R documentation

of ‘iconplot.Rd’

June 12, 2018

---

iconplot *Icon Plots for Visualization of Contingency Tables*

Description

An icon plot is a graphical representation of a contingency table. `iconplot(` computes a icon plot of a data matrix (matrix or data frame) or of an object of class `table`. Based on argument `grp.xy` the data set is split into groups. Similarly the graphics region is divided into panels. Then the elements of the groups are visualized within the associated panels.

Usage

```
 iconplot(data
,  grp.xy = 2 ~ 1
,  grp.color = NULL
,  grp.icon = NULL
,  colors
,  icons
,  vars.to.factors
,  panel.reverse.y = FALSE
,  panel.space.factor = 0.05
,  panel.prop.to.size = c(FALSE, FALSE)
,  panel.margin = 0.03
,  panel.frame = TRUE
,  panel.adjust = c(0.5, 0.5)
,  icon.horizontal = TRUE
,  icon.stack.type = c("lt", "lb", "rt", "rb")[1]
,  icon.cex = NA
,  icon.aspect = 1
,  icon.stack.len = NA
,  icon.space.factor = 0.3
,  icon.grey.levels = 2
,  icon.frame = TRUE
,  icon.draft = TRUE
,  lab.side = c("bl", "br", "tl", "tr")[1]
,  lab.parallel = c(TRUE, TRUE)
```

```
    , lab.cex = 1
    , lab.boxes = 2
    , lab.color = c("#CCCCCC", "white")
    , lab.type = c("expanded", "compact")[2]
    , lab.n.max = c(20, 30)
    , lab.legend = c("cols","rows","skewed","horizontal","vertical")[2]
    , packer = c("icons", "numbers", "panel.legend", "stars")[1]
    , panel.text =  NULL
    , mar = rep(1, 4)
    , main
    , verbose = !TRUE
    , ...)
```

Arguments

`data`	a data matrix, a data frame or an object of class `table`. Note: If the column or dimension names are elements of the following set of reserved names: `.sign, .fraction, .color, .icon, .job.no, .x0, .x1, .y0, .y1` strange results may occur; therefore, avoid these variable names.
`grp.xy`	a formula specifying how the data set is divided into groups and defines in which panel an element of the data is represented. The formula `y1 ~ x1` means that the data set is split according to the levels of the variables `y1` and `x1` into groups. If `n.level.y1` and `n.level.x1` are the numbers of levels of the two variables the plotting region is divided like a chessboard into `n.level.y1` rows and `n.level.x1` columns. In this way we get `n.level.y1 * n.level.x1` fields that are called panels. In each of the panels the elements of the associated group are represented by pictogram elements or icons. If the argument `grp.xy` hasn't been set by default the first variable of the data set defines the grouping of the data along the x-axis and the second one the grouping along the y-axis. Instead of variable names the indices of the variables can be used. The definition of recursive groupings is allowed and is expressed by operator "+": `y ~ 3 + 4` means that the horizontal range of the graphical region is split twice: At first the segmentation of the region is computed according to variable 3, in the second step the subranges of step 1 will be divided as a function of the levels of variable 4. A `"0"` on one side of the ~ character indicates that no splitting of the correspondent region is desired.
`grp.color`	defines how the data are grouped with respect to coloring. The name of the variable used for coloring the icons (or pictogram elements) has to be assigned to `grp.color`. `colors[i]` defines the color of the icon belonging to the level `i` of the variable fixed by `grp.color`.
`grp.icon`	defines how the data are grouped with respect the associated icon. The name of the variable used for selecting symbols or icons has to be fixed by argument `icons`. The symbol (icon) representing an observation depends on its level of the variable specified by `grp.icon`. If additional variables – separated by a + character – are found the values of these variables will be used in the call of a icon generating function. For details see paragraph 'Details'.
`colors`	set of colors used for pictogram elements.

`icons` defines the icons or the set of icons used by `iconplot`. If `icons` is a vector `icons[i]` is used to represent the observations whose level of the variable fixed by `grp.icon` is `i`. There are some alternatives to define the icons or pictogram elements:
* default: the default symbol is a rectangle.
* vector of numbers: numbers specify plotting characters of the graphics system similar to `points(..., pch = 13)`.
* list of raster images: the images are used as icons.
* character vector: `icons[i]` with an extension indicating a `pnm`, `ppm`, `jpg` or `png` image file: `iconplot` tries to use the image of the file as icon.
Otherwise `icons[i]` is interpreted as the name of an internal icon generating function.
* list of functions: `icons[i]` is interpreted as an icon generating function and is called to compute the icon for level `i`.
* list of icon descriptions. For details see paragraph "Details".
Note: If an image file is defined by an internet link it is temporarily downloaded using `tempfile()` and `download.file()`.
Note: Mixtures of these alternative definition don't work usually. Therefore, it is recommended to use one type of definition only.

`vars.to.factors`
controls the transformation of variables to factors. If missing it is set to `TRUE` for each of the relevant variables. If `vars.to.factors` is a vector and if its elements don't have names the variables `1:length(vars.to.factors)` are transformed. If `vars.to.factors` consists of named elements the names indicate the variables to be transformed.
If `vars.to.factors[i] == FALSE` variable `i` will not be transformed.
If `vars.to.factors[i] == 1` variable `i` is transformed to a factor.
If `vars.to.factors[i] < 1` the range of variable `i` is cut into groups in a way that we approximately get `round(1/vars.to.factors[i])` groups and each of the groups approximately contain `100 * vars.to.factors[i]` percent of the data. If `vars.to.factors[i] > 1` the range of variable `i` will be cut into `floor(vars.to.factors[i])` subranges of equal size and you get a factor variable with `floor(vars.to.factors[i])` levels.

`panel.reverse.y`
logical, if `TRUE` the vertical axis is reversed.

`panel.space.factor`
relative space inserted between the panels.

`panel.prop.to.size`
a vector containing two elements which controls the sizes of the panels. The first entry determines the widths of the panels and the second one their heights. `panel.prop.to.size[1] == 0` means all panels are of the same width.
If `panel.prop.to.size[2] == 0` the panels are of the same height. A value of 1 indicates that sizes should be computed proportional to the frequencies of the levels. Otherwise the sizes of the panels are fixed proportional to:
`frequencies^panel.prop.to.size`.

`panel.margin` controls the margins around the regions of the panels. If this argument is a vector of length four the elements refer to the four sides of the plot: bottom, left, top, and right. If this argument is set to `c(0, 0.1, 0.5, 0)` we get no additional margin below the panels and on the right-hand side. However, there will be an upper margin of size `100 * 0.5` percent of the height of the area containing the panels and a margin of size code100 * 0.1 percent of the width on the left-hand is provided.

`panel.frame`	logical, if `TRUE` a border line is drawn around each of the panels.
`panel.adjust`	controls the adjustment of the panels within their regions. This argument modifies the internal coordinates and do usually not change the appearance of the plot.
`panel.text`	vector of strings. The text `panel.text[i]` is written into `panel[i]`. The texts can be used for short describitions of the contents of the panels. To get an idea of the numbering of the panels you can set `panel.text = 1:20`.
`icon.horizontal`	logical, if `TRUE` the stacks of icons or pictogram elements are plotted horizontally. This argument effects the way how icons are put into the panels.
`icon.stack.type`	defines the method of plotting the stacks of icons: "r" or "l" are shortcuts for "right" or "left". "t", "b" correspond to "top" and "bottom", respectively. Note: Fractional parts of frequencies are represented by smaller icons. Adding the letter `"s"` (as a abbreviation for "shrinkage") to the argument `icon.stack.type` both dimensions of the icons are reduced. If `icon.stack.type` is a vector its elements define the different types of stacking for the panels.
`icon.cex`	size of icons; this argument is similar to `cex` of `points()`.
`icon.aspect`	aspect ratio of icons: width / height.
`icon.stack.len`	maximal number of icons gathered to build a stack. If this length is decreased the number of stacks (rows or columns of icons) will increase.
`icon.space.factor`	relative space between two icons.
`icon.grey.levels`	controls the coloring of icons of class raster or images. If `icon.grey.levels` is of length 1 and if it is an integer the argument defines the number of color- or grey-levels of the icons. If `icon.grey.levels` consists of two or more values between 0 and 1 they are interpreted as grey limits. Level 0 represents `"black"`, and 1 indicates the maximal brightness. The default setting generates one color only (besides black and white).
`icon.frame`	logical, if `TRUE` a border is drawn around each of the pictograms.
`icon.draft`	logical, if `TRUE` raster images are generated by calling `rasterImage()` with the setting `interpolate = TRUE`.
`mar`	this argument is delivered to the graphics device via `par()` and manipulates the margins of the plot.
`main`	defines the title of the plot.
`lab.side`	defines one or two sides that are used for margin information: "l" indicates the "left" side, "b" identifies the "bottom" as well as "r" the "right" and "t" the "top" side.
`lab.parallel`	logical, if `FALSE` margin labels are perpendicularly constructed to the axes. If `lab.parallel` is a vector the first element is used for controlling the labels of the bottom or top side and the second one specifies the orientation of the y-labels. If one elements is set to 0.5 the labels of the last `xy` grouping variable are printed perpendicularly only.
`lab.legend`	a character string indicating the kind of legend out of the vector `c("cols", "rows", "skewed", "horizontal", "vertical")`. Assigning a number of the set `1:5` to the argument is interpreted as an index of the set of the five types of legends.

	`"cols"`: vertical legends, side by side at the bottom side of the plot. `"rows"`: horizontal legends, line by line at the bottom side of the plot. `"skewed"`: horizontal legends, line by line and the level names are rotated. `"horizontal"`: horizontal legends, side by side at the bottom side of the plot. `"vertical"`: vertical legends, line by line at the right side of the plot.
`lab.cex`	sets the size of the characters of the labels and the legends.
`lab.boxes`	defines the types of boxes around the margin labels: `lab.boxes == 0`: no boxes are drawn. `lab.boxes >= 1`: small boxes around the labels are drawn. `lab.boxes >= 2`: big boxes around the labels are drawn. `lab.boxes %% 1`: defines the size of the separation line between the names of the variables and the names of the levels.
`lab.color`	The first element defines the color of the box containing the names of variables or levels in the margins. The second element sets the color of the separation line between the variable names and the level names within the margins.
`lab.type`	defines the design style of margin labeling: "c" or "e" are shortcuts for "compact" or "expanded".
`lab.n.max`	is an integer vector consisting of three elements. The first element sets the number of characters during printing the labels of the levels. The second element defines the maximal number of level names to be plotted in the margins. `lab.n.max[3]` limits the number of labels of the color- or icon-legend.
`packer`	defines the packer(s) which are used to fill the panels. If "icons" is an element of `packer` the observations will be represented by icons, pictogram elements or symbols. If the character string "numbers" is found in `packer` in each of the panels the numbers of its observations will be printed into the areas of the panels. The packer "panel.legend" plots the level combinations into the panels. This may be a useful feature as long as the number of the panels is small. Otherwise the texts of level combinations will overlap each other. The argument `cex` controls the size of the text strings.
`verbose`	logical, if `TRUE` internal information is printed during the computation.
`...`	arguments that will be passed to the graphics functions and suitable ones to the icon generating functions.

Details

`iconplot()` constructs an icon plot of a data matrix and a contingency table. In an icon plot each observation of the data set is represented by a small symbol or an image called pictogram or icon. A cell of a contingency table is visualized by a set of icons. The icons of a cell are plotted within a rectangular region which we call panel and an icon plot consists of a lot of panels containing the icons of the cells.

Similar to the layout of contingency tables the set of panels are arranged in a grid-like manner. Considering a high dimensional contingency table you can concentrate on some of the variables and can construct suitable margin tables. Equivalently you can build a lot of icon plots to emphasize your viewpoint. By varying the actual arguments of `iconplot()` a huge set of appearances of plots results and the nicest one for your purpose can be choosen. `table`, `matrix` or `data frames` can be used as data input of `iconplot()`. Tables are allowed to have fractional or negative entries; these cases may occur by computing the difference of two tables or by changing the units of counting. Internally a table will be expanded to a data matrix. Fractional numbers are coded in a data matrix by the additional column or variable `.fraction`, to handle negative numbers the new variable `.sign` is added.

The argument `grp.xy` of `iconplot` defines the variables used for grouping and splitting the data dependent on the levels of the specified variables. Each group is represented within a panel as stated above. Let's have a look at an example: Consider you have a 2x3 contigency table and would like to represent it by an icon plot. So a plot to be constructed should have 2x3 panels and the number of icons of the panels should be given by the cell entries. To get an icon plot with desired panel structure you define the xy-grouping by `grp.xy = 1 ~ 2`. This means: The data set has to be split according to the two levels of the first variable and the y-range of the plot has to be divided in two rows of panels. On the other side the second variable defines the grouping concerning the the x-range and three columns of panels appear. As a result a icon plot is generated that consists of six panels arranged in two rows and three columns. The panels of a fixed level of the first variable are placed side by side, whereas the panels of a fixed level of the second variable are stacked one upon the other and a layout known from a chessboard results. As an example try: `x <- as.table(matrix(1:6, 2, 3)); iconplot(x, grp.xy = 1 ~ 2)` `grp.xy = 0 ~ 1 + 2` leads a double grouping on the x-axis and no vertical grouping. `grp.xy = 1 + 2 ~ 3 + 4` presums four or more variables and splits the graphics region twice along the x- and twice along the y-direction.

Within a panel the entry of one cell is represented. Several arguments control the way how the icons are placed in a panel. The absolute size of the icons can be defined by `icon.cex`. `icon.aspect` fixes the aspect ratio of the pictograms (width / height). The elements in a panel are assembled into stacks; the maximal length of these stacks can be set by `icon.stack.len`; horizontal stacks are plotted if `icon.horizontal` is `TRUE`. Framing icons and spacing between them is controlled by the arguments `icon.frame` and `icon.space.factor`.

The icons or pictogram elements may be colored dependent on the levels of a variable. The variable has to be established by argument `grp.color`. A set of colors can be defined by argument `colors`. Accordingly, the symbols or images are determined by `grp.icon` and `icons`.

An icon or pictogram element can be generated by an icon generating function. The result of an icon generating function describes a standardized icon by a set of segments, polygons, splines and texts which are combined in a list. `segments`: `segments` are defined by a matrix or a data frame of 5 or 6 columns: Columns 1 to 4 keep the coordinates of the starting and ending points of the segments: `x.0, y.0, x.1, y.1`.
The 5th column contains the widths of the segments. The coordinates and the widths have to be choosen in a way that the icon fits pretty well into a plotting field of size 100mm x 100mm assuming the coordinates of the world window defined by: `usr = c(0, 100, 0, 100)`.
If the 6th column is available it defines the coloring of the segments. A value of "0" codes the color "white" and the other values are interpreted as usually: "1" means "black" and any other color is processed as `col` in `points`, for example. An `NA` value instead of a color instructs `iconplot()` to color the segment dependent on the associated level of the variable fixed by `grp.color`. Segment objects must have the class attribute `"segments"`.

`polygon`: Polygons are defined by a matrix or data frame of 2 or 3 columns. Colums 1 and 2 store the coordinates of the vertices of the polygon. A third column fixes the coloring of the polygon. The class attribute of this kind of element has to be set to `"polygon"`.

`spline`: Splines are defined by a matrix or data frame of 3 or 4 columns. Colums 1 and 2 store the coordinates of the points which form the basis of the spline. The third column keeps the line width of the curve. The optional fourth column shows how to color the spline. Splines are identified by class attribute `"spline"`.

`text`: Text elements of a generated icon are defined by a data frame of 3, 4 or 5 columns. The first two columns of the object store the coordinates of the positions of the text(s). The third element stores the text information and the fourth is used to set the size of the characters. The fifth fixes the

coloring of the text. The class attribute of a text element is `"text"`. There are some internal icon generating functions. Here is a list of them:
`BI`, `TL`, `cross.simple`, `cross`, `circle.simple`, `circle`, `car.simple`, `car`, `nabla`, `walkman`, `smiley.blueeye`, `smiley.normal`, `smiley`, `smiley.sad`, `mazz.man`, `bike`, `bike2`, `heart`, `bend.sign`, `fir.tree`, `comet`, `coor.system`.

Value

`iconplot()` returns a list consisting of three elements. The first element is the matrix `jobs` whose lines show some attributes of the panels. In a row of this matrix you find the number of the panel `.job.no` and the location of the panel (in user coordinates: `xmins, xmaxs, ymins, ymaxs`). The second element is a copy of the modified data matrix which is used for the construction of the icon plot: Besides the data delivered by the user there are columns showing the colors, icons and coordinates of the pictogram elements. The third element contains the output of `par()` and describes the graphics device during the computation; this list differs from the parameter settings after leaving `iconplot()` because the state of graphics parameter before calling `iconplot()` is restored. These three lists may be helpful if you want to add further graphical elements to the plot.

Note

Remark: the version of `iconplot` of this package is an experimental version. Therefore, in the future some of the features may be changed and it is not sure that the function works as described on all types of graphics devices. In case of errors feel free to write a mail. Additional information and examples are found on the web page
http://www.wiwi.uni-bielefeld.de/lehrbereiche/statoekoinf/comet/wolf/wolf_aplpack.

Author(s)

Hans Peter Wolf

See Also

`mosaicplot`, `pairs`, `puticon`

Examples

```
# HairEyeColor data, grouping by color
iconplot(HairEyeColor,
         grp.color        = 1,
         grp.xy           = NULL,
         colors           = c("black", "brown", "red", "gold"),
         icon.space.factor = 0,
         icon.aspect      = 2,
         main = "grouping by color")
# HairEyeColor data, grouping by color and symbols
iconplot(HairEyeColor,
         grp.icon         = "Sex",
         grp.color        = "Hair",
         grp.xy           = NULL,
         colors           = c("black", "brown", "red", "gold"),
         icons            = 18:17,
         icon.frame       = FALSE,
         lab.cex          = 0.8,
         icon.space.factor = 0,
         lab.parallel     = !FALSE,
         main = "grouping by color and icons")
# HairEyeColor data, grouping by x and color
iconplot(HairEyeColor,
         grp.xy            = "0 ~ 1",
         grp.color         = 2,
         colors            = c("black", "brown", "red", "gold"),
         icon.stack.type   = "tr",
         icon.space.factor = c(0, 0.4),
         lab.cex           =0.7,
         main = "grouping by x and by colors")
# 2-dim, 1 split in y, 1 split in x, grouping by color
iconplot(HairEyeColor,
         grp.xy          = "1 ~ 3",
         grp.color       = 2,
         colors          = c("brown", "blue", "brown3", "green"),
         panel.frame     = FALSE,
         icon.stack.type = "bl",
         lab.cex         = 0.7,
         main = "grouping by x and y and by colors")
# 3-dim, 2 splits in x, 1 split in x, margin labs on the right
iconplot(HairEyeColor,
         grp.xy             = "2 ~ 1 + 3 ",
         grp.color          = 2,
         panel.space.factor = c(0, .1),
         panel.margin       = c(.05,.03,.03,.01),
         icon.stack.type    = "lb",
         icon.stack.len     = 7,
         icon.frame         = FALSE,
         icon.space.factor  = .0,
         lab.parallel       = c(TRUE, FALSE),
         lab.color          = c("lightblue","green"),
         lab.side           = "br",
         lab.boxes          = 0.2,
         lab.type           = "compact",
         lab.cex            = 0.8,
         main = "grouping: 2~1+3 and by color, margin labs variations")
# 3-dim, 3 splits in y, icon.aspect = NA
iconplot(HairEyeColor,
         grp.xy             = "3 + 2 ~ 1",
         grp.color          = 3,
         panel.margin       = 0,
         panel.space.factor = 0.1,
         icon.stack.type    = "lb",
         icon.horizontal    = TRUE,
         icon.stack.len     = 5,
         icon.space.factor  = c(.1, .3),
         icon.aspect        = NA,
         icon.frame         = FALSE,
         lab.boxes          = 0.3,
         lab.color          = "grey",
         lab.side           = "tl",
         lab.parallel       = TRUE,
         lab.cex            = 0.7,
         lab.type           = "compact",
         main = "grouping: 3 + 2 ~ 1 and by color")
# 3-dim, plotting characters as icons
data <- as.table(array(0:23, 2:4))
```

```
iconplot(data,
         grp.xy          = 1 + 2 ~ 3,
         grp.color       = 3,
         grp.icon        = 2,
         icon.aspect     = 2,
         icon.horizontal = TRUE,
         icons           = 15:18,
         icon.stack.type = c("lb", "lt", "rb","rt")[3],
         icon.frame      = FALSE,
         lab.cex         = .6,
         lab.type        = "compact",
         main = "1+2 ~ 3")
# 3-dim contingency table: panels of different sizes, 1 split in y, 2 in x
# packer numbers
  ## because of computation time
iconplot(Titanic,
         grp.xy             = 1~3+2,
         grp.color          = 1,
         packer             = c("icons", "numbers"),
         panel.prop.to.size = 0.5,
         panel.frame        = !TRUE,
         panel.margin       = .01,
         icon.aspect        = 0.15,
         icon.stack.type    = "lt",
         icon.space.factor  = 0.0,
         icon.frame         = FALSE,
         lab.side           = c("bl","br","tl","tr")[1],
         lab.type           = "compact",
         lab.cex            = 0.8,
         lab.boxes          = 1.1,
         lab.color          = "lightgreen",
         lab.parallel       = TRUE,
         main = "different sizes of panels")

# 3-dim contingency table: panels of different sizes, 3 splits in y
  ## because of computation time
iconplot(Titanic,
         grp.xy             = "4 + 3 + 1 ~ 0" ,
         grp.color          = 4,
         colors             = c("green", "red"),
         packer             = c("icons", "numbers"),
         panel.frame        = FALSE,
         panel.margin       = .01,
         panel.prop.to.size = .3,
         panel.space.factor = 0.05,
         panel.reverse.y    = TRUE,
         icon.space.factor  = 0.5,
         lab.side           = "l",
         lab.type           = "compact",
         lab.parallel       = c(FALSE, TRUE),
         lab.cex            = 0.7,
         main = "Titanic data, different sizes of panels")

#  3-dim contingency table: panels of different sizes
  ## because of computation time
iconplot(Titanic,
         grp.xy             = "0 ~ 4 + 3 + 1 " ,
         grp.color          = 4,
         colors             = c("green", "red"),
         panel.frame        = FALSE,
         panel.margin       = .01,
         panel.prop.to.size = .2,
         panel.space.factor = 0.05,
         panel.reverse.y    = TRUE,
         icon.space.factor  = 0.5,
         lab.side           = "b",
         lab.type           = "compact",
         lab.boxes          = 0.2,
         lab.parallel       = c(FALSE, TRUE),
         lab.cex            = 0.6,
         lab.color          = c("lightblue"),
         main = "Titanic data, different widths of panels")

# 3-dim contingency table: panels of different sizes, 3 splits in x
  ## because of computation time
iconplot(Titanic,
         grp.xy             = 3 + 2 ~ 1,
         grp.color          = 2,
         panel.prop.to.size = 0.66,
         icon.space.factor  = 0.4,
         panel.space.factor = 0.1,
         lab.type           = "c",
         lab.cex            = 0.7,
         lab.boxes          = 1.2,
         lab.color          = c("lightblue"),
         main = "Titanic: panel.prop.to.size = 0.66")

# comparing iconplot and mosaic plot
# par(mfrow = 2:1)
iconplot(HairEyeColor,
         grp.xy             = 2 ~ 1 + 3 ,
         lab.parallel       = c(TRUE, TRUE),
         colors             = "red",
         panel.reverse.y    = TRUE,
         panel.prop.to.size = TRUE,
         icon.space.factor  = 0.5,
         icon.aspect        = 2,
         lab.cex            = .6,
         lab.boxes          = 1,
         lab.color          = "grey",
         # lab.side           = "lt",
         panel.margin       = c(0.00,.035,0.0,.050),
         main = 'HairEyeColor: grp.xy = 2 ~ 1 + 3')
mosaicplot(HairEyeColor)
# par(mfrow = c(1,1))
# relative frequences
data <- as.table(Titanic / max(Titanic))
iconplot(data,
         grp.xy             = 1 ~ 2 + 3,
         grp.color          = 4,
         panel.frame        = FALSE,
         panel.space.factor = 0.05,
         icon.horizontal    = !TRUE,
         icon.space.factor  = 0.103,
         icon.stack.type    = "b",
         icon.aspect        = 0.5,
         main = "Titanic: relative frequencies", colors = c("black", "green"))
# negative and fractional cell entries
  ## because of computation time
data <- HairEyeColor; Exp <- margin.table(data, 1)
for( d in 2:length(dim(data)) ){
  Exp <- outer( Exp, margin.table(data, d) ) / sum(data)
}
Diff <- Exp - data
cat("observed:\n"); print(data)
cat("expected:\n"); print(round(Exp, 3))
cat("deviation: expected - observed:\n"); print(round(Diff,3))
iconplot(Diff,
         grp.xy          = 1 + .sign ~ 2 + 3,
         grp.color       = ".sign",
         colors          = c( "red", "green"),
         panel.reverse.y = TRUE,
         panel.frame     = FALSE,
         icon.stack.type = c("t","b"),
         lab.boxes       = 1.2,
         lab.color       = "lightgreen",
         main = "deviations from expectation: HairEyeColor")

# relative differences of expectations, split according sign
data <- margin.table(Titanic, c(2,1,4)); pT <- prop.table(data)
eT <- outer(outer(margin.table(pT,1), margin.table(pT,2)), margin.table(pT,3))
data <- as.table(pT - eT); data <- data / max(data)
iconplot(data,
         grp.xy = Survived + Sex + .sign ~ Class,
         grp.color          = ".sign",
         panel.frame        = FALSE,
         panel.reverse.y    = TRUE,
         panel.space.factor = 0.05,
         icon.horizontal    = !TRUE,
         icon.stack.type    = rep(c("t","b"), 2),
         icon.aspect        = 2,
         icon.space.factor  = 0.1,
         lab.boxes          = 1.2,
         lab.color          = "lightgrey",
         main = "Titanic: difference to expectation")
# using a foto as icon, rentals of flats in Goettingen 2015/12
rentels <-
 structure(list(Rooms = c(2, 3, 2, 2, 3, 2, 2, 3, 2, NA, 2, 2,
 3, 4, 4, NA, 3, 2, 3, 2, 4, 2, 1, 2), qm = c(43.13, 86, 48, 66.62,
 76, 49, 59, 97, 45, 87, 46.39, 71, 65, 100, 75, 178, 94.07, 56,
 97, 70, 132, 43, 24, 48), Eur = c(365, 480, 480, 660, 500, 410,
 440, 1200, 450, 696, 420, 710, 747.5, 1300, 450, 990, 900, 520,
 1020, 1005, 924, 610, 375, 420)), class = "data.frame",
 row.names = c(NA, 24L))
fname <- system.file("src", "tm1.jpg", package="aplpack") # fname <- "tm1.jpg"
print(fname)
iconplot(rentels,
         grp.xy              = Eur ~ qm,
         vars.to.factors     = c(1, .5, .3),
         panel.frame         = FALSE,
         panel.space.factor  = 0.2,
         panel.prop.to.size  = 0.7,
         icons               = fname,
         icon.frame          = FALSE,
         icon.space.factor   = 0.05,
         lab.parallel        = c(TRUE, TRUE),
         lab.legend          = "cols",
         main = "rentels of flats in Goettingen 2015/12")
# size by .fractions, color by rooms
data <- cbind(rentels, .fraction = (rentels[,3] / max(rentels[,3]))^.5)
iconplot(data,
         grp.xy             = Eur ~ qm,
         grp.color          = Rooms,
         vars.to.factors    = c(1,.5, .3),
         panel.frame        = FALSE,
         panel.space.factor = 0.1,
         panel.prop.to.size = 0.7,
         icons              = fname,
         icon.stack.type    = "s",
         icon.frame         = FALSE,
         icon.space.factor  = 0.05,
         lab.cex            = 0.8,
         main               = "size fby .fractions, color by rooms")
# jpg files as icons
  ## because of computation time
data <- as.table(Titanic[2:3,,,,drop=FALSE]) / 10
fname1 <- system.file("src", "walkman-r.jpg", package="aplpack") # fname1 <- "walkman-r.jpg"
fname2 <- system.file("src", "pw-esch.jpg", package="aplpack")   # fname2 <- "pw-esch.jpg"
p.set <- c(fname1, fname2)
iconplot(data,
         grp.xy             = 2 ~ 3+1,
         grp.color          = 1,
         grp.icon           = 3,
         icons              = p.set,
         colors             = c("blue", "green"),
         panel.space.factor = 0.05,
         panel.prop.to.size = c(.5, .5, 1),
         icon.aspect        = 1,
         icon.space.factor  = .10,
         icon.horizontal    = TRUE,
         icon.draft         = FALSE,
         icon.stack.type    = c("lb", "lt", "rb","rt")[1],
         icon.grey.levels   = list(2, 10),
         lab.side           = "t", lab.cex = .7,
         main = "walkman and pw icons, scaled subset of Titanic")

# files of different types as icons
  ## because of computation time
fname3 <- system.file("src", "pw-esch.ppm", package="aplpack") # fname3 <- "pw-esch.ppm"
fname4 <- system.file("src", "pw-esch.png", package="aplpack") # fname4 <- "pw-esch.png"
p.set <- c(fname2, fname3, fname4)
iconplot(trees,
         grp.xy             = Girth ~ Height,
         grp.icon           = Height,
         grp.color          = Volume,
         vars.to.factors    = c(Volume = 4, Girth = 3, Height = 3),
         panel.space.factor = 0.05,
         panel.prop.to.size = c(.7, .45),
         panel.frame        = FALSE,
         icons              = p.set,
         icon.cex           = 14,
         icon.grey.levels = 6, icon.space.factor   = 0.05 )

# using raster graphics objects as icons
data <- as.table(Titanic[1:2,,,,drop=FALSE])/10
image1 <- as.raster(  matrix( c(1,0,1,1,0,1,1,0,1), ncol = 3, nrow = 3))
image2 <- as.raster(  matrix( c(1,0,1,0,0,0,1,0,1), ncol = 3, nrow = 3))
iconplot(data,
         grp.xy            = 2 ~ 4+1,
         grp.color         = 1,
         grp.icon          = 4,
         colors            = c("blue", "green"),
         icons             = list(image1, image2),
         icon.aspect       = 1,
         icon.space.factor = .10,
         icon.horizontal   = TRUE,
         icon.draft        = FALSE,
         icon.stack.type   = c("lb", "lt", "rb","rt")[1],
         icon.grey.levels  = list(2, 10),
         lab.side = "t", lab.cex = .7, main = "some Titanic data")
# using internal generator "fir.tree"
  ## because of computation time
data <- trees
iconplot(data,
         grp.color          = 3,
         grp.xy             = 1 ~ 2,
         vars.to.factor     = c(5, 5, 8),
         icons              = "fir.tree",
         colors             = rainbow(8, start = .1, end = .5),
         icon.frame         = FALSE,
         lab.legend         = 2,
         lab.cex            = 0.7,
         main = "grouping by vars and by colors")

# using different internal generators
data <- trees
iconplot(data,
         grp.color          = 1,
         grp.xy             = 1 ~ 2,
         grp.icon           = 2,
         colors             = c("orange", "green", "orange", "red"),
         icons = c("nabla", "BI", "walkman", "car.simple", "bike", "circle"),
         vars.to.factor     = c(3,6),
         lab.legend         = 2,
         lab.cex            = 0.7,
         main = "grouping by vars, by icons and by colors")
# Traveller plot proposed by M. Mazziotta and A. Pareto
Mazzi.Pareto <-
 structure(list(Region = c("Piemonte", "Valle d'Aosta", "Lombardia",
 "Trentino-Alto Adige", "Veneto", "Friuli-Venezia Giulia", "Liguria",
 "Emilia-Romagna", "Toscana", "Umbria", "Marche", "Lazio", "Abruzzo",
 "Molise", "Campania", "Puglia", "Basilicata", "Calabria", "Sicilia",
 "Sardegna"), Mean = c(98.74, 104.07, 101.38, 106.1, 104.38, 105.55,
 102.76, 103.62, 101.84, 103.52, 102.05, 97.88, 102.9, 91.43,
 94.12, 96.78, 93.55, 92.59, 96.29, 100.45), Penalty = c(0.43,
 4.23, 0.64, 0.63, 0.77, 0.34, 0.29, 0.46, 0.27, 0.22, 0.15, 0.82,
 1.3, 1.02, 0.37, 0.21, 2.37, 0.51, 0.31, 0.76), MPI = c(98.3,
 99.84, 100.74, 105.47, 103.61, 105.21, 102.47, 103.16, 101.57,
 103.3, 101.9, 97.06, 101.6, 90.42, 93.75, 96.58, 91.18, 92.08,
 95.98, 99.69)), .Names = c("Region", "Mean", "Penalty", "MPI"
 ), row.names = c(NA, -20L), class = "data.frame")
dm <- cbind(Mazzi.Pareto,
            col = as.factor(rep(1:4, 5)),        # as.factor!!
            row = as.factor(rep(1:5, each = 4))) # as.factor!!
iconplot(dm, verbose = !TRUE, x.text = 60,  y.text = -10,  #t3s
         grp.xy            = row ~ col,
         grp.icon          = 0 + Mean +  Penalty + Region,
         vars.to.factor    = FALSE,
         icons             = "mazz.man",
         panel.reverse.y   = TRUE,
         icon.space.factor = 0,
         icon.frame        = FALSE,
         lab.parallel      = TRUE,
         lab.side          = c("",""),
         main = "Traveller plot")
# definition of a check list, tally or 'Krebholz'
check.list <- function(x, colors = rainbow(length(x))){
  num.split <- function(x, div = 5){
    x.name <- as.character(substitute(x))
    xn <- lapply( x, function(x)
       c(rep(div, x %/% div), if( 0 < ( h <- x %% div) ) h )
    )
    len <- max(sapply(xn, length))
    xn <- lapply( xn, function(x) c(x, rep(0, len - length(x) )))
    xn <- matrix( unlist(xn), ncol = len, byrow = TRUE )
    xn <- as.table(xn)
    dimnames(xn) <- list( seq( along = x ), 1:len)
    names(dimnames(xn)) <- c(x.name, "Blocks")
    xn
  }
  x.split <- num.split(x)
  rownames(x.split) <- paste(sep = ":", 1:length(x), x)
  iconplot(x.split,
           grp.xy             = 1 ~ 2,
           grp.col            = 1,
           colors             = colors,
           panel.space.factor = c(0.4, 0.3),
           panel.frame        = FALSE,
           icon.stack.len     = 5,
           icon.space.factor  = c(0.4, 0),
           icon.asp           = NA,
           icon.frame         = FALSE,
           lab.side           = "l",
           lab.cex            = 0.7,
           main = paste("score of", substitute(x)))
  x.split
}
set.seed(13); data <- sample(1:50, size = 15)
check.list(data)
```

Index