Visualization of large multivariate datasets with the `tabplot` package

Martijn Tennekes and Edwin de Jonge

May 2, 2012
(A later version may be available on CRAN)

Abstract

The tableplot is a powerful visualization method to explore and analyse large multivariate datasets. In this vignette, the implementation of tableplots in R is described.

1 Introduction

The tableplot is a visualization method that is used to explore and analyse large datasets. Tableplots are used to explore the relationships between the variables, to discover strange data patterns, and to check the occurrence and selectivity of missing values.

A tableplot applied to the diamonds dataset of the `ggplot2` package (where some missing values were added) is illustrated in Figure 1. Each column represents a variable. The whole data set is sorted according to one column (in this case, carat), and then grouped into row bins. Algorithm 1 in Appendix A describes the creation of a tableplot into detail.

Tableplots are aimed to visualize multivariate datasets with several variabels (up tot a dozen) and a large number of records, say at least one thousand. Tableplots can also be generated for datasets with less records, but they may be less useful. The maximum size of datasets that can be visualized with the `tabplot` package depends on the R's memory, or, when using the `ff` package, on the limitations of that package.

A graphical user interface for generating tableplots is implemented in the package `tabplotGTK`.

2 Getting started with the `tableplot` function

The diamonds dataset is very suitable to demonstrate the `tabplot` package. To illustrate the visualization of missing values, we add several `NA`'s.

```
> require(ggplot2)
> data(diamonds)
> ## add some NA's
> is.na(diamonds$price) <- diamonds$cut=="Ideal"
> is.na(diamonds$cut) <- (runif(nrow(diamonds))>0.8)
```

A tableplot is simply created by the function `tableplot`:

```
> tableplot(diamonds)
```

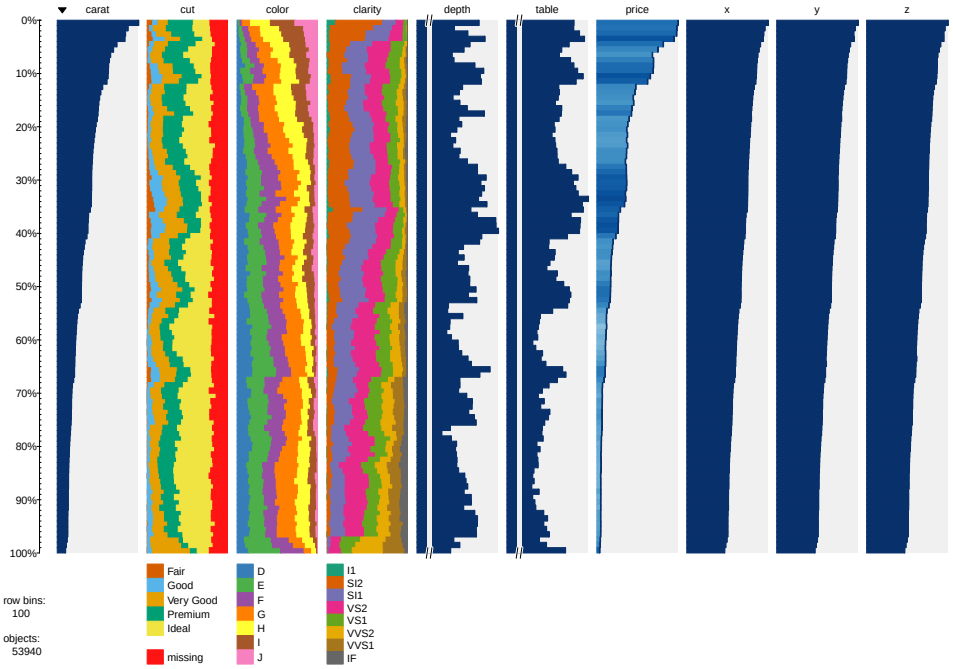

Figure 1: Tableplot of the diamonds dataset

The result is depicted in Figure 1. By default, all variables of the dataset are depicted. With the argument `colNames`, we can specify which variables are plotted. The dataset is by default sorted according to the values of the first variable. With the argument `sortCol`, we can specify on which variable(s) the data is sorted.

```
> tableplot(diamonds,
+           colNames=c("carat", "price", "cut", "color", "clarity"),
+           sortCol="price")
```

The result is illustrated in Figure 2.

Setting an appropriate number of row bins (by the argument `nbins`) is important, like in a histogram. A good number of row bins is a trade of

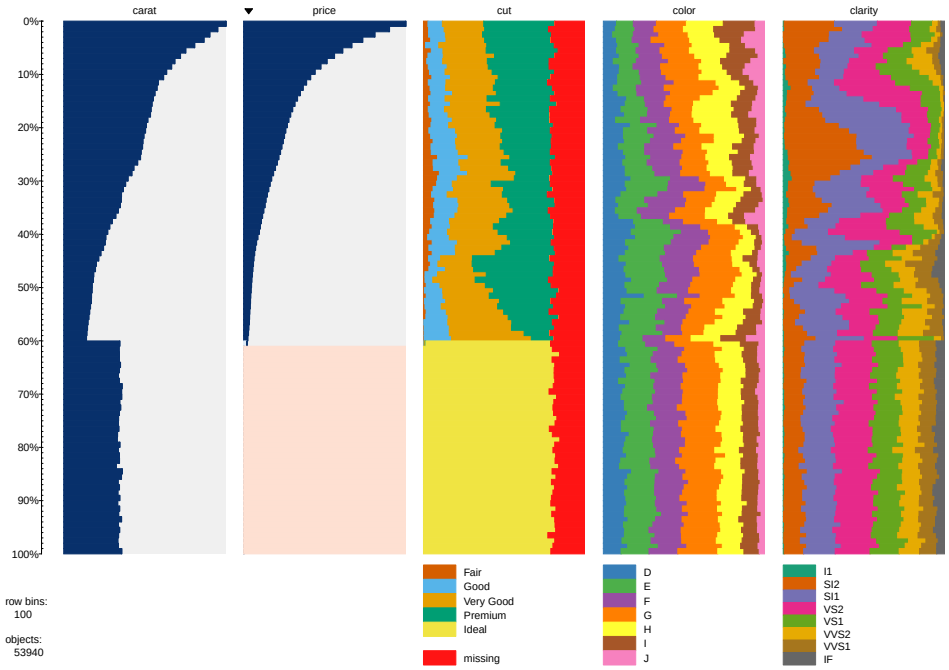

Figure 2: Tableplot of the diamonds dataset: sorted by price

between good polished but meaningless data, and detailed, but noisy data. In practice, we found that the default number of 100 usually is a good starting point.

The percentages near the vertical axis indicate which subset of the data in terms of units (rows) is depicted. The range from 0% to 100% in Figure 2 means that all units of the data are plotted.

We can focus our attention to the 5% most expensive diamonds by setting the `from` argument to 0 and the `to` argument to 5:

```
> tableplot(diamonds,
+           colNames=c("carat", "price", "cut", "color", "clarity"),
+           sortCol="price",
+           from=0,
+           to=5)
```

Observe that in the obtained tableplot in Figure 3, the number of row bins is still 100, so that the number of units per row bin is now 27 instead of 540. Therefore, much more detail can be observed in this tableplot.

The vertical axis contains two sets of tick marks. The small tick marks correspond with the row bins and the large tick marks correspond with the percentages between `from` and `to`. The latter are determined by R's base function `pretty`.

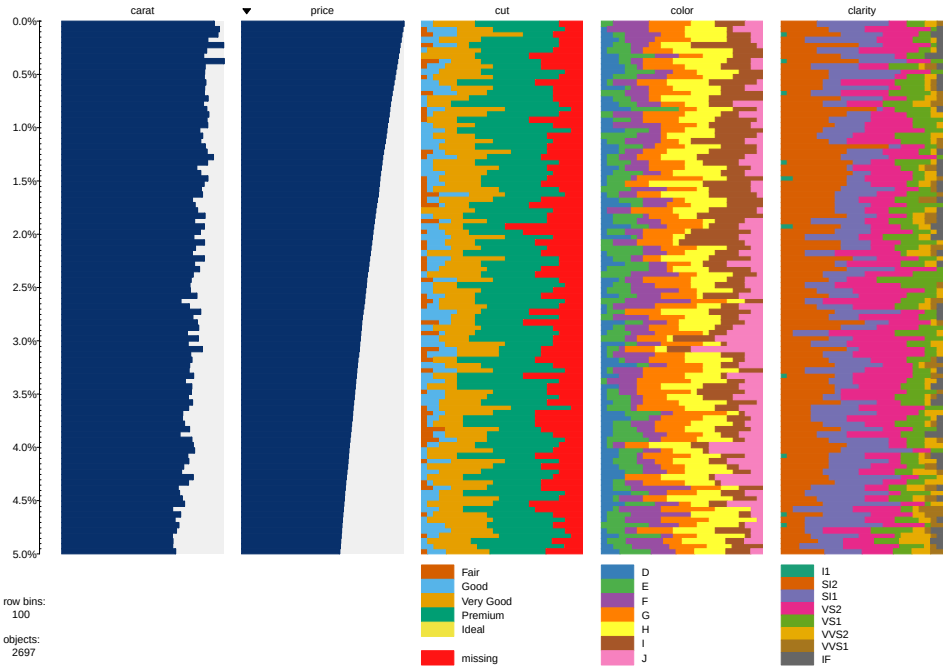

Figure 3: Tableplot of the diamonds dataset: zooming in

3 Customizing the tableplot

3.1 Continuous variables

For each bin of a continuous variable, the mean value is calculated (see Algorithm 1). When the distribution of these mean values is exponential, it is useful to apply a logarithmic transformation. The argument `scales` can be set to linear mode `"lin"`, logarithmic mode `"log"`, or the default value `"auto"`, which automatically determines which of the former two modes is used.

Observe that the x-axes of the variables depth and table in Figure 1 are broken. The x-axis of a variable i is broken if either

$$0 < max(m_{i1}, m_{i2}, \ldots, m_{in}) \qquad \text{AND}$$
$$\texttt{bias\_brokenX} \cdot max(m_{i1}, m_{i2}, \ldots, m_{in}) < min(m_{i1}, m_{i2}, \ldots, m_{in})$$

OR

$$0 > min(m_{i1}, m_{i2}, \ldots, m_{in}) \qquad \text{AND}$$
$$\texttt{bias\_brokenX} \cdot min(m_{i1}, m_{i2}, \ldots, m_{in}) > max(m_{i1}, m_{i2}, \ldots, m_{in}),$$

where `bias_brokenX` is a bias parameter that should be a number between 0 and 1. If `bias_brokenX=1` then the above conditions are always false,

which implies that the x-axes are never broken. On the other hand, if `bias_brokenX=0` then the x-axes are always broken. By default, `bias_brokenX =0.8`, which mean that an x-axis is broken if (in case of a variable with positive values) the minimum value is at least 0.8 times the maximum value. In the diamonds dataset, this applies to the variables depth and table.

3.2 Categorical variables

The color palettes of categorical variables can be customized with the argument `pals`. Several qualitative palettes are implemented. They can be shown by

```
> tablePalettes()
```

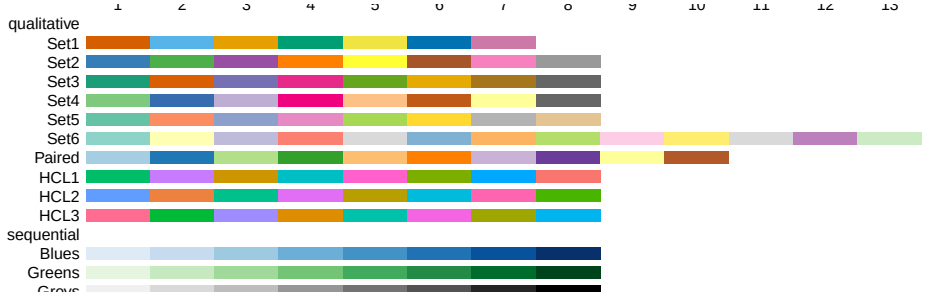

Figure 4: Color palettes

The default palette is a combination of Set1 and Set2. It has the advantage that each category has a unique color for variables with up to 16 categories.

Suppose we want a to use the default palette for the variable `cut`, but starting with the seventh color, pink. Further we want the fifth palette for the variable `color`, but without the first color (black), and a custom palette, say a rainbow palette, for the variable `clarity`:

```
> tableplot(diamonds, pals=list("Set1(7)", "Set5", rainbow(8)))
```

3.3 Filtering data

The argument `filter` serves as a filter condition for the data. The following code generates a tableplot of premium cut diamonds that cost less than 5000$.

```
> tableplot(diamonds, filter="price < 5000 & cut=='Premium'")
```

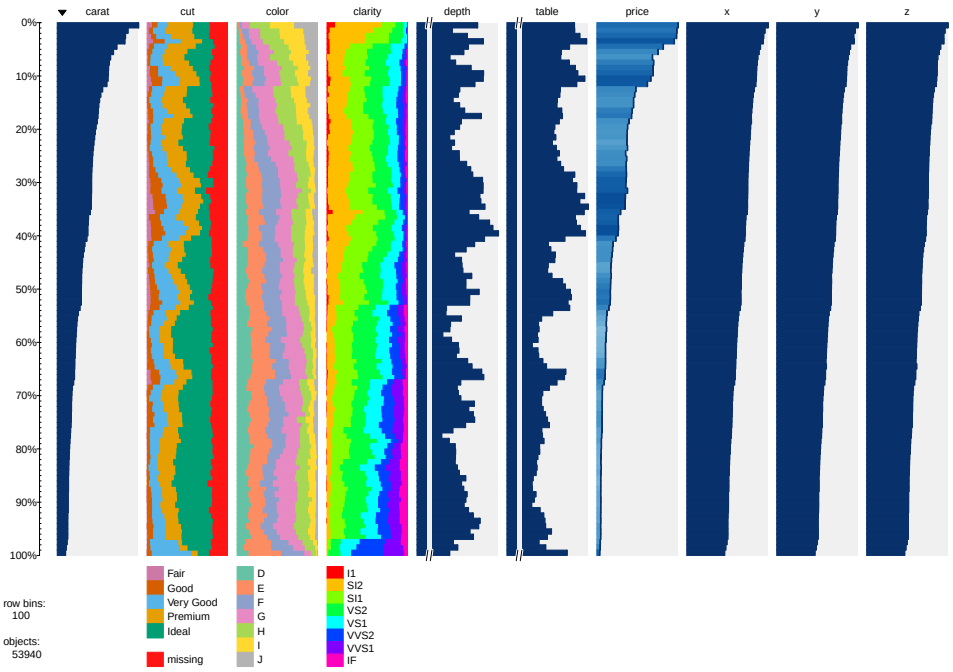

Figure 5: Tableplot of the diamonds dataset: other colour palettes

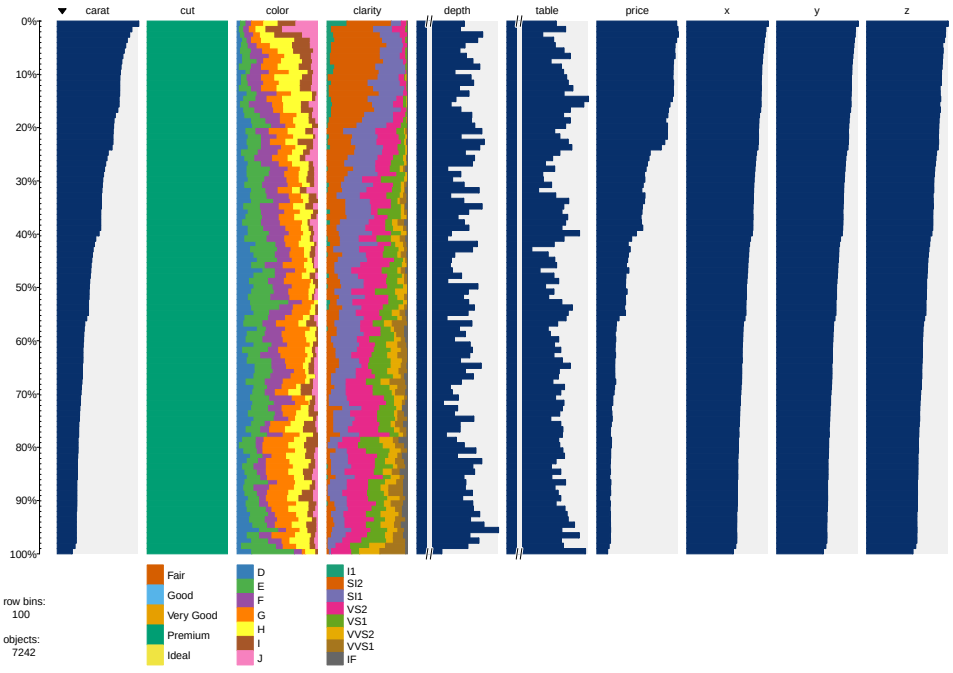

Figure 6: Tableplot of the diamonds dataset: filtered

3.4 The tabplot object

The function `tableplot` returns a tabplot-object, that can be used to make minor changes to the tableplot, for instance the order of columns or the color palettes. Of course, these changes can also be made by generating a new tableplot, such as in the examples above. However, if it takes considerable time to generate a tableplot, then it is practical to make minor changes immediately.

The output of the `tableplot` function can be assigned to a variable. The graphical output can be omitted by setting the argument `plot` to FALSE.

```
> tab <- tableplot(diamonds, plot=FALSE)
```

The tabplot-object is a list that contains all information to depict a tableplot. The generic functions `summary` and `plot` can be applied to the `tabplot` object.

```
> summary(tab)
```

```
      general                variable1                variable2
 dataset  :diamonds    name       :carat        name       :cut
 variables:10          type       :numeric      type       :categorical
 objects  :53940       sort       :decreasing   sort       :NA
 bins     :100         scale_init :auto         categories:6
 from     :0%          scale_final:lin
 to       :100%
      variable3                variable4                variable5
 name       :color         name       :clarity       name       :depth
 type       :categorical   type       :categorical   type       :numeric
 sort       :NA            sort       :NA            sort       :NA
 categories:7              categories:8              scale_init :auto
                                                     scale_final:lin

       variable6              variable7              variable8
 name       :table      name       :price      name       :x
 type       :numeric    type       :numeric    type       :numeric
 sort       :NA         sort       :NA         sort       :NA
 scale_init :auto       scale_init :auto       scale_init :auto
 scale_final:lin        scale_final:lin        scale_final:lin

       variable9              variable10
 name       :y          name       :z
 type       :numeric    type       :numeric
 sort       :NA         sort       :NA
 scale_init :auto       scale_init :auto
 scale_final:lin        scale_final:lin
```

```
> plot(tab)
```

The function `tableChange` is used to make minor changes to a tabplot-object. Suppose we want the columns in the order of 2, and we want to change all color palettes to default starting with the second color.

```
> tab2 <- tableChange(tab,
+                     colNames=c("carat", "price", "cut", "color", "clarity"),
+                     pals=list("Set1(2)"))
> plot(tab2)
```

With the function `tableSave`, tableplots can be saved to a desired grahical output format: pdf, eps, svg, wmf, png, jpg, bmp, or tiff.

```
> tableSave(tab,
+           filename="diamonds.png",
+           width=5,
+           height=3,
+           fontsize = 6,
+           legend.lines = 6)
```

4 Resources

- Summary of the package: `help(package=tabplot)`
- The main help page: `?tabplot`
- Project site: http://code.google.com/p/tableplot/
- References:
  - Tennekes, M., Jonge, E. de, Daas, P.J.H. (2011) Visual profiling of large statistical datasets. Paper presented at the 2011 New Techniques and Technologies for Statistics conference, Brussels, Belgium. (paper, presentation)

A Tableplot creation algorithm

A tabplot is basically created by Algorithm 1.

Algorithm 1 Create tableplot

Input: Tabular dataset t, column i_s of which the distribution is of interest[a], number of row bins n.

$t' \leftarrow$ sort t according to the values of column i_s.
Divide t' into n equally sized row bins according to the order of t'.
for each column i **do**
    if i is numeric **then**
        $m_{ib} \leftarrow$ mean value per bin b
        $c_{ib} \leftarrow$ fraction of missing values per bin b
    end if
    if i is categorical **then**
        $f_{ijb} \leftarrow$ frequency of each category j (including missing values) per bin b
    end if
end for
for each column i **do**
    if i is numeric **then**
        Plot a bar chart of the mean values $\{m_{i1}, m_{i2}, \ldots, m_{in}\}$, optionally with a logarithmic scale. The fraction of missing values $\{c_{i1}, c_{i2}, \ldots, c_{in}\}$ determines the lightness of the bar colour. The lighter the colour, the more missing values occur in bin b. If all values are missing, a light red bar of full length is drawn.
    end if
    if i is categorical **then**
        Plot a stacked bar chart according to the frequencies $\{f_{i1b}, f_{i2b}, \ldots\}$ for each bin b. Each category is shown is a distinct colour. If there are missing values, they are depicted by a red colour.
    end if
end for

Output: Tableplot

[a] The dataset t can also be sorted according to multiple columns.