strucplots—Visualizing higher-dimensional Contingency Tables in **vcd**

by David Meyer
Wirtschaftsuniversität Wien, Austria
David.Meyer@R-project.org

July 30, 2005

The `strucplot` framework in **vcd** can be used to visualize higher-dimensional contingency tables and integrates techniques such as mosaic displays and association plots. The main idea is to visualize the tables' cells arranged in rectangular form. To allow for more than two dimensions, the plot area is constructed using recursive conditional splits, given the tiles representing the levels of the previous dimension. This principle defines a graph 'family' with still many parameters left such as:

- the split direction for each dimension
- the content of the tiles
- the graphical parameters of the tiles' content
- the spacing between the tiles
- the labeling of the tiles

The document at hand gives an introduction to the framework, whereas labeling and shading issues are described in separate vignettes.

1 Framework Overview

The strucplot framework is highly modularized; Figure 1 shows the hierarchical relationship between the various components. The core function is `strucplot()` which processes the parameters, sets up the graphical layout using grid viewports (see Figure 2), and coordinates panel, labeling, shading, and spacing functions to produce the plot. The actual workhorse is the chosen panel function: currently, low-level panel functions for association plots (`struc_assoc`) and mosaic plots (`struc_mosaic`) are available. In addition, there are high-level wrapper functions such as `mosaic()` and `assoc()` to `strucplot()` that conveniently set the parameters—it is these high-level functions that the user normally would like to use, although `strucplot()` is fully functional on its own. `mosaic()` and `assoc` are generic functions: for both, there exist a formula method and a default method. Other functions, such as `doubledecker()`, `pairs()`, and `cotabplot`, are on even higher level and make use of codemosaic() and `assoc()` to produce plots.

2 Mosaic and Association Plots

As an example, consider the famous 'Survival on the Titanic' data containing two binary variables (survival to the disaster and gender), as well as two categorical variables (class and age). The 'flattened' contincency table can be obtained using the `structable()` function (quite similar to `ftable()`, but allowing the specification of split directions as in mosaic():

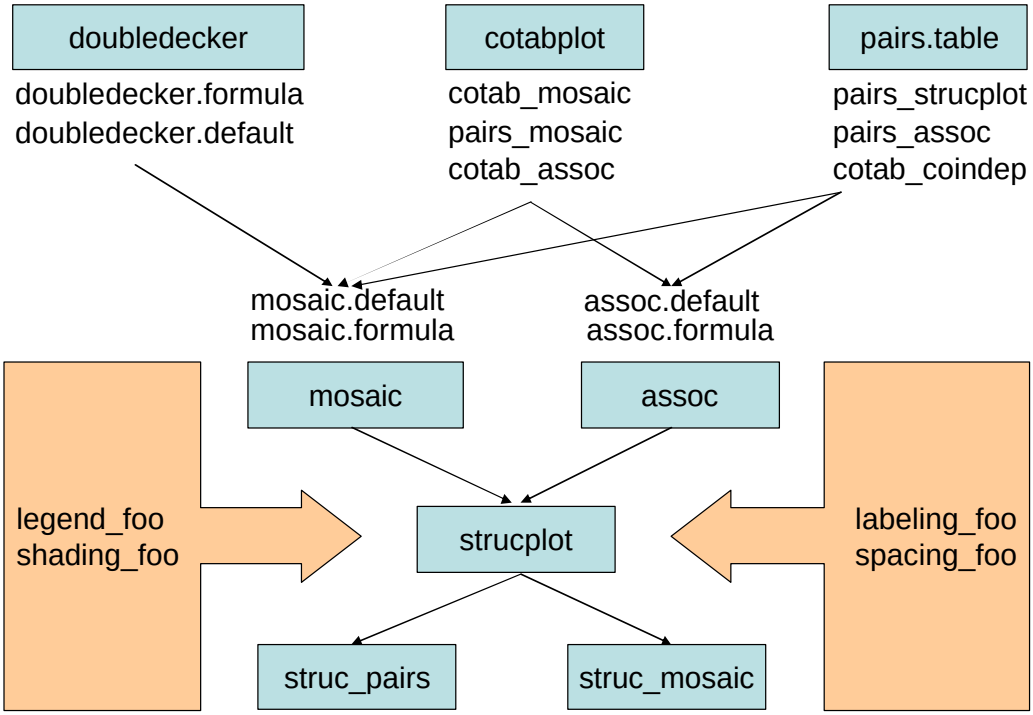

Figure 1: Strucplot components.

```
> structable(Titanic)
```

```
            Sex      Male     Female
            Survived   No Yes     No Yes
Class Age
1st   Child             0   5      0   1
      Adult             118 57     4 140
2nd   Child             0   11     0   13
      Adult             154 14     13  80
3rd   Child             35  13     17  14
      Adult             387 75     89  76
Crew  Child             0   0      0   0
      Adult             670 192    3   20
```

Let us first visualize the contingency table by the means of a mosaic plot (Hartigan and Kleiner, 1984) which is basically an area-proportional visualization of (typically observed) frequencies, composed of tiles (corresponding to the cells) created by recursive vertical and horizontal splits of a square. Thus, the area of each tile is proportional to the corresponding cell entry *given* the dimensions of previous splits (see Figure 3):

```
> mosaic(Titanic, type = "expected", main = TRUE, sub = "Expected values")
```

The small bullets indicate zero entries in the corresponding cell.
It is also possible to visualize the expected values instead of the observed values (see Figure 4):

```
> mosaic(Titanic, type = "expected", main = TRUE, sub = "Expected values")
```

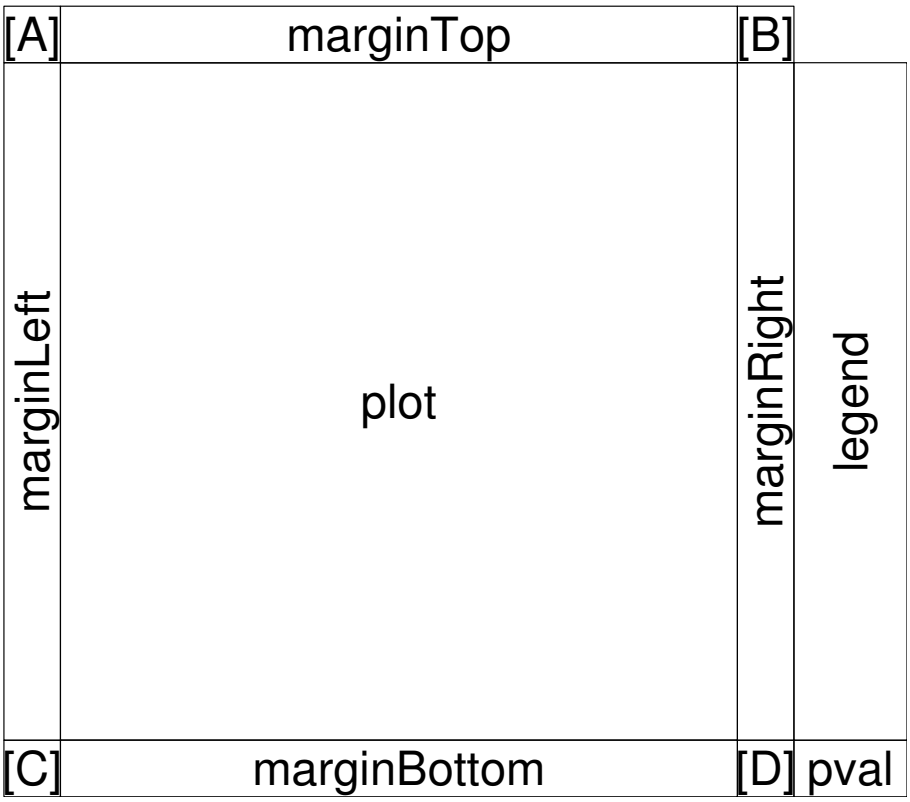

Figure 2: Viewport layout for strucplots with their names. [A] = "cornerTL", [B] = "cornerTR", [C] = "cornerBL", [D] = "cornerBR".

The Pearson residuals (standardized deviations of observed from expected values) are preferably visualized using association plots (Cohen, 1980). In contrast to `assocplot()`, **vcd**'s `assoc()` function scales to more than two variables (see Figure 5):

```
> assoc(Titanic, compress = FALSE)
```

The `compress` argument keeps distances between tiles equal for better comparison.

So far, we have visualized the full table. Parts of the data can be selected either using `margin.table()` on the input data, or the formula interfaces. The following example compares the age distribution among classes, given gender, and also shows how to use grid's viewport framework to put two plots besides each other:

```
> pushViewport(viewport(layout = grid.layout(ncol = 2)))

> pushViewport(viewport(layout.pos.col = 1))
> mosaic(~ Class + Age, data = Titanic[,"Male",,],
+        margin = c(bottom = 0), sub = "Male", newpage = FALSE)
> popViewport()

> pushViewport(viewport(layout.pos.col = 2))
> mosaic(~ Class + Age, data = Titanic[,"Female",,],
+        margin = c(bottom = 0), sub = "Female", newpage = FALSE)
> popViewport(2)
```

note the use of the `margin` argument: it takes a vector with up to four values whose unnamed components are recycled, but "overruled" by the named arguments. Thus, in the example, only the

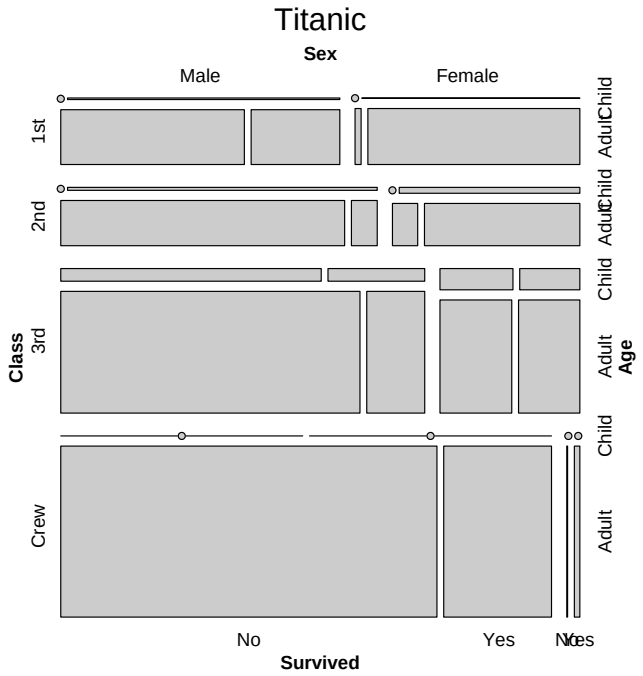

Figure 3: Mosaic plot for the Titanic data.

bottom margin is set to 0, leaving the default size of 2 lines unchanged for the other margins. This idea applies to almost all vectorized arguments in the strucplot framework (with `split_vertical` as a prominent exception).

Since mosaic displays are "conditional plots" by definition, we can also use one single mosaic for stratified plots. The formula interface of `mosaic()` allows the specification of conditioning variables (see Figure 7):

```
> mosaic(~ Class + Age | Sex, data = Titanic, split = TRUE)
```

Note, however, that the plots in the "pseudo-strata" are distorted since they are not corrected for the marginal distribution(s) of the conditioning variables. Indeed, the `cotabplot()` function would do a much better job here (see next section).

For both mosaic plots and association plots, the splitting of the tiles can be controlled using the `split_vertical` argument (default: alternating splits starting with a vertical one). Sensible recycling rules apply. For compatibility with `mosaicplot()`, the `mosaic()` function also allows the use of a `"direction"` argument taking a vector of `"h"` and `"v"` characters (see Figures 8 and 9):

```
> mosaic(Titanic, direction = "h")
> mosaic(Titanic, split_vertical = c(TRUE, TRUE, TRUE, FALSE))
```

3 Higher-level functions

The last plot pretty much resembles a doubledecker plot, except for the spacing. Since spacing is also modular in the strucplot framework, the `doubledecker()` function was straightforward to implement: it really is just a wrapper for `mosaic()`, setting the right defaults. Figure 10 shows a doubledecker plot of the Titanic data, explaining the probability of Survival by Age, given Sex, given Class:

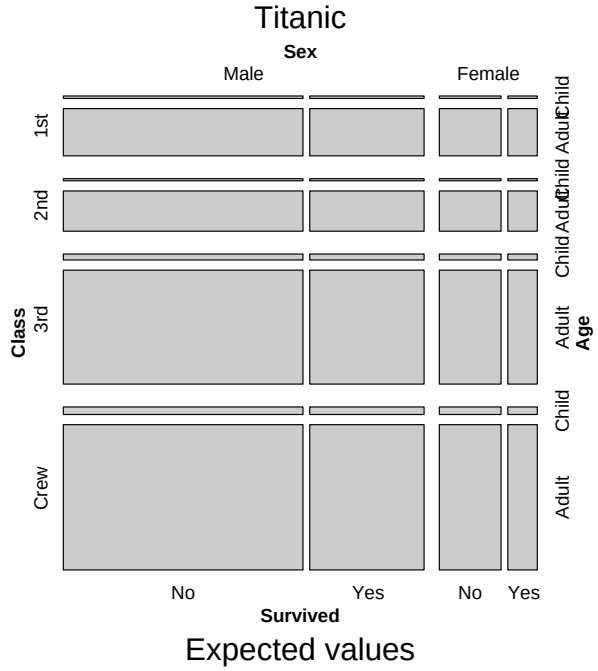

Figure 4: Mosaic plot for the Titanic data (expected values).

```
> doubledecker(Titanic)
> ## equivalent to:
> doubledecker(Survived ~ Class + Sex + Age, data = Titanic)
```

Another high-level function is `pairs.table()`, that is, the S3-method for objects of class "table" of the `pairs()` generic. This function produces a matrix of strucplots (currently, association or mosaic plots) in the off-diagonal cells, and the variable names (with, optionally, univariate statistics) in the diagonal cells. Figure 11 shows mosaic plots visualizing mutual independence in the lower triangle, association plots for the same in the upper triangle, and bar charts in the diagonal.

```
> pairs(Titanic, upper_panel = pairs_assoc)
```

Each cell's row and column define two variables X and Y used for the specification of four different types of independence: '`pairwise`', '`total`', '`conditional`' and '`joint`'. The pairwise mosaic matrix shows bivariate marginal relations between X and Y, collapsed over all other variables. The total independence mosaic matrix shows mosaic plots for mutual independence, i.e., for marginal and conditional independence among all pairs of variables. The conditional independence mosaic matrix shows mosaic plots for marginal independence of X and Y given all other variables. The joint independence mosaic matrix shows mosaic plots for joint independence of all pairs (X, Y) of variables from the others.

Since the matrix is symmetric, the upper and lower part can be used to display different types of independence models, or different strucplots (currently, only association and mosaic plots are supported). The available panel functions (`pairs_assoc()` and `pairs_mosaic()`) are just simple wrappers to `assoc()` and `mosaic()`. Obviously, seeing patterns in strucplot matrices becomes increasingly difficult with higher dimensionality. Typically, therefore, this plot is used with a suitable residual-based shading (described in a separate vignette).

Another useful high-level plot is produced by `cotabplot()`: this function can be used for plotting stratified strucplots in a lattice-like display. Whereas `pairs()` is conditioning on variables,

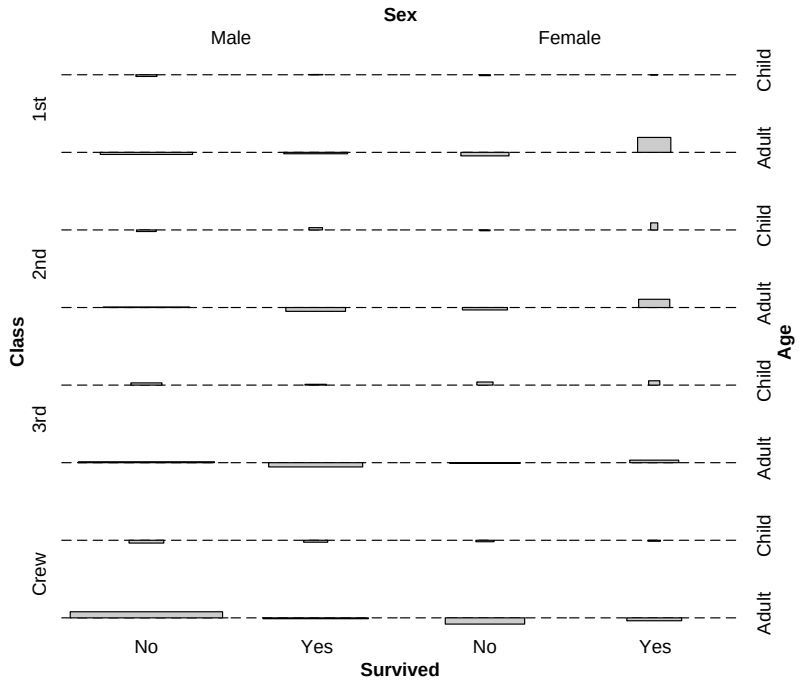

Figure 5: Association plot for the Titanic data.

`cotabplot()` is conditioning on variable *levels*. The plot in Figure 12 shows the Survival rate among the classes, given all four level combinations of Sex and Age.

```
> cotabplot(~ Class + Survived | Sex + Age, data = Titanic)
```

4 Technical Notes on Panel Functions

All strucplot components (including the panel functions) are parameterized through the use of generating functions, that is, functions with all parameters in their signature returning a function to be called by `strucplot()`. This returned function has access to all parameters passed to the generating function thanks to lexical scope. The idea is to use module-specific 'code factories' that generate parameterized workhorses rather than struggling with monolithic functions with endless parameter lists. As a simple example, consider `struc_mosaic()` and `struc_assoc`, the generating functions for the mosaic and the association plots:

```
struc_mosaic <- function(zero_size = 0.5)
  function(residuals, observed, expected = NULL, spacing, gp, split_vertical)
  ...


struc_assoc <- function(compress = TRUE, xlim = NULL, ylim = NULL,
                        yspace = unit(0.5, "lines"), xscale = 0.9)
  function(residuals, observed = NULL, expected, spacing, gp, split_vertical)
  ...
```

Both functions have specific parameters in their signature, but both return a function with a standardized interface that subsequently is called by `strucplot` to produce the actual plot. As can

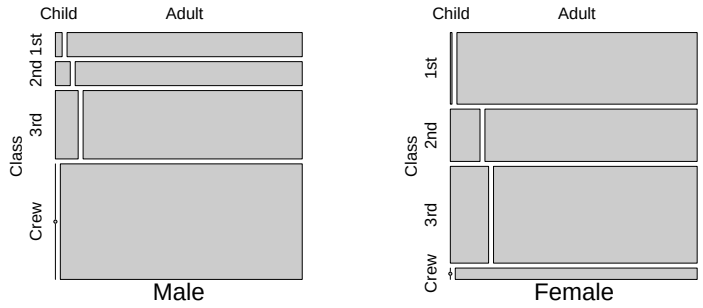

Figure 6: Age distribution among classes, given gender.

be seen, the only information `strucplot()` passes to the panel functions are residuals, observed and expected values, a structure containing the between-tile spacing information, an object of class `"gpar"` for the tiles' graphical parameters, and a logical vector with the splitting information for each dimension. The workhorse function, of course, is free to use only part or all of this information. High-level interface functions such as `mosaic()` and `assoc` call `strucplot()` by specifying the corresponding function along with the specific parameters. Now, there a several ways of doing the same thing, as illustrated by the following equivalent calls:

```
> strucplot(Titanic, panel = struc_mosaic(zero_size = 1))
> strucplot(Titanic, panel = struc_mosaic, panel_args = list(zero_size = 1))
> strucplot(Titanic, panel = myworkhorse)
```

where `myworkhorse` could be, e.g., a function with hard-wired parameters, but with an interface complying with the workhorse signature. To enable `strucplot()` to distinguish between generating functions and workhorse functions, the generating functions must be of class `"panel_generator"`.

All panel functions are supposed to produce conditional hierarchical plots by the means of nested viewports, corresponding to the provided splitting information. Thus, at the end of the plotting, each tile is associated with a particular viewport. Each of those viewports has to be conventionally named, enabling other strucplot modules, in particular the labeling functions, to access specific tiles after they have been plotted. Note that the viewport tree is popped by default. The following example shows how to access parts of the plot after it has been drawn (see Figure 13):

```
> mosaic(~ Sex + Class, data = Titanic, pop = FALSE)
> current.vpTree() ## output not shown

> seekViewport("cell..Sex.Male")
> grid.rect(gp = gpar(col = "red", lwd = 4))

> seekViewport("cell..Sex.Male..Class.Crew")
> grid.circle(r = 0.2, gp = gpar(fill = "cyan"))
```

The naming convention for the viewports is: `cell..`*`[Variable 1].[Level 1]..[Variable 2].[Level 2]`* .... Clearly, these names depend on the splitting.

In addition to the viewports, the main graphical elements get names following a similar construction method. This allows to change graphical parameters of plot elements after the plotting (see Figure 14):

```
> assoc(Titanic)
> getNames() ## output not shown
```

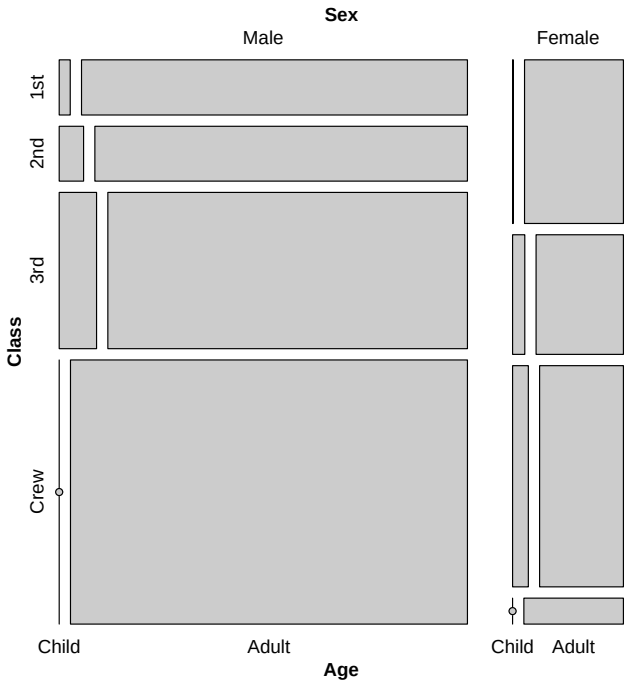

Figure 7: Mosaic plot for the Titanic data (expected values).

```
> grid.edit("rect..Class.1st..Age.Adult..Sex.Female..Survived.Yes",
            gp = gpar(fill = "red"))
```

References

Cohen, A. (1980). On the graphical display of the significant components in a two-way contingency table. *Communications in Statistics—Theory and Methods*, A9:1025–1041.

Hartigan, J. and Kleiner, B. (1984). A mosaic of television ratings. *The American Statistician*, 38:32–35.

Figure 8: Mosaic plot for the Titanic data—alternative splitting (1).

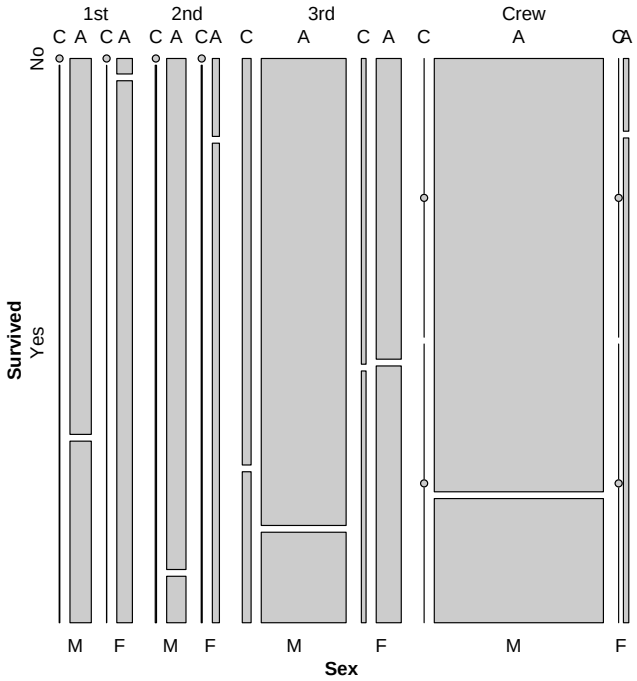

Figure 9: Mosaic plot for the Titanic data—alternative splitting (2).

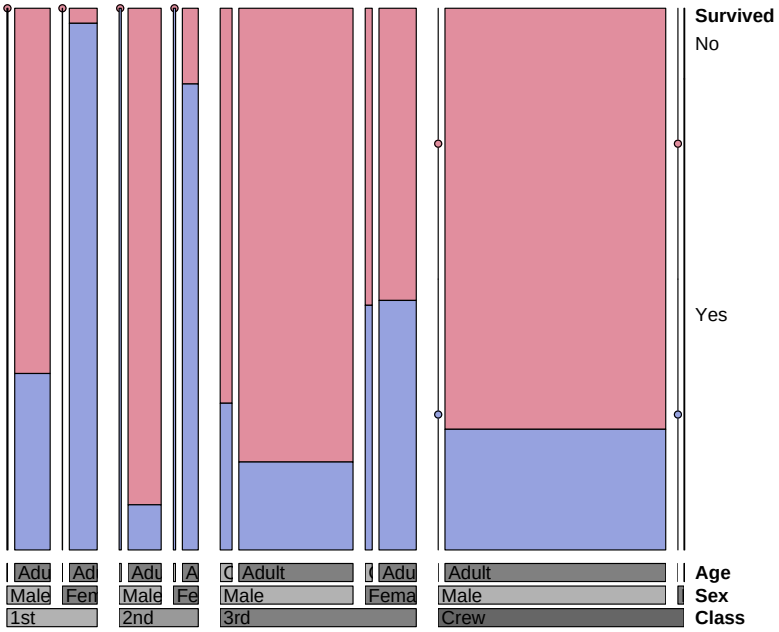

Figure 10: Doubledecker plot for the Titanic data.

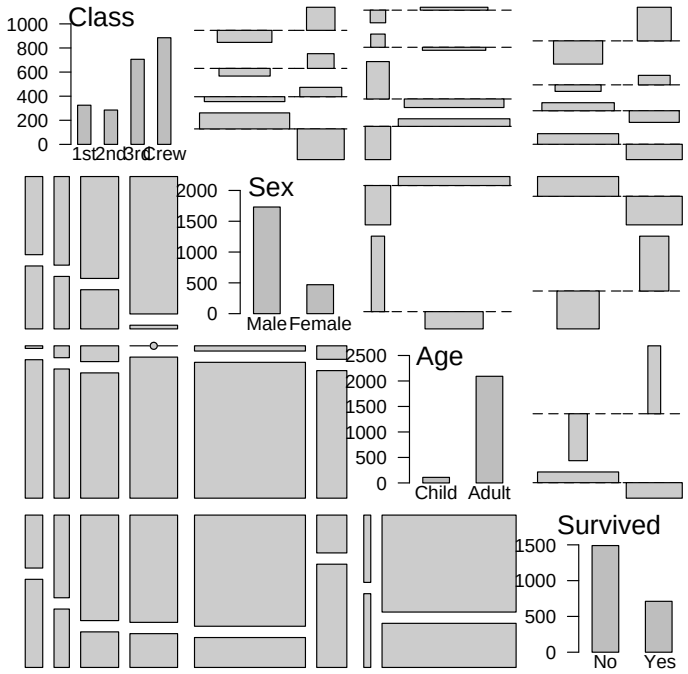

Figure 11: Pairs plot for the Titanic data.

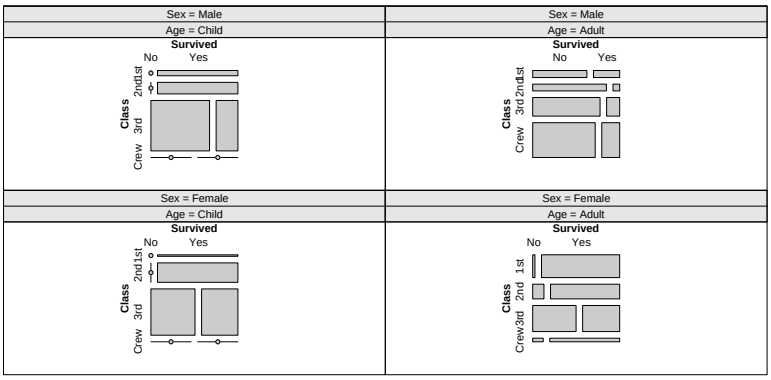

Figure 12: cotabplot for the Titanic data.

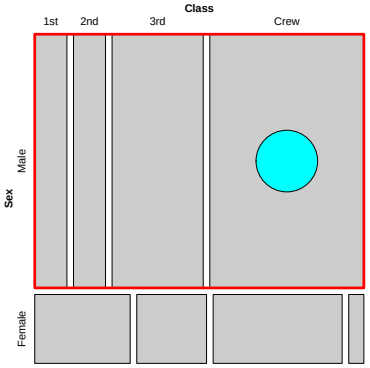

Figure 13: Adding elements to a mosaic plot after drawing.

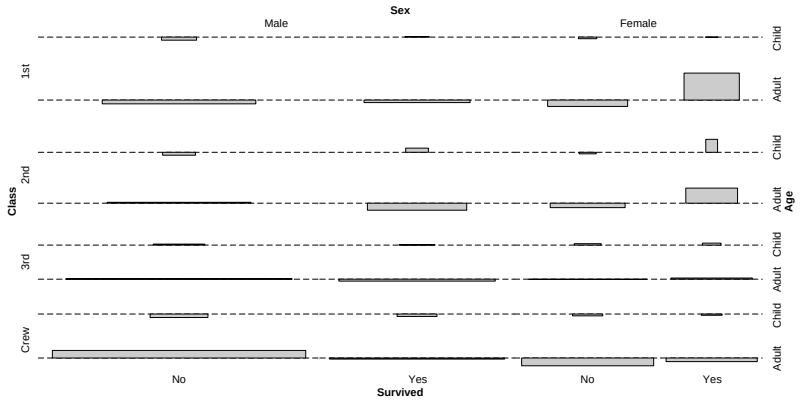

Figure 14: Changing graphical parameters of elements after drawing.