sparkTable: Generating Graphical Tables for Websites and Documents with *R*

Alexander Kowarik
Statistics Austria
Guglgasse 13
1110 Vienna

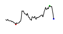
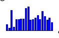
Bernhard Meindl
Statistics Austria
Guglgasse 13
1110 Vienna

Matthias Templ
Statistics Austria
Guglgasse 13
1110 Vienna

May 15, 2014

Abstract

Visual analysis of data is important to understand the main characteristics, main trends and relationships in data sets and it can be used to assess the data quality.

Using the *R* package **sparkTable**, statistical tables holding quantitative information can be enhanced by including spark-type graphs such as sparklines  and sparkbars . These kind of graphics have initially been proposed by [Tufte, 2001] and are considered as simple, intense and illustrative graphs that are small enough to fit in a single line. Thus they can easily enrich tables and texts with additional information in a comprehensive visual way.

The *R*-package **sparkTable** is using a clean `S4`-class design and provides methods to create different type of spark graphs that can be used in websites, presentations and documents. We also implemented an easy way also for non-experts to create highly complex tables. In this case, graphical parameters can be interactivly changed, variables can be sorted, graphs can be added and removed in an interactive manner to produce custom-tailored graphical tables – standard tables that are enriched with graphs – that can be displayed in a browser and exported to various formats.

1 Introduction

Readers can be easily understand simple visual presentations of data like charts published every day in newspapers, internet, television and scientific articles. For readers it is often much easier to understand statistics presented as a chart or a map, rather than long lists of numbers [see, e.g., United Nations Economic Commission for Europe, 2009]. The visual presentations of data should illustrate trends and relationships quickly and easily. The technical possiblities to present data gets even more sophisticated and extended in the last century [see, e.g., Bosch and de Jonge, 2008].

Tables:
Tables are a comprehensive way to summarize the most important facts about complex data. United Nations Economic Commission for Europe [2009] claims that a small, well-formatted table can provide a great deal of information that readers can quickly absorb. In their contribution [United Nations Economic Commission for Europe, 2009], but also in Miller [2004], guidelines are written how to design good tables. However, the possibility of including graphics in tables (and text) is missing.

Tuftes pioneer work:
In this contributions several examples show how to improve tables by inclusion of small graphics in the tables and it is pointed out that a good table may also include visual graphics. This originally proposed by Tufte [2001]. He introduced the concept of sparklines and their application that is not only limited to tables. These *intense, simple, word-sized graphics* sparklines can be embedded in flowing text and also match with other propositions he has made. For example, they can easily be used together with the small multiples approach [Tufte, 1990]. According to Tufte it is always important to answer the question *compared to what?* when looking at graphs. Thus, the concept of small multiples is simply putting multiple graphs next to each other to allow direct visual comparisons between different (graphical) objects. He also states that small multiples are the best design solution for presentation for a wide range of problems.

Another proposition Tufte made is to *show the data above all.* In order to do so he introduced the term of data-ink ratio. This ratio can be seen as the ink in a graph used to actually show data by the total ink used to print a graph. While the additional ink used for labels, axes, ticks and grids among others may help to improve graphs, it turns away attention from the most important part of the graph - the data values itself. Thus, in his view the data-ink ratio should be increased whenever possible and sparklines are a way to do so since it is possible to show a high amount of data, trends and variations in small space. This concept is also critised. Inbar et al. [2007] evaluated people's acceptance of the minimalist approach (by increasing data/ink ratio) to visualize information and a standard bar-graph and a minimalist version [both in Tufte, 2001] were compared. The majority of evaluators was more attracted by the standard graph. However, they didn't evaluate the case of placing sparklines in text or tables. Traditionally, the output in many areas of statistics is pretty much focused on tables [United Nations Economic Commission for Europe, 2009], this is especially true for official statistics. Large tables with lots of number provided a lot of information, but make it hard to grasp it. Visualization can provide tools to make effects in the data visible on first sight and provide the possibility to show more of the data then by just showing the numbers.

Software to produce sparklines:
To publish sparklines in newspapers has a long history. For example, the Huntsville Times, reports the use of sparklines in their sports section on March 21, 2005. To produce this kind of sparklines and graphical tables, various software products are available. *SAS*© and *JMP*© include facilities, but they offer also the link (batch call) to the **sparkTable** [Kowarik et al., 2012] package to produce tables including sparklines [Hill, 2011]. Also *Microsoft Excel* allows to produce sparklines and there are implementations to produce simple sparklines for *Photoshop*, *Prism*, *DeltaMaster* and *matplotlib*. Most of these solutions are closed-source and commerical.

2 The sparkTable package

Our contribution goes a step beyond the mentioned available tools. It is not just a reimplementation of sparklines in another software. Various improvements has been made and new features are available to the user. The presented software is free and open-source, it's features are design in a well-specified object-oriented manner with defined classes and methods. The presented software allows to save the sparklines in different output formats for easy inclusion in LaTeX or websites with many optional graphical features, i.e. the sparklines are highly customizable. The software features the simple generation of graphical tables with any statistical content that is automatically calculated from microdata. Finally, interactive features to create sparklines and sparktables are available and available and clickable in the browser.

In the following, a short introduction into the *R* package **sparkTable** [Kowarik et al., 2012] is given by example.

The package **sparkTable** is available as an add-on package to *R* [R Core Team, 2014], a free environment for statistical computing and graphics. The package first was uploaded to the Comprehensive R Archive Network (CRAN) in May 2010 and has been improved various times.

The initial task was to find out ways how common and frequent output such as tables and reports of statistical organistations could be improved using spark-type graphs and to find a technical solution to create these visualisations. Since *R* is gaining momentum in national statistical institues all over the world it was a natural choice to create an addon-package for *R* containing such functionality. The reasons are many, among them the possibility to easily share and distribute the package and to instantly get feedback from users.

The goal was to create a free and open source software tool that allows the quick creation of spark-type graphics that could easily be included in various documents such as reports or webpages. Many examples for graphical tables have been produced and it turned out that graphical tables are an excellent tool to improve traditional existing outputs of national statistical offices. Some of these graphical tables are presented below.

sparkTable can also be used also by non experts in *R*. Interactive clickable features have been added to the package. Because especially for graphical tables some process of abstraction is still required, it has been made much easier to customize and change graphical tables by using interactive web applications.

Instead of a formal description of classes and methods and internal implementation of the package, the functionality of the package is decribed by example.

3 Command line usage of sparkTable by example

The first step is to install and load the package. We assume that *R* has already been started. Then the package can be simply installed from CRAN and loaded as shown in Listing 1.

```
install.packages("sparkTable")
library("sparkTable")
```

Listing 1: Installing and Loading the sparkTable package

Help files can be accessed from *R* by typing `help(package=sparkTable)` into the console. This results in a browseable, linked help archive. Each method, function and data set that comes with the package is documented and also has some example code, which allows the user to become familiary with the package.To demonstrate the facilities of **sparkTable** example data sets available in the package are used.

Table 1 shows the most important functions and classes of the **sparkTable** package. Basic functions are available to produce sparklines, e.g. `newSparkLine` to create an object of class *sparkline* that can be used as input for the plotting function `plotSparks`. In addition, funtions are available to produce graphical tables and calculate summary statistics for tables, see Section 3.1 and 3.2 for practical examples and more explanations.

In the following it is shown how to create basic spark graphics in 3.1 while we demonstrate how to produce graphical tables that include some sparks in 3.2. We finish this Section by introducing the new interactive ways that can be used to change, modify and adjust objects generated with package **sparkTable** in 4.

Table 1: Most important functions of package **sparkTable**

function	description	comment
`newSparkLine` `newSparkBar` `newSparkBox` `newSparkHist`	creation of sparklines, sparkbars, sparkbox-plots and sparkhistograms that may serve as the base for creating graphical tables	creates objects of class *sparkline*, *sparkbar*, *sparkbox* and *sparkhist*
`plotSparks`	plots objects of class *sparkline*, *sparkbar*, *sparkhist* or *sparkbox*	
`summaryST`	summary for a data frame in a graphical table	creates an object of class *sparkTable*. Customize the output with `setParameter`
`newSparkTable`	function to create an object of class *sparkTable*	
`plotSparkTable`	plot objects of class *sparkTable*	object might produced by function `summaryST`
`getParameter`, `setParameter`	basic functions to set parameters for objects of class *sparkline*, *sparkbar*, *sparkbox*, *sparkTable* (or *geoTable*)	

3.1 Basic sparks

The dataset **pop** that is included in the package is used. This dataset contains the Austrian population size for different age groups from 1981 to 2009. We start by loading the data set into the current *R* session.

```
data(pop)
```

Listing 2: Loading the Austrian population data set

This data set - available in long-format - contains three variables. **time** holds the year, variable **variable** the different age groups and variable **value** the corresponding population numbers. Next, an object is generated that contains all the information that is required to plot a first sparkline. This can be achieved using function `newSparkLine()`. Calling the function and only setting argument *values* to use the population data of Austria stored in vector *pop_ges* as shown in Listing 3, we obtain an new object **sl** with default settings for a sparkline plot.

```
pop_ges <- pop$value[pop$variable=="Insgesamt"]
sline <- newSparkLine(values=pop_ges)
```

Listing 3: Creating a first sparkline object

The resulting object **sline** is of class *sparkline* and has the default parameter settings. Later it is shown how to change specific settings of the graph using the generic function `setParameter`. Note that it is also possible to directly change parameters using `newSparkLine()` directly. The help-files that can be accessed using `?newSparkLine` give an overview on the parameters that can be set or rather changed.

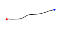
After creating a sparkline object it is then possible to generate a graphic by using function `plotSparks()`. Using this generic function it is possible to plot not only objects of class *sparkline* but also objects of class *sparkBar* (generated with `newSparkBar()`), *sparkBox* (generated with `newSparkBox()`) and *sparkHist* (generated with `newSparkHist()`). `?plotSparks()` required a spark* object as input and allows to specify the desired output format. Currently, one of the following output formats can be selected by changing the function argument *outputType*:

- *pdf:* the spark-graphics is saved in adobe pdf-format;
- *eps:* the graph is saved as an encapsulated postscript file;
- *png:* the graphic is written to disk as a portable network graph.

Using argument *filename* the user can specify an output filename for the resulting graphic. Listing 4 shows how to save the sparkline represented in obj `sline` as a pdf-grafik with filename *first-sl.pdf*.

```
plotSparks(object=sline, outputType="pdf", filename="first-sl")
```

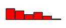
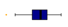
Listing 4: Creating a pdf from a sparkline object

This graph can now easily be included in the current text showing the development  of the Austrian population from 1981 to 2009.

Similarily, other types as graphics can be produced. Listing 5 highlights how to create other supported kind of word-graphs. In addition, some of the default parameters are changed.

```
v1 <- as.numeric(table(rpois(50,2))); v2 <- rnorm(50)
sbar <- newSparkBar(values=v1, barSpacingPerc=5)
sbox <- newSparkBox(values=v2, boxCol=c("black", "darkblue"))
shist <- newSparkHist(values=v2,
  barCol=c("black", "darkgreen", "black"))
```

Listing 5: Creating a sparkbar, a sparkhist and a sparkbox object using simulated data

Since they were exported as *pdf*-files, the produced  sparkbar, the  sparkbox and the sparkhist can be easily integrated into a LaTeX-document as shown here.

Changing parameters of spark objects can also be achieved using function `setParameter()`. In the help-file (accessible via `?setParameter`) all possible values that can be changed are explained and listed. Some of the options that might be changed are now listed below:

- *width:* available vertical space of the graph;
- *height:* available horizontal space of the graph;
- *values:* data points that should be plotted;
- *lineWidth:* the thickness of the line in a sparkline-plot;
- *pointWidth:* the radius of special points (minimum, maximus, last observation) of a sparkline;
- *showIQR:* whether the inter quartile range (IQR) be plotted in a sparkline-graph;
- *outCol:* color of outlying observations in a sparkbox-graph;
- *barSpacingPerc:* space in percent of the total horizontal space used between the bars in a sparkbar-graph.

In Listing 6 we now show how to change some graphical parameters of the spark graphics we have just produced.

```
sl <- setParameter(sline, type="lineWidth", value=5)
sl <- setParameter(sline, type="pointWidth", value=15)
```

Listing 6: changing graphical parameters of sparks

For example the thickness of the line was changed in the sparkline object `sline` from the default value of 1 to 5. Additionally, we changed the size of the points for special values in the sparkline to 15. If this line is plotted, the sparkline showing Austrian population numbers looks different.

It is worth noting that the current settings can be queried using function `getParameter()` in an analogue way as setting parameters by just leaving out the argument *values*. Detailed information about the kind of information that can be extracted using this function is available from the help page that is accessible via `?getParameter`).

3.2 Graphical Tables

While in Section 3.1 the basic usage of the package **sparkTable** to produce sparklines is demonstrated, this section shows how to enrich tables by including spark graphs. In addition it is shown how spark-graphs can be used to visualize regional indicators in so called checkerplots [Templ et al., 2013].

To plot a graphical table, the first step is to create a suitable input object - in this case an object of class *sparkTable*. Objects of this kind contain the data itself in a suitable format as well as a list containing the type of graphs of functions on the values that should be listed in the resulting table. Moreover, a vector of variable names must be listed to specify for which variables in the data input graphs should be produced. These steps can be achieved using function `newSparkTable()`. An example using the Austrian population data is presented in Listing 7.

```
# prepare data
pop <- pop[,c("variable","value","time")]
pop$time <- as.numeric(as.character(pop$time))
dat <- reshapeExt(pop, idvar="variable", varying=list(2))

# prepare content
cc <- list(
  y1981=function(x) { head(x,1) },
  sline=newSparkLine(lineWidth=2, pointWidth=6),
  sbar=newSparkBar(),
  y2009=function(x) { tail(x,1) }
)
vt <- rep("value",4) # set variables

stab <- newSparkTable(dataObj=dat, tableContent=cc, varType=vt)
```

Listing 7: Creation of a sparkTable

The required code to create a *sparkTable* object shown in Listing 7 can be splitted into four sections. In the first three lines the input data set is prepared for analysis. The function `reshapeExt()` is used to transform a data set into long format that is a suitable input for function `newSparkTable()`. Note that the function `reshapeExt()` is an extension to function `reshape()` from the stats package. The next step is to prepare a list specifying the content of each column of the required table. In this case the table is specified by four columns. In the first and last column of the table we apply a function to data extracting the first and last value of the time series for a specific group, respectively. Note that the provided function can be defined by the user and it can be arbitraty complex. For the second column - see the line `sline =newSparkLine(lineWidth=2, pointWidth=6)` – sparklines are created and for the third column sparkbars should be plotted. Therefore we need to set the second and third list-element to objects of the corresponding classes. In Listing 7 it can be seen that it is also possible to change parameters directly. We used this possibility when changing the thickness of the line and the size of points for sparklines as it was already described in Section 3.1. It should be also noted that the names of this input list correspond to column names in the resulting final table. For example, the first column in the final table will have column name *y1981*.

After the object **stab** has been created, the graphical table can be created. The required code is shown in Listing 8.

```
plotSparkTable(object=stab, outputType='tex', filename="first-
    stab", graphNames='first-stab')
```

Listing 8: Generating output of a sparkTable object

The function `plotSparkTable()` is applied on the object **stab** that just has been created in Listings 7. The format of the output can be specified. By setting argument *outputType* to *'tex'*, the functions write LaTeX-code in the file specified with argument *filename*. Additionally, some instructions on how to include the table into a document are printed in the *R*-session. The final result of this example is shown in Table 2.

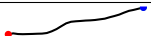
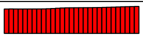
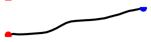
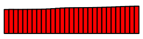
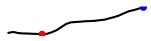
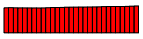
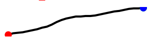
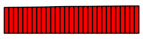
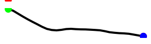
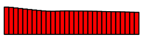
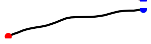
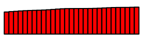
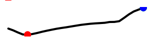
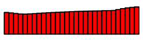
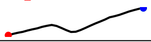
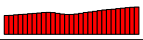
	y1981	sline	sbar	y2009
Insgesamt	7553326			8355260
Maenner	3570172			4068047
Frauen	3983154			4287213
Maenner auf 1.000 Frauen	896			949
0-19 Jahre	2184224			1763948
20-64 Jahre	4212971			5140425
65+ Jahre	1156131			1450887
75+ Jahre	454278			665415

Table 2: A graphical table featuring data values and sparkgraphs applied to Austrian population data.

3.3 Geographical Tables

The concept to create geographical tables is based on the concept of the so called checkerplot [Templ et al., 2013], which is also implemented in the package **sparkTable** in the function `checkerplot()`. Listing 9 shows how to generate a geographical table for the EU using the function `newGeoTable` and `plotGeoTable` and the result is shown in Table 3.

```
  data(popEU,package="sparkTable")
  data(debtEU,package="sparkTable")
  data(coordsEU,package="sparkTable")
  popEU <- popEU[popEU$country%in%coordsEU$country,]
  debtEU <- debtEU[debtEU$country%in%coordsEU$country,]
  EU <- cbind(popEU,debtEU[,-1])
  EUlong <- reshapeExt(EU,idvar="country",v.names=c("pop","debt
      "),
    varying=list(2:13,14:25),geographicVar="country",timeValues
        =1999:2010)
  l <- newSparkLine()
  l <- setParameter(l, 'lineWidth', 2.5)
  content <- list(function(x){"Population:"},l,function(x){"
      Debt:"},l)
  varType <- c(rep("pop",2),rep("debt",2))
  xGeoEU <- newGeoTable(EUlong, content, varType,geographicVar=
      "country",
    geographicInfo=coordsEU)
  plotGeoTable(xGeoEU, outputType="tex", graphNames="out1"
      ,filename="testEUT",
    transpose=TRUE)
```

Listing 9: Creating a geoTable

For the geographical representation in a grid, geographical coordinates have to be provided. Internally, an allocation problem is solved by linear programming to find the nearest free grid cell from the provided coordinates under serveral contraints [Templ et al., 2013], resulting in, for example, Skandinavian countries placed in the north, south mediteranian countries in the south, etc. Again, the sparklines in each grid can be modified by function `setParameter`.

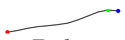
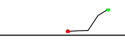
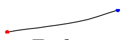
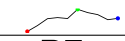
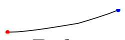
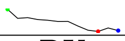
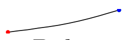
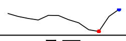
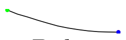
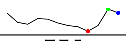
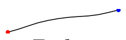
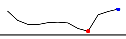
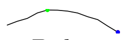
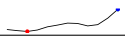
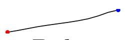
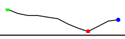
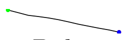
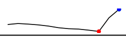
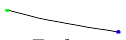
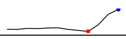
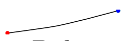
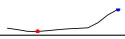
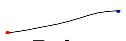
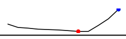
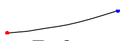
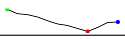
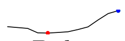
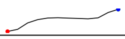
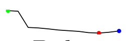
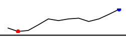
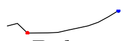
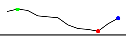
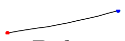
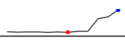
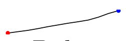
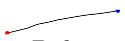
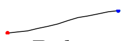
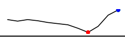
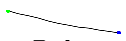
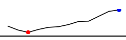
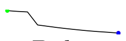
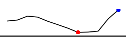
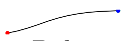
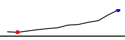
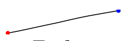
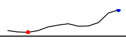
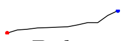
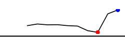
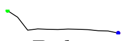
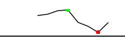
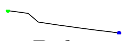
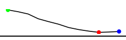
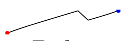
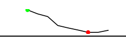
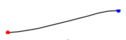
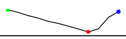
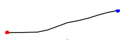
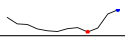
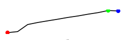
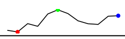
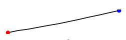
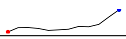
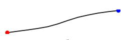
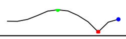
IS *Pop :*  *Debt :* 		**NO** *Pop :*  *Debt :* 	**SE** *Pop :*  *Debt :* 	**FI** *Pop :*  *Debt :* 	**EE** *Pop :*  *Debt :* 
	NL *Pop :*  *Debt :* 	**DE** *Pop :*  *Debt :* 	**DK** *Pop :*  *Debt :* 	**LT** *Pop :*  *Debt :* 	**LV** *Pop :*  *Debt :* 
UK *Pop :*  *Debt :* 	**IE** *Pop :*  *Debt :* 	**BE** *Pop :*  *Debt :* 	**CZ** *Pop :*  *Debt :* 	**PL** *Pop :*  *Debt :* 	**SK** *Pop :*  *Debt :* 
LU *Pop :*  *Debt :* 	**CH** *Pop :*  *Debt :*	**LI** *Pop :*  *Debt :*	**AT** *Pop :*  *Debt :* 	**HU** *Pop :*  *Debt :* 	**RO** *Pop :*  *Debt :* 
PT *Pop :*  *Debt :* 	**FR** *Pop :*  *Debt :* 	**SI** *Pop :*  *Debt :* 	**HR** *Pop :*  *Debt :* 	**BG** *Pop :*  *Debt :* 	**TR** *Pop :*  *Debt :* 
	ES *Pop :*  *Debt :* 	**IT** *Pop :*  *Debt :* 	**MT** *Pop :*  *Debt :* 	**GR** *Pop :*  *Debt :* 	**CY** *Pop :*  *Debt :* 

Table 3: A geographical table featuring sparkgraphs for all EU countries.

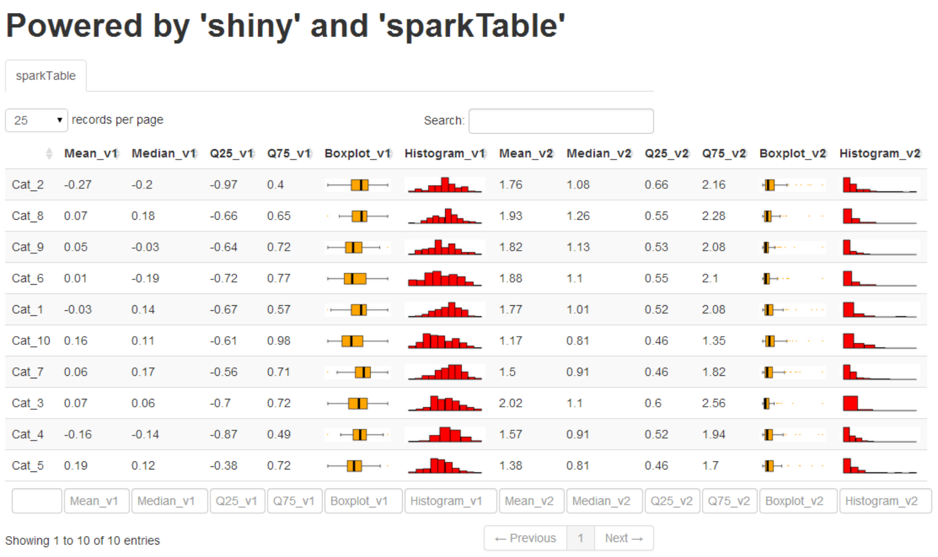
4 Interactive features

Until recently **sparkTable** only provided rigid views of graphical table in documents and on homepages. However, when viewing a graphical table on a homepage some interactivity is benefitional.

Two functions using the functionality from **shiny** [RStudio and Inc., 2014] allows for interactive graphical tables.

Calling the shiny app:
`runShinyApp()` is a function to generate a shiny app to view your graphical table directly in a browser with some basic interactive function like sorting and filtering. At the moment, methods for objects of class `sparkTable` and `data.frame` are implemented.

Figure 1 shows the output of the function from a `sparkTable` object, whereas figure 2 shows the output of a `data.frame` object.



Powered by 'shiny' and 'sparkTable'

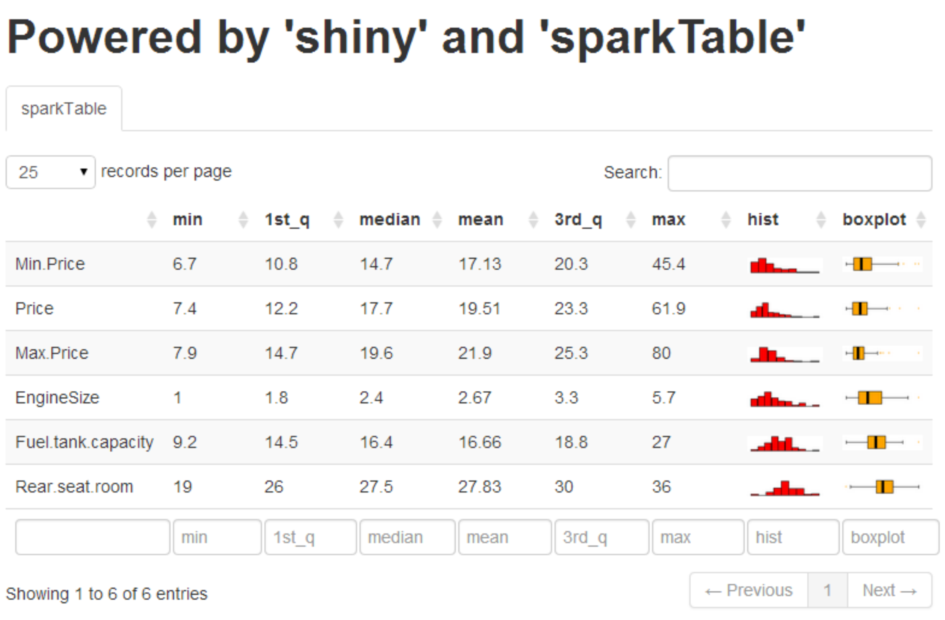
sparkTable

25 records per page — Search:

	Mean_v1	Median_v1	Q25_v1	Q75_v1	Boxplot_v1	Histogram_v1	Mean_v2	Median_v2	Q25_v2	Q75_v2	Boxplot_v2	Histogram_v2
Cat_2	-0.27	-0.2	-0.97	0.4			1.76	1.08	0.66	2.16		
Cat_8	0.07	0.18	-0.66	0.65			1.93	1.26	0.55	2.28		
Cat_9	0.05	-0.03	-0.64	0.72			1.82	1.13	0.53	2.08		
Cat_6	0.01	-0.19	-0.72	0.77			1.88	1.1	0.55	2.1		
Cat_1	-0.03	0.14	-0.67	0.57			1.77	1.01	0.52	2.08		
Cat_10	0.16	0.11	-0.61	0.98			1.17	0.81	0.46	1.35		
Cat_7	0.06	0.17	-0.56	0.71			1.5	0.91	0.46	1.82		
Cat_3	0.07	0.06	-0.7	0.72			2.02	1.1	0.6	2.56		
Cat_4	-0.16	-0.14	-0.87	0.49			1.57	0.91	0.52	1.94		
Cat_5	0.19	0.12	-0.38	0.72			1.38	0.81	0.46	1.7		
	Mean_v1	Median_v1	Q25_v1	Q75_v1	Boxplot_v1	Histogram_v1	Mean_v2	Median_v2	Q25_v2	Q75_v2	Boxplot_v2	Histogram_v2

Showing 1 to 10 of 10 entries — ← Previous 1 Next →

Figure 1: Shiny output of a sparkTable object



Powered by 'shiny' and 'sparkTable'

sparkTable

25 records per page — Search:

	min	1st_q	median	mean	3rd_q	max	hist	boxplot
Min.Price	6.7	10.8	14.7	17.13	20.3	45.4		
Price	7.4	12.2	17.7	19.51	23.3	61.9		
Max.Price	7.9	14.7	19.6	21.9	25.3	80		
EngineSize	1	1.8	2.4	2.67	3.3	5.7		
Fuel.tank.capacity	9.2	14.5	16.4	16.66	18.8	27		
Rear.seat.room	19	26	27.5	27.83	30	36		
	min	1st_q	median	mean	3rd_q	max	hist	boxplot

Showing 1 to 6 of 6 entries — ← Previous 1 Next →

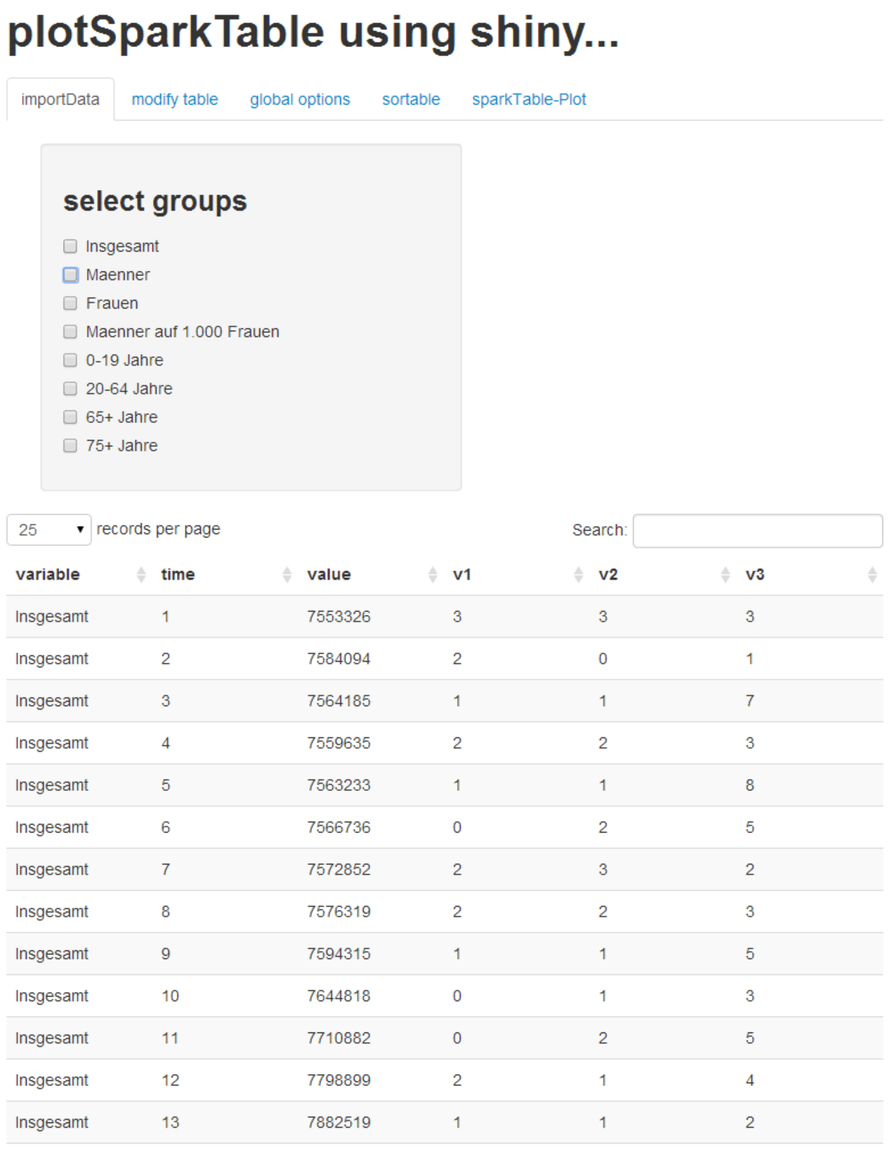
Figure 2: Shiny output of a data.frame object

In the following, the key-features of `shiny_sparkTable()` are discussed.

4.1 Tab 1: import sparkTable objects

The provided app is organized in tabs. In each of the five tabs, specific changes can be made.

Figure shows the first tab - *importData* - that is available to the user when the function is called for the first time. We note that this function must be started with a *sparkTable*-object as input.



plotSparkTable using shiny...

select groups

- ☐ Insgesamt
- ☐ Maenner
- ☐ Frauen
- ☐ Maenner auf 1.000 Frauen
- ☐ 0-19 Jahre
- ☐ 20-64 Jahre
- ☐ 65+ Jahre
- ☐ 75+ Jahre

25 records per page　　Search:

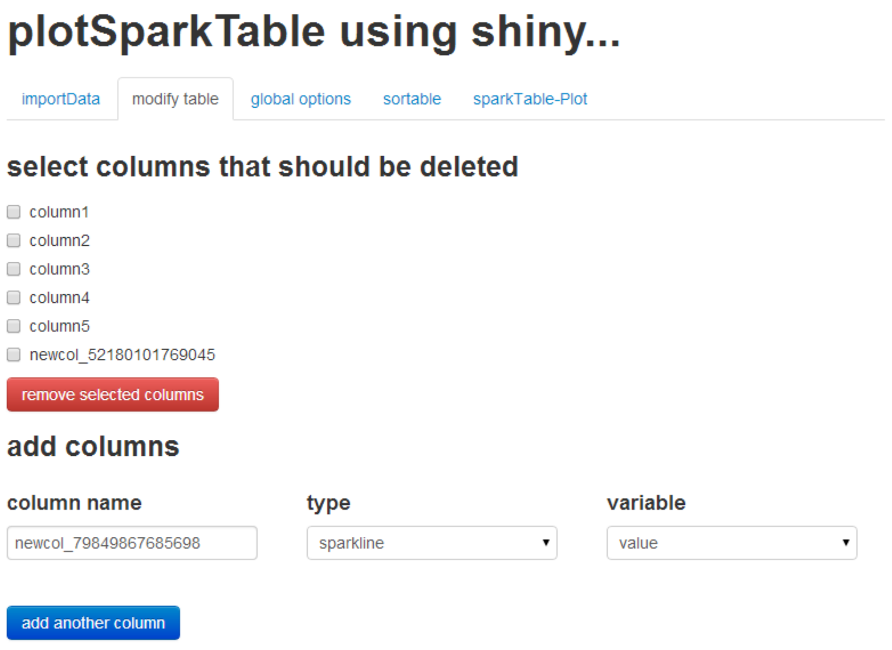
variable	time	value	v1	v2	v3
Insgesamt	1	7553326	3	3	3
Insgesamt	2	7584094	2	0	1
Insgesamt	3	7564185	1	1	7
Insgesamt	4	7559635	2	2	3
Insgesamt	5	7563233	1	1	8
Insgesamt	6	7566736	0	2	5
Insgesamt	7	7572852	2	3	2
Insgesamt	8	7576319	2	2	3
Insgesamt	9	7594315	1	1	5
Insgesamt	10	7644818	0	1	3
Insgesamt	11	7710882	0	2	5
Insgesamt	12	7798899	2	1	4
Insgesamt	13	7882519	1	1	2

Figure 3: Data options applicable for a *sparkTable* object.

In the upper part of this page, all possible groups that are available in the input data object are listed. In the lower part of the page an interactive table is shown. By using the mouse to toggle check-boxes it is possible to choose the groups that should be included in the final output.

4.2 Tab 2: preparation of the table

The second tab - *modify table* - that is shown in Figure 4 is also partitioned into two parts.


plotSparkTable using shiny...

select columns that should be deleted

- ☐ column1
- ☐ column2
- ☐ column3
- ☐ column4
- ☐ column5
- ☐ newcol_52180101769045

add columns

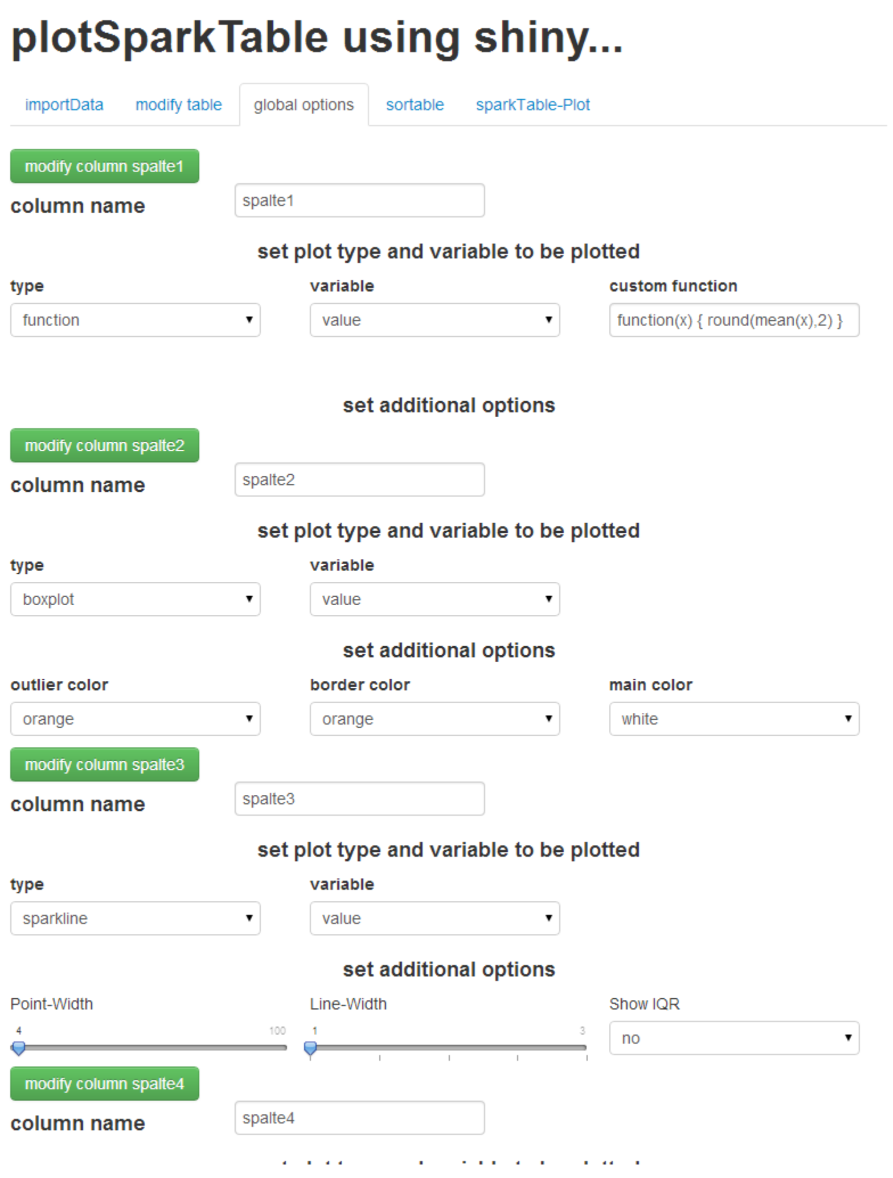
column name	type	variable
newcol_79849867685698	sparkline	value

Figure 4: Possiblities for modifications of the loaded *sparkTable* object

In the upper part, all the columns that are currently present in the output table are listed. It is then possible to select columns that should be deleted. After selecting the corresponding columns by toggling the check boxes and clicking on the *remove selected columns*-button, these columns are dropped from the table. In this tab it is however also possible to add an additional column to the sparkTable. In this case the user needs to provide a column name and can select both the type of the column and the variable that should be used by selecting values from a drop-down menu. For the type of the variable all supported types of graph (*sparkline*, *sparkbar*, *sparkhist* and *sparkbox*) are available as well as the special type *function* that can be used to calculated some statistics from the selected variable. An example would be to calculate the maximum data-value.

4.3 Tab 3: global options

In the *global options*-tab (see Figure 5) it is possible to change the specifics of the current output.



column name: spalte1

set plot type and variable to be plotted

type: function
variable: value
custom function: function(x) { round(mean(x),2) }

set additional options

column name: spalte2

set plot type and variable to be plotted

type: boxplot
variable: value

set additional options

outlier color: orange
border color: orange
main color: white

column name: spalte3

set plot type and variable to be plotted

type: sparkline
variable: value

set additional options

Point-Width: 4 (4–100)
Line-Width: 1 (1–3)
Show IQR: no

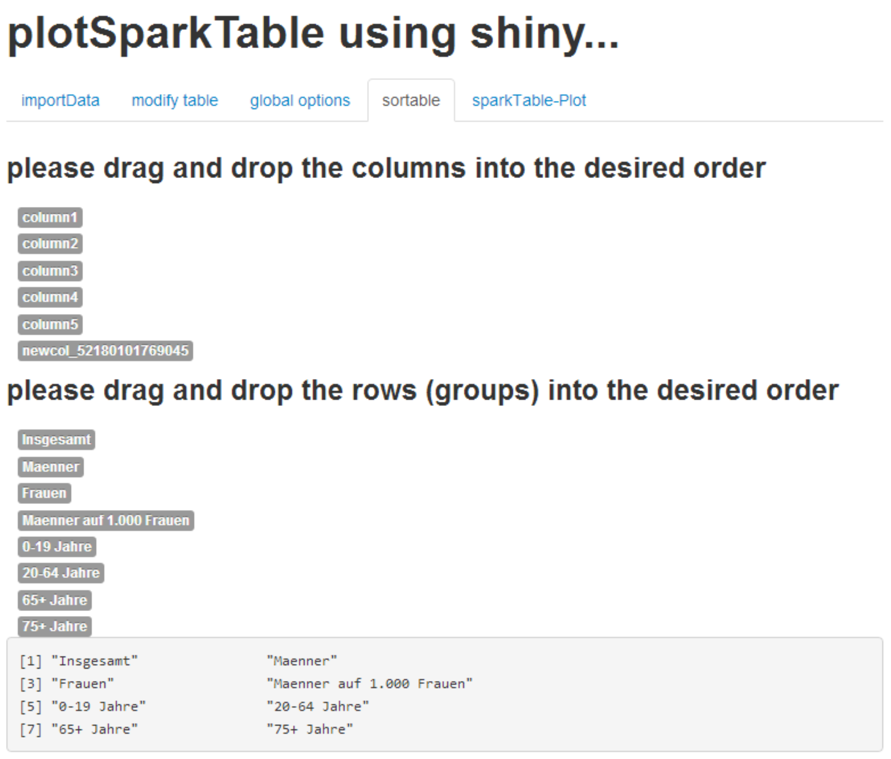
column name: spalte4

Figure 5: Global options

For each column, a block of options is listed. The use has the option to change the column name by modifying the current heading in a text box. It is also possible to change the type of a column by selecting a new type from a drop-down box or to adjust graphical parameters (such as point sizes or line width or colors) for some plots. For columns that are of type *function*, the function definition can be altered. For the adjustment of graphical parameters, suitable inputs such as sliders or drop-down boxes are used. After the desired modification has been done, the user is required to press the modify-button for the currently changed column. The page will refresh and additional changes can be done. We note, however, that is is currently only possible to change one variable after another.

4.4 Tab 4: ordering of columns

In the fourth tab (*sortable*) that is shown in Figure 6 it is possible to interactivly change the ordering of columns (in the first part of the page) and rows (in the second part of the page) by using the mouse to drag boxes around until the desired ordering has been reached.



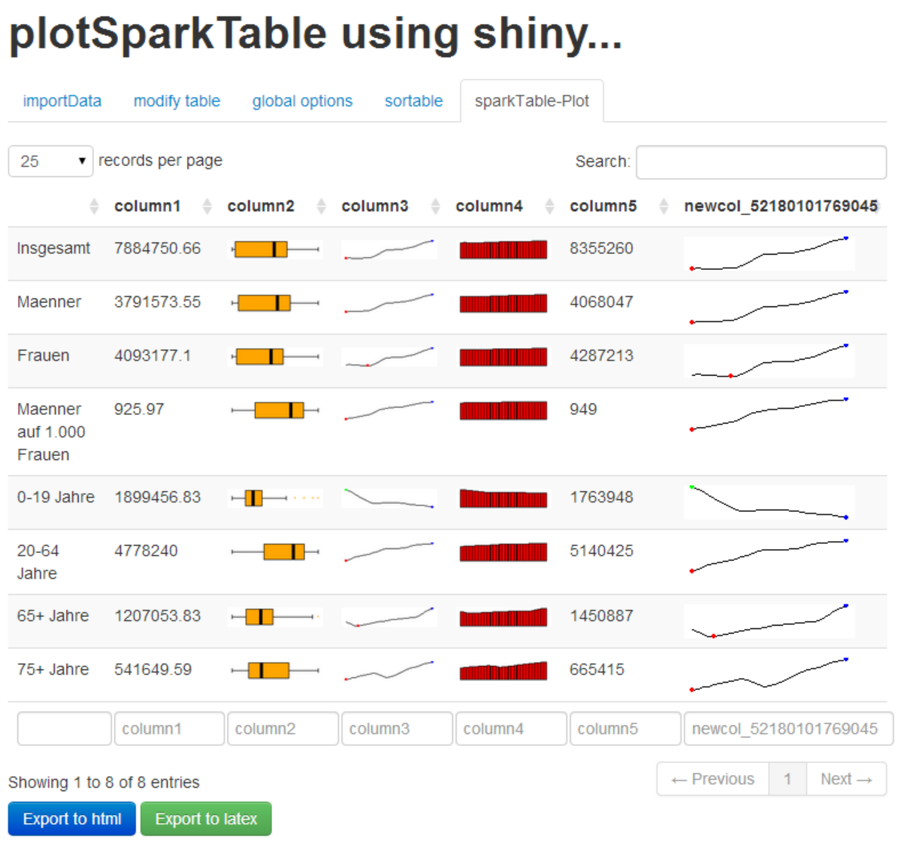
```
[1] "Insgesamt"               "Maenner"
[3] "Frauen"                  "Maenner auf 1.000 Frauen"
[5] "0-19 Jahre"              "20-64 Jahre"
[7] "65+ Jahre"               "75+ Jahre"
```

Figure 6: Sort variables and groupes in the sparkTable

The current result is automatically applied to the sparkTable, that is always visible in the last tab.

4.5 Tab 5: the graphical table

In the tab *sparkTable-Plot* (see Figure 7), the current graphical table with all the current options is displayed. At the bottom of the page the user is presented with two buttons. Using *Export to html*, the graphical table can be exported as an html-file while pressing the *Export to latex*-button exports the final output into tex-format.



	column1	column2	column3	column4	column5	newcol_52180101769045
Insgesamt	7884750.66				8355260	
Maenner	3791573.55				4068047	
Frauen	4093177.1				4287213	
Maenner auf 1.000 Frauen	925.97				949	
0-19 Jahre	1899456.83				1763948	
20-64 Jahre	4778240				5140425	
65+ Jahre	1207053.83				1450887	
75+ Jahre	541649.59				665415	
	column1	column2	column3	column4	column5	newcol_52180101769045

Figure 7: Preview of the output sparkTable

5 Conclusions

The package **sparkTable** provides a flexible and interactive way to produce graphical tables for printed or web publication. Graphical tables have the key advantage to provide more information and insight in the same amount of space. A strict class oriented implementation using S4 classes organise the internal production of graphical tables and sparklines in a pre-defined manner. The users itself have highly costumizable functions available to create sparklines and tables in a flexible manner. The sparklines can be modified easily and the tables can express any statistics provided.

Additionally, interactivity is added with the help of the package **shiny**. Especially these interactive features allows non-experts in *R* to create graphical tables enriched by all kind of sparklines.

The results can be exported to various formats like *html* or $\LaTeX$.

References

O. Bosch and E. de Jonge. Visualising official statistics. *Statistical Journal of the IAOS: Journal of the International Association for Official Statistics*, 25(3):103–116, 2008.

E. Hill. Jmp© 9 add-ins: Taking visualization of sas© data to new heights. In *SAS Global Forum 2011*, 2011.

O. Inbar, N. Tractinsky, and J. Meyer. Minimalism in information visualization: Attitudes towards maximizing the data-ink ratio. In *Proceedings of the 14th European Conference on Cognitive Ergonomics: Invent! Explore!*, ECCE '07, pages 185–188, New York, NY, USA, 2007. ACM. ISBN 978-1-84799-849-1.

Alexander Kowarik, Bernhard Meindl, and Matthias Templ. *sparkTable: Sparklines and graphical tables for tex and html*, 2012. R package version 0.9.7.

J.E. Miller. *The Chicago Guide to Writing about Numbers*. Chicago Guides to Writing, Editing, and. University of Chicago Press, 2004. ISBN 9780226526300.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL `http://www.R-project.org/`.

RStudio and Inc. *shiny: Web Application Framework for R*, 2014. URL `http://CRAN.R-project.org/package=shiny`. R package version 0.9.1.

Matthias Templ, Beat Hulliger, Alexander Kowarik, and Karin Fürst. Combining geographical information and traditional plots: the checkerplot. *International Journal of Geographical Information Science*, 27(4):685–698, 2013.

Edward R. Tufte. *Envisioning Information*. Graphics Press, 1990. ISBN 978-0-9613921-1-6.

Edward R. Tufte. *The Visual Display of Quantitative Information.* 2nd edition, 2001. ISBN 0961392142.

United Nations Economic Commission for Europe. *Making Data Meaningful. Part 2: A Guide to Presenting Statistics*, 2009. ECE/CES/STAT/NONE/2009/3.