# compute new rank
tax_mutate(dietswap, loud_genus = toupper(Genus)) %>%
  tt_get() %>%
  head()

# overwrite a current rank
tax_mutate(dietswap, Genus = toupper(Genus)) %>%
  tt_get() %>%
  head()

# overwrite all ranks
tax_mutate(dietswap, across(everything(), .fns = toupper)) %>%
  tt_get() %>%
  head()

# add a new rank at the beginning
```

```
tax_mutate(dietswap, Root = "Bacteria", .before = 1) %>%
  tt_get() %>%
  head()

# this is an error as ranks can't be any other class than character
# tax_mutate(dietswap, Genus = 1:ntaxa(dietswap))
```

tax_name *Simple way to set unique taxa_names for phyloseq object*

Description

If your current taxa_names aren't what you want (e.g. they are long DNA sequences), this function will help you set sensible unique names.

It combines:

- a prefix like tax, asv, or otu (pick an appropriate prefix or set your own)
- a unique (sequential) number
- classification information from a chosen taxonomic rank (optional)

Usage

```
tax_name(
  ps,
  prefix = c("tax", "asv", "otu")[1],
  rank = NA,
  pad_number = TRUE,
  sep = "_"
)
```

Arguments

`ps`	phyloseq object
`prefix`	e.g. 'tax', 'asv', or 'otu' (or set your own)
`rank`	name of taxonomic rank from which to use classifications in new names
`pad_number`	should unique numbers have zeros added to the front (e.g. 001, 002) to be made the same number of characters?
`sep`	character to separate the unique number and any taxonomic classification info (relevant if rank given)

Details

Don't confuse this with the phyloseq function `taxa_names()` or the newer microViz function `tax_rename()`.

Value

phyloseq object

See Also

tax_rename for a more informative taxon naming tool

phyloseq::taxa_names for accessing and manually setting names

Examples

```
library(phyloseq)
# get example data
data("enterotype")
ps <- enterotype
head(taxa_names(ps)) # these are mostly fine (except the -1), but imagine you wanted new names

# consider storing the original names for reference (e.g. if they are DNA sequences)
old_taxa_names <- taxa_names(ps)

ps <- tax_name(ps)
taxa_names(ps) %>% head()

# probably better to include the genus info to make these names more informative
ps <- tax_name(ps, rank = "Genus")
taxa_names(ps) %>% head()

# store new names with old names in dataframe for reference
names_df <- tibble::tibble(old = old_taxa_names, new = taxa_names(ps))

# alternative settings
tax_name(ps, pad_number = FALSE) %>%
  taxa_names() %>%
  head()
tax_name(ps, prefix = "whateveryoulike") %>%
  taxa_names() %>%
  head()
tax_name(ps, rank = "Genus", sep = "-") %>%
  taxa_names() %>%
  head()
```

---

tax_names2rank *Add taxa_names as last column in phyloseq tax_table*

Description

The taxa names in your phyloseq may specify a further unique classification of your taxa, e.g. ASVs, that is not otherwise represented in the tax_table itself. This function fixes that, and allows you to include this level in taxatree_plots for example.

Usage

```
tax_names2rank(data, colname = "unique")
```

Arguments

`data`	phyloseq object, or psExtra or tax_table (taxonomyTable)
`colname`	name of new rank to add at right side of tax_table

Value

same class object as passed in to data

`tax_palette` *Make a fixed taxa colour palette e.g. for comp_barplot*

Description

Makes a named palette vector from your phyloseq dataset considering overall abundance to assign colours (or some other sorting)

Usage

```
tax_palette(
  data,
  rank,
  n,
  by = sum,
  pal = "brewerPlus",
  add = c(other = "lightgrey"),
  ...
)
```

Arguments

`data`	phyloseq or psExtra
`rank`	taxonomic rank name or "unique"
`n`	number of colours / taxa (not including "other")
`by`	tax sorting method for tax_sort e.g. sum
`pal`	palette name from distinct_palette function
`add`	name = value pairs appended to end of output, or NA for none
`...`	other args are passed to tax_sort

Value

named character vector

Examples

```
library(ggplot2)
data(dietswap, package = "microbiome")

myPal <- tax_palette(dietswap, rank = "Genus", pal = "brewerPlus", n = 40)
myPal %>% tax_palette_plot() # just to check the palette

# plot one subset of data
dietswap %>%
  ps_filter(nationality == "AFR", timepoint == 1, sex == "male") %>%
  comp_barplot(
    tax_level = "Genus", n_taxa = 15,
    bar_outline_colour = NA, bar_width = 0.7,
    palette = myPal, label = NULL
  )

# plot a different subset of data (top taxa differ but colours are the same)
dietswap %>%
  ps_filter(nationality != "AFR", timepoint == 1, sex == "male") %>%
  comp_barplot(
    tax_level = "Genus", n_taxa = 15,
    bar_outline_colour = NA, bar_width = 0.7,
    palette = myPal, label = NULL
  )
```

`tax_palette_plot` *tax_palette plotting helper function*

Description

Check the named palette colour vector you created with tax_palette()

Usage

```
tax_palette_plot(named_pal_vec, max_n = NA)
```

Arguments

`named_pal_vec`	vector of colours named by taxa (e.g. tax_palette output)
`max_n`	NA to display all colours, or limit this

Value

ggplot

Examples

```
library(ggplot2)
data(dietswap, package = "microbiome")

myPal <- tax_palette(dietswap, rank = "Genus", pal = "brewerPlus", n = 40)
myPal %>% tax_palette_plot() # just to check the palette
```

---

tax_prepend_ranks *Add rank prefixes to phyloseq tax_table values*

Description

Prepend the start of rank names to each taxon at each rank (useful particularly in case of duplicated taxa names across ranks, e.g. dietswap dataset)

Usage

```
tax_prepend_ranks(ps, sep = ": ", nchar = 1)
```

Arguments

ps	phyloseq object
sep	characters to paste in between rank initial and taxon name
nchar	number of characters to use from start of rank_names

Value

phyloseq

See Also

tax_fix for fixing other tax_table problems

Examples

```
data("dietswap", package = "microbiome")
phyloseq::tax_table(dietswap) %>% head()
dietswap %>%
  tax_prepend_ranks() %>%
  phyloseq::tax_table() %>%
  head()
```

tax_rename *Make new phyloseq taxa names from classification and taxon abundance info*

Description

Pairs classification at the given rank with a numeric ranking suffix (based on abundance or prevalence data) to automatically create informative taxa names.

Usage

```
tax_rename(
  ps,
  rank,
  sort_by = sum,
  transform_for_sort = "identity",
  pad_digits = "auto",
  sep = " ",
  ...
)
```

Arguments

`ps`	phyloseq object
`rank`	name of rank to use in new taxa names
`sort_by`	how to sort taxa for numbering within rank-based groups (a tax_sort option)
`transform_for_sort`	named of transformation to apply to taxa before sorting
`pad_digits`	how long should the numeric suffixes be? see details
`sep`	character to separate the rank prefixes from numeric suffixes
`...`	additional arguments passed to tax_sort

Details

e.g. "Bacteroides 003" for the third most abundant Bacteroides OTU or ASV.

Taxa are returned in original order, and otu_table is returned un-transformed.

pad_digits options:

- "auto" –> minimum digits to have equal length numbers within groups
- "max" –> minimum digits to have equal length numbers across all groups
- A number: e.g.
  - 3 –> 001, 002, ..., 042, ..., 180, ...
  - 1 –> 1, 2, ..., 42, ..., 180, ...

Value

phyloseq object

See Also

`phyloseq::taxa_names` for accessing and manually setting names

Examples

```
library(phyloseq)
data("ibd", package = "microViz")

ps <- ibd %>%
  tax_filter(min_prevalence = 3) %>%
  tax_fix()

# show a few of the current, uninformative names
taxa_names(ps) %>% head(15)
taxa_names(ps) %>% tail(15)

# change names to genus classification plus number
psNewNames <- ps %>% tax_rename(rank = "Genus")

taxa_names(psNewNames) %>% head(15)
taxa_names(psNewNames) %>% tail(15)

# demonstrate some alternative argument settings
psNewNames2 <- ps %>% tax_rename(
  rank = "Family", sort_by = prev, pad_digits = "max", sep = "-"
)

taxa_names(psNewNames2) %>% head(15)
taxa_names(psNewNames2) %>% tail(15)

ps %>%
  tax_rename(rank = "Genus", pad_digits = 2) %>%
  taxa_names() %>%
  head(15)

# naming improvement on plots example
library(ggplot2)
library(patchwork)

# Overly aggressive OTU filtering to simplify and speed up example
psExample <- ps %>% tax_filter(min_prevalence = 0.4)

# before OTU renaming
before <- psExample %>%
  ps_filter(activity == "inactive") %>%
  tax_names2rank("Taxon") %>%
  comp_barplot(
    tax_level = "Taxon", n_taxa = 12, other_name = "Other",
    merge_other = FALSE, bar_outline_colour = "grey60"
  ) +
  coord_flip() +
  ggtitle("Original taxon names :(")

# after OTU renaming
after <- psExample %>%
  ps_filter(activity == "inactive") %>%
  tax_rename(rank = "Genus", pad_digits = "max") %>%
  tax_names2rank("Taxon") %>%
  comp_barplot(
    tax_level = "Taxon", n_taxa = 12, other_name = "Other",
    merge_other = FALSE, bar_outline_colour = "grey60"
  ) +
  coord_flip() +
  ggtitle("New taxon names :)", "tax_rename(rank = 'Genus', sort_by = sum)")

before + after & theme(legend.text = element_text(size = 8))

# ordination example
psExample %>%
  tax_rename(rank = "Genus", sort_by = sum) %>%
  tax_names2rank("otu") %>%
  tax_transform("clr", rank = "otu") %>%
  ord_calc() %>%
  ord_plot(
    size = 2, colour = "ibd", shape = "circle", alpha = 0.5,
    plot_taxa = 1:10,
    tax_vec_length = 0.5,
    tax_lab_style = tax_lab_style(
      type = "text", max_angle = 90, check_overlap = TRUE,
      size = 2.5, fontface = "bold"
    ),
    tax_vec_style_all = vec_tax_all(alpha = 0.1)
  ) +
  coord_fixed(clip = "off") +
  stat_chull(aes(colour = ibd)) +
  scale_colour_brewer(palette = "Dark2") +
  theme(panel.grid = element_line(size = 0.1))
```

`tax_reorder` *Reorder taxa in phyloseq object using vector of names*

Description

Reorder taxa in phyloseq object using vector of names

Usage

```
tax_reorder(
  ps,
  tax_order,
  tree_warn = TRUE,
  unmatched_warn = TRUE,
  ignore = c("other", "Other")
)
```

Arguments

`ps`	phyloseq object
`tax_order`	Names of taxa in desired order; at least some must match. (Numerical indices are also possible)
`tree_warn`	If phylogenetic tree is present in phyloseq phy_tree slot, taxa cannot be reordered. Default behaviour of tax_sort is to remove the phylogenetic tree and warn about this. tree_warn = FALSE will suppress the warning message, but still remove the tree!
`unmatched_warn`	Warn if any names (or indices) given in tax_order are not found within (range of) taxa_names(ps) - these will be ignored
`ignore`	Values that you do not want to be used for reordering taxa (useful for comp_barplot when custom palette names are used to set tax_order)

Value

phyloseq object (always without phy_tree)

Examples

```
data("dietswap", package = "microbiome")
new_order <- c(
  "Fusobacteria", "Cyanobacteria", "Verrucomicrobia", "Spirochaetes",
  "Actinobacteria", "Firmicutes", "Proteobacteria", "Bacteroidetes"
)
tax_agg(dietswap, rank = "Phylum") %>%
  ps_get() %>%
  phyloseq::taxa_names()

tax_agg(dietswap, rank = "Phylum") %>%
  ps_get() %>%
  tax_reorder(tax_order = new_order) %>%
  phyloseq::taxa_names()

# partial reordering (of the frontmost positions only) is possible
tax_agg(dietswap, rank = "Phylum") %>%
  ps_get() %>%
  tax_reorder(tax_order = c("Cyanobacteria", "Bacteroidetes")) %>%
  phyloseq::taxa_names()
```

tax_scale *Mean-center and SD-scale taxa in phyloseq*

Description

Wrapper for applying base scale function to phyloseq otu_table

Usage

```
tax_scale(data, center = TRUE, scale = TRUE, do = NA, keep_counts = TRUE)
```

Arguments

data	phyloseq or psExtra or otu_table
center	if TRUE: center each taxon by subtracting its mean
scale	if TRUE, divide each centred taxon by its standard deviation (or divide by RMS if not centred!)
do	alternative argument that overrides center and scale options! takes "both", "scale", "center" or "neither"
keep_counts	if TRUE, retain the original count data in psExtra counts slot

Examples

```
data("dietswap", package = "microbiome")
ps <- dietswap
ps %>%
  otu_get() %>%
  .[1:6, 1:6]

# standard use (mean center and SD scale)
tax_scale(ps) %>%
  otu_get() %>%
  .[1:6, 1:6] # Aerococcus is NaN as standard deviation = 0 (0 prevalence)

# RMS scale only (directly on otu_table)
otu_get(ps) %>%
  tax_scale(center = FALSE) %>%
  .[1:6, 1:6] # Aerococcus is NaN as standard deviation = 0 (0 prevalence)

# example using alternative `do` argument (to center only, no scaling)
tax_scale(ps, do = "center") %>%
  otu_get() %>%
  .[1:6, 1:6]

# preserves existing info
tax_transform(ps, "compositional", rank = "Genus") %>% tax_scale()

# drops other psExtra objects previously calculated with unscaled data
```

```
psxDist <- tax_agg(ps, "Genus") %>% dist_calc()
psxDist
psxDist %>% tax_scale()
tax_scale(psxDist) %>% info_get()
```

tax_select — *Subset phyloseq object by (partial) taxa names*

Description

Convenient name-based taxa selection/filtering of phyloseq object, including approximate name matching. Takes a phyloseq with tax table and a (partial) taxonomic name, or a list/vector of taxonomic names (full or partial matches).

Usage

```
tax_select(
  ps,
  tax_list,
  ranks_searched = "all",
  strict_matches = FALSE,
  n_typos = 1,
  deselect = FALSE
)
```

Arguments

`ps`	phyloseq object
`tax_list`	e.g. c('g__Bifidobacterium', 'g__Akkermansia', 'g__Bacteroides', 'g__Streptococcus')
`ranks_searched`	'all' or a list of which taxonomic ranks should be searched for the names in tax_list?
`strict_matches`	only perfect full name matches allowed if TRUE
`n_typos`	how many typos to allow in each name? uses agrep approximate matching if > 0
`deselect`	if TRUE, the matching taxa will be REMOVED instead!

Details

tax_select will also search the otu names/rownames, BUT only for perfect matches.

Value

phyloseq object with fewer taxa

See Also

`ps_select` for selecting variables in phyloseq sample_data

`agrep` for the function that powers the approximate matching in tax_select

Examples

```
# Get example phyloseq object data
data("dietswap", package = "microbiome")
pSeq <- dietswap

# SELECTION EXAMPLES #
a <- pSeq %>% tax_select(tax_list = "Bif", n_typos = 0, ranks_searched = "Genus")
b <- pSeq %>% tax_select(tax_list = "Bifidobacterium", n_typos = 0)
c <- pSeq %>% tax_select(tax_list = "Bif", n_typos = 1)
identical(a, b) # TRUE
identical(a, c) # FALSE

pSeq %>% tax_select(tax_list = "Bifidobactrium") # default 1 typo allowed
one <- pSeq %>% tax_select(tax_list = "Akkarmensia", n_typos = 2)
two <- pSeq %>% tax_select(tax_list = "Akkermansia", n_typos = 0)
identical(one, two) # TRUE

# DESELECTION EXAMPLE # #
pSeq %>% tax_select(tax_list = "Bif", strict_matches = FALSE, deselect = TRUE)
# Incorrect example
# pSeq %>% tax_select(tax_list = "Bif", strict_matches = TRUE) # fails
```

`tax_sort` *Sort taxa in phyloseq otu_table and tax_table*

Description

Multiple ways of sorting taxa are possible and determined by the by argument. The by argument must be one of:

- 'rev' to reverse the current order
- 'name' (sort alphabetically by `at`)
- 'asis' to keep current order as is
- a sample name (descending abundance sorting within that sample)
- summary stat. function e.g. `sum` or `mean`

The `at` argument must be "names" for sorting unique taxa, or a rank name, for sorting at that rank. `at` is ignored when by is "rev".

Usage

```
tax_sort(
  data,
  by = "name",
  at = "names",
  ...,
  tree_warn = TRUE,
  verbose = TRUE,
  trans = "identity",
  use_counts = TRUE
)
```

Arguments

`data`	psExtra or phyloseq
`by`	how to sort, see description
`at`	"names" or a taxonomic rank to apply sorting method to, as specified in `by`.
`...`	used if summary function given, or pass `undetected` arg for tax_transform("binary") if by = "prev" or "prevalence"
`tree_warn`	If phylogenetic tree is present in phyloseq phy_tree slot, taxa cannot be re-ordered. Default behaviour of tax_sort is to remove the phylogenetic tree and warn about this. tree_warn = FALSE will suppress the warning message, but still remove the tree!
`verbose`	passed to phyloseq_validate verbose (if TRUE: message about suspicious values in tax_table, and how to fix)
`trans`	name of transformation to apply to taxa before sorting (taxa are returned un-transformed)
`use_counts`	use count data if available, instead of transformed data

Details

Don't forget to pass `na.rm = TRUE` to `...` if using a summary stat function in `by`

Value

sorted phyloseq or psExtra

Examples

```
library(phyloseq)
data("dietswap", package = "microbiome")
dietswap

# reverse current order
dietswap %>%
  tax_sort("rev") %>%
  tax_table() %>%
  head(30)

# sort alphabetically by a taxonomic rank (or "names" for taxa_names)
dietswap %>%
  tax_sort(by = "name", at = "Phylum") %>%
  tax_table() %>%
  head(30)

# sequentially sorting by higher ranks
# sets tax_table in nested alphabetical order
dietswap %>%
  tax_sort(at = "names") %>%
  tax_sort(at = "Genus") %>%
  tax_sort(at = "Family") %>%
  tax_sort(at = "Phylum") %>%
  tax_table() %>%
  head(30)

# sort by function e.g. total sum or median abundance
dietswap %>%
  tax_sort(by = sum) %>%
  taxa_names() %>%
  head(20)

# transform to compositional data (proportions) before sorting
# note that abundances are returned untransformed
dietswap %>%
  tax_sort(by = sum, trans = "compositional") %>%
  taxa_names() %>%
  head(20)

# order by descending abundance in a single named sample
dietswap %>%
  tax_sort(by = "Sample-1") %>%
  otu_table() %>%
  .[1:8, 1:4]


# sum order should always equal mean order if non-negative abundances
# don't forget to add na.rm = TRUE if you expect NAs in otu_table somehow
dietswap %>%
  tax_sort(by = sum, na.rm = TRUE) %>%
  taxa_names() %>%
  head(20)

# if your phyloseq object has a phylogenetic tree,
# tax_sort will remove the tree, and warn you about this
# unless you disable that warning with tree_warn = FALSE

# You can sort by abundance at higher taxonomic ranks,
# without losing lower rank info
# e.g. sort (descending) by phyla abundances
dietswap %>%
  tax_sort(by = sum, at = "Phylum") %>%
  tax_table() %>%
  head()

# You can sort by ascending abundance (or prevalence etc) by reversing after
dietswap %>%
  tax_sort(by = "prev", at = "Phylum") %>%
  tax_sort(by = "rev") %>%
  tax_table() %>%
  head()
```

`tax_sort_ord` *Order taxa in phyloseq by their loading vectors*

Description

`tax_sort_ord` reorders taxa in a phyloseq object based on the relative length of their taxa scores / "loading" vector lengths on 1 or 2 ordination axes.

`ord_order_taxa` gets the taxa names in order from the ordination contained in a psExtra object. This is used internally by `tax_sort_ord`.

Usage

```
tax_sort_ord(ps, ord, axes = 1:2, scaling = 2)

ord_order_taxa(ord, axes = 1:2, scaling = 2)
```

Arguments

`ps`	phyloseq object to be sorted
`ord`	psExtra with ordination object
`axes`	which axes to use for sorting? numerical vector of length 1 or 2
`scaling`	Type 2, or type 1 scaling. For more info, see `https://sites.google.com/site/mb3gustame/constrained-analyses/redundancy-analysis`. Either "species" or "site" scores are scaled by (proportional) eigenvalues, and the other set of scores is left unscaled (from ?vegan::scores.cca)

See Also

- These functions were created to support ordering of taxa bars on `ord_plot_iris`
- `ps_sort_ord` for ordering samples in phyloseq by ordination

tax_top *Get names of "top" n taxa*

Description

Simple wrapper around tax_sort that:

1. optionally aggregates taxa at `rank`
2. sorts the aggregated taxa according to `by`
3. returns the top `n` number of taxa names

Usage

```
tax_top(data, n = 10, by = sum, rank = "unique", use_counts = FALSE, ...)
```

Arguments

`data`	phyloseq object or psExtra
`n`	how many taxa names to return, or NA for all (can return fewer than n values, if there are fewer to return)
`by`	how to sort taxa (see `?tax_sort()`), defaults to `sum` which sorts by total abundance across all samples
`rank`	taxonomic rank to aggregate at before calculating ("unique" = no aggregation)
`use_counts`	use count data if available, instead of transformed data
`...`	Arguments passed on to `tax_sort` `verbose` passed to phyloseq_validate verbose (if TRUE: message about suspicious values in tax_table, and how to fix) `trans` name of transformation to apply to taxa before sorting (taxa are returned un-transformed)

Value

vector of taxa names at chosen rank

See Also

`tax_agg` for more info on taxonomic aggregation

`tax_sort` for more info on sorting taxa

Examples

```
data("dietswap", package = "microbiome")
tax_top(dietswap)
tax_top(dietswap, n = 4, by = "prev", rank = "Phylum", undetected = 30)
```

tax_transform *Transform taxa in phyloseq object and record transformation*

Description

Transform taxa features, and optionally aggregate at specified taxonomic rank beforehand. You can pipe the results of `tax_agg` into `tax_transform`, or equivalently set the rank argument in `tax_transform`.

Usage

```
tax_transform(
  data,
  trans,
  rank = NA,
  keep_counts = TRUE,
  chain = FALSE,
  zero_replace = 0,
  add = 0,
  transformation = NULL,
  ...
)
```

Arguments

`data`	a phyloseq object or psExtra output from `tax_agg`
`trans`	any valid taxa transformation (e.g. from `microbiome::transform`)
`rank`	If data is phyloseq: data are aggregated at this rank before transforming. If NA, runs tax_agg(data, rank = NA). If rank is NA and data is already psExtra, any preceding aggregation is left as is.
`keep_counts`	if TRUE, store the pre-transformation count data in psExtra counts slot
`chain`	if TRUE, transforming again is possible when data are already transformed i.e. multiple transformations can be chained with multiple tax_transform calls
`zero_replace`	Replace any zeros with this value before transforming. Either a numeric, or "halfmin" which replaces zeros with half of the smallest value across the entire dataset. Beware: the choice of zero replacement is not tracked in the psExtra output.
`add`	Add this value to the otu_table before transforming. If `add` != 0, `zero_replace` does nothing. Either a numeric, or "halfmin". Beware: this choice is not tracked in the psExtra output.
`transformation`	deprecated, use `trans` instead!
`...`	any extra arguments passed to `microbiome::transform` or pass undetected = `a number` when using trans = "binary"

Details

This function often uses `microbiome::transform` internally and can perform the same transformations, including many from `vegan::decostand` (where the default MARGIN = 2). See below for notes about some of the available transformations.

`tax_transform` returns a `psExtra` containing the transformed phyloseq object and extra info (used for annotating `ord_plot` ordinations):

- tax_transform (a string recording the transformation),
- tax_agg (a string recording the taxonomic aggregation rank if specified here or earlier in `tax_agg`).

A few commonly used transformations:

- "clr", or "rclr", perform the centered log ratio transformation, or the robust clr, using `microbiome::transform`
- "compositional" converts the data into proportions, from 0 to 1.
- "identity" does not transform the data, and records this choice for `ord_plot`
- "binary" can be used to transform tax abundances into presence/abundance data.
- "log2" which performs a log base 2 transformation (don't forget to set zero_replace if there are any zeros in your data)

Value

`psExtra` object including phyloseq and extra info

(r)clr transformation note

If any values are zero, the clr transform routine first adds a small pseudocount of min(relative abundance)/2 to all values. To avoid this, you can replace any zeros in advance by setting zero_replace to a number > 0.

The rclr transform does not replace zeros. Instead, only non-zero features are transformed, using the geometric mean of non-zero features as denominator.

Binary transformation notes

By default, otu_table values of 0 are kept as 0, and all positive values are converted to 1 (like `decostand(method = "pa")`). You can set a different threshold, by passing e.g. undetected = 10, for example, in which case all abundances of 10 or below would be converted to 0. All abundances above 10 would be converted to 1s.

Beware that the choice of detection threshold is not tracked in the psExtra.

See Also

`microbiome::transform` for some more info on available transformations

`vegan::decostand` for even more transformation options

`tax_agg`

Examples

```
data("dietswap", package = "microbiome")

# aggregate taxa at Phylum level and center log ratio transform the phyla counts
tax_transform(dietswap, trans = "clr", rank = "Phylum")

# this is equivalent to the two-step method (agg then transform)
tax_agg(dietswap, rank = "Phylum") %>% tax_transform("clr")

# does nothing except record tax_agg as "unique" and tax_transform as "identity" in psExtra info
dietswap %>% tax_transform("identity", rank = NA)

# binary transformation (convert abundances to presence/absence or detected/undetected)
tax_transform(dietswap, trans = "binary", rank = "Genus")
# change detection threshold by setting undetected argument (default is 0)
tax_transform(dietswap, trans = "binary", rank = "Genus", undetected = 50) %>%
  otu_get() %>%
  .[1:6, 1:4]

# log2 transformation after replacing all zeros with a pseudocount of 1
tax_transform(dietswap, trans = "log2", rank = "Family", zero_replace = 1)

# log2 transformation after replacing all zeros with a pseudocount of half
# the minimum non-zero count value in the aggregated dataset
tax_transform(dietswap, trans = "log2", rank = "Family", zero_replace = "halfmin")
```

---

upgrade_ps_extra_to_psExtra

Convert old format "ps_extra" objects to new "psExtra" objects

---

Description

This will only be necessary if you have saved old format "ps_extra" objects generated by an old microViz version (< 0.10.0), and you cannot or do not want to regenerate these old format objects from your original phyloseq object.

Usage

```
upgrade_ps_extra_to_psExtra(ps_extra)
```

Arguments

ps_extra	an old format "ps_extra" object, as generated by old microViz versions (< 0.10.0)

Value

new format "psExtra" S4 object

Examples

```
# read your old saved 'ps_extra' object that you want to keep using
# oldObject <- readRDS("old-object-path.rds")
# newObject <- upgrade_ps_extra_to_psExtra(oldObject)
# continue with your next analysis or plotting steps...
```

---

varAnnotation *Helper to specify a HeatmapAnnotation for variables in cor_heatmap*

Description

Helper to specify a HeatmapAnnotation for variables in cor_heatmap

Usage

```
varAnnotation(
  ...,
  name,
  annotation_legend_param = list(),
  show_legend = TRUE,
  gp = grid::gpar(col = NA),
  border = FALSE,
  gap = grid::unit(2, "mm"),
  show_annotation_name = TRUE,
  annotation_label = NULL,
  annotation_name_gp = grid::gpar(),
  annotation_name_offset = NULL,
  annotation_name_rot = NULL,
  annotation_name_align = FALSE,
  annotation_name_side = "auto",
  .data = NULL,
  .vars = NULL,
  .side = NULL
)
```

Arguments

`...`	Name-value pairs where the names correspond to annotation names and values are the output of variable annotation functions such as anno_var_box(), or manually specified AnnotationFunction objects
`name`	Name of the heatmap annotation, optional.
`annotation_legend_param`	A list which contains parameters for annotation legends. See `color_mapping_legend,ColorMapping-me` for all possible options.
`show_legend`	Whether show annotation legends. The value can be one single value or a vector.
`gp`	Graphic parameters for simple annotations (with `fill` parameter ignored).

`border`	border of single annotations.
`gap`	Gap between annotations. It can be a single value or a vector of `unit` objects.
`show_annotation_name`	Whether show annotation names? For column annotation, annotation names are drawn either on the left or the right, and for row annotations, names are draw either on top or at the bottom. The value can be a vector.
`annotation_label`	Labels for the annotations. By default it is the same as individual annotation names.
`annotation_name_gp`	Graphic parameters for annotation names. Graphic parameters can be vectors.
`annotation_name_offset`	Offset to the annotation names, a `unit` object. The value can be a vector.
`annotation_name_rot`	Rotation of the annotation names. The value can be a vector.
`annotation_name_align`	Whether to align the annotation names.
`annotation_name_side`	Side of the annotation names.
`.data`	OPTIONAL phyloseq or psExtra, only set this to override use of same data as in heatmap
`.vars`	OPTIONAL selection vector of variables (names, numbers or logical), only set this if providing .data argument to override default
`.side`	OPTIONAL string, indicating the side for the variable annotations: only set this to override default

Value

HeatmapAnnotation object

See Also

`taxAnnotation()`

Index