Package ‘treemap’

July 2, 2014

Type Package

Title Treemap visualization

Version 2.2

Date 2014-03-31

Author Martijn Tennekes

Maintainer Martijn Tennekes <mtennekes@gmail.com>

Description A treemap is a space-filling visualization of hierarchical structures. This package offers great flexibility to draw treemaps.

License GPL-3

LazyLoad yes

Imports
colorspace, data.table (>= 1.8.8), ggplot2, grid, gridBase,igraph, RColorBrewer, shiny (>= 0.8.0)

NeedsCompilation no

Repository CRAN

Date/Publication 2014-03-31 15:37:39

R topics documented:

`treemap-package` *Treemap package*

Description

Package:	treemap
Type:	Package
Version:	2.2
Date:	2014-03-31
License:	GPL-3
LazyLoad:	yes

Details

A treemap is a space-filling visualization of hierarchical structures. This package offers great flexibility to draw treemaps.

The main function is `treemap`. See also `itreemap` for a graphical user interface to create treemaps. By default Tree Colors are used, which are colors from the HCL color space. Use `treecolors` to experiment with the parameter settings.

Author(s)

Martijn Tennekes <`mtennekes@gmail.com`>

`business` *Fictitious Business Statistics Data*

Description

Fictitious (aggregated) business statistics data. The index variables (`NACE1` to `NACE4`) are derived from the Statistical Classification of Economic Activities in the European Community (NACE). The variables `turnover(.prev)` and `employees(.prev)` have values for NACE codes in the business economy domain only.

References

Statistical Classification of Economic Activities in the European Community (NACE) Eurostat - Structural business statistics (SBS)

`GNI2010` *GNI 2010 Data*

Description

Gross national income (per capita) in dollars per country in 2010.

References

Website of The World Bank - Health Nutrition and Population Statistics

`itreemap` *Interactive user interface for treemap*

Description

This function is an interactive web-based user interface for creating treemaps. Interaction is provided for the four main input arguments of (`treemap`) besides the data.frame itself, namely `index`, `vSize`, `vColor` and `type`. Zooming in and out is possible. Command line outputs are generated in the console.

Usage

```
itreemap(dtf = NULL, index = NULL, vSize = NULL, vColor = NULL,
  type = NULL, height = 700, command.line.output = TRUE)
```

Arguments

`dtf`	a data.frame (`treemap`) If not provided, then the first data.frame in the global workspace is loaded.
`index`	index variables (up to four). See `treemap`.
`vSize`	name of the variable that determine the rectangle sizes.
`vColor`	name of the variable that determine the rectangle colors. See `treemap`.
`type`	treemap type. See `treemap`.
`height`	height of the plotted treemap in pixels. Tip: decrease this number if the treemap doesn’t fit conveniently.
`command.line.output`	if `TRUE`, the command line output of the generated treemaps are provided in the console.

Examples

```
## Not run:
data(business)
itreemap(business)

## End(Not run)
```

random.hierarchical.data

Create random hierarchical data

Description

This function generates random hierarchical data. Experimental.

Usage

```
random.hierarchical.data(n = NULL, method = "random", number.children = 3,
  children.root = 4, depth = 3, nodes.per.layer = NULL,
  labels = c("LETTERS", "numbers", "letters"), labels.prefix = NULL,
  sep = ".", colnames = c(paste("index", 1:depth, sep = ""), "x"),
  value.generator = rlnorm, value.generator.args = NULL)
```

Arguments

`n`	number of leaf nodes. This is a shortcut argument. If specified, the method is set to `"random.arcs"` with a nodes.per.layer such that the average number of children per layer is as constant as possible.
`method`	one of `"random"`**:** Random tree where for each node, the number of children, is determined by a random poisson generator with `lambda=number.children`, until the maximum depth specified by `depth` is reached. The number of children of the root node is set to `children.root`. `"random.arcs"`**:** Random tree where the exact number of nodes in each layer must be speficied by `nodes.per.layer`. The arcs between the layers are random, with the restriction that each node is connected. `"full.tree"`**:** Each node has exactly `number.children` children.
`number.children`	the number of children. For `method="random"` this is the average number of children and for `method="full.tree"`, it is the exact number of children. In the latter case, it can also be a vector that specifies the number of children for each layer.
`children.root`	number of children of the root node. For `method="random"` only.
`depth`	depth of the tree. Note that for `method="random"`, this depth may not be reached.
`nodes.per.layer`	exact number of nodes per layer, that is needed for `method="random.arcs"`
`labels`	one of `"letters"`, `"LETTERS"`, `"numbers"`, `"numbers1"`, `"numbers0"`, `"hex"`, `"bits"`. The label set for `"numbers1"` is `1:9`, and for `"numbers0"` it is `0:9`. `"numbers"` is equal to `"numbers0"`, except that is starts from `1`.
`labels.prefix`	vector of label prefixes, one for each layer
`sep`	seperator character

`colnames`	names of the columns. The first `depth` columns are the index columns (from highest to lowest hierarchical layer), and the last column is stored with random values
`value.generator`	function that determine the random values for the leaf nodes
`value.generator.args`	list of arguments passed to `value.generator`

Examples

```
d <- random.hierarchical.data(200)
treemap(d, index=names(d)[1:(ncol(d)-1)], vSize="x")

d <- random.hierarchical.data(number.children=5)
treemap(d, index=names(d)[1:(ncol(d)-1)], vSize="x")

d <- random.hierarchical.data(method="full.tree", number.children=3, value.generator=runif)
treemap(d, index=names(d)[1:(ncol(d)-1)], vSize="x")
```

`tmPlot` *Create a treemap (deprecated)*

Description

This function is migrated to `treemap`.

Usage

```
tmPlot(...)
```

Arguments

`...`	passed on to `treemap`

`treecolors` *Interactive tool to experiment with Tree Colors*

Description

Tree Colors are color palettes for tree data structures. They are used in `treemap` by default (`type="index"`). With this tool, users can experiment with the parameters (in `treemap` stored in `palette.HCL.options`). Tree Colors can directly be obtained by `treepalette` with `method="HCL"`.

Usage

```
treecolors(height = 700)
```

Arguments

`height`	height of the plotted treemap in pixels. Tip: decrease this number if the treemap doesn't fit conveniently.

Examples

```
## Not run:
treecolors()

## End(Not run)
```

treegraph *Create a tree graph*

Description

This function draws a tree graph. By default, a radial layout is used.

Usage

```
treegraph(dtf, index = names(dtf), directed = FALSE, palette.HCL.options,
  show.labels = FALSE, rootlabel = "", vertex.layout, vertex.layout.params,
  truncate.labels = NULL, vertex.size = 3, vertex.label.dist = 0.3,
  vertex.label.cex = 0.8, vertex.label.family = "sans",
  vertex.label.color = "black", mai = c(0, 0, 0, 0), ...)
```

Arguments

`dtf`	a data.frame or data.table. Required.
`index`	the index variables of dtf (see `treemap`)
`directed`	logical that determines whether the graph is directed (`TRUE`) or undirected (`FALSE`)
`palette.HCL.options`	list of advanced options to obtain Tree Colors from the HCL space (when `palette="HCL"`). This list contains: `hue_start`: number between 0 and 360 that determines the starting hue value (default: 30) `hue_end`: number between `hue_start` and `hue_start + 360` that determines the ending hue value (default: 390) `hue_perm`: boolean that determines whether the colors are permuted such that adjacent levels get more distinguishable colors. If `FALSE`, then the colors are equally distributed from `hue_start` to `hue_end` (default: TRUE) `hue_rev`: boolean that determines whether the colors of even-numbered branched are reversed (to increase discrimination among branches)

- `hue_fraction`: number between 0 and 1 that determines the fraction of the hue circle that is used for recursive color picking: if 1 then the full hue circle is used, which means that the hue of the colors of lower-level nodes are spread maximally. If 0, then the hue of the colors of lower-level nodes are identical of the hue of their parents. (default: .5)
- `chroma`: chroma value of colors of the first-level nodes, that are determined by the first index variable (default: 60)
- `luminance`: luminance value of colors of the first-level nodes, i.e. determined by the first index variable (default: 70)
- `chroma_slope`: slope value for chroma of the non-first-level nodes. The chroma values for the second-level nodes are `chroma+chroma_slope`, for the third-level nodes `chroma+2*chroma_slope`, etc. (default: 5)
- `luminance_slope`: slope value for luminance of the non-first-level nodes (default: -10)

For "depth" and "categorical" types, only the first two items are used. Use `treecolors` to experiment with these parameters.

`show.labels`	show the labels
`rootlabel`	name of the rootlabel
`vertex.layout`	layout algorithm. See `layout` for options. Note: if the `igraph` package is not loaded, use `igraph::` as prefix (see example).
`vertex.layout.params`	list of arguments passed to `vertex.layout`
`truncate.labels`	number of characters at which the levels are truncated. Either a single value for all index variables, or a vector of values for each index variable
`vertex.size`	vertex.size (see `igraph.plotting`)
`vertex.label.dist`	vertex.label.dist (see `igraph.plotting`)
`vertex.label.cex`	vertex.label.cex (see `igraph.plotting`)
`vertex.label.family`	vertex.label.family (see `igraph.plotting`)
`vertex.label.color`	vertex.label.color (see `igraph.plotting`)
`mai`	margins see `par`
`...`	arguments passed to `plot.igraph`

Value

(invisible) igraph object

Examples

```
data(business)
treegraph(business, index=c("NACE1", "NACE2", "NACE3", "NACE4"), show.labels=FALSE)
treegraph(business[business$NACE1=="F - Construction",],
    index=c("NACE2", "NACE3", "NACE4"), show.labels=TRUE, truncate.labels=c(2,4,6))
treegraph(business[business$NACE1=="F - Construction",],
    index=c("NACE2", "NACE3", "NACE4"), show.labels=TRUE, truncate.labels=c(2,4,6),
    vertex.layout=igraph::layout.fruchterman.reingold)
```

`treemap` *Create a treemap*

Description

A treemap is a space-filling visualization of hierarchical structures. This function offers great flexibility to draw treemaps. Required is a data.frame (`dtf`) that contains one or more hierarchical index columns given by `index`, a column that determines the rectangle area sizes (`vSize`), and optionally a column that determines the rectangle colors (`vColor`). The way how rectangles are colored is determined by the argument `type`.

Usage

```
treemap(dtf, index, vSize, vColor = NULL, type = "index", title = NA,
  title.legend = NA, algorithm = "pivotSize", sortID = "-size",
  palette = NA, palette.HCL.options = NULL, range = NA,
  fontsize.title = 14, fontsize.labels = 11, fontsize.legend = 12,
  fontcolor.labels = NULL, fontface.labels = c("bold", rep("plain",
  length(index) - 1)), fontfamily.title = "sans",
  fontfamily.labels = "sans", fontfamily.legend = "sans",
  border.col = "black", border.lwds = c(length(index) + 1, (length(index) -
  1):1), lowerbound.cex.labels = 0.4, inflate.labels = FALSE,
  bg.labels = NULL, force.print.labels = FALSE, overlap.labels = 0.5,
  align.labels = c("center", "center"), xmod.labels = 0, ymod.labels = 0,
  position.legend = NULL, drop.unused.levels = TRUE, aspRatio = NA,
  vp = NULL)
```

Arguments

`dtf`	a data.frame. Required.
`index`	vector of column names in `dtf` that specify the aggregation indices. It could contain only one column name, which results in a treemap without hierarchy. If multiple column names are provided, the first name is the highest aggregation level, the second name the second-highest aggregation level, and so on. Required.
`vSize`	name of the column in `dtf` that specifies the sizes of the rectangles. Required.
`vColor`	name of the column that, in combination with `type`, determines the colors of the rectangles. The variable can be scaled by the addition of "*<scale factor>" or "/<scale factor>". Note: when omitted for `"value"` treemaps, a contant value of 1 is taken.

`type` type of the treemap, which determines how the rectangles are colored:

- `"index"`**:** colors are determined by the `index` variables. Different branches in the hierarchical tree get different colors. For this type, `vColor` is not needed.
- `"value"`**:** the numeric `vColor`-column is directly mapped to a color palette. This palette is diverging, so that values of 0 are assigned to the mid color (white or yellow), and negative and positive values are assigned to color based on two different hues colors (by default reds for negative and greens for positive values). For more freedom, see `"manual"`.
- `"comp"`**:** colors indicate change of the `vSize`-column with respect to the numeric `vColor`-column in percentages. Note: the negative scale may be different from the positive scale in order to compensate for the ratio distribution.
- `"dens"`**:** colors indicate density. This is analogous to a population density map where `vSize`-values are area sizes, `vColor`-values are populations per area, and colors are computed as densities (i.e. population per squared km).
- `"depth"`**:** each aggregation level (defined by `index`) has a distinct color. For this type, `vColor` is not needed.
- `"categorical"`**:** `vColor` is a factor column that determines the color.
- `"color"`**:** `vColor` is a vector of colors in the hexadecimal (#RRGGBB) format
- `"manual"`**:** The numeric `vColor`-column is directly mapped to a color palette. Both palette and range should be provided. The palette is mapped linearly to the range.

`title` title of the treemap.

`title.legend` title of the legend.

`algorithm` name of the used algorithm: `"squarified"` or `"pivotSize"`. The squarified treemap algorithm (Bruls et al., 2000) produces good aspect ratios, but ignores the sorting order of the rectangles (`sortID`). The ordered treemap, pivot-by-size, algorithm (Bederson et al., 2002) takes the sorting order (`sortID`) into account while aspect ratios are still acceptable.

`sortID` name of the variable that determines the order in which the rectangles are placed from top left to bottom right. Only applicable when `algorithm=="pivotSize"`. Also the values "size" and "color" can be used, which refer to `vSize` and `vColor` respectively. To inverse the sorting order, use "-" in the prefix. By default, large rectangles are placed top left.

`palette` one of the following:

- **a color palette:** i.e., a vector of hexadecimal colors (#RRGGBB)
- **a name of a Brewer palette:** See `RColorBrewer::display.brewer.all()` for the options. The palette can be reversed by prefixing with a "-". For treemap types "value" and "comp", a diverging palette should be chosen (default="RdYlGn"), for type "dens" a sequential (default="OrRd"). The default value for "depth" is "Set2".
- **"HCL":** Tree Colors are color schemes derived from the Hue-Chroma-Luminance color space model. This is only applicable for qualitative palettes, which are applied to the treemap types "index", "depth", and "categorical". For "index" and "categorical" this is the default value.

`palette.HCL.options`
list of advanced options to obtain Tree Colors from the HCL space (when `palette="HCL"`). This list contains:

- `hue_start`: number between 0 and 360 that determines the starting hue value (default: 30)
- `hue_end`: number between `hue_start` and `hue_start + 360` that determines the ending hue value (default: 390)
- `hue_perm`: boolean that determines whether the colors are permuted such that adjacent levels get more distinguishable colors. If `FALSE`, then the colors are equally distributed from `hue_start` to `hue_end` (default: TRUE)
- `hue_rev`: boolean that determines whether the colors of even-numbered branched are reversed (to increase discrimination among branches)
- `hue_fraction`: number between 0 and 1 that determines the fraction of the hue circle that is used for recursive color picking: if 1 then the full hue circle is used, which means that the hue of the colors of lower-level nodes are spread maximally. If 0, then the hue of the colors of lower-level nodes are identical of the hue of their parents. (default: .5)
- `chroma`: chroma value of colors of the first-level nodes, that are determined by the first index variable (default: 60)
- `luminance`: luminance value of colors of the first-level nodes, i.e. determined by the first index variable (default: 70)
- `chroma_slope`: slope value for chroma of the non-first-level nodes. The chroma values for the second-level nodes are `chroma+chroma_slope`, for the third-level nodes `chroma+2*chroma_slope`, etc. (default: 5)
- `luminance_slope`: slope value for luminance of the non-first-level nodes (default: -10)

For "depth" and "categorical" types, only the first two items are used. Use `treecolors` to experiment with these parameters.

`range`
range of values that determine the colors. Only applicable for types "value", "comp", and "dens". When omitted, the range of actual values is used. This range is mapped to `palette`.

`fontsize.title`
font size of the title

`fontsize.labels`
font size(s) of the data labels, which is either a single number that specifies the font size for all aggregation levels, or a vector that specifies the font size for each aggregation level. Use value `0` to omit the labels for the corresponding aggregation level.

`fontsize.legend`
font size for the legend

`fontcolor.labels`
Specifies the label colors. Either a single color value, or a vector of color values one for each aggregation level. By default, white and black colors are used, depending on the background (`bg.labels`).

`fontface.labels`
either a single value, or a vector of values one for each aggregation level. Values can be integers If an integer, following the R base graphics standard: 1 = plain,

2 = bold, 3 = italic, 4 = bold italic, or characters: `"plain"`, `"bold"`, `"italic"`, `"oblique"`, and `"bold.italic"`.

`fontfamily.title`
: font family of the title. Standard values are "serif", "sans", "mono", "symbol". Mapping is device dependent.

`fontfamily.labels`
: font family of the labels in each rectangle. Standard values are "serif", "sans", "mono", "symbol". Mapping is device dependent.

`fontfamily.legend`
: font family of the legend. Standard values are "serif", "sans", "mono", "symbol". Mapping is device dependent.

`border.col` color of borders drawn around each rectangle. Either one color for all rectangles or a vector of colors, or one for each aggregation level

`border.lwds` thicknesses of border lines. Either one number specifies the line thicknesses (widths) for all rectangles or a vector of line thicknesses for each aggregation level.

`lowerbound.cex.labels`
: multiplier between 0 and 1 that sets the lowerbound for the data label font sizes: 0 means draw all data labels, and 1 means only draw data labels if they fit (given `fontsize.labels`).

`inflate.labels` logical that determines whether data labels are inflated inside the rectangles. If `TRUE`, `fontsize.labels` does not determine the fontsize anymore, but it still determines the minimum fontsize in combination with `lowerbound.cex.labels`.

`bg.labels` background color of high aggregation labels. Either a color, or a number between 0 and 255 that determines the transparency of the labels. In the latter case, the color itself is determined by the color of the underlying rectangle. For "value" and "categorical" treemaps, the default is (slightly) transparent grey (`"#CCCCCCDC"`), and for the other types slightly transparent: `220`.

`force.print.labels`
: logical that determines whether data labels are being forced to be printed if they don't fit.

`overlap.labels` number between 0 and 1 that determines the tolerance of the overlap between labels. 0 means that labels of lower levels are not printed if higher level labels overlap, 1 means that labels are always printed. In-between values, for instance the default value .5, means that lower level labels are printed if other labels do not overlap with more than .5 times their area size.

`align.labels` object that specifies the alignment of the labels. Either a character vector of two values specifying the horizontal alignment (`"left"`, `"center"`, or `"right"`) and the vertical alignment (`"top"`, `"center"`, or `"bottom"`), or a list of sush character vectors, one for each aggregation level.

`xmod.labels` the horizontal position modification of the labels in inches. Either a single value, or a vector that specifies the modification for each aggregation level.

`ymod.labels` the vertical position modification of the labels in inches. Either a single value, or a vector that specifies the modification for each aggregation level.

`position.legend`	position of the legend: `"bottom"`, `"right"`, or `"none"`. For "categorical" and "index" treemaps, `"right"` is the default value, for "index" treemap, `"none"`, and for the other types, `"bottom"`.
`drop.unused.levels`	logical that determines whether unused levels (if any) are shown in the legend. Applicable for "categorical" treemap type.
`aspRatio`	preferred aspect ratio of the main rectangle, defined by width/height. When set to `NA`, the available window size is used.
`vp`	`viewport` to draw in. By default it is not specified, which means that a new plot is created. Useful when drawing small multiples, or when placing a treemap in a custom grid based plot.

Value

A list is silently returned:

`tm`	a `data.frame` containing information about the rectangles: indices, sizes, original color values, derived color values, depth level, position (x0, y0, w, h), and color.
`type`	argument type
`vSize`	argument vSize
`vColor`	argument vColor
`algorithm`	argument algorithm
`vpCoorX`	x-coordinates of the treemap within the whole plot
`vpCoorY`	y-coordinates of the treemap within the whole plot
`aspRatio`	aspect ratio of the treemap
`range`	range of the color values scale

References

Bederson, B., Shneiderman, B., Wattenberg, M. (2002) Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies. ACM Transactions on Graphics, 21(4): 833-854.

Bruls, D.M., C. Huizing, J.J. van Wijk. Squarified Treemaps. In: W. de Leeuw, R. van Liere (eds.), Data Visualization 2000, Proceedings of the joint Eurographics and IEEE TCVG Symposium on Visualization, 2000, Springer, Vienna, p. 33-42.

Examples

```
#########################################
### quick example with Gross National Income data
#########################################
data(GNI2010)
treemap(GNI2010,
       index=c("continent", "iso3"),
       vSize="population",
       vColor="GNI",
       type="value")

#########################################
### extended examples with fictive business statistics data
#########################################
data(business)

#########################################
### treemap types
#########################################

# index treemap: colors are determined by the index argument
## Not run:
# large example which takes some time...
treemap(business,
        index=c("NACE1", "NACE2", "NACE3"),
        vSize="turnover",
        type="index")

## End(Not run)
treemap(business[business$NACE1=="C - Manufacturing",],
        index=c("NACE2", "NACE3"),
        vSize=c("employees"),
        type="index")

# value treemap: colors are derived from a numeric variable given by vColor
# (when omited, all values are set to 1 as in the following example)
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        title.legend="number of NACE4 categories",
        type="value")

# comparisson treemaps: colors indicate change of vSize with respect to vColor
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        vColor="employees.prev",
        type="comp")

# density treemaps: colors indicate density (like a population density map)
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="turnover",
        vColor="employees/1000",
        type="dens")

## Not run:
# depth treemap: show depth
treemap(business,
        index=c("NACE1", "NACE2", "NACE3"),
        vSize="turnover",
        type="depth")
```

```
## End(Not run)

# categorical treemap: colors are determined by a categorical variable
business <- transform(business, data.available = factor(!is.na(turnover)), x = 1)
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="x",
        vColor="data.available",
        type="categorical")

## Not run:
# color treemap
business$color <- rainbow(nlevels(business$NACE2))[business$NACE2]
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="x",
        vColor="color",
        type="color")

# manual
business$color <- rainbow(nlevels(business$NACE2))[business$NACE2]
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="turnover",
        vColor="employees",
        type="manual",
        palette=terrain.colors(10),
        range=c(-50000, 500000))

## End(Not run)

#########################################
### graphical options: control fontsizes
#########################################

## Not run:
# draw labels of first index at fontsize 12 at the center,
# and labels of second index at fontsize 8 top left
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        fontsize.labels=c(12, 8),
        align.labels=list(c("center", "center"), c("left", "top")),
        lowerbound.cex.labels=1)


# draw all labels at fontsize 12 (only if they fit)
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        fontsize.labels=12,
        lowerbound.cex.labels=1)
```

```
# draw all labels at fontsize 12, and if they don't fit, reduce to a minimum of .6*12
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        fontsize.labels=12,
        lowerbound.cex.labels=.6)

# draw all labels at maximal fontsize
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        lowerbound.cex.labels=0,
        inflate.labels = TRUE)

# draw all labels at fixed fontsize, even if they don't fit
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        fontsize.labels=10,
        lowerbound.cex.labels=1,
        force.print.labels=TRUE)

#########################################
### graphical options: color palettes
#########################################

## for comp and value typed treemaps all diverging brewer palettes can be chosen
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        vColor="employees.prev",
        type="comp",
        palette="RdBu")

## draw warm-colored index treemap
palette.HCL.options <- list(hue_start=270, hue_end=360+150)
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        type="index",
        palette.HCL.options=palette.HCL.options)

# terrain colors
business$employees.growth <- business$employees - business$employees.prev
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        vColor="employees.growth",
        type="value",
        palette=terrain.colors(10))

# Brewer's Red-White-Grey palette reversed with predefined range
```

```
treemap(business,
        index=c("NACE1", "NACE2"),
        vSize="employees",
        vColor="employees.growth",
        type="value",
        palette="-RdGy",
        range=c(-30000,30000))

## End(Not run)
```

treepalette *Obtain hierarchical color palettes (Tree Colors)*

Description

Obtain hierarchical color palettes, either the so-called Tree Colors from the HCL color space model, or by using an existing color palette. The former method, which is recommended, is used by default in `treemap` (type `"index"`) and `treegraph`. Use `treecolors` to experiment with this method.

Usage

```
treepalette(dtf, index = names(dtf), method = "HCL", palette = NULL,
  palette.HCL.options, return.parameters = TRUE, prepare.dat = TRUE)
```

Arguments

`dtf`	a data.frame or data.table. Required.
`index`	the index variables of dtf
`method`	used method: either `"HCL"` (recommended), which is based on the HCL color space model, or `"HSV"`, which uses the argument `palette`.
`palette`	color palette, which is only used for the HSV method
`palette.HCL.options`	list of options to obtain Tree Colors from the HCL space (when `palette="HCL"`). This list contains:

- `hue_start`: number between 0 and 360 that determines the starting hue value (default: 30)
- `hue_end`: number between `hue_start` and `hue_start + 360` that determines the ending hue value (default: 390)
- `hue_perm`: boolean that determines whether the colors are permuted such that adjacent levels get more distinguishable colors. If `FALSE`, then the colors are equally distributed from `hue_start` to `hue_end` (default: TRUE)
- `hue_rev`: boolean that determines whether the colors of even-numbered branched are reversed (to increase discrimination among branches)

- `hue_fraction`: number between 0 and 1 that determines the fraction of the hue circle that is used for recursive color picking: if 1 then the full hue circle is used, which means that the hue of the colors of lower-level nodes are spread maximally. If 0, then the hue of the colors of lower-level nodes are identical of the hue of their parents. (default: .5)
- `chroma`: chroma value of colors of the first-level nodes, that are determined by the first index variable (default: 60)
- `luminance`: luminance value of colors of the first-level nodes, i.e. determined by the first index variable (default: 70)
- `chroma_slope`: slope value for chroma of the non-first-level nodes. The chroma values for the second-level nodes are `chroma+chroma_slope`, for the third-level nodes `chroma+2*chroma_slope`, etc. (default: 5)
- `luminance_slope`: slope value for luminance of the non-first-level nodes (default: -10)

For "depth" and "categorical" types, only the first two items are used. Use `treecolors` to experiment with these parameters.

`return.parameters`
should a data.frame with color values and parameter options be returned (`TRUE`), or just the vector of color values (`FALSE`)?

`prepare.dat` data is by default preprocessed, except for interal use

Value

Either a vector of colors, or a data.frame is return (see `return.parameters`).

Index