Colors and Residual-based Shadings in Strucplots

by David Meyer

Wirtschaftsuniversität Wien, Austria

David.Meyer@R-project.org

September 19, 2005

Unlike other graphics functions in base R, the strucplot framework allows almost full control over the graphical parameters of all plot elements. In particular, the user can *individually* modify the graphical appearance of tiles in association plots, mosaic plots, and sieve plots. Built on top of this functionality, the framework supplies a set of shading functions choosing colors appropriate for the visualization of log-linear models. The tiles' graphical parameters are set using the `gp` argument of the strucplot functions. This argument basically expects an object of class `gpar` whose components are arrays of the same shape (length and dimensionality) than the data table (see Section 1). For convenience, however, the user can also supply a function that computes such an object given a vector of residuals, or, alternatively, a generating function that takes parameters and returns such a function (see Section 2). We provide several shading functions, including support for both the HSV and the HCL color space, and the visualization of significance tests (see Section 3).

1 Specifying graphical parameters of strucplot displays

As an example, consider the 'UCBAdmissions' data. In the table collapsed over departments, we would like to highlight the (incidentally wrong) impression that there were to much male students accepted compared to the presumably discriminated female students (see Figure 1):

```
> (ucb <- margin.table(UCBAdmissions, 1:2))

        Gender
Admit     Male Female
  Admitted 1198   557
  Rejected 1493  1278

> (fill_colors <- matrix(c("dark cyan", "gray", "gray", "dark magenta"),
+     ncol = 2))

     [,1]         [,2]
[1,] "dark cyan" "gray"
[2,] "gray"       "dark magenta"

> mosaic(ucb, gp = gpar(fill = fill_colors, col = 0))
```

As the example shows, we create a fourfold table with appropriate colors (dark cyan for admitted male students and dark magenta for rejected female students) and supply them to the `fill` component of the `gpar` object passed to the `gp` argument of `mosaic()`. For optical clarity, we addtionally hide the tiles' borders by setting the `col` component to 0 (white).

If the parameters specified in the `gpar` object are incomplete, strucplot functions will recycle them along the last splitting dimension. In the following example based on the 'Titanic' data, we will highlight all cells corresponding to survived passengers (see Figure 2):

1

Figure 1: Mosaic plot for the 'UCBAdmissions' data with highlighted cells.

```
> mosaic(Titanic, gp = gpar(fill = c("gray", "dark magenta")), spacing = spacing_highlighting,
+      labeling_args = list(abbreviate = c(Age = 3), rep = c(Survived = FALSE)))
```

Note that `spacing_highlighting()` sets the spaces between tiles in the last dimension to 0. The `labeling_args` argument ensures that labels do not overlapp (see the separate vignette on labeling for more information).

2 Customizing residual-based shadings

This flexible way of specifying graphical parameters is the basis for a suite of shading functions that modify the tiles' appearance with respect to a vector of residuals, resulting from deviations of the observed from the expected values under a given log-linear model. The idea is to visualize at least the size of the residuals, but some shadings, additionally, indicate overall significance. One particular shading, the maximum shading, even allows to identify those cells that cause the rejection of the null hypothesis.

Conceptually, the strucplot framework offers three alternatives to add residual-based shading to plots:

1. Pre-computing the graphical parameters (e.g., fill colors), encapsulating them into an object of class `"gpar"` as demonstrated in the previous section, and passing this object to the `gp` argument.

2. Providing a function to the `gp` argument that takes residuals as input and returns an object as described in 1.

3. Providing a *generating* function taking parameters and returning a function as described in 2. The parameters of this generating function are either directly specified (which is not possible for all shading functions), or as a list to the `shading_args` argument.
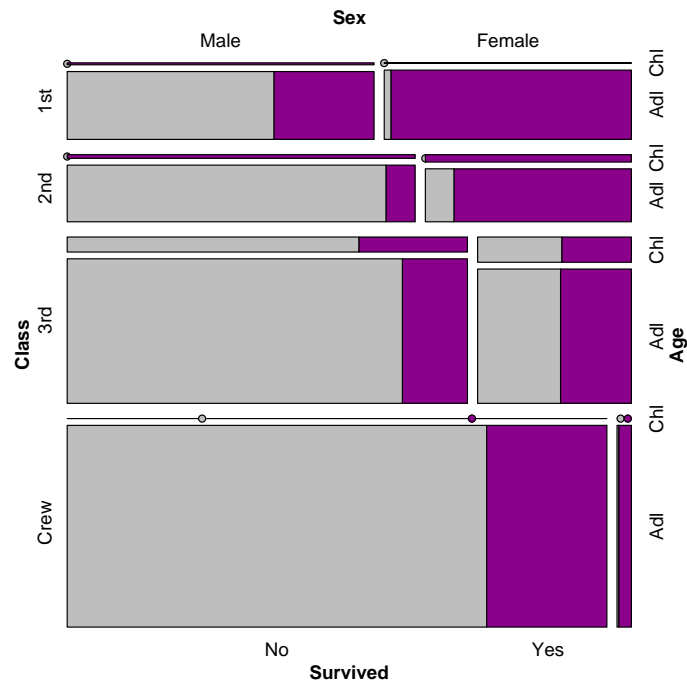
Figure 2: Recycling of parameters, used for highlighting the survived passengers in the 'Titanic' data.

For each of these approaches, we will demonstrate the necessary steps to obtain a binary shading that visualizes the sign of the residuals by a corresponding fill color (for simplicity, we will treat 0 as positive).

The first method is precomputing the graphical parameters "by hand". We will use 'light blue' color for positive and 'light salmon' color for negative residuals (see Figure 3):

```
> expected <- independence_table(ucb)
> (residuals <- (ucb - expected)/sqrt(expected))

         Gender
Admit          Male    Female
  Admitted  4.784093 -5.793466
  Rejected -3.807325  4.610614

> (fillcols <- ifelse(residuals > 0, "lightblue", "lightsalmon"))

         Gender
Admit     Male          Female
  Admitted "lightblue"   "lightsalmon"
  Rejected "lightsalmon" "lightblue"

> mosaic(ucb, gp = gpar(fill = fillcols))
```

For implementing method 2, we need to create a 'shading function' that—technically speaking—computes `"gpar"` objects from residuals. For that, we can just reuse the code from the previous step:

```
> shade_fun <- function(res) gpar(fill = ifelse(res > 0, "lightblue",
+      "lightsalmon"))
```
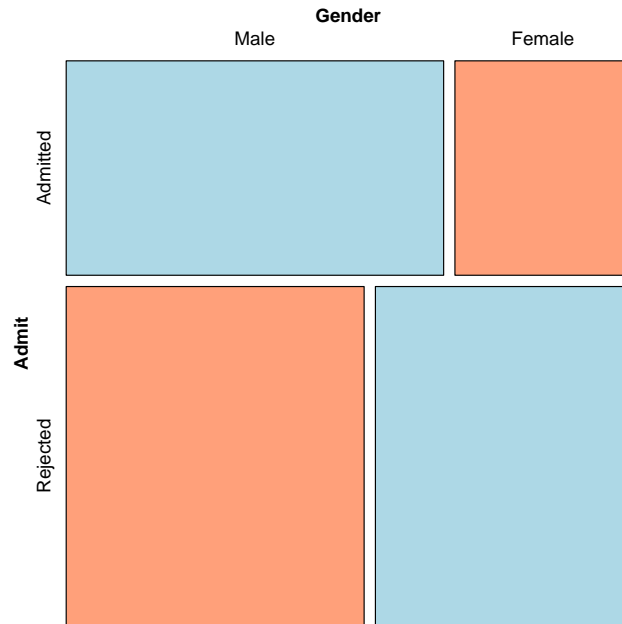
Figure 3: Binary shading visualizing the sign of the residuals.

To create a mosaic display with binary shading, it now suffices to specify the data table along with `shade_fun()`:

```
> mosaic(ucb, gp = shade_fun)
```

`mosaic()` internally calls `strucplot()` which computes the residuals from the specified independence model (total independence by default), passes them to `shade_fun()`, and uses the `"gpar"` object returned to finally create the plot.

Our `shade_fun()` function might be useful, but suffers from a major flaw: the colors are hardwired. We cannot simply extend the argument list to include, e.g., a `fill = c("lightblue", "lightsalmon")` argument because `strucplot()` will neither know how to handle it, nor let us change the defaults. In fact, the interface of shading functions is fixed, they are expected to take exactly one argument: a table of residuals. This is where generating functions (method 3) come into play. We simply wrap our shading function in another function that takes all additional arguments it needs to use, possibly preprocesses them, and returns the actual shading function. This function will have access to the parameters since in R, nested functions are lexically scoped. This kind of wrapping function is also called a *generating function* since it returns ('creates') a 'parameterized' shading function with the minimal standard interface `strucplot()` requires. The follwing example shows the necessary extensions for our running example:

```
> myshading_binary <- function(col = c("lightblue", "lightsalmon")) {
+     if (length(col) != 2)
+         stop("Need exactly two colors!")
+     function(res) gpar(fill = ifelse(res > 0, col[1], col[2]))
+ }
```

In the call of `mosaic()` (figure not shown), using the new `myshading_binary()` function, we can now simply change the colors:

```
> mosaic(ucb, gp = myshading_binary(c("red", "blue")))
```

4

The procedure described so far is a rather general concept, applicable to a wide family of user-level **grid** graphics. Indeed, the customization of other components of the strucplot framework (labeling, spacing, legend, and core functions) follows the same idea. Now for the shading functions, this is not the whole story. Since shading functions can be used for visualizing significance (see next section), it makes sense for generating functions to have access to the model, i.e., observed and expected values, residuals, and degrees of freedom. For example, the `shading_max()` generating function computes a permutation distribution of the maximum statistic and p values for specified significance levels based on the observed table to create data-driven cut-off points. If this was done in the shading function itself, the permutation statistic would be recomputed every time the shading function is called, resulting in possibly severe performance loss and numerical inconsistencies. Therefore, generating functions for shadings are required to take at least the parameters `observed`, `expected`, `residuals`, and `df` (these are provided by the strucplot framework), followed by other parameters controlling the shading appearance (to be specified by the user):

```
> myshading_binary2 <- function(observed = NULL, residuals = NULL, expected = NULL,
+     df = NULL, col = c("lightblue", "lightsalmon")) {
+     if (length(col) != 2)
+         stop("Need exactly two colors!")
+     function(res) gpar(fill = ifelse(res > 0, col[1], col[2]))
+ }
> class(myshading_binary2) <- "generating_function"
```

In some sence, generating functions for shadings are parameterized both by the user and the strucplot framework. For shading functions that require model information, the user-specified parameters are to be passed to the `gp_args` argument instead, and for this to work, the generating function needs a class attribute to be distinguishable from the "normal" shading functions. For others (like our simple `myshading_binary2()`) this is optional, but recommended for consistency:

```
> mosaic(ucb, gp = myshading_binary2, gp_args = list(c("red", "blue")))
```

This final version pretty much resembles `shading_binary()`, one of the standard shading functions provided by the **vcd** package.

3   An overview of the shading functions in vcd

Apart from `shading_binary()`, there are currently two basic shadings available: `shading_hcl()` and `shading_hsv()`, as well as two derived functions: `shading_Friendly()` building upon `shading_hsv()`, and `shading_max()` building upon `shading_hcl()`. `shading_hsv()` and `shading_hcl()` provide the same conceptual tools, but use different color spaces: the Hue-Saturation-Value (HSV) and the Hue-Chroma-Luminance (HCL) scheme, respectively. We will first expose the basic concept of these shading functions using the HSV space, and then briefly explain the differences to the HCL space (a detailed discussion can be found in Zeileis et al., 2005). The HCL space is trickier to use, but preferable to the HSV space from a perceptual point of view.

In the HSV space, colors are specified in three dimensions: Hue, Saturation ('colorfulness'), and Value ('brightness', amount of gray). These three dimensions are used by `shading_hsv()` to visualize information about the residuals and the underlying independence model. The hue indicates the residuals' sign: by default, blue for positive, and red for negative residuals. The saturation of a residual is set according to its size: high saturation for large, and low saturation for small residuals. Finally, the overall brightness is used to indicate the significance of a test statistic: bright colors for significant, and dark colors for non-significant results.

As an example, we will visualize the association of hair and eye color in the 'HairEyeColor' data set (see Figure 4)

```
> haireye <- margin.table(HairEyeColor, 1:2)
> mosaic(haireye, gp = shading_hsv)
```
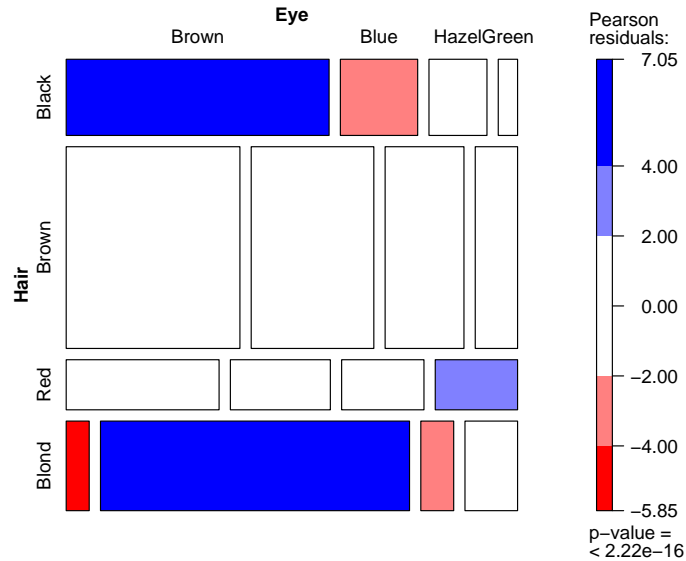
Figure 4: Shaded residuals in the 'HairEyeColor' data set—two cut-off points.

Large positive residuals (greater than 4) can be found for brown eyes/black hair and blue eyes/blond hair, they are colored in saturated red. On the other hand, there is a large negative residual (less than −4) for brown eyes/blond hair, colored deep blue. There are also three medium-sized positive (negative) residuals between 2 and 4 (−2 and −4): the colors for them are less saturated. Residuals between −2 and 2 are shaded in white. The heuristic for choosing the cut-off points 2 and 4 is that the Pearson residuals are approximately standard normal which implies that the highlighted cells are those with residuals *individually* significant at approximately the $\alpha = 0.05$ and $\alpha = 0.0001$ level, respectively. These default cut-off points can be changed to alternative values using the `interpolate` argument (see Figure 5):

```
> mosaic(haireye, gp = shading_hsv, gp_args = list(interpolate = 1:4))
```

The `interpolate` argument also accepts a function, which then is called with the absolute residuals to get a vector of cut-off points. Thus, it is possible to automatically choose the cut-off points in a data-driven way. For example, one might think that the extension from four cut-off points to a continuous shading—visualizing the whole range of significance levels—could be useful. We simply need a one-to-one mapping from the residuals to the p values:

```
> ipol <- function(x) pmin(x/4, 1)
```

Note that this `ipol()` function maps residuals greater than 4 to a significance level of 1. However, the resulting plot (Figure 6) is deceiving:

```
> mosaic(HairEyeColor, gp = shading_hsv, gp_args = list(interpolate = ipol),
+     labeling_args = list(abbreviate = c(Sex = 3)))
```

Too much color makes it difficult to interpret the image, and the subtle color differences are hard to catch. Therefore, we only included shadings with discrete cut-off points.

The third remaining dimension, the value, is used for visualizing the significance of a test statistic. The user can either directly specify the p value, or, alternatively, a function that computes
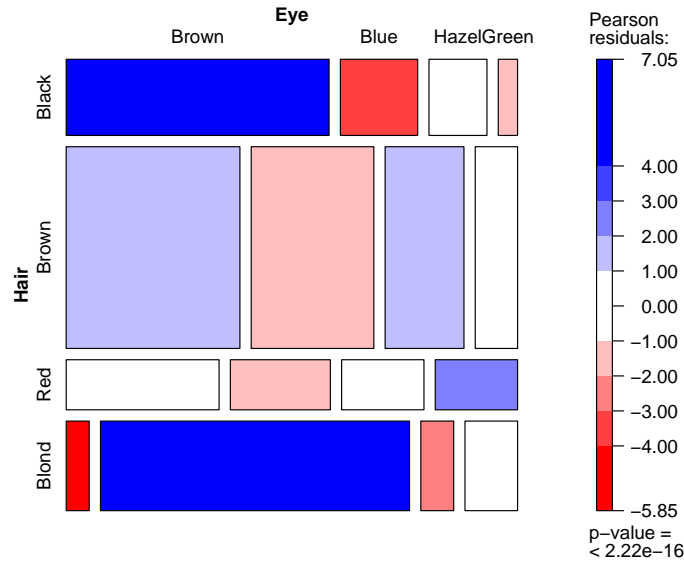
Figure 5: Shaded residuals in the 'HairEyeColor' data set—four cut-off points.

it, to the p.value argument. Such a function must take observed and expected values, residuals, and degrees of freedom as arguments. If nothing is specified, the p value is computed from a χ^2 distribution with df degrees of freedom. The level argument is used to specify the confidence level: if p.value is smaller than 1 - level, bright colors are used, otherwise dark colors are employed. The following example using the 'Bundesliga' data shows the relationship of home goals and away goals of the German premier's soccer league in 1995: although there are two "larger" residuals (one greater than 2, one less then −2), the χ^2 test does not reject the null hypothesis of independence. Consequently, the colors appear dark (see Figure 7):

```
> bl <- xtabs(~HomeGoals + AwayGoals, data = Bundesliga, subset = Year ==
+     1995)
> mosaic(bl, gp = shading_hsv)
```

A shading function building upon shading_hsv() is shading_Friendly(), miming the shading introduced by Friendly (1994). In addition to the defaults of the HSV shading, it uses the border color and line type to redundantly code the residuals' sign. The following example again uses the 'Bundesliga' data from above, this time using the Friendly scheme and, in addition, an alternative legend (see Figure 8):

```
> mosaic(bl, gp = shading_Friendly, legend = legend_fixed, zero_size = 0)
```

(The zeroSize = 0 argument removes the bullets indicating zero observed values. This feature does not exist in the original SAS implementation of the Friendly mosaic plots.)

As introduced before, the default shading scheme is not shading_hsv() but shading_hcl() due to the better perceptual characteristics of the HCL color space. Figure 9 depicts color scales for red and blue hue (left and right side, respectively) in the HSV and HCL spaces (upper and lower part, respectively), varying colorfulness and brightness (x and y-axis, respectively). The boxes represent the diverging color palettes used for the shadings. For the HSV space, we can can see that the effect of changing the level of brightness ('value') is not the same for different levels
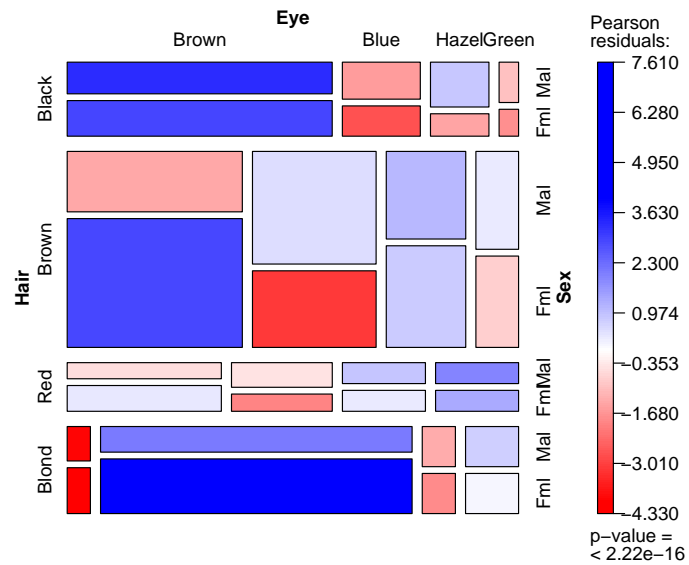
7

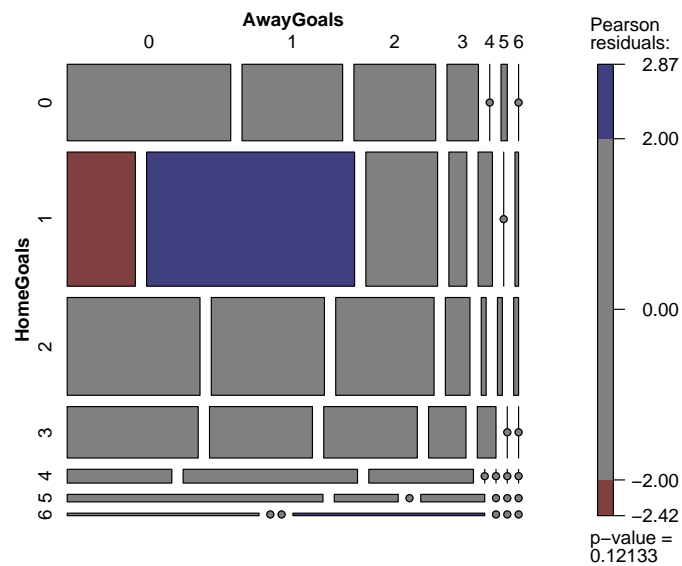Figure 6: The 'HairEyeColor' data with continuous shading.

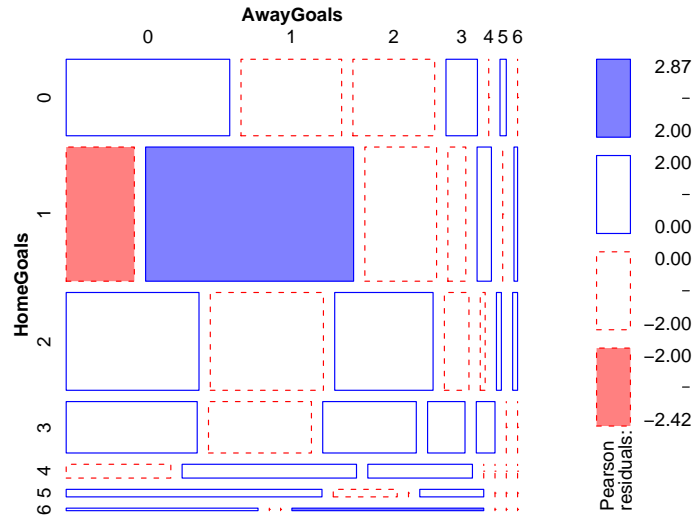Figure 7: Non-significant χ^2 test using part of the 'Bundesliga' data.

Figure 8: The 'Bundesliga' data for 1995 using the Friendly shading and a legend with fixed bins.

of saturation, and again not the same for the two different hues. In fact, in the HSV space all dimensions are confounded, which obviously is problematic for coding information. In contrast, the HCL color space offers perceptual uniform colors: as can be seen from the lower part, the chroma is homogeneous for different levels of luminance. Unfortunately, this comes at the price of the space being irregularly shaped, making it difficult to automatically select diverging color palettes. The following example again illustrates the 'HairEyeColor' data, this time with HCL colors (Figure 10 depicts the default palette, and Figure 11 an alternative setting):

```
> mosaic(haireye, gp = shading_hcl)

> mosaic(haireye, gp = shading_hcl, gp_args = list(h = c(130, 43), c = 100,
+       l = c(90, 70)))
```

A more 'advanced' function building upon shading_hcl() is shading_max(), using the maximum statistic both to conduct the independence test and to visualize significant *cells* causing the rejection of the independence hypothesis (Meyer et al., 2003). The level argument of shading_max() then can be used to specify several confidence levels from which the corresponding cut-off points are computed. By default, two cut-off points are computed corresponding to confidence levels of 90% and 99%, respectively. In the following example, we investigate the effect of a new treatment for rheumatoid arthritis on a group of female patients using the maximum shading (see Figure 12):

```
> mosaic(~Treatment + Improved, data = Arthritis, subset = Sex == "Female",
+       gp = shading_max)
```

The maximum test is significant although the residuals are all in the $[-2, 2]$ interval, as the cut-off points are even smaller. The shading_hcl() function with default cut-off points would not have shown any color. In addition, since the test statistic is the maximum of the absolute Pearson residuals, *every* colored residual violates the null hypotheses of independence, and thus, the 'culprits' can immediately be identified.
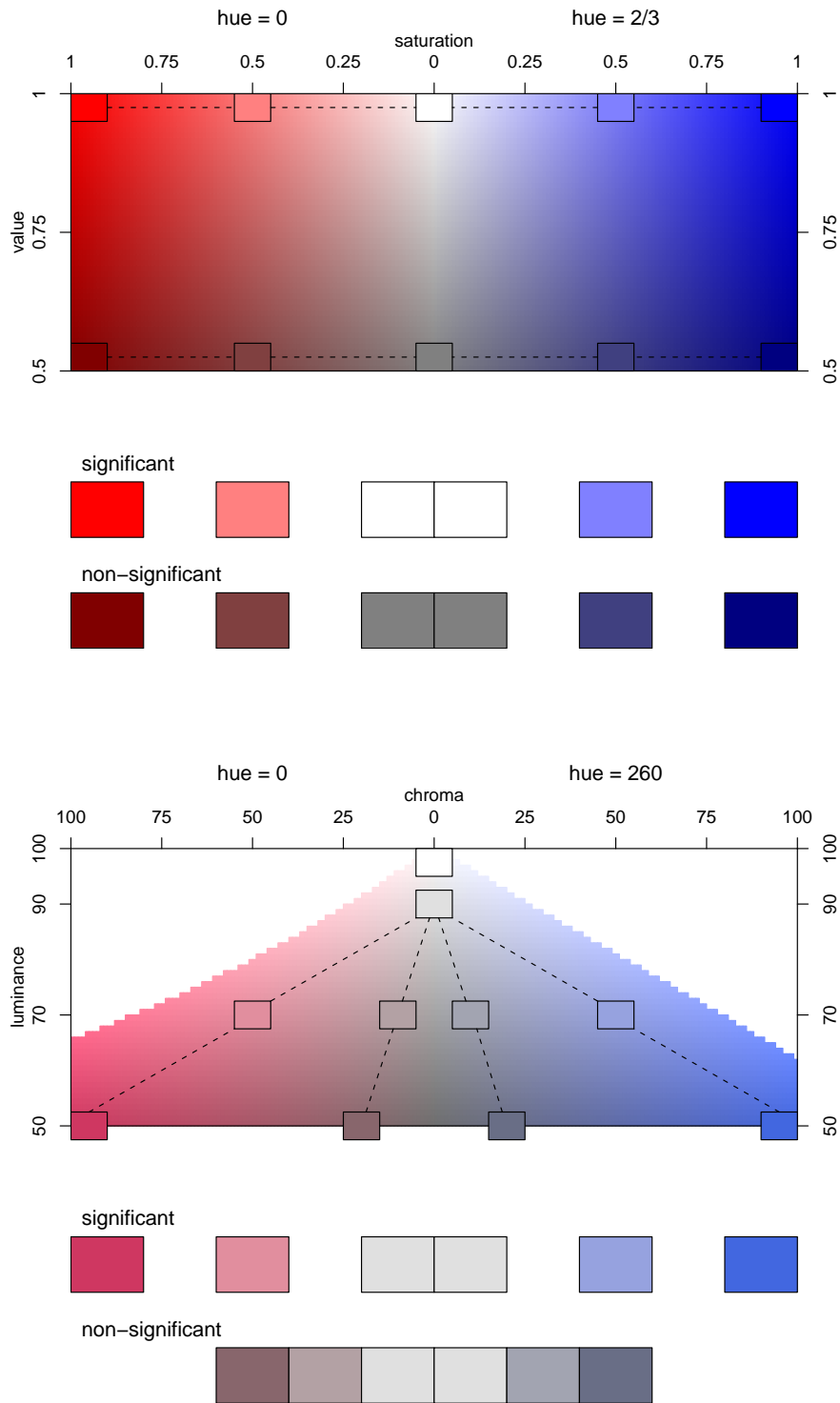
9

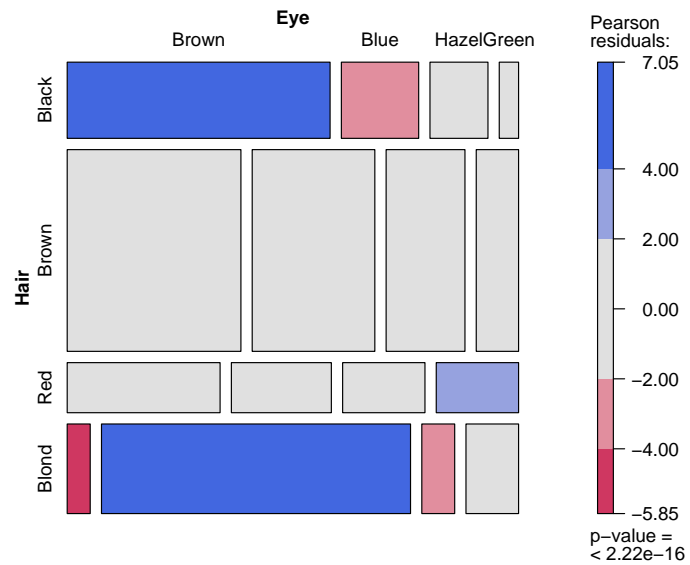Figure 9: Residual-based shadings in HSV (upper) and HCL space (lower).

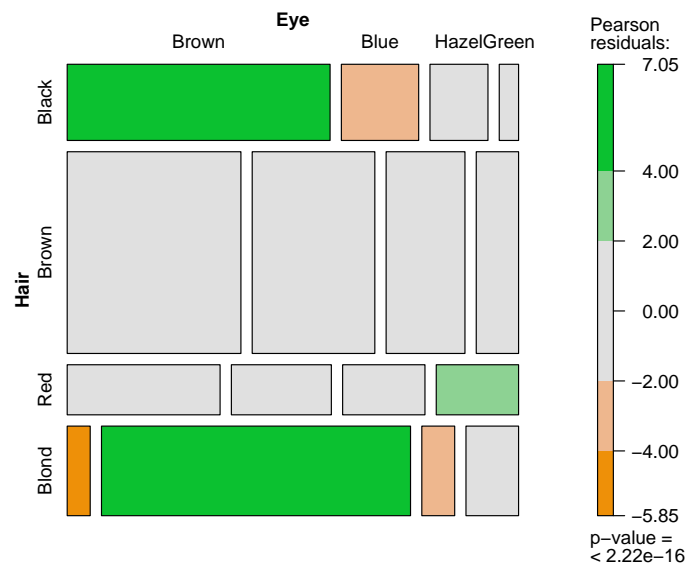Figure 10: The 'HairEyeColor' data, using default HCL color palette.

Figure 11: The 'HairEyeColor' data, using a custom HCL color palette.
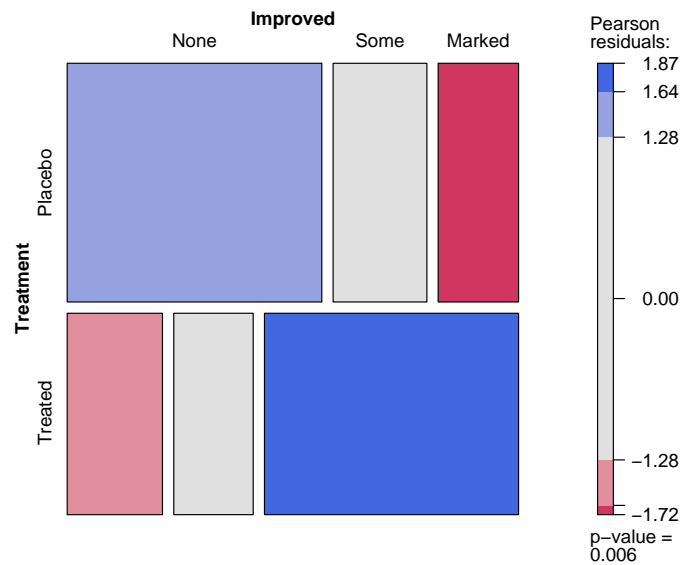
Figure 12: Significant maximum test on female patients of the 'Arthritis' data.

References

Friendly, M. (1994). Mosaic displays for multi-way contingency tables. *Journal of the American Statistical Association*, 89:190–200.

Meyer, D., Zeileis, A., and Hornik, K. (2003). Visualizing independence using extended association plots. In Hornik, K., Leisch, F., and Zeileis, A., editors, *Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria*. ISSN 1609-395X.

Zeileis, A., Meyer, D., and Hornik, K. (2005). Residual-based shadings for visualizing (conditional) independence. Report 20, Department of Statistics and Mathematics, Wirtschaftsuniversität Wien, Research Report Series.