SCHUMPETER DISCUSSION PAPERS

Creating Reproducible Publication Quality Graphics with R: A Tutorial

Martin Meermeyer

SDP 2015-006
ISSN 1867-5352

SCHUMPETER DISCUSSION PAPERS 2015-006

Creating Reproducible Publication Quality Graphics with R: A Tutorial

Martin Meermeyer

Abstract

The publication of results of an empirical analysis often includes graphical representations. One of the particular strengths of the statistical computing language and environment R are the graphical capabilities. Beside the relatively simple base graphic system a number of more sophisticated alternative graphic systems are available within R. Compared to these the base graphic system is easy to use but nevertheless highly customizable and therefore advantageous for many research projects. However, even in the base graphic system the number of graphical functions and parameters is large. Due to this it can be hard to find the functions and parameters which must be used to customize a graphic for publication purposes. This paper provides a brief tutorial for this along with a set of simple rules how to structure the code to achieve a sufficient level of reproducibility for a data analysis in general and for associate graphical results in specific. The paper summarizes experiences from a R tutorial hold at the University of Wuppertal for a number of times. The code snippets in the text and the code to reproduce the given examples in the appendix is copy-and-paste-ready and can easily be adopted for own tasks.

Keywords: base graphic system, plots, legends.

1. Introduction

The statistical computing language and environment R (R Core Team 2015) provides three major graphic systems. The *base graphic system* is a static device where graphical objects are arranged by means of a paper and pencil model. Due to this it is very intuitive and easy to learn. More flexible but yet more complicated graphic systems are the *lattice graphic system* and the *grid graphic system* which are described in Sarkar (2008) and Wickham (2009). An exhaustive reference for the base and the grid graphic system is Murrell (2011). Two detailed references to solve technical problems which frequently occur in the preparation of graphics in R are Abedin and Mittal (2014) and Chang (2012). Fundamental concepts on how to turn data and statistics into graphical representations are described in Few (2012) and Robbins (2013). Lewin-Koh (2015) provides an up to date overview of the different graphic systems and graphic related packages which are available in R today. To get an idea of the wide range of possibilities the R Graph Gallery (2015) hosts a large collection of examples from different scientific fields.

Reproducibility is one of the key concepts of scientific research. One approach, which is especially useful if the research contains a data analysis, is *literate programming* where the code of an scientific data analysis is embedded into the corresponding text in the word processor. An extensive introduction into the concepts of reproducible research and literate programming is

Stodden, Leisch, and Peng (2014). With a focus on R and the popular IDE (Integrated Development Environment) RStudio Gandrud (2013) explains the implementation of reproducible research for complex scientific projects using the knitr-architecture introduced by Xie (Xie 2013). At the present knitr is the most sophisticated system of literate programming. A disadvantage of literate programming is a compartively high degree of complexity of the required tools. The acquisition of the skills to handle these tools successfully can be time consuming. For scientific projects in which the methodological part is not dominating and which frequently occur in academics, for instance in seminar, bachelor or master theses, employing such tools is often not efficient. The aim of this paper is to show how a sufficient level of reproducibility can be achieved by employing a few fundamental rules and techniques. The focus is on the reproducible preparation of high quality plots and graphics for publication purposes, i. e. for papers, theses and presentations, using solely functions from the base graphic system.

This paper does not provide an introduction to R and the base graphic system because there already exists a number of high quality resources for that. Therefore it is assumed that the reader is familiar with the basic concepts, for instance with the built-in documentation, the import of data sets, with high-level and low-level plotting commands or how to query or set graphical parameters using the function `par()`. Here only subjects which are not covered in detail elsewhere are addressed. A cross-sectional dataset is used as example throughout this paper. Plotting naturally ordered data, like time series or geographical data, is a topic of its own and will not be discussed.

In the next section it is described how reproducibility can principally be ensured if a data analysis is conducted with R. To exploit the graphical capabilities of R entirely it is essential that the data (notably the categorical variables) are properly coded which is covered in section 3. Full control over the layout of a plot can only be achieved if its fundamental dimensions are defined precisely. In section 4 it is explained how this can be done by using separate graphic windows for each and every plot along with some advices how to deal with these windows in practice. At the end of this section it is explained how the plots can be conveniently incorporated into text processing software in a way that is in accordance with the principle of reproducibility. The usage of the base graphic system is endorsed in this paper. Nevertheless, one disadvantage of this system compared to the more sophisticated graphic systems is the handling of legends. Since this may be a primary reason to switch to another graphic system a workaround to solve this issue is explained in section 5. In section 6 the key aspects are summarized.

2. Ensuring reproducibility

To ensure the reproducibility of the results of a scientific data analysis carried out with R it must be possible to generate the final results without any user interaction. To enforce this principle any R-script should always be executed entirely with `source()` and be starting with the following lines:

```
## Clear R-workspace
rm(list=ls(all=TRUE))

## Set location of R-script as working directory
path.act <- dirname(sys.frame(1)$ofile)
setwd(path.act)

## Close all graphic devices
graphics.off()
```

Removing all objects from the workspace at the beginning in conjunction with the complete execution of the code by `source()` will also help to avoid errors because the execution will fail if something is wrong or missing. If the analysis requires the handling of very large data sets or lengthy computations this can be of course cumbersome. In this case the analysis should be split up to several consecutive scripts and the objects holding the intermediate or final analytical results should be stored in and loaded from `RData`-files using the functions `save()` and `load()`. In the last script the final results are then prepared for publication. If the working directory is set automatically as location of the R-scripts manual path specifications in the scripts are usually unnecessary if the data set or the `RData`-files are stored in the same location as the R-scripts. Note that the code to determine the location of the R-script only works if the entire script is executed with `source()`. The call to close all open graphic devices will be necessary to keep the desktop clean when the graphical output is created in the way recommended in this paper.

If an additional package is needed to conduct an analysis or to prepare the output ,such a package, say **SemiPar**, can be loaded by

```
## Load required package or install it if it is missing
if(!require("SemiPar")) install.packages("SemiPar", dependencies=TRUE)
```

With the expression `require("SemiPar")` the specified package is tried to load. If this is successful the function returns a `TRUE` and a `FALSE` otherwise. In the latter case the expression to install the package is executed. Altogether the specified package is automatically installed if it is not available in the local installation. This ensures that an analysis can be reproduced on a factory fresh R-installation given that the specified package is available online. Note that the automatic installation of packages from a remote resource can principally involve security issues.

3. Preparing the data

One of the most convenient features of R in general and of R graphic systems in particular is the handling of categorical data. To access this capability the categorical variables must be appropriately coded as `factors`. In many data sets found in the social sciences the categorical variables are represented by dummy variables for the different levels of the categorical variables or by numeric values. In both cases it is necessary to recode the data. Recoding the data can be performed within R itself, but it is often much more convenient to conduct this with an arbitrary spreadsheet software and a few column-wise search and replace operations.

To show the benefits of an appropriate coding and for demonstration purposes throughout this paper four variables from a data set regarding the union membership of U.S. workers with 534 observations are employed. The data were originally taken from Berndt (1991) and are available in the package **SemiPar** (Wand 2014). From this data set the numeric variables *wage* (hourly wage in USD) and *age* (age in years) as well as the categorical variables *union.member* (levels: yes, no) and *sector* (levels: manufacturing, construction, other) are used:

```
data("trade.union")
tradeUnion <- trade.union[,c("wage","age","union.member","sector")]
```

In the original data set the categorical variables are coded with numbers, please refer to the documentation of the data set for details. Using `summary()` to calculate the descriptive statistics of the four variables yields:

```
      wage             age         union.member        sector
 Min.   : 1.000   Min.   :18.00   Min.   :0.0000   Min.   :0.0000
 1st Qu.: 5.250   1st Qu.:28.00   1st Qu.:0.0000   1st Qu.:0.0000
 Median : 7.780   Median :35.00   Median :0.0000   Median :0.0000
 Mean   : 9.024   Mean   :36.83   Mean   :0.1798   Mean   :0.2753
 3rd Qu.:11.250   3rd Qu.:44.00   3rd Qu.:0.0000   3rd Qu.:0.0000
 Max.   :44.500   Max.   :64.00   Max.   :1.0000   Max.   :2.0000
```

For the two categorical variables the descriptive statistics are almost meaningless. The only exception is the mean of the variable *union.member*. Since this variable has exactly two levels and the value of 1 represents the membership the mean is the proportion of union members in the sample.

The recoding can be performed with the following two lines:

```
tradeUnion$union.member <- factor(tradeUnion$union.member, labels=c("no","yes"))
tradeUnion$sector <- factor(tradeUnion$sector, labels=c("other","manuf","constr"))
```

Note that the order of the labels must correspond to the order of the numeric coding. After transforming the categorical variables into factors with self-explanatory labels the output from `summary()` for the categorical variables is appropriate:

```
     wage             age        union.member    sector
 Min.   : 1.000   Min.   :18.00   no :438      other :411
 1st Qu.: 5.250   1st Qu.:28.00   yes: 96      manuf : 99
 Median : 7.780   Median :35.00                constr: 24
 Mean   : 9.024   Mean   :36.83
 3rd Qu.:11.250   3rd Qu.:44.00
 Max.   :44.500   Max.   :64.00
```

The order of the factor levels will affect the layout of plots and statistics. Here the order still corresponds to the numbers which were originally used to represent the levels of the categorical variables. For data sets imported from textfiles the levels of the factors are ordered alphabetically. Specifying a particular order for the levels can be achieved by

```
tradeUnion$union.member <- factor(tradeUnion$union.member, levels=c("yes","no"))
tradeUnion$sector <- factor(tradeUnion$sector, levels=c("constr","manuf","other"))
```

In the descriptive statistics, for instance, the different levels now appear in the specified order:

```
      wage              age         union.member    sector    
 Min.   : 1.000   Min.   :18.00   yes: 96      constr: 24  
 1st Qu.: 5.250   1st Qu.:28.00   no :438      manuf : 99  
 Median : 7.780   Median :35.00                other :411  
 Mean   : 9.024   Mean   :36.83                            
 3rd Qu.:11.250   3rd Qu.:44.00                            
 Max.   :44.500   Max.   :64.00                            
```

For bivariate plots an appropriate coding of categorical variables as factors is beneficial as well. With the two fundamental types of variables, numeric and categorical, there are four possible combinations for bivariate plots. The four default plot types are shown in Figure 1 using the four variables mentioned above. The scatterplot in panel (a) is appropriate to plot a numeric variable against another numeric variable. The conditional boxplot in panel (b) is a reasonable graphical representation of a numerical variable plotted against a factor. If a factor is plotted against another variable spineplots are meaningful graphical representations. In panel (c) the default spineplot variant of a factor plotted against a numeric variable is shown. Panel (d) shows the default spineplot variant of a factor plotted against another

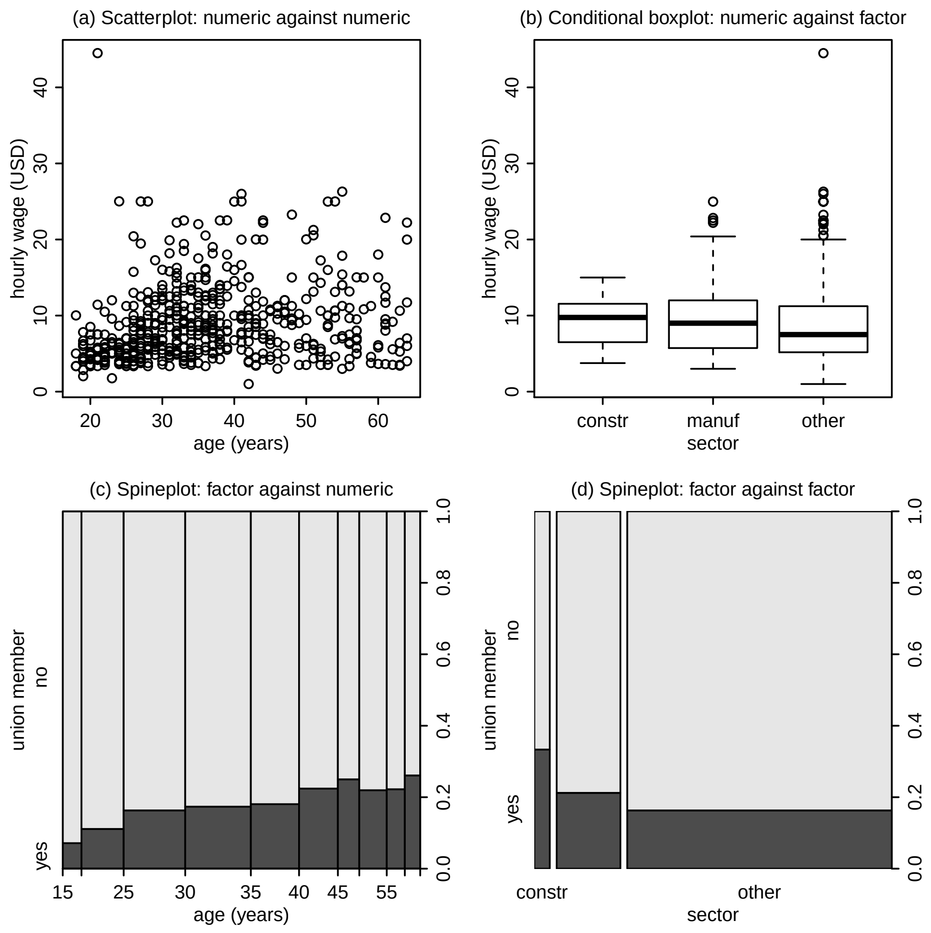

Figure 1: The four standard bivariate plot types for cross-sectional data.

factor. Details on boxplots and spineplots will not be discussed here but can be found in the documentation of the functions `boxplot()` and `spineplot()` and the references cited there. The different plots can essentially be produced with the following 4 lines:

```
plot(wage ~ age,             data=tradeUnion)
plot(wage ~ sector,          data=tradeUnion)
plot(union.member ~ sector, data=tradeUnion)
plot(union.member ~ age,    data=tradeUnion)
```

Obviously the appropriate plot types are chosen automatically. Note that Figure 1 is prepared for printing here, the code to reproduce the figure is provided in section A.3. For the boxplot and the spineplots also the functions `boxplot()` and `spineplot()` could be used alternatively.

4. Controlling plot dimensions and basic font size

For scientific plots visual clarity and parsimony are desirable properties, see Robbins (2013, chap. 6) for more details on this. To achieve this the overall dimensions (size and aspect ratio) and the font size for labels and annotations must be chosen appropriately. The size and aspect ratio depend on the content of the plot and should correspond to the amount and type of information it presents, i. e. the elements of a plot should neither look squeezed nor scattered. The aspect ratio of scatterplots, for instance, is usually chosen as 1:1 whereas time series plots are frequently plotted in a landscape format. In the latter case an aspect ratio that corresponds to the golden ratio $(1+\sqrt{5})/2 \approx 1.618 : 1$ often provides an harmonic impression. For conditional boxplots the width obviously depends on the number of the conditioning factor levels. Hence, in most cases the size and the aspect ratio must be chosen individually based on general principles and rules of thumb. Only for plots which contain continuous lines, for instance termplots, smoothed curves or time series, there exists a data driven way to choose the aspect ratio known as *banking to 45 degrees*. The idea is described in detail in Cleveland, McGill, and McGill (1988), an example illustrating the benefits of this technique can be found in Jacoby (1997, p. 85ff). The disadvantage of this technique is the lack of control over the final size of the plot.

For labels and annotations in plots a font without serifs should be used. The font size should be smaller or at least not larger than the size of the main text in the corresponding document. If the font size itself does not represent an information (as in word clouds) the font sizes in the plot should not vary or should be as small as possible if a variation could not be avoided.

To have full control over the dimensions of a plot and the pointsize for titles, labels and annotations it is beneficial to set these in advance. In the base graphic system this can be easily achieved by opening a separate window with the specified dimensions into which the plot is directly drawn. These separate windows are a specific type of *graphic devices* in R. Searching for `Devices` in the R help gives a list of alternative graphic devices. One of the functions to create a new graphic window is `x11()`, which works properly on the three major platforms Windows, Linux and Mac OS and in conjunction with the popular IDE RStudio. By the following lines the dimensions as well as the pointsize can be specified directly:

```
## Specify fundamental plot dimensions
width.cm  <- 7
height.cm <- 7
pointsize <- 8

x11(width       = width.cm/2.54,
    height      = height.cm/2.54,
    pointsize   = pointsize)
```

For users who run the RGui directly (in MDI- as well as in the SDI-mode) in conjunction with an arbitrary code editor no special attention is needed when separate graphic windows are created. If the IDE RStudio (or any other IDE with an embedded graphic device) is used opening separate graphic windows with `x11()` can be inconvenient with the default settings. The new graphic window will appear in the upper right hand corner of the primary screen. The argument `xpos` of the function `x11()`, which is `-20` by default, is responsible for the horizontal position of the new graphic window on the primary screen. RStudio has a graphic device of its own which is embedded in the window of RStudio. If the RStudio-window is maximized on a single screen the new graphic window is usually covered by the window of RStudio. Every time the script is executed the graphic window must be brought to the top with a click on the corresponding icon in the task bar. This is bothersome and unfortunately there is no other way to bring the graphic device to the top automatically at the present. Depending on the available hardware there are two workarounds for this. On a single screen the size of the RStudio window can be reduced so that the freshly opened device fits on the free space of the screen. With `xpos=-1` the new device will appear at the right margin of the screen. With two screens available and the RStudio-window maximized on the primary screen the new device can be shifted to the secondary screen by setting the argument `xpos` to a value which is larger than the physical horizontal resolution of the primary screen. On Windows-systems this only works if the secondary screen is arranged to the right of the primary screen in the Windows screen management. Combined with the argument `ypos` new devices can be freely arranged on the secondary screen. If a data analysis is carried out on different computers it is convenient to define the default global horizontal position at the beginning of the script as in the code of section A.1.

The standard layout of plots in R is prepared for screen displays. Due to this a lot of space is wasted in terms of area in the plot which is literally empty. For printing purposes especially the margins around the plot region, the length of the axis tickmarks and the perpendicular distance of the axis tickmark labels can be reduced by setting the corresponding parameters with the function `par()`:

```
## Specify global plot parameters
par(mar = c(3, 3, 2, 1),  # Margins
    mgp = c(1.5, 0.5, 0), # Distance of axis tickmark labels (second value)
    tcl = -0.3)           # Length of axis tickmarks
```

For details on the parameters refer to the documentation of `par()`. The unit of measurement for this parameters is “the height of a line of text”. This relative size directly depends on the pointsize specified in the call of `x11()`. Due to the relative sizes the proportions of the plot are preserved even if the dimension of the plot or the pointsize are modified. When the margins are reduced in size the main title of the plot and the axis titles are usually beyond the margins. The perpendicular position of the axis titles can be controlled with the first value of the graphical parameter `mgp`, but for the main title there is no such parameter. This can be considered as a bug in the base graphic system. The low-level graphic function `title()` allows for the exact control of the perpendicular distances and other aspects of the main and axis titles. Alternatively, the function `mtext()` can be used to add main and axis titles to a plot. The latter function is used in the following:

```
## Annotations
main    <- "(a) Scatterplot: numeric against numeric"
label.y <- "hourly wage (USD)"
label.x <- "age (years)"

## Generate plot and add annotations
plot(wage ~ age, data=tradeUnion, xlab=NA, ylab=NA)
mtext(main,    side=3, line=0.5)
mtext(label.y, side=2, line=1.5)
mtext(label.x, side=1, line=1.5)
```

Since the three titles are added separately the perpendicular positions of these can be set independently from each other using the argument `line`. Note that the default axis titles must be suppressed here by setting these to `NA` in the call of `plot()`.

The effects of the graphical parameters are illustrated in Figure 2. The left hand side shows the scatterplot from Figure 1 (a) which was plotted into a single quadratic device with a side length of 7 cm and a pointsize of 8 using the default graphical parameters and the standard arguments `main`, `xlab` and `ylab` for the main and axis titles in `plot()`. The scatterplot on the right hand side was plotted into an identical device with the graphical parameters and functions used in the two preceding code snippets. To highlight the distances to the margins a frame around each plot was added here. Obviously the plot on the right hand side is appropriate for printing whereas the left is not. The code to reproduce the figures can be found in section A.4.

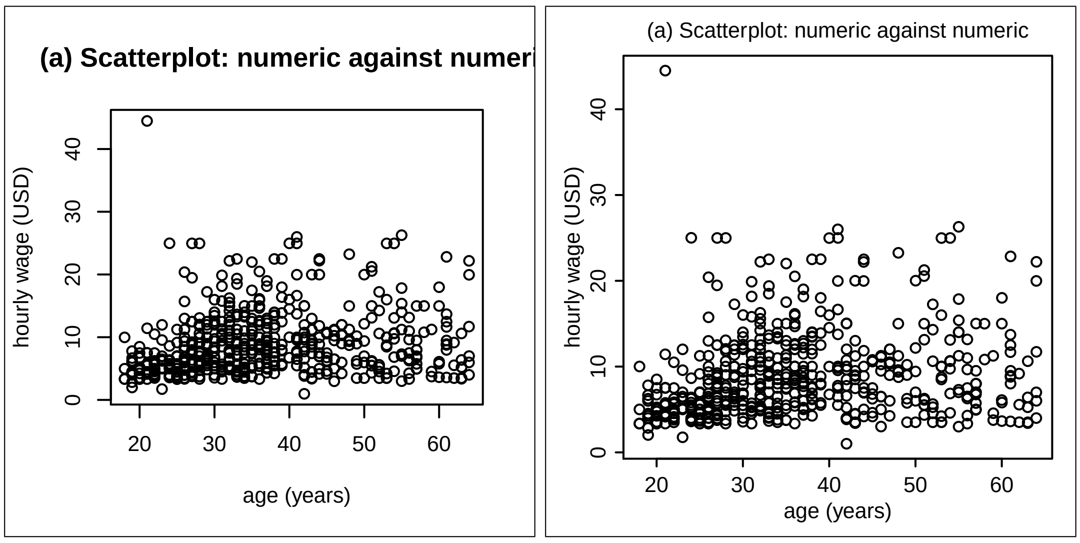

Figure 2: A scatterplot with standard and adjusted graphical parameters.

To insert the plots into a document these must be stored as graphic files. The appropriate function depends on the operating system and the desired graphics format. On Windows-systems the function `savePlot()` can be directly used to save the content of a graphic window in an arbitrary format, refer to the documentation of this function for details:

```
## Specify filename/filetype and save plot
filename  <- "fig-2x2-expl"
filetype  <- "pdf"
savePlot(filename=filename, type=filetype)
```

On Linux- and Mac OS-machines `savePlot()` can only be used to store a plot in a raster graphics format like PNG, BMP or JPG. For PDF-files the function `dev.copy2pdf()` and for EPS-files the function `dev.copy2eps()` must be used on these systems. For the sake of clarity it is reasonable to shift the code lines with the definition of the filename and filetype to the top of the code. The examples code in the appendix is structured like this.

Since the plots are stored separately these can usually be incorporated into the document by the text processing software automatically. By this every change in the plots due to any kind of modification in the analysis itself or in the preparation of the plots will directly become visible within the document. This mechanism ensures the reproducibility of the graphical results of an analysis. If the document preparation system LaTeX is used it is convenient to print the code which incorporates the figure into the document directly to the console. When the name of the figure is specified in advance it can be used as filename as well as label for the figure:

```
## LaTeX-Code to include the figure
cat(fill=TRUE)
cat("\\begin{figure}[ht]", fill=TRUE)
cat("\\centering", fill=TRUE)
cat("\\includegraphics{",filename,"}", sep="", fill=TRUE)
cat("\\caption{XXX}", sep="", fill=TRUE)
cat("\\label{", filename, "}", sep="", fill=TRUE)
cat("\\end{figure}", fill=TRUE)
cat(fill=TRUE)
```

For the text processor Microsoft Word it is convenient to incorporate compatible graphic files with the field function `INCLUDEPICTURE`. An empty field can be added at the cursor position by pressing `CTRL+F9` and the required field command which needs to be inserted into it can be printed to the console:

```
## Microsoft Word field function to include the figure
cat(fill=TRUE)
cat("INCLUDEPICTURE \".\\\\", filename, ".", filetype,
    "\" \\d \\x \\y", sep="", fill=TRUE)
cat(fill=TRUE)
```

After pasting the field command into the empty field it must be refreshed by pressing `F9`. The plot will directly appear if the field values are set to be visible in the document. If the document is in the mode to show the field functions instead of the field values the mode must be switched by pressing `ALT+F9` to make the plot visible.

5. Adding separate legends to plots

The base graphic system is often sufficient to prepare the graphics from a scientific analysis. Compared to the more sophisticated graphic systems there are nevertheless some shortcomings. One of these is the handling of legends. Compared to the lattice graphic system the legends in the base graphic system are intended to be printed within the plot itself by the low-level graphic function `legend()`. For many types of plots this is not a satisfying solution since there may be not enough space for a legend or it may be necessary to check that the legend does not cover other graphical objects. Therefore, in this section it is described how to place a legend separately by using solely functions from the base graphic system. For illustration the scatterplot of the wage against the age of the workers (see Figure 1 (a)) is augmented by the information regarding the affiliation to a sector. The affiliation is represented with different characters and the definition of the characters is provided in terms of a legend. The resulting plot is shown in Figure 3. The complete code to reproduce this figure is given in section A.5. In the following only the key parts of the code are explained in detail.

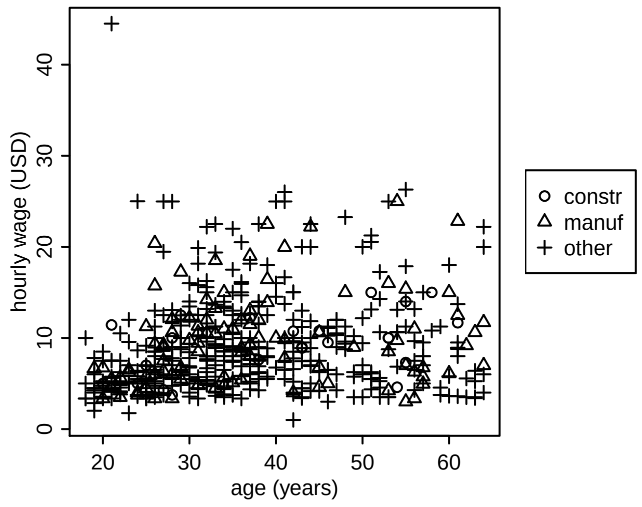

Figure 3: Legend added to a separate region

To add a separate legend firstly the entire space of the graphic window is divided into two regions using the function `layout()`. The first region is for the plot itself and the second for the legend. With this function the graphic window can be flexibly split into different regions which can be addressed separately by arbitrary high-level plotting commands. The basic usage is described in the documentation of `layout()` and in detail in Murrell (2011, section 3.3.2). In conjunction with predefining the size of a graphic window it is problematic to specify the region dimensions in absolute widths with `layout()`. If, for instance, the graphic window is 9 cm in width providing the arguments that the device should be split into two regions with widths of 7 and 2 cm produces an error. To avoid this, the specification of the region widths in the call of `layout()` must be reduced by approximately 0.1 cm. Corrections in absolute units like this must be adjusted each time when the size of the graphic window is modified and are therefore inconvenient. In the following code the width of the region for the legend is specified in centimeters first and then translated into a relative layout specification:

```
## Divide device into 2 regions for the plot (1) and the legend (2)
width.legend.cm <- 2
fraction.legend <- round((width.legend.cm/width.cm)*100, digits=0)
layoutvec       <- rep(c(1,2), times=c(100 - fraction.legend, fraction.legend))
layoutmatrix    <- matrix(layoutvec, 1, 100)
layout(layoutmatrix)
```

First the relative width of the region for the legend is expressed by a percentage integer value. This value is used to build a vector which represents the horizontal division of the graphic window and this vector is transformed into a matrix which is necessary to pass it to `layout()`. Reverting the order of `layoutvec` by

```
layoutvec <- rev(layoutvec)
```

will yield the legend to be placed to the left of the plot. It is reasonable to shift the code line with the definition of the legend width to the top of the code. This is done in the code to reproduce Figure 3 in section A.5.

By calling the function `layout()` the graphical parameter for the overall font magnification `cex` is set to 0.66. To maintain the original font size this parameter has to be set back to a value of 1 after the call of `layout()`. Graphical parameters which are set by `par()` affect each single region defined by `layout()`. Since graphical objects which have already been added to the plot remain unchanged due to the underlying paper and pencil model of the base graphic system, the graphical parameters controlled by `par()` can be set for each single region separately. Altogether `par()` should be called after `layout()` and before the first high-level plotting command. If changes of graphical parameters are necessary in subsequent regions `par()` must be called prior to the corresponding high-level plotting commands.

To add the legend to a separate region via the low-level plotting command `legend()` a high-level plotting command must be called beforehand to address the second region. This can be done by plotting an arbitrary single point with plot type `"n"` (nothing). To keep the region empty the axes and axis titles must also be suppressed in `plot()`. Before this dummy plot is produced the left and right margin of the second region are set to 0. After that the legend can be added to the dummy plot as usual:

```
## Add legend in region 2
par(mar=c(3,0,1,0))
plot(x=1, y=1, type="n", axes=FALSE, xlab=NA, ylab=NA)
legend(x="left", pch=c(1:3), legend=levels(tradeUnion$sector))
```

Here, the legend is left aligned and vertically centered in reference to the box around the plot. To achieve this the top and bottom margin of the legend region must be identical to those of the plot region. With two additional lines of code this can be done automatically, in section A.6 the code for this is provided.

6. Summary

Reproducibility is a key feature of any scientific data analysis. Using the techniques described in this paper reproducibility can be guaranteed for the analysis itself as well as for the graphical results. A technique using single graphic windows for each new plot is advocated here to control the overall dimensions and the layout of plots precisely. The plots are produced solely with functions from the base graphic system of R which is considered to be sufficient for many problems. The advantage of the base graphic system in comparison to the more complex graphic systems is its simplicity due to the underlying paper and pencil model. A particular disadvantage of the base graphic system is the handling of legends which can be circumvented as described in the last section. Using the copy-and-paste ready code in the appendix the figures shown in this paper can be reproduced directly. At the same time it is easy to adopt and modify this code for own tasks.

References

Abedin J, Mittal HV (2014). *R Graphs Cookbook.* 2nd edition. Packt Publishing Ltd.

Berndt ER (1991). *The Practice of Econometrics: Classic and Contemporary.* Addison-Wesley, Reading, MA.

Chang W (2012). *R Graphics Cookbook.* O'Reilly Media.

Cleveland WS, McGill ME, McGill R (1988). "The Shape Parameter of a Two-Variable Graph." *Journal of the American Statistical Association*, **83**(402), 289–300.

Few S (2012). *Show Me the Numbers: Designing Tables and Graphs to Enlighten.* 2nd edition. Analytics Press.

Gandrud C (2013). *Reproducible Research with R and R Studio.* CRC Press.

Jacoby WG (1997). *Statistical Graphics for Univariate and Bivariate Data.* Sage.

Lewin-Koh N (2015). *CRAN Task View: Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization.* URL `http://cran.r-project.org/web/views/Graphics.html`.

Murrell P (2011). *R Graphics.* 2nd edition. CRC Press.

R Core Team (2015). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org/`.

R Graph Gallery (2015). URL `http://rgraphgallery.blogspot.de/`.

Robbins NB (2013). *Creating More Effective Graphs.* Chart House.

Sarkar D (2008). *Lattice: Multivariate Data Visualization with R.* Springer Science & Business Media.

Stodden V, Leisch F, Peng RD (2014). *Implementing reproducible research.* CRC Press.

Wand M (2014). *SemiPar: Semiparametic Regression*. R package version 1.0-4.1, URL http://CRAN.R-project.org/package=SemiPar.

Wickham H (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer Science & Business Media.

Xie Y (2013). *Dynamic Documents with R and knitr*. CRC Press.

A. Code Examples

To facilitate the copy-and-paste of code the headline is suppressed in the appendix. After pasting the code into an editor or IDE the white spaces may be lost and with these the indentions and vertical alignments. To avoid redundancies the code is organized in blocks. The code in the sections A.1 and A.2 must always be executed before using the code to reproduce the figures in the sections A.3, A.4 and A.5. The code in subsection A.6 can be used to replace the corresponding block of code in subsection A.5.

A.1. Basic header

```
## Clear R-workspace
rm(list=ls(all=TRUE))

## Set location of R-script as working directory
path.act <- dirname(sys.frame(1)$ofile)
setwd(path.act)

## Close all graphic devices
graphics.off()

## Standard horizontal graphic window position
global.xpos <- -20
```

A.2. Code to prepare the data set

```
## Load required package or install it if it is missing
if(!require("SemiPar")) install.packages("SemiPar", dependencies=TRUE)

## Load data set and select variables
data("trade.union")
tradeUnion <- trade.union[,c("wage","age","union.member","sector")]

## Build factors
tradeUnion$sector       <- factor(tradeUnion$sector, labels=c("other","manuf","constr"))
tradeUnion$union.member <- factor(tradeUnion$union.member, labels=c("no","yes"))

## Reorder factor levels
tradeUnion$sector       <- factor(tradeUnion$sector, levels=c("constr","manuf","other"))
tradeUnion$union.member <- factor(tradeUnion$union.member, levels=c("yes","no"))
```

A.3. Code to reproduce Figure 1

```
## Specify filename, filetype and fundamental plot dimensions
filename  <- "fig-2x2-expl"
filetype  <- "pdf"
width.cm  <- 14
height.cm <- 14
pointsize <- 8
```

```
## Create new graphic window
x11(width       = width.cm/2.54,  # width in inch
    height      = height.cm/2.54, # height in inch
    pointsize   = pointsize,      # pointsize in big points (1/72 inch)
    xpos        = global.xpos)    # horizontal position of device

## Split window into four independent plot regions
layoutmatrix <- matrix(c(1,2,
                         3,4), ncol=2, byrow=TRUE)
layout(layoutmatrix)

## Specify plot parameters for actual region
par(cex = 1,            # global font magnification
    mar = c(3,3,2,2),   # margins
    mgp = c(1.5,0.5,0), # distance of axis tickmark labels (2nd)
    tcl = -0.3)         # length of axis tickmarks

#### Scatterplot ####
## Annotations
main    <- "(a) Scatterplot: numeric against numeric"
label.y <- "hourly wage (USD)"
label.x <- "age (years)"

## Plot and annotations
plot(wage ~ age, data=tradeUnion, xlab=NA, ylab=NA)
mtext(main,    side=3, line=0.5)
mtext(label.y, side=2, line=1.5)
mtext(label.x, side=1, line=1.5)

#### Conditional Boxplot ####
## Annotations
main    <- "(b) Conditional boxplot: numeric against factor"
label.y <- "hourly wage (USD)"
label.x <- "sector"

## Plot and annotations
plot(wage ~ sector, data=tradeUnion, xlab=NA, ylab=NA)
mtext(main,    side=3, line=0.5)
mtext(label.y, side=2, line=1.5)
mtext(label.x, side=1, line=1.5)

#### Spineplot ####
## Annotations
main    <- "(c) Spineplot: factor against numeric"
label.y <- "union member"
label.x <- "age (years)"

## Plot and annotations
plot(union.member ~ age, data=tradeUnion, xlab=NA, ylab=NA)
mtext(main,    side=3, line=0.5)
mtext(label.y, side=2, line=1.5)
mtext(label.x, side=1, line=1.5)
```

```
#### Spineplot ####
## Annotations
main    <- "(d) Spineplot: factor against factor"
label.y <- "union member"
label.x <- "sector"

## Plot and annotations
plot(union.member ~ sector, data=tradeUnion, xlab=NA, ylab=NA)
mtext(main,    side=3, line=0.5)
mtext(label.y, side=2, line=1.5)
mtext(label.x, side=1, line=1.5)

## Save plot in the current working directory
savePlot(filename=filename, type=filetype)
```

A.4. Code to reproduce Figure 2

```
#### Standard design and titles with standard interfaces ####

## Specify filename, filetype and fundamental plot dimensions
filename  <- "fig-comp-standard-graph-1"
filetype  <- "pdf"
width.cm  <- 7
height.cm <- 7
pointsize <- 8

## Create new graphic window
x11(width       = width.cm/2.54,  # width in inch
    height      = height.cm/2.54, # height in inch
    pointsize   = pointsize,      # pointsize in big points (1/72 inch)
    xpos        = global.xpos)    # horizontal position of device

## Annotations
main    <- "(a) Scatterplot: numeric against numeric"
label.y <- "hourly wage (USD)"
label.x <- "age (years)"

## Plot and annotations
plot(wage ~ age, data=tradeUnion, main=main, xlab=label.x, ylab=label.y)

## Save plot in the current working directory
savePlot(filename=filename, type=filetype)

#### Parsimonious design and titles with mtext() ####

## Specify filename, filetype and fundamental plot dimensions
filename  <- "fig-comp-standard-graph-2"
filetype  <- "pdf"
width.cm  <- 7
height.cm <- 7
pointsize <- 8
```

```
## Create new graphic window
x11(width       = width.cm/2.54,  # width in inch
    height      = height.cm/2.54, # height in inch
    pointsize   = pointsize,      # pointsize in big points (1/72 inch)
    xpos        = global.xpos)    # horizontal position of device


## Specify plot parameters for actual region
par(cex = 1,            # global font magnification
    mar = c(3,3,2,1),   # margins
    mgp = c(1.5,0.5,0), # distance of axis tickmark labels (2nd)
    tcl = -0.3)         # length of axis tickmarks

## Plot and annotations
plot(wage ~ age, data=tradeUnion, xlab=NA, ylab=NA)
mtext(main,    side=3, line=0.5)
mtext(label.y, side=2, line=1.5)
mtext(label.x, side=1, line=1.5)

## Save plot in the current working directory
savePlot(filename=filename, type=filetype)
```

A.5. Code to reproduce Figure 3

```
## Specify filename, filetype and fundamental plot dimensions
filename        <- "fig-legend"
filetype        <- "pdf"
width.cm        <- 9
width.legend.cm <- 2
height.cm       <- 7
pointsize       <- 8

## Create new graphic window
x11(width       = width.cm/2.54,  # width in inch
    height      = height.cm/2.54, # height in inch
    pointsize   = pointsize,      # pointsize in big points (1/72 inch)
    xpos        = global.xpos)    # horizontal position of device

## Divide device into 2 regions: 1) plot, 2) legend
fraction.legend <- round((width.legend.cm/width.cm)*100, digits=0)
layoutvec       <- rep(c(1,2), times=c(100 - fraction.legend, fraction.legend))
#layoutvec      <- rev(layoutvec) # Legend on the left
layoutmatrix    <- matrix(layoutvec, 1, 100)
layout(layoutmatrix)

## Specify plot parameters for actual region
par(cex = 1,            # global font magnification
    mar = c(3,3,1,1),   # margins
    mgp = c(1.5,0.5,0), # distance of axis tickmark labels (2nd)
    tcl = -0.3)         # length of axis tickmarks

## Annotations
main    <- NA
label.y <- "hourly wage (USD)"
label.x <- "age (years)"
```

```
## Plot and annotations in region 1
plot(wage ~ age, data=tradeUnion, pch=as.numeric(tradeUnion$sector), xlab=NA, ylab=NA)
mtext(main,    side=3, line=0.5)
mtext(label.y, side=2, line=1.5)
mtext(label.x, side=1, line=1.5)

## Add legend in region 2
par(mar=c(3,0,1,0))
plot(x=1, y=1, type="n", axes=FALSE, xlab=NA, ylab=NA)
legend(x="left", pch=c(1:3), legend=levels(tradeUnion$sector))

## Save plot in the current working directory
savePlot(filename=filename, type=filetype)
```

A.6. Code to set vertical margins of the legend region automatically

```
## Add legend in region 2
old.mar <- par("mar")
par(mar=c(old.mar[1],0,old.mar[3],0))
plot(x=1, y=1, type="n", axes=FALSE, xlab=NA, ylab=NA)
legend(x="left", pch=c(1:3), legend=levels(tradeUnion$sector))
par(mar=old.mar)
```

Affiliation:

Martin Meermeyer
Institut für Empirische Wirtschafts- und Sozialforschung
Westfälische Hochschule - Campus Bocholt
Münsterstraße 265
46397 Bocholt, Germany
E-mail: m.meermeyer@gmail.com