# DescTools

## A Hardworking Assistant for Descriptive Statistics

## <preliminary blueprint version>

by Andri Signorell

Helsana Versicherungen AG, Health Sciences, Zurich
HWZ University of Applied Sciences in Business Administration, Zurich

andri@signorell.net

November, 5th, 2016

R sometimes makes ordinary tasks difficult. Virtually every data analysis project starts with describing data. The first thing to do will often be calculating summary statistics for all variables while listing the occurrence of nonresponse and missing data and producing some kind of graphics. This is a three-click process in SPSS, but regardless of the normality of this task, base R does not contain higher level functions for quickly describing huge datasets (meant regarding the number of variables, not records) adequately in a more or less automated way. Sure, there are facilities like summary (base), describe (Hmisc), stat.desc (library pastecs), but all of them are lacking some functionality or flexibility we would have expected. What we in particular missed ever since is a combination of numerical and graphical description of data.

R comes with several functions for computing summary statistics, including mean, var, median, range and others. But then there are quite a few commonly used functions, which curiously are missing in the stats package, think of e.g. skewness, kurtosis but also the Gini-coefficent, Cohen's Kappa or Somers' delta. This led to a rank growth of libraries implementing just one specific missing thing. There are plenty of "misc"-libraries out there, containing these functions and tests. We would normally end up using a dozen libraries, each time using just one single function out of it and suffering huge variety concerning NA-handling, recycling rules and so on.
R has been developed in a university environment. This will be clear at the latest then when you find yourself working in a corporate environment, where Word document format is pretty ubiquitous and you realize that only MS-Office (and no LATEX) is installed on your system (and the IT guys won't give you admin rights). We were forced in this situation to write code for doing our reporting in MS-Word. (This works quite well for Windows, but not for Mac unfortunately.)
The first version of "DescTools" arose after completion of a project, where we had to describe a dataset under deadline pressure, and we started to gather our newly created functions and put them together.
This collection has meanwhile grown to a considerably versatile toolset for descriptive statistics, providing rich univariate and bivariate descriptions of data without expecting the user to say much.
There are numerous basic statistic functions and tests, possibly flexible and enriched with different approaches (if existing). Confidence intervals are extensively provided.

Recognizing that most problems can be satisfactorily visualized with bar-, scatter- and dotplots, still some more specific plot types are used in special cases and thus included in the library. Some of them are rather new, and some of them are based on types found scattered in the myriads of R packages found out there (partly rewritten to meet the design goals of the package).

The aim of this document is to show how data description can be accomplished with relative ease compared to the standard R interface.

Users, even expert statisticians, do not always screen the data.

B. D. Ripley, Robust statistics (2004)

# 1 Introduction

The analyst's sacred duty before beginning any sort of statistical analysis is to take a preliminary look at the data with three main goals in mind: first, to check for errors and anomalies; second, to understand the distribution of each of the variables on its own; and third, to begin to understand the nature and strength of relationships among variables.

Errors should, of course, be corrected, since even a small percentage of erroneous data values can drastically influence the results and might completely invalidate the analysis. Understanding the distribution of the variables, especially the outcomes, is crucial to choosing the appropriate multipredictor regression method. Finally, understanding the nature and strength of relationships is the first step in building a more formal statistical model from which to draw conclusions.

To prevent the analyst to bypass these steps the describing process must be quick and simple. So the principal goal of DescTools is to make data description easier, less costly, less time consuming and less error-prone. One outstanding feature of the package is the combination of numerical results and graphical representation which can mostly be automated and reported to the console, but as well quite easily be exported to a Word Document.

The proper description of data depends on the nature of the measurement. The key distinction for statistical analysis is between numerical and categorical variables. The temperature of the pizza is a numerical variable, while the driver delivering it is categorical. The delivery time is numerical, whereas the area of the customer is categorical. A secondary but sometimes important distinction within numerical variables is whether the variable can take on a whole continuum or just a discrete set of values. So the temperature would be continuous, while number of pizzas ordered (count) would be discrete.

A numerical variable taking on a continuum of values is called continuous and one that only takes on a discrete set of values is called discrete. A secondary distinction sometimes made with regard to categorical variables is whether the categories are ordered or unordered. So, for example, categories of quality (low, medium, high) would be ordered, while the operator would be unordered.

A categorical variable is ordinal if the categories can be logically ordered from smallest to largest in a sense meaningful for the question at hand (we need to rule out silly orders like alphabetical); otherwise it is unordered or nominal. Some overlap between types is possible. For example, we may break a numerical variable (such as exact total amount) into ranges or categories. Conversely, we may treat a categorical variable as a numerical score, for example, by assigning values one to three to the ordinal responses Low, Medium, High. Most of the basic analysis methods for numerical scores (e.g., linear regression or t-tests) have interpretations based on average scores. So assigning scores to a categorical variable is effective if average scores are readily interpretable. [3]

A describing procedure has to take all these types and properties into account. The function Desc has been designed to describe variables depending on their type with some reasonable statistic measures and an adequate graphic representation. It includes code for describing logical variables, factors (ordered and unordered), integer variables (typically counts), numeric variables, dates and tables and matrices.

Data frames will be split into their variables and the single variable will be described. A formula interface is implemented to easily describe variables in dependence of others.

The output can either be sent to the R-console or as well directly redirected to a MS-Word document.
The latter works only in Windows with MS-Office installed, but Mac users can leave the wrd argument away and add a plotit = TRUE argument to have the full results in the console.

Note: For all the following examples in this document, `library(DescTools)` must be declared.

## 2 Categorical Variables

The first variable to be described is an unordered `factor`. Factors are typically best detailed by a frequency table of their levels. The default order of the output table is following a pareto rule, the most frequent levels first.
Ordered factors would be sorted after their natural order by default. The default order can be changed by setting the `ord` argument to either `"desc"` (for descending frequency order), `"asc"` (ascending order), `"name"` (alphabetical order) or `"level"` (order of the levels).

Factors sometimes tend to have lots of levels. Listing all of them might not be informative. Thus the frequency table is by default truncated in the case that there are more than a dozen values. This can be avoided by setting the argument `maxrows=Inf`. The same argument can also be used to list either only a defined number of levels by setting `maxrows` to the desired number or restricting the maximum number by defining the maximum cumulative percentage. If e.g. `maxrows=0.7` is set, then as much levels will be displayed as are needed to just exceed the cumulative percentage of 70%.

The number formats are controlled by the `options` "`fmt.abs`" and "`fmt.perc`". These formats define the representation of the counts and of the percentages. For getting the following results, the options must be set to:

```
options(fmt.abs=structure(list(digits=0, big.mark="'"), class="fmt"))
options(fmt.per=structure(list(digits=5, leading="drop"), class="fmt"))
```

The argument `plotit` can be set to produce a plot in one step. This value can also be defined as `options(plotit=TRUE)` in order to save the pains to request it every time.
(See: `?DescToolsOptions` and `?Desc.factor` for more details).

```
Desc(d.pizza$driver, plotit=TRUE)
```

**Desc**

```
  length      n    NAs levels unique  dupes
   1'209  1'204      5      7      7      y

      level freq  perc cumfreq cumperc
1 Carpenter  272  .226     272    .226
2    Carter  234  .194     506    .420
3    Taylor  204  .169     710    .590
4    Hunter  156  .130     866    .719
5    Miller  125  .104     991    .823
6    Farmer  117  .097   1'108    .920
7   Butcher   96  .080   1'204   1.000
```

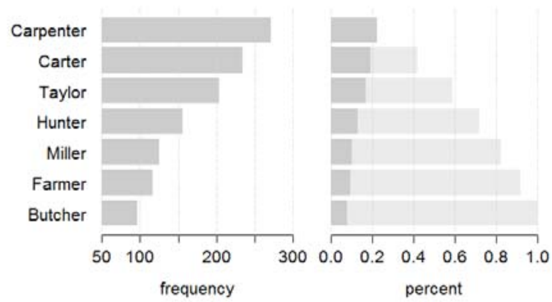



**Figure 10.2** Frequency plot of a categorical variable

If there are missing values, they will be listed in the first row, together with the length of the vector and the number of levels.

Synopsis

| | |
|---|---|
| `length` | total number of elements in the vector, NAs are included here |
| `n` | number of valid cases, NAs, NaNs, Inf etc. are not counted here |

| | |
|---|---|
| `NAs` | number of missing values |
| `levels` | number of levels |
| `unique` | number of unique (observed) values.<br>**Note:** This is not necessarily the same number as levels, as there might be empty levels. Thus the number of levels might be higher than the number of unique values (but not conversely). |
| `dupes` | y(es) or n(o), reporting if there are any duplicate values in the vector. If "n" (for no) is reported then there are only unique values in the variable. This might typically be the case for identifiers. |
| `freq` | the count (absolute frequency) of the specific level. The order of a factors frequency table is by default chosen as "absolute frequency-decreasing". |
| `perc` | the relative frequency of the specific level |
| `cumfreq` | the cumulative frequencies of the levels |
| `cumperc` | the same for the percentage values |

The graphical representation consists of two horizontal barplots. The left one is displaying the absolute frequencies with truncated x-axis. The left plot will always display the percentages with fixed x-axis limits set to 0 and 1. The cumulative frequencies can be displayed or be left away.

The plot can be customized with several arguments:

```
plot(Desc(d.pizza$driver), main = NULL, maxlablen = 25, type = c("bar", "dot"),
     col = NULL, border = NULL, xlim = NULL, ecdf = TRUE))
```

If the labels exceed a certain length, they will be truncated. The length where this happens can be controlled with the argument `maxlablen`. The cumulative bars can be blown off with `ecdf=FALSE`. The other arguments follow the meaning of those in the function `barplot`.

# 3 Numerical Variables

## 3.1 Numeric

The next variable, the temperature of the delivered pizza, is numeric. Numeric variables are described by the most common statistical measures for location, variation and shape.

Several features of the output are worth some consideration. The largest and smallest values should be scanned for outlying or incorrect values. In real world data erroneous (or awkwardly coded) values are often found at the ends of a variable. Therefore the values and their frequencies (numbers in brackets) are reported. In the example below "(2)" means that the value 20.2 can be found twice in the variable.

The mean (or median) and standard deviation (or interquartile range IQR, resp. the median absolute deviation mad) should be assessed as general measures of the location and spread of the data. The quantiles deliver a good overall impression of the distribution. In the current example we note that 90% of the data lie between 26 and 60 degrees and the inner 50% between 42 and 55.

The skewness and kurtosis are usually more easily assessed by graphical means, though their numerical values are included in the output. A large difference between the mean and median is another cue for the skewness. In right-skewed data with a positive value of the skewness, the mean is larger than the median, while in left-skewed data (skewness < 0), the mean is smaller than the median.

```
Desc(d.pizza$temperature, main="", plotit=TRUE)
```

```
 length       n     NAs  unique      0s    mean  meanSE
  1'209   1'170      39     375       0  47.937   0.291

    .05     .10     .25  median     .75     .90     .95
 26.700  33.290  42.225      50  55.300  58.800  60.500

  range      sd   vcoef     mad     IQR    skew    kurt
 45.500   9.938   0.207   9.192  13.075  -0.842   0.051

lowest : 19.3, 19.4, 20, 20.2 (2), 20.35
highest: 63.8, 64.1, 64.6, 64.7, 64.8
```

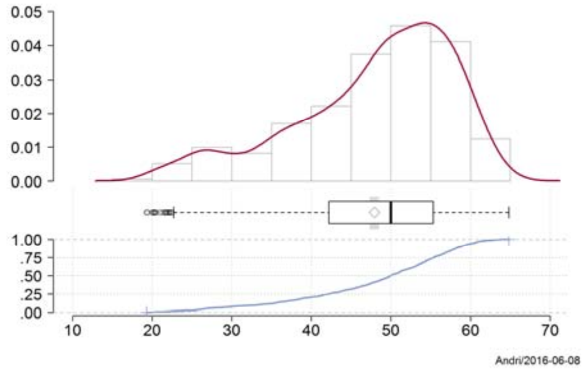



**Figure 3.1** Distribution of a numeric variable.

**PlotFdist**

The plot in figure 3.1 as produced by the function `PlotFdist` combines a histogram with a density plot, a boxplot and the plot of the empirical distribution function (ECDF). The scale for the x-axis is synchronized over all plots. The median can thus be found on the boxplot as also in the ecdf-plot.
The maximum and the minimum value are tagged with a tiny vertical dash upon the ecdf-line. The mean is shown in the boxplot as grey cross, the grey bar is its confidence interval.

Let's enumerate the features in detail. The first measures length, n, NAs, unique have again the same meaning as above. NAs are silently removed from all subsequently calculations.

| | |
|---|---|
| `0s` | total number of zero values. |
| `mean` | the arithmetic mean of the vector. |
| `meanSE` | standard error of the mean, sd(x) / sqrt(n). (See also: function `MeanCI(...)`) This can be used to construct the confidence intervals for the mean, defined as `qt(p = 0.025, df = n-1) * sd(x) / sqrt(n)`. |
| `.05, .., .95` | quantiles of x, starting with 5%, 10%, 1. quartile, median etc. |
| `rng` | range of x, `max(x) - min(x)` |
| `sd` | standard deviation |
| `vcoef` | variation coefficient, defined as `sd(x) / mean(x)` |
| `mad` | median absolute deviation |
| `IQR` | inter quartiles range |
| `skew` | skewness of x |
| `kurt` | kurtosis of x |
| `lowest` | the smallest 5 values. If there are bindings, the frequency of each value will be reported in brackets. |
| `highest` | same as lowest, but on the other end |

Transformations can easily be entered in place.

```
Desc(1/d.pizza$temperature, digits=3, main="")
title(expression(frac(1,x)))
```

```
 length       n     NAs  unique      0s    mean  meanSE
  1'209   1'170      39     375       0   0.022   0.000

    .05     .10     .25  median     .75     .90     .95
  0.017   0.017   0.018   0.020   0.024   0.030   0.037

  range      sd   vcoef     mad     IQR    skew    kurt
  0.036   0.006   0.289   0.004   0.006   2.027   4.244

lowest : 0.015, 0.015, 0.015, 0.016, 0.016
highest: 0.049, 0.050 (2), 0.050, 0.052, 0.052
```

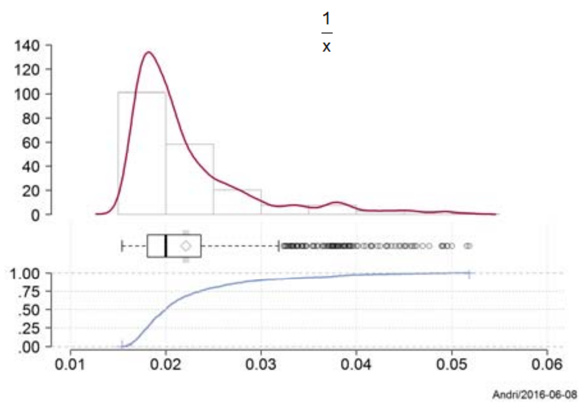



**Figure 3.2** Distribution of a numeric variable.

There are several approaches commonly used for graphical comparing the variable's distribution to a reference distribution. The two most seen are firstly superposing the reference density curve over the variable's histogram and the second using a Q-Q-plot. A Q-Q-plot is used to compare the shapes of distributions, providing a graphical view of how properties such as location, scale, and skewness are similar or different in the two distributions.

```
z <- LinScale(z, newlow=0, newhigh = 32)[,1]

PlotFdist(z, args.curve = list(expr="dchisq(x, df=5)", col="darkgreen"),
          args.boxplot=NA, args.ecdf=NA)

legend(x="topright", legend=c("kernel density", expression(chi["df=5"]^2-distribution)),
       fill=c(getOption("col1", hred), "darkgreen"), text.width = 5)
```

**LinScale**

We get

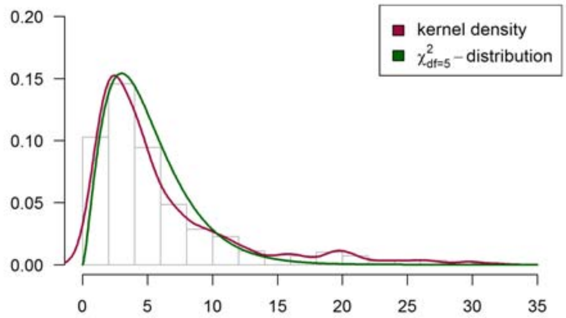



**Figure 3.3** Overlay of fitted $\chi^2$-function.

This makes it clear, that this is not the best way to decide, whether the red curve follows our hypothesized distribution or not. Where does random start?

**PlotQQ**

The better approach is to use a QQ-plot, which by the way solves the x-axis scaling problem we had in the overlay solution. The function `PlotQQ` is a wrapper for plotting QQ-plots with other than normal distributions.
A `qqline` is inserted on which the points are likely to lie (approximately) if the two distributions being compared are similar.
It sometimes might be hard to judge, if the points are (too) far away from the `qqline` or not.
An idea to check the general variability is to use simulated sets with the desired distribution. If our points exceed the confidence intervals, something is likely to be wrong.
In our example everything's fine, of course, as we sampled from the tested distribution.

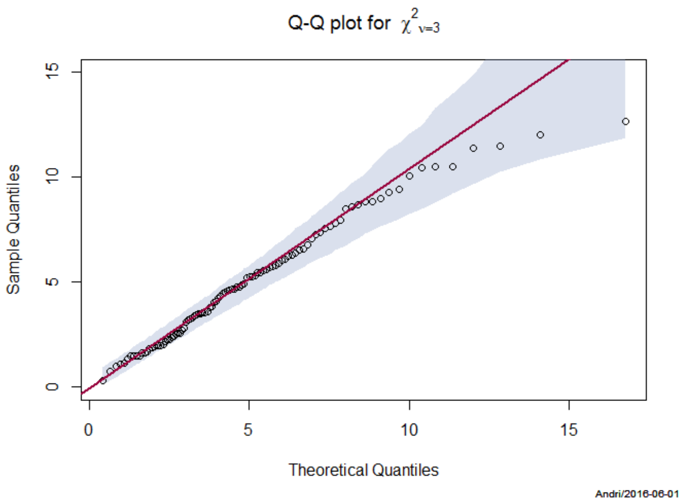



**Figure 3.4** QQ plot for a $\chi^2$-distributed variable.

```
set.seed(159)
z <- rchisq(100, df=5)

PlotQQ(z, function(p) qchisq(p, df=5), type="n", main=NA, args.qqline = NA)

x <- qdist(ppoints(z))
y <- replicate(1000, sort(rchisq(100, df=5)))
ci <- apply(y, 1, quantile, c(0.025,0.975))

DrawBand(x = c(x, rev(x)), y = c(ci[1,], rev(ci[2,])), col=SetAlpha(hblue, 0.3))
PlotQQ(zz, function(p) qchisq(p, df=5), add=TRUE,
       args.qqline=list(col=hred,lwd=2, probs=c(0.1, 0.6)))

title(main=expression("Q-Q plot for" ~~ {chi^2}[nu == 3]))
```

What do the tests say about ozone being gamma distributed?

```
AndersonDarlingTest(na.omit(ozone), "pgamma", shape = m^2/v, scale = v/m)

##         Anderson-Darling test of goodness-of-fit
##         Null hypothesis: Gamma distribution
##         with parameters shape = 1.6310, scale = 25.8300
##
## data:  na.omit(ozone)
## An = 0.66365, p-value = 0.5896
```

The observation seems compatible with the hypothesis.
Let's superpose the model distribution curve to both, the histogram and the cumulative distribution function.

```
ozone <- airquality$Ozone; m <- mean(ozone, na.rm = TRUE); v <- var(ozone, na.rm = TRUE)

PlotFdist(ozone, args.hist = list(breaks=15),
  args.curve = list(expr="dgamma(x, shape = m^2/v, scale = v/m)", col=hecru),
  args.curve.ecdf = list(expr="pgamma(x, shape = m^2/v, scale = v/m)", col=hecru),
  na.rm = TRUE, main = "Airquality - Ozone")

legend(x="topright",
       legend=c(expression(plain("gamma:  ") * Gamma * " " * bgroup("(", k * " = " *
           over(bar(x)^2, s^2) * " , " * theta * plain(" = ") * over(s^2, bar(x)), ")") ),
                "kernel density"),
       fill=c(hecru, getOption("col1", hred)), text.width = 0.25)
```

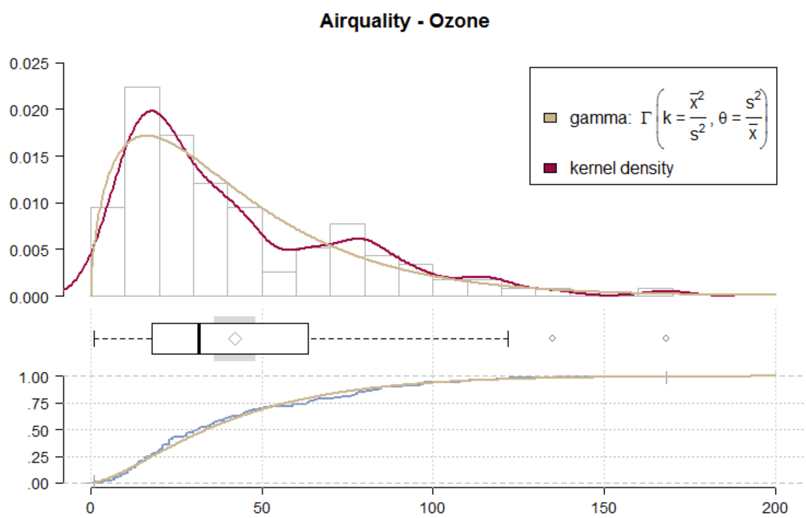



**Figure 3.5** Compare empirical distribution with a gamma distribution.

### 3.2 Numeric data with few unique values

If there's a numeric variable with only one or two handfuls of unique values then a description by means of a histogram and a density curve is not really adequate. The density curve will start oscillating and the bins in the histograms would lose their continuous nature.
Therefore we change the graphic representation from a histogram to a histogram like h-type plot leaving the density curve off.

In the numerical results the extreme values will be replaced by a full frequency representation with absolute values and percentages.

```
Desc(d.pizza$weekday, plotit=TRUE)
```

```
length      n    NAs  unique     0s   mean  meanSE
 1'209  1'177     32       7      0   4.44    0.06

   .05    .10    .25  median    .75    .90     .95
  1.00   1.00   3.00    5.00   6.00   7.00    7.00

 range     sd  vcoef     mad    IQR   skew    kurt
  6.00   2.02   0.45    2.97   3.00  -0.34   -1.17


   level  freq   perc  cumfreq  cumperc
1      1   144  12.2%      144    12.2%
2      2   117   9.9%      261    22.2%
3      3   134  11.4%      395    33.6%
4      4   147  12.5%      542    46.0%
5      5   171  14.5%      713    60.6%
6      6   244  20.7%      957    81.3%
7      7   220  18.7%    1'177   100.0%
```

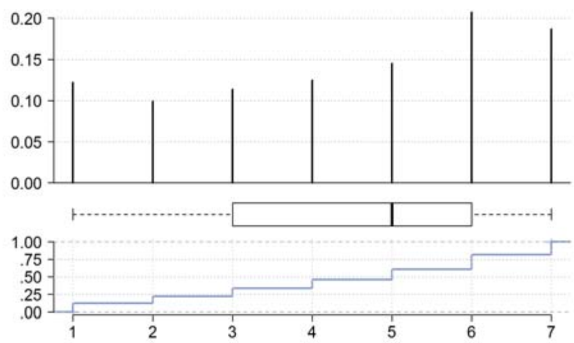



**Figure 3.6** Distribution of a numeric variable.

### 3.3 Count data (discrete)

The next variable is a count variable, whose nature is somewhat between numeric and factors as far as descriptive measures are concerned. In fact, if there are only just a few unique values, then the factor representation (frequencies) might be more appropriate than the numeric description (with densities etc.). We draw the line between factor and numeric representation at a dozen of unique values in x. Beyond that number, the numeric description will be reported and for fewer values the factor representation will be used.

```
Desc(d.pizza$count, plotit=TRUE)
```

```
  length      n    NAs unique     0s   mean meanSE
   1'209  1'197     12      8      0  3.444  0.045

     .05    .10    .25 median    .75    .90    .95
       1      2      2      3      4      6      6

     rng     sd  vcoef    mad    IQR   skew   kurt
       7  1.556  0.452  1.483      2  0.454 -0.363


  level freq  perc cumfreq cumperc
1     1  108  .090     108    .090
2     2  259  .216     367    .307
3     3  300  .251     667    .557
4     4  240  .201     907    .758
5     5  152  .127    1059    .885
6     6   97  .081    1156    .966
7     7   34  .028    1190    .994
8     8    7  .006    1197   1.000
```

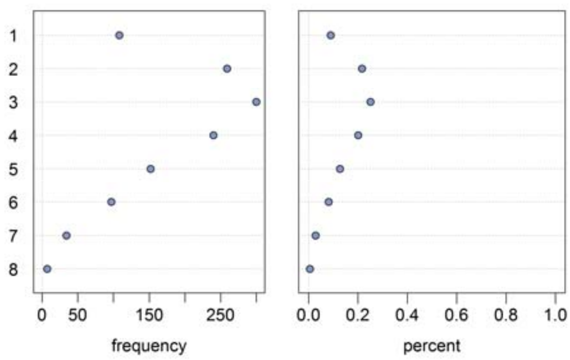



**Figure 3.7** Distribution of a count variable.

The plot is produced as a (horizontal) `dotchart`. More than 12 unique values are truncated (a warning is placed in the plot area). The `maxrows` argument can be used to override this default (`Inf` for all).

Two dotcharts are created, the left one shows the absolute frequencies, the right one the percentages. On the left plot the x-axis might be adapted to the data (as R does by default). The percentages will always be displayed on a 0:1-range.
The plot width is adapted to the length of the labels. If the labels get too long, they will be truncated and displayed with ellipsis (...).

# 4 Logical values

Dichotomous variables do not have real dense (univariate) information. The variable `wine_ordered` for example contains only two values, 0 and 1. Still it is usually interesting to know, how many NAs there are, besides the frequencies of course. The individual frequencies are reported together with a confidence interval, calculated by `BinomCI` using the option `"Wilson"`.

```
Desc(d.pizza$wine_ordered, plotit=TRUE)
```

```
  length      n    NAs unique
   1'209  1'197     12      2

  freq  perc lci.95 uci.95¹
0 1010  .844   .822   .863
1  187  .156   .137   .178

¹ 95%-CI Wilson
```

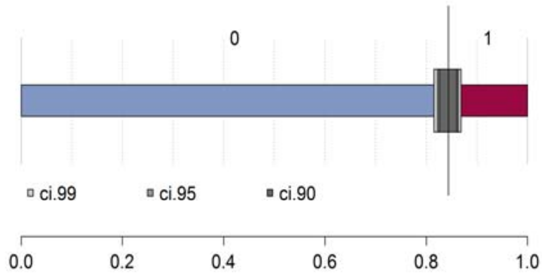



**Figure 4.1** Distribution of a numeric variable.

This is basically a univariate horizontal stacked `barplot`, with confidence intervals on the confidence levels of 0.90, 0.95 and 0.99. The vertical line denominates the point estimator.

# 5 Time variables

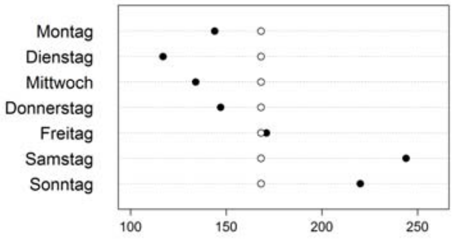
## 5.1 Dates

A date variable is harder to describe in a univariate context. What characteristics would one want to know from a date? We would normally choose a description similar to numeric values, supplemented by an analysis of the weekday and month for grasping anomalies concerning extreme, invalid or missing values.

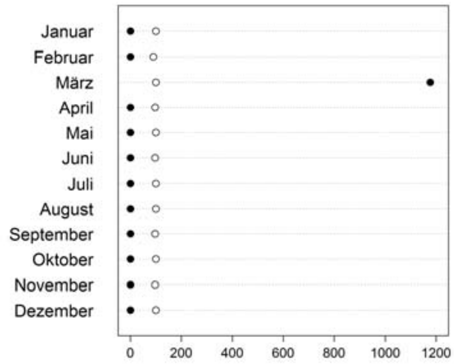
```
Desc(d.pizza$date, plotit=TRUE)
```

```
  length      n    NAs unique
   1'209  1'177     32     31

lowest : 2014-03-01 (42), 2014-03-02 (46), 2014-03-03 (26), 2014-03-04 (19)
highest: 2014-03-28 (46), 2014-03-29 (53), 2014-03-30 (43), 2014-03-31 (34)
```

```
Weekdays:
       level freq  perc  cumfreq cumperc   exp  res
1     Montag  144  .122      144    .122 168.1 -1.9
2   Dienstag  117  .099      261    .222 168.1 -3.9
3   Mittwoch  134  .114      395    .336 168.1 -2.6
4 Donnerstag  147  .125      542    .460 168.1 -1.6
5    Freitag  171  .145      713    .606 168.1   .2
6    Samstag  244  .207      957    .813 168.1  5.9
7    Sonntag  220  .187     1177   1.000 168.1  4.0

        Chi-squared test for given probabilities

data:  table(xd)
X-squared = 78.8785, df = 6, p-value = 6.09e-15
```



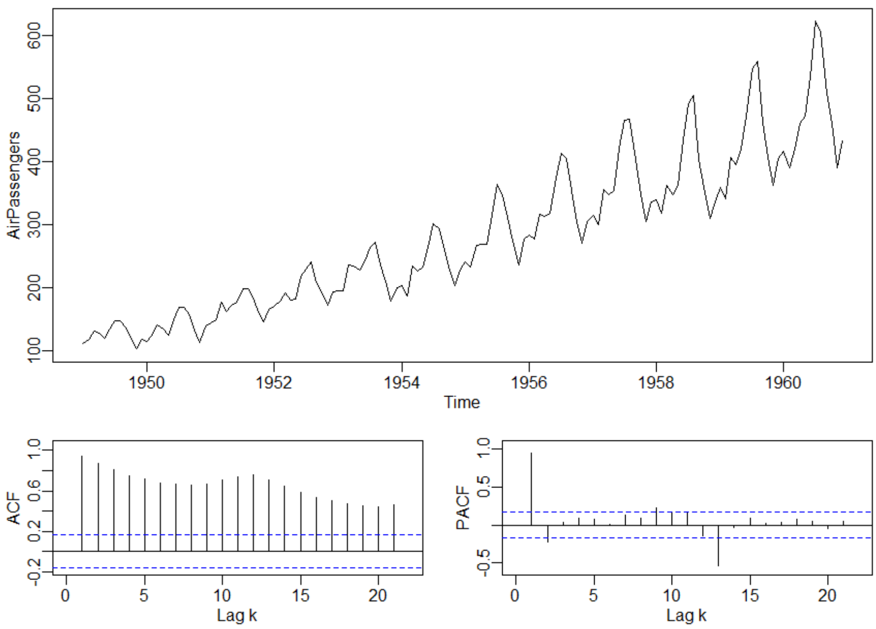

```
Months:
       level freq perc cumfreq cumperc  exp prs.res
1      Januar    0    0       0       0 99.7   -10.0
2     Februar    0    0       0       0 93.3    -9.7
3        März 1177    1    1177       1 99.7   107.9
4       April    0    0    1177       1 96.5    -9.8
5         Mai    0    0    1177       1 99.7   -10.0
6        Juni    0    0    1177       1 96.5    -9.8
7        Juli    0    0    1177       1 99.7   -10.0
8      August    0    0    1177       1 99.7   -10.0
9   September    0    0    1177       1 96.5    -9.8
10    Oktober    0    0    1177       1 99.7   -10.0
11   November    0    0    1177       1 96.5    -9.8
12   Dezember    0    0    1177       1 99.7   -10.0

        Chi-squared test for given probabilities

data:  tab
X-squared = 12719.19, df = 11, p-value < 2.2e-16
```




## 5.2 Timeseries ACF-plot

This produces a combined plot of a time series and its autocorrelation and partial autocorrelation, which is used in every introductory course for time-series.

```
PlotACF(AirPassengers)
```



# 6 data.frames

## 6.1 Overview

After that, every single variable will be described according to the type of its class.

Let's start with a quick description of some variables out of the integrated `data.frame` `d.pizza`.

```
library(DescTools)

# the results (and the plots) will either be displayed in the console
Desc(d.pizza[,c("driver","temperature","count","weekday","wine_ordered","date")],
plotit=TRUE)

# ... or we can start a new word instance and send the results directly to a word document
wrd <- GetNewWrd()
Desc(d.pizza[,c("driver","temperature","count","weekday","wine_ordered","date")], wrd=wrd)
```

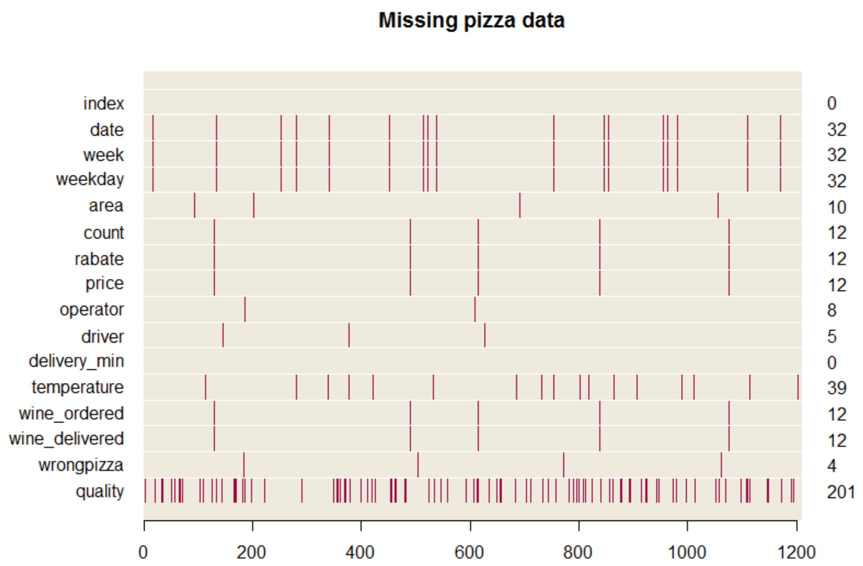
```
'data.frame':   1209 obs. of  4 variables:
 1 $ driver      : Factor w/ 7 levels "Butcher","Carpenter",..: 7 1 1 7 3 7 7 7 7 3 ...
 2 $ temperature : num  53 56.4 36.5 NA 50 27 33.9 54.8 48 54.4 ...
 3 $ count       : int  5 2 3 2 5 1 4 NA 3 6 ...
 4 $ weekday     : num  6 6 6 6 6 6 6 6 6 6 ...
 5 $ wine_ordered: int  0 0 0 0 0 0 1 NA 0 1 ...
 6 $ date        : Date, format: "2014-03-01" "2014-03-01" "2014-03-01" "2014-03-01" ...
```

First a simple `Str()` of the `data.frame` is performed. The result is no more than that of a `str()` command, extended with an enumeration of the variables.

**Str**

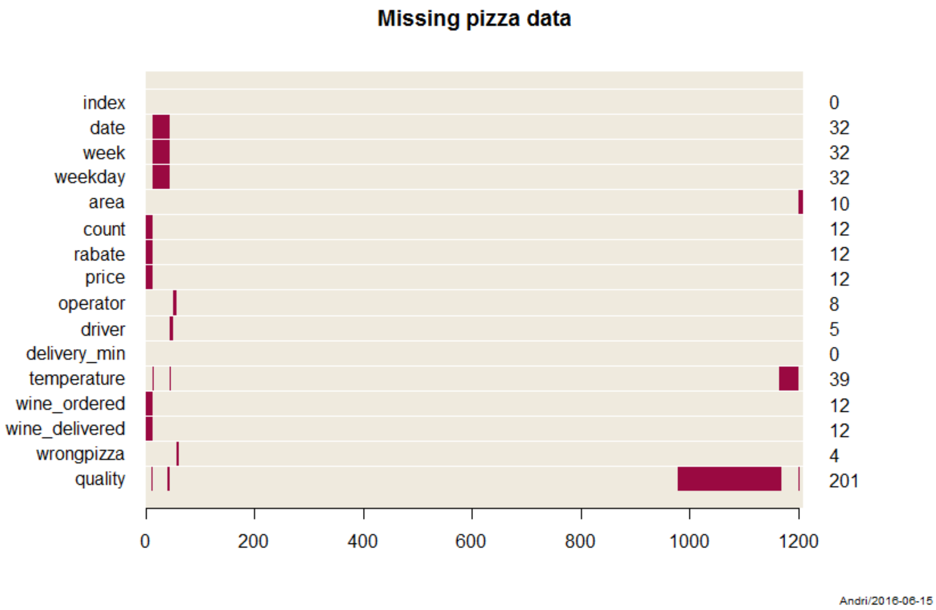
## 6.2 Missing data

An interesting idea for creating a visual representation of missing data was brought to my attention by Henk Harmsen. The following plot symbolizes each missing value with a vertical line. The x-axis represents the index of the record. On the right side are the numbers of missings noted.



The missing values can be clustered such as to display several areas of missing values. This can be helpful for detecting dependencies or patterns within the missings.

```
PlotMiss(d.pizza, main="Missing pizza data", clust = TRUE)
```




# 7 Pairwise Numeric ~ Categorical

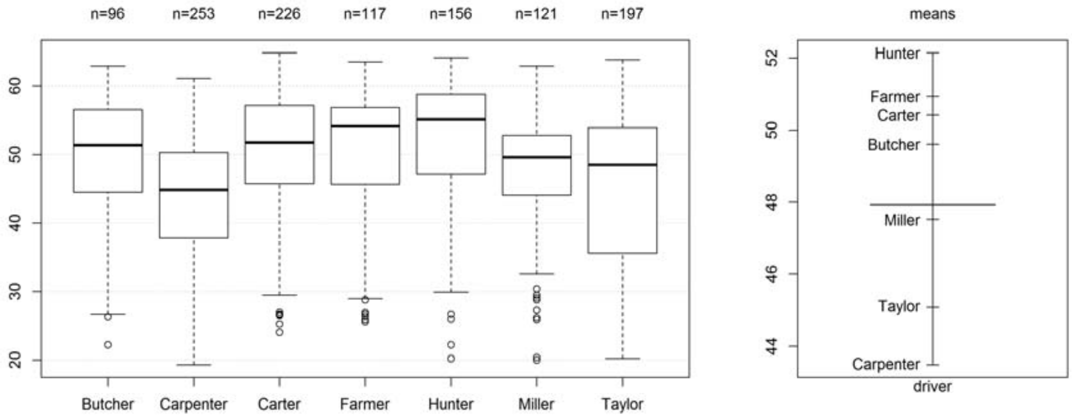
## 7.1 Boxplot and Designplot

`Desc` implements a formula interface allowing to define bivariate descriptions straight forward.

A numeric variable vs. a categorical is best described by group wise measures. Here the valid pairs are reported first. Missing values in the single groups are documented in the results table and missing values on the grouping factor are mentioned with a warning at the end of the table, if existing at all.

```
Desc(temperature ~ driver, d.pizza, digits=1, plotit=TRUE)

Summary:
n pairs: 1'209, valid: 1'166 (96%), missings: 43 (4%), groups: 7

          Butcher  Carpenter    Carter    Farmer    Hunter    Miller    Taylor
mean         49.6      43.5¹      50.4      50.9     52.1²      47.5      45.1
median       51.4      44.8¹      51.8      54.1     55.1²      49.6      48.5
sd            8.8        9.4       8.5       9.0       8.9       8.9      11.4
IQR          12.0       12.5      11.3      11.2      11.6       8.8      18.4
n              96        253       226       117       156       121       197
np          0.082      0.217     0.194     0.100     0.134     0.104     0.169
NAs             0         19         8         0         0         4         7
0s              0          0         0         0         0         0         0

¹ min, ² max

Kruskal-Wallis rank sum test:
  Kruskal-Wallis chi-squared = 141.9349, df = 6, p-value < 2.2e-16
```

```
Warning:
  Grouping variable contains 5 NAs (0.414%).
```



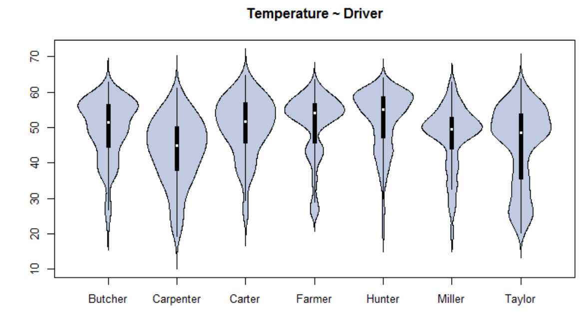

a boxplot combined with a means-plot as used in anova

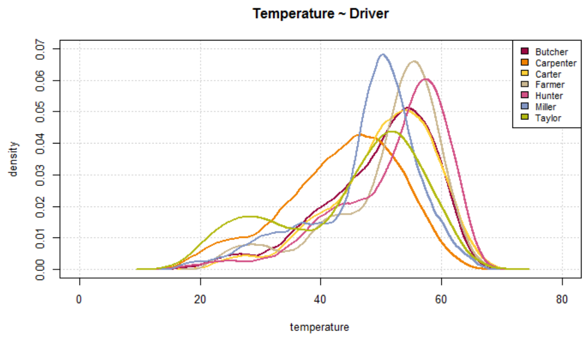
## 7.2 Comparing distributions

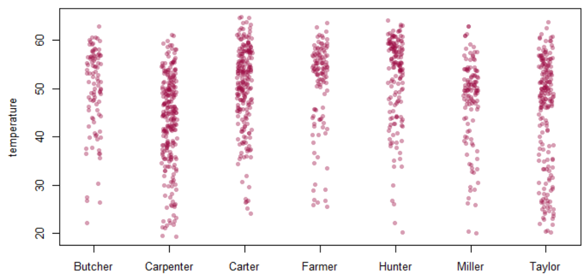
How should we compare distributions graphically, moving beyond a simple boxplot? PlotViolin serves the same utility as a side-by-side boxplot, but provides more detail about the single distribution. We started with John Verzani's Violinplot and rewrote it so that it takes exactly the same parameters as the boxplot-function.

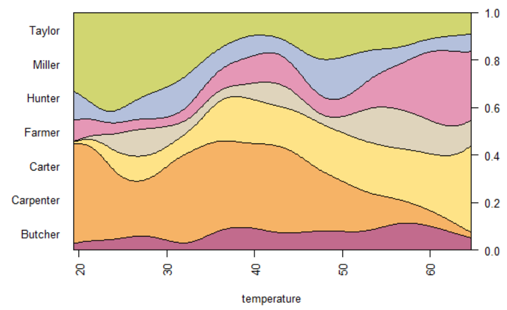
Another idea is to plot several densities within the same plot. PlotMultiDens does this while setting the xlim- and ylim-values to an appropriate value, ensuring all density lines are fully visible. For a smaller number of variables, say up to two handfuls, this will be the most direct way to compare their distributions. (Note: For violins this limit lies much higher as they do not overlap and so mutually hide.)

```
PlotViolin(temperature ~ driver, data=d.pizza, col = SetAlpha(hblue,0.5),
           main="Temperature ~ Driver")

PlotMultiDens(temperature ~ driver, data=d.pizza, xlab="temperature",
              main="Temperature ~ Driver", panel.first=grid(),
              col=PalHelsana(), lwd=2 )
```



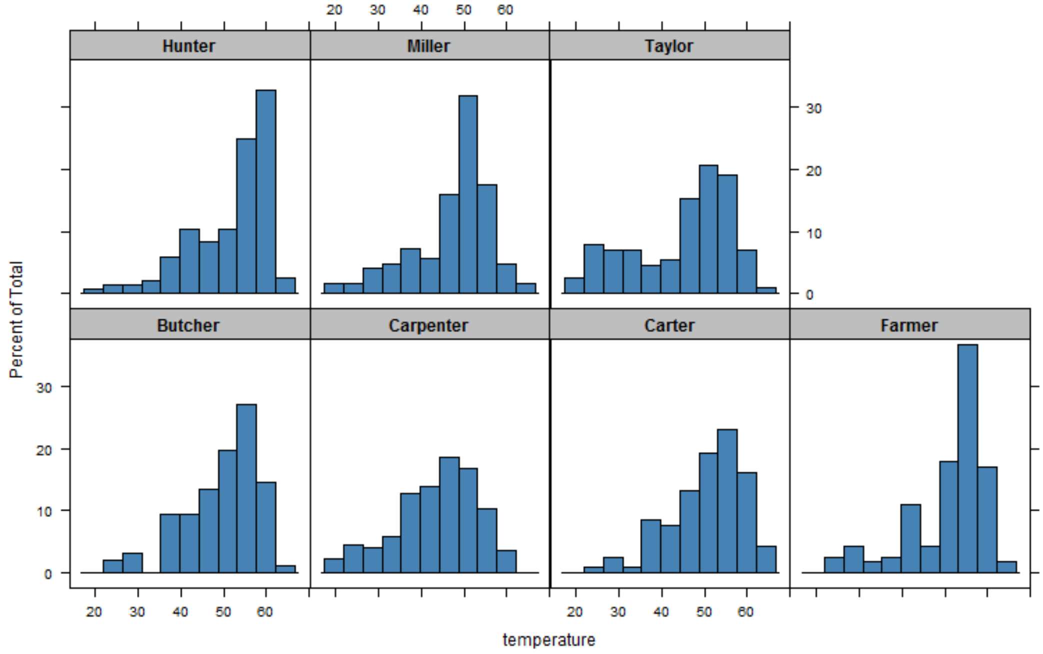

For small datasets a stripchart might be the best way to plot the data.
The conditional density-plot at the right allows grasping the proportions within the total density.

```
stripchart(temperature ~ driver, d.pizza, vertical=TRUE,
           method="jitter", pch=16, col=SetAlpha(hred,0.4))

d.frm <- na.omit(d.pizza[,c("temperature","driver")])
par(las=2, mar=c(4.1,10.1,5.1, 5.1))
cdplot(x=d.frm$temperature, y=d.frm$driver, ylab="", xlab="temperature",
       col=SetAlpha(PalHelsana(), 0.6))
```

## 7.3 Trellis

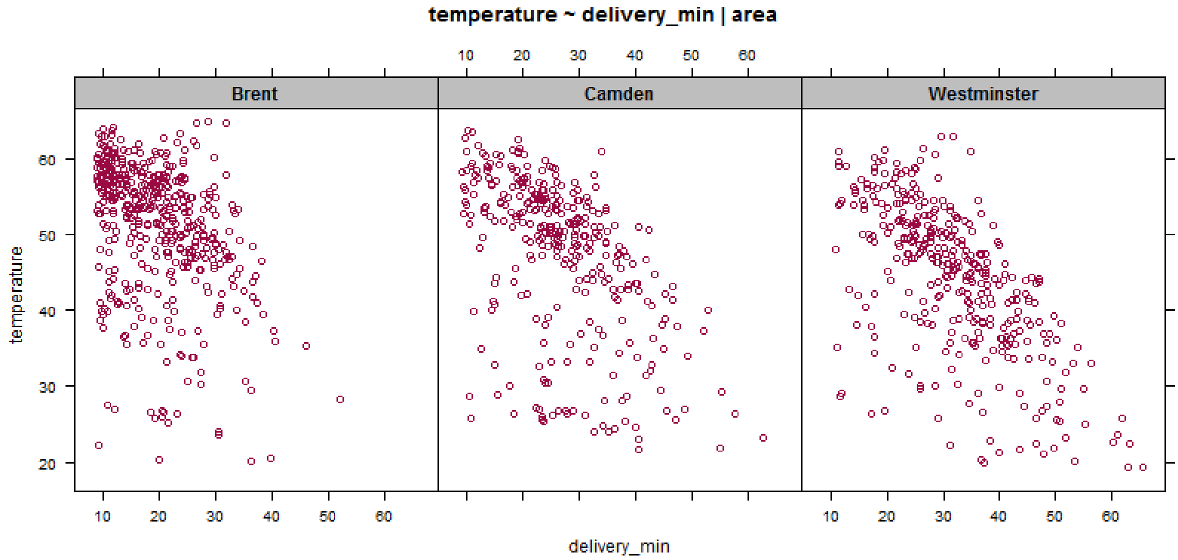
The classic way is to spend a full plot for every single variable. There's an interesting link, demonstrating this technique: http://www.statmethods.net/advgraphs/trellis.html.
But first or all, let's readjust Deepayan's rather peculiar default colours. (Sorry Deepayan!)

```
library(lattice)
trellis.par.set(strip.background = list(col = gray(0.5)),
                add.text = list(col = 'white'))

myStripStyle <- function(which.panel, factor.levels, ...) {
  panel.rect(0, -0.5, 1, 1,
             col = "grey",
             border = 1)
  panel.text(x = 0.5, y = 0.25,
             font=2,
             lab = factor.levels[which.panel],
             col = "black")
}

histogram( ~ temperature | driver, data=d.pizza, col="steelblue", strip=myStripStyle)
```



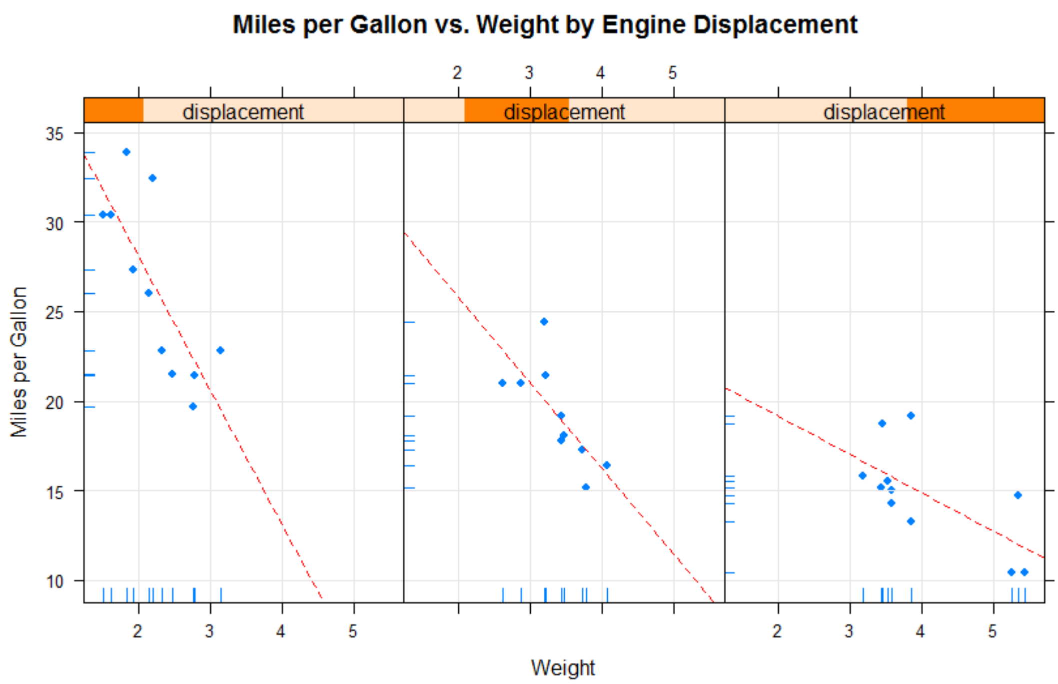

Again here a scatterplot is highly informative.

```
xyplot(temperature ~ delivery_min | area, d.pizza,
       main='temperature ~ delivery_min | area', col=hred, strip=myStripStyle)
```



Another nice combination of several elements like rug, grid and lmline:

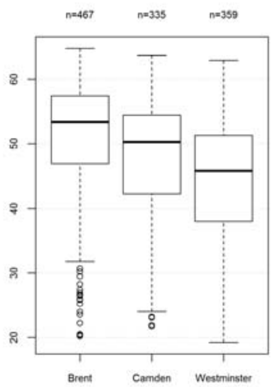
```
library(lattice)

displacement <- equal.count(mtcars$disp, number=3, overlap=0)

mypanel <- function(x, y) {
  panel.xyplot(x, y, pch=19)
  panel.rug(x, y)
  panel.grid(h=-1, v=-1)
  panel.lmline(x, y, col="red", lwd=1, lty=2)
}

xyplot(mpg ~ wt | displacement, data=mtcars,
       layout = c(3, 1),
       aspect = 1.5,
       main   = "Miles per Gallon vs. Weight by Engine Displacement",
       xlab   = "Weight",
       ylab   = "Miles per Gallon",
       panel  = mypanel)
```

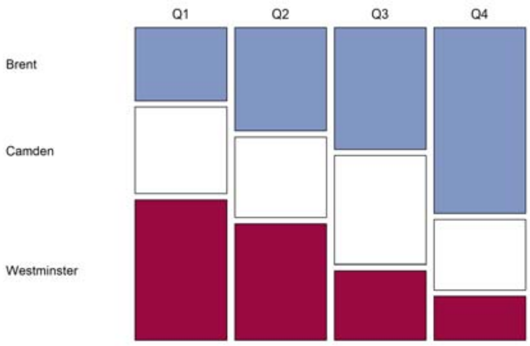


# 8 Pairwise Categorical ~ Numeric

No, it's not the same as numeric ~ categorical. The design is such, that the response variable is categorical and the predictor numeric. With a model one would set up a multinomial regression (or logistic in the case of 2 categories).

```
Desc(area ~ temperature, data=d.pizza, digits=1, wrd=wrd)

Summary:
n pairs: 1'209, valid: 1'161 (96%), missings: 48 (4%), groups: 3


              Brent       Camden  Westminster
mean           51.1²        47.4        44.3¹
median         53.4²        50.3        45.9¹
sd              8.7         10.1         9.8
IQR            10.5         12.2        13.2
n               467          335         359
np            0.402        0.289       0.309
NAs               7            9          22
0s                0            0           0

¹ min, ² max

Kruskal-Wallis rank sum test:
  Kruskal-Wallis chi-squared = 115.83, df = 2, p-value < 2.2e-16
Warning:
  Grouping variable contains 10 NAs (0.827%).


Proportions of area in the quantiles of temperature:

                Q1    Q2    Q3    Q4
  Brent       0.244 0.345 0.405 0.618
  Camden      0.289 0.266 0.363 0.236
  Westminster 0.467 0.389 0.232 0.146
```




# 9 Pairwise Categorical ~ Categorical

Two categorical variables are described by a contingency table, as shown in the vignette Tables.

# 10 Pairwise Numeric ~ Numeric

## 10.1 Boxplot and Designplot

Two numerical variables have no obvious standard description as their relationship can have manifold forms. Thus we're going to report only the simple correlation coefficients (Pearson, Spearman and Kendall) and a hopefully helpful scatterplot.

The variables are plotted as xy-scatterplots with interchanging mutual dependency, supplemented with either a LOESS or a spline smoother.

```
Desc(temperature ~ delivery_min, d.pizza, plotit=TRUE)
```

```
Summary:
n pairs: 1'209, valid: 1'170 (97%), missings: 39 (3%)


Pearson corr. : -0.575
Spearman corr.: -0.573
Kendall corr. : -0.422
```

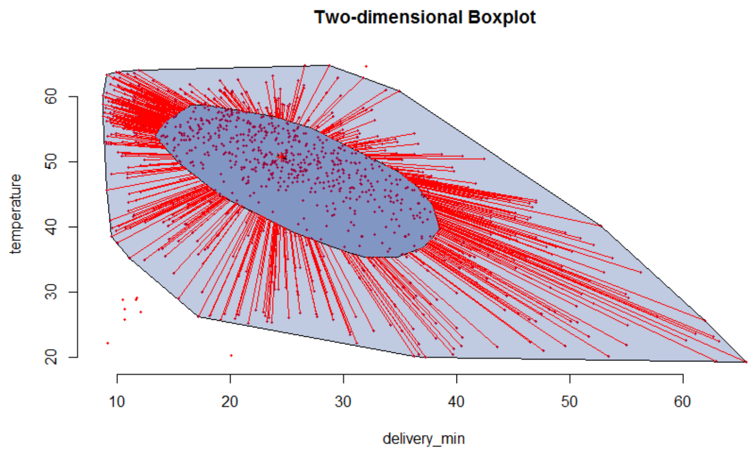
Scatterplots for two numeric variables:


**Figure 10.2** Mosaicplot of Eye colour ~ Hair colour.

## 10.2 Boxplot in 2 dimensions: PlotBag

This function transposes the boxplot idea in the 2-dimensional space. The points are outliers, the lightblue area is the area within the fences in a normal boxplot and the darkblue area is the inner quartile range.
The median is plotted as orange point in the middle.
This code is taken verbatim from Peter Wolf's aplpack package.

```
d.frm <- d.pizza[complete.cases(d.pizza[,c("temperature","delivery_min")]),]

PlotBag(x=d.frm$delivery_min, y=d.frm$temperature, xlab="delivery_min",
        ylab="temperature", main="Two-dimensional Boxplot")
```




## 11 Table One

Create a table summarizing continuous, categorical and dichotomous variables, optionally stratified by one or more variables, while performing adequate statistical tests.

```
# define some special formats for count data, percentages and numeric results
# (those will be supported by TOne)
options(fmt.abs=structure(list(digits=0, big.mark="'"), class="fmt"))
options(fmt.per=structure(list(digits=1, fmt="%"), class="fmt"))
options(fmt.num=structure(list(digits=1, big.mark="'"), class="fmt"))

ToWrd(TOne(x=d.pizza[, c("temperature","delivery_min","driver","wine_ordered")],
           grp=d.pizza$quality),
      wrd=GetNewWrd())
```

will produce the following table:

| var | total | low | medium | high | |
|---|---|---|---|---|---|
| n | 1'008 | 156 (15.5%) | 356 (35.3%) | 496 (49.2%) | |
| temperature | 47.9 (9.9) | 32.9 (7.8) | 45.6 (7.4) | 53.6 (6.5) | *** 1 |
| delivery_min | 25.7 (10.8) | 33.9 (11.7) | 26.5 (10.1) | 22.6 (9.5) | *** 1 |
| driver | | | | | *** 3 |
| Butcher | 79 (8.0%) | 10 (6.5%) | 36 (10.1%) | 33 (6.7%) | |
| Carpenter | 225 (22.6%) | 59 (38.1%) | 90 (25.4%) | 76 (15.4%) | |
| Carter | 196 (19.4%) | 11 (7.1%) | 72 (20.3%) | 113 (22.9%) | |
| Farmer | 94 (9.7%) | 10 (6.5%) | 26 (7.3%) | 58 (11.7%) | |
| Hunter | 130 (13.0%) | 8 (5.2%) | 43 (12.1%) | 79 (16.0%) | |
| Miller | 109 (10.4%) | 16 (10.3%) | 35 (9.9%) | 58 (11.7%) | |
| Taylor | 171 (16.9%) | 41 (26.5%) | 53 (14.9%) | 77 (15.6%) | |
| wine_ordered (= 1) | 161 (16.1%) | 32 (20.8%) | 63 (17.9%) | 66 (13.4%) | . 3 |

[1]) Kruskal-Wallis test, [2]) Fisher exact test, [3]) Chi-Square test

## 12 Multiple pairwises

The formula supports the dot symbol, meaning every variable in the data besides the ones already present in the formula. The following code produces a plot for driver, operator and area versus the response variable temperature:

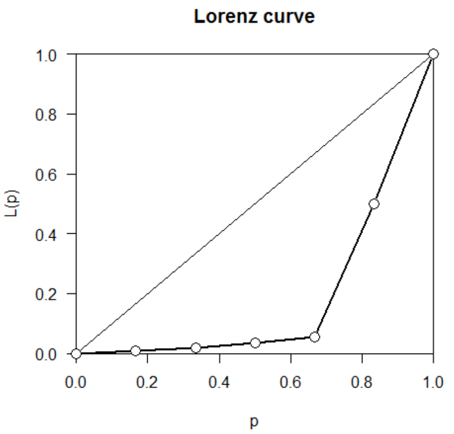
```
Desc(temperature ~ ., data=d.pizza[,c("temperature","driver","operator","area")],
     digits=1)
```

This can as well be reversed in the sense that the dot is defined as response variable and so all the variables will be plotted against one predictor variable.

## 13 Concentration

Lorenz-curves can be found in other libraries. This implementation starts with that from the library ineq, adding some value by calculating confidence intervals for the Gini coefficient.

```
x <- c(10, 10, 20, 20, 500, 560)

lc <- Lc(x)
plot(lc)
points(lc$p, lc$L, cex=1.5, pch=21, bg="white", col="black", xpd=TRUE)

Gini(x)
Gini(x, unbiased = FALSE)

Gini(x, conf.level = 0.95)
```



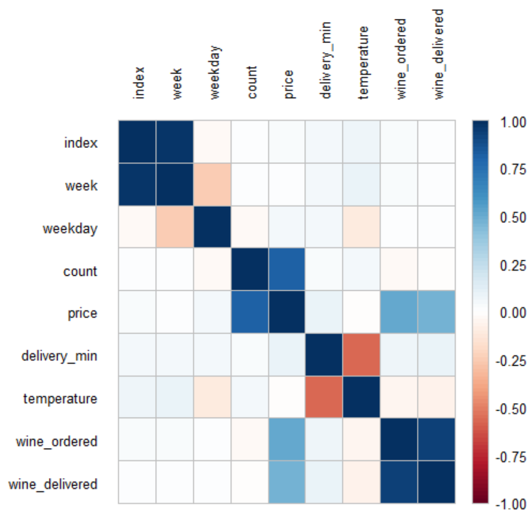

```
> Gini(x)
[1] 0.7535714

> Gini(x, unbiased = FALSE)
[1] 0.6279762

> Gini(x, conf.level=0.95)
     gini    lwr.ci    upr.ci
0.7535714 0.2000000 0.8967742
```

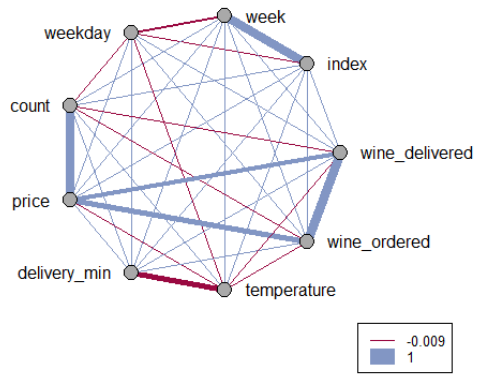
# 14 Multivariate graphical description

## 14.1 Correlation plot

These functions produce a graphical display of a correlation matrix. In the classic matrix representation the cells of the matrix can be shaded or coloured to show the correlation value. In the right circular representation the correlations are coded in the line width of the connecting lines. Red means a negative correlation, blue a positive one.

```
par(mfrow=c(1,2))
m <- cor(d.pizza[,which(sapply(d.pizza, is.numeric))], use="pairwise.complete.obs")

PlotCorr(m, col=PalDescTools("RedWhiteBlue1", 100), border="grey",
        args.colorlegend=list(labels=Format(seq(1,-1,-.25), 2), frame="grey"))

PlotWeb(m, col=c(hred, hblue))
```



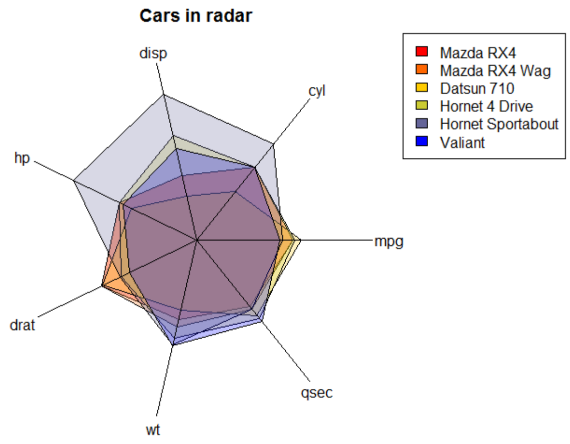




## 14.2 PlotPolar (Radarplot)

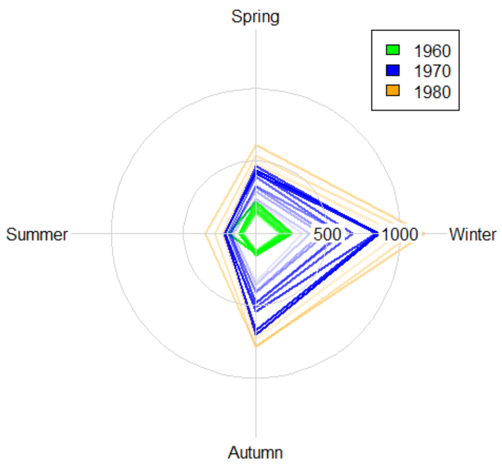
This function produces a polar plot but can also be used to draw radarplots or spiderplots.

**A)**

```
d.car <- scale(mtcars[1:6,1:7], center=FALSE)

# let's have a palette with thransparent colors
cols <- SetAlpha(colorRampPalette(c("red","yellow","blue"), space = "rgb")(6), 0.25)

PlotPolar(d.car, type="l", fill=cols, main="Cars in radar")
PolarGrid(nr=NA, ntheta=ncol(d.car), alabels=colnames(d.car), lty="solid", col="black")
legend(x=2, y=2, legend=rownames(d.car), fill=SetAlpha(cols, NA))
```



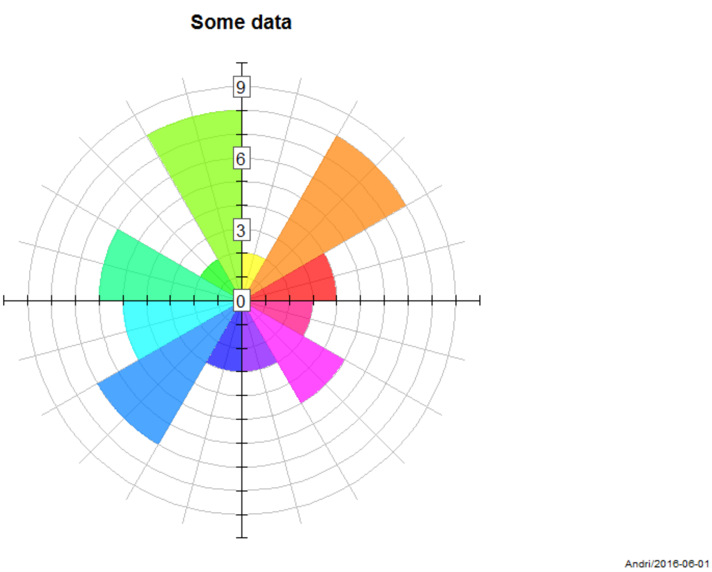

**A)**




**B)**

```
B)

m <- matrix(UKgas, ncol=4, byrow=TRUE)

cols <- c(SetAlpha(rep("green", 10), seq(0,1,0.1)),
          SetAlpha(rep("blue", 10), seq(0,1,0.1)),
          SetAlpha(rep("orange", 10), seq(0,1,0.1)))

PlotPolar(r=m, type="l", col=cols, lwd=2 )
PolarGrid(ntheta=4, alabels=c("Winter","Spring","Summer","Autumn"), lty="solid")

legend(x="topright", legend=c(1960,1970,1980), fill=c("green","blue","orange"))
```

A barplot in polar coordinates can be produced by means of the function DrawAnnulusSector.



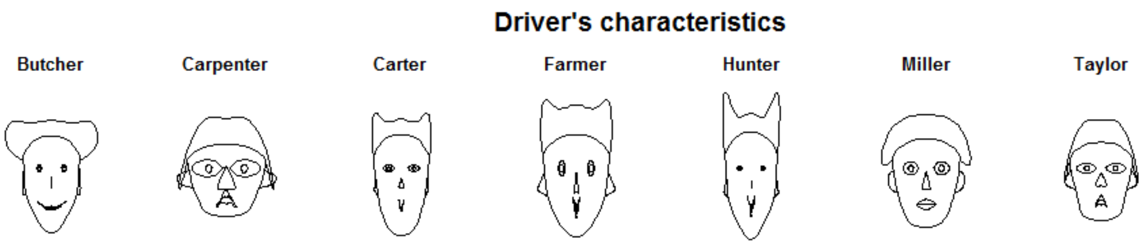

```
x <- c(4,8,2,8,2,6,5,7,3,3,5,3)
```

```
theta <- (0:12) * pi / 6
PlotPolar(x, type = "n", main="Some data")
PolarGrid(nr = 0:9, ntheta = 24, col="grey", lty=1, rlabels = NA, alabels = NA)
DrawAnnulusSector(x=0, y=0, radius.in=0, radius.out=x,
                  angle.beg = theta[-length(theta)], angle.end = theta[-1],
                  col=SetAlpha(rainbow(12), 0.7), border=NA)

segments(x0 = -10:10, y0 = -.2, y1=0.2)
segments(x0=-10, x1=10, y0 = 0)
segments(y0 = -10:10, x0 = -.2, x1=0.2)
segments(y0=-10, y1=10, x0 = 0)

BoxedText(x=0, y=c(0,3,6,9), labels = c(0,3,6,9), xpad = .3, ypad=.3, border="grey35")
```

## 14.3 PlotFaces

A nice idea for the concrete representation of your customer's profile is to produce a Chernoff faces plot. The rows of a data matrix represent cases and the columns the variables.

```
m <- data.frame( lapply(
d.pizza[,c("temperature","price","delivery_min","wine_ordered","weekday")]
                         , tapply, d.pizza$driver, mean, na.rm=TRUE))

PlotFaces(m, ncol=7, nrow=1, main="Driver's characteristics")
```

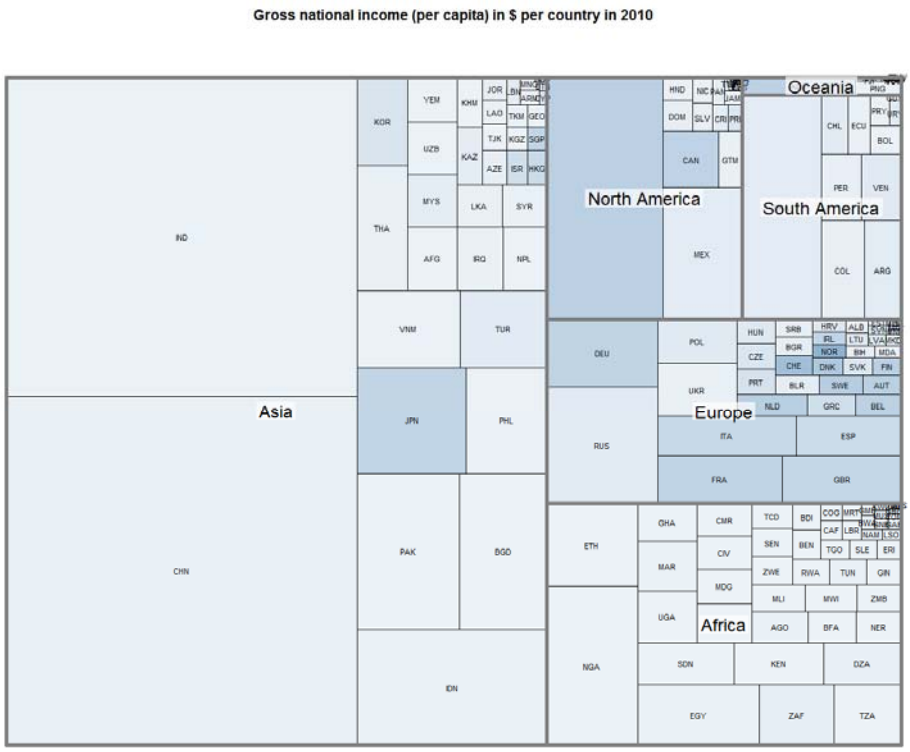



## 14.4 PlotTreemap

This function produces a treemap.

```
# get some data
data(GNI2010, package="treemap")
gn <- GNI2010[,c("iso3","population","continent","GNI")]
gn <- gn[gn$GNI!=0,]

# define a color
gn$col1 <- SetAlpha("steelblue", LinScale(gn$GNI, newlow=0.1, newhigh=0.6))


b <- PlotTreemap(x=gn$population, grp=gn$continent, col=gn$col1, labels=gn$iso3,
                main="Gross national income (per capita) in $ per country in 2010",
                labels.grp=NA, cex=0.7)
```

```
# get the midpoints
mid <- do.call(rbind, lapply(lapply(b, "[", 1), data.frame))

# and write the continents' text
DrawBoxedText(x=mid$grp.x, y=mid$grp.y, labels=rownames(mid), cex=1.5, bold=TRUE,
              border=NA, col=SetAlpha("white",0.7) )
```




# 15 Supplements to base R plots

## 15.1 Lineplots

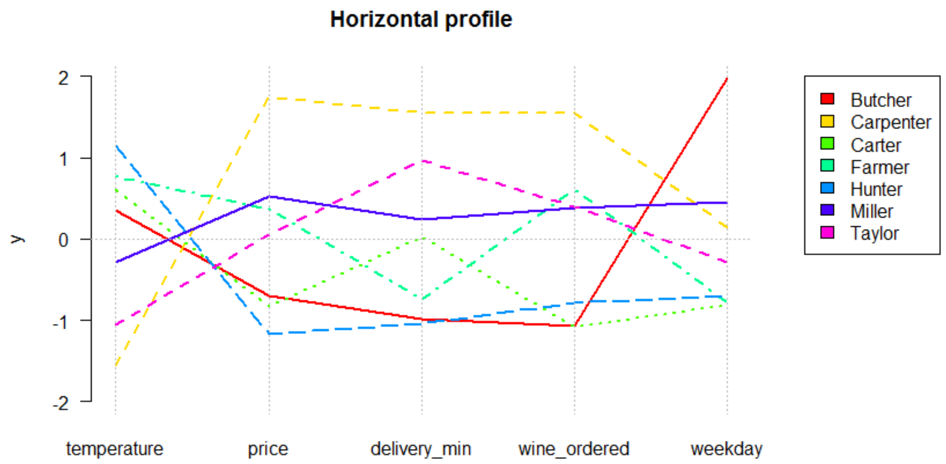
There are many flavours of line plots. Most (all?) of them can be handled by the function `matplot`.
We generally desist from defining own functions, that only set suitable arguments for another already existing function, as we fear we would run into a forest of new functions, loosing overview.

Yet the parametrization of `matplot` can be a haunting experience and so we integrate some common examples here in the sense of a “How-To” tutorial.

Let’s for example have a horizontal profile of the driver’s characteristics.

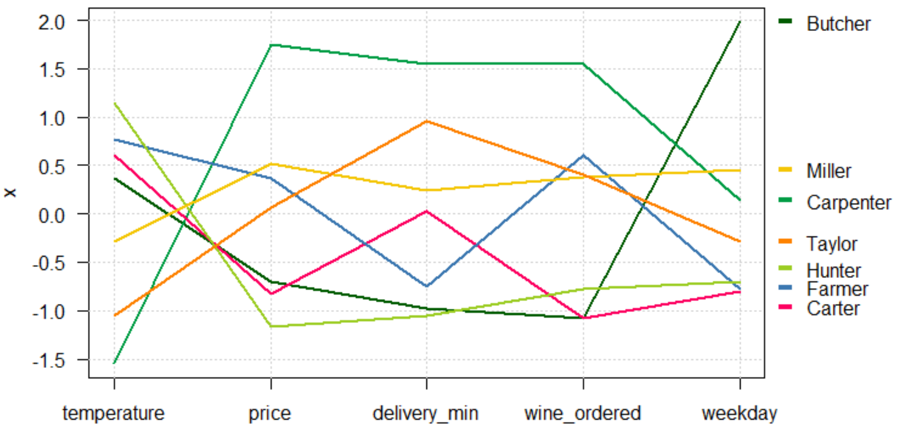
```
m <-
data.frame(lapply(d.pizza[,c("temperature","price","delivery_min","wine_ordered","weekday")],
                tapply, d.pizza$driver, mean, na.rm=TRUE))
(ms <- data.frame(lapply(m, scale)))         # lets scale that
```

```
          temperature        price  delivery_min  wine_ordered     weekday
Butcher      0.3605689 -0.69917381   -0.98046684    -1.0738446   1.9826284
Carpenter   -1.5481318  1.74805901    1.54851320     1.5445402   0.1389367
Carter       0.6105633 -0.82596309    0.02841316    -1.0840337  -0.8062020
Farmer       0.7718643  0.36562860   -0.74842415     0.6105001  -0.7800183
Hunter       1.1473246 -1.16829499   -1.04738479    -0.7792855  -0.7038441
Miller      -0.2918676  0.52072004    0.23662429     0.3794541   0.4596817
Taylor      -1.0503216  0.05902424    0.96272512     0.4026695  -0.2911825

x <- 1:ncol(ms)
y <- t(ms)

windows(8.8,5)
par(mar=c(5,4,4,10)+.1)
matplot(x, y, type="l", col=rainbow(nrow(ms)), xaxt="n", las=1, lwd=2, frame.plot=FALSE,
ylim=c(-2,2),
        xlab="", main="Horizontal profile")
abline(h=0, v=1:5, lty="dotted", col="grey")
par(xpd=TRUE)
legend(x=5.5, y=2, legend=rownames(ms), fill=rainbow(nrow(ms)))
axis(side=1, at=1:5, labels=colnames(ms), las=1, col="white")
```




The same, but with less code and a nifty and better readable legend at the right side.

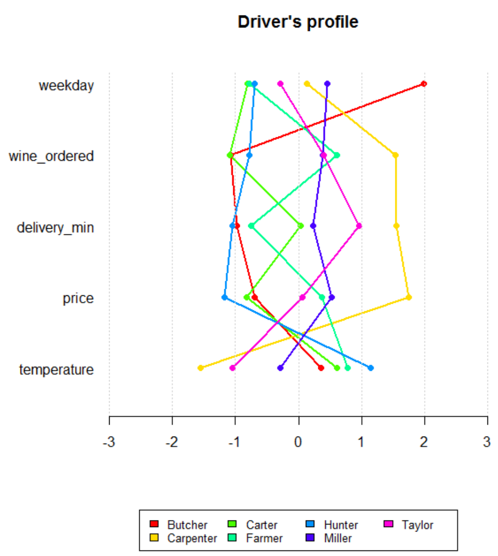
```
PlotLinesA(t(ms), col=PalTibco(), lwd=2)
```

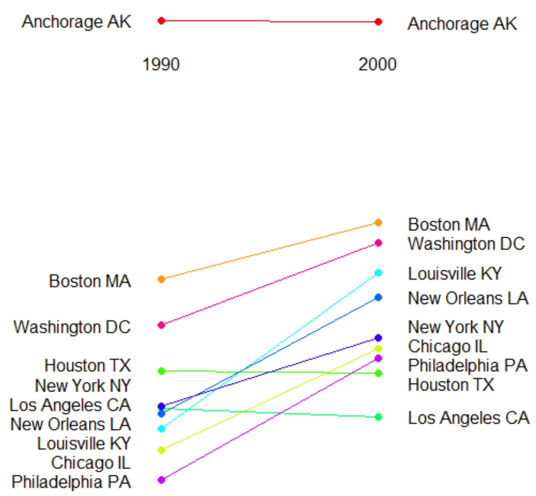



Andri/2016-04-27

And again the same, but on the vertical axis. (A)

```
par(mar=c(8,8,5,2))
matplot(x=y, y=x, type="l", pch=1:5, frame.plot=FALSE, axes=FALSE, xlab="", ylab="",
lty="solid",
        col=rainbow(nrow(ms)), xlim=c(-3,3), ylim=c(0.5,ncol(ms)), main="Driver's profile",
lwd=2)
matpoints(x=y, y=x, col=rainbow(nrow(ms)), pch=16)
grid(ny=NA)
axis(side=1, las=1)
mtext(colnames(ms), side=2, at=1:ncol(ms), las=2)
par(xpd=TRUE)
legend(x=0, y=-1, legend=rownames(ms), fill=rainbow(nrow(ms)), xjust=0.5, ncol=4, cex=0.8)
```




**A)** **B)**

## 15.2 "Bumpchart"

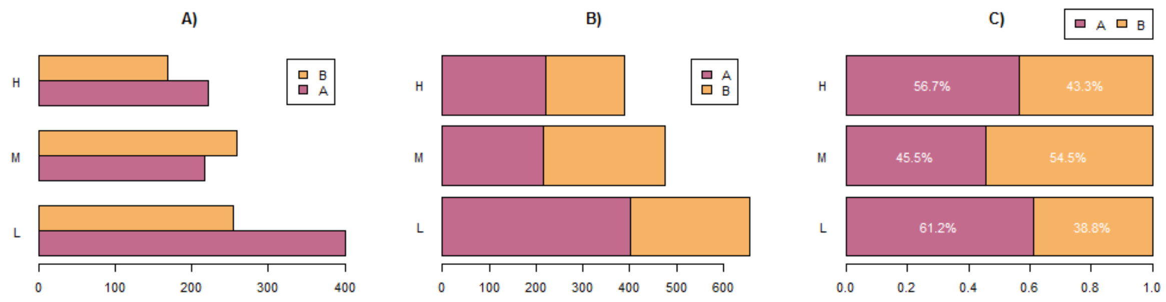
Plot B is sometimes called bumpchart (Jim Lemon).

```
# example from plotrix (bumpchart)
edu <- matrix(c(90.4,90.3,75.7,78.9,66,71.8,70.5,70.4,68.4,67.9,
                67.2,76.1,68.1,74.7,68.5,72.4,64.3,71.2,73.1,77.8), ncol=2, byrow=TRUE)
rownames(edu) <- c("Anchorage AK","Boston MA","Chicago IL",
                   "Houston TX","Los Angeles CA","Louisville KY","New Orleans LA",
                   "New York NY","Philadelphia PA","Washington DC")
colnames(edu) <- c(1990,2000)

par(mar=c(5,10,5,10))
matplot(x=1:2, y=t(edu), type="l", frame.plot=FALSE, axes=FALSE, xlab="",
        ylab="", lty="solid", col=rainbow(10))
matpoints(x=1:2, y=t(edu), pch=16, frame.plot=FALSE, axes=FALSE, xlab="",
          ylab="", lty="solid", col=rainbow(10))

sapply( 1:2, function(i) mtext(rownames(edu), side=2*i,
                               at=SpreadOut(edu[,i], mindist=1.1), line=1, las=1 ))
mtext(colnames(edu), side=3, at=1:2, line=-3.5, las=1 )
```

## 15.3 Barplot horizontal

A simple barplot, once with absolute values, once with percentages.

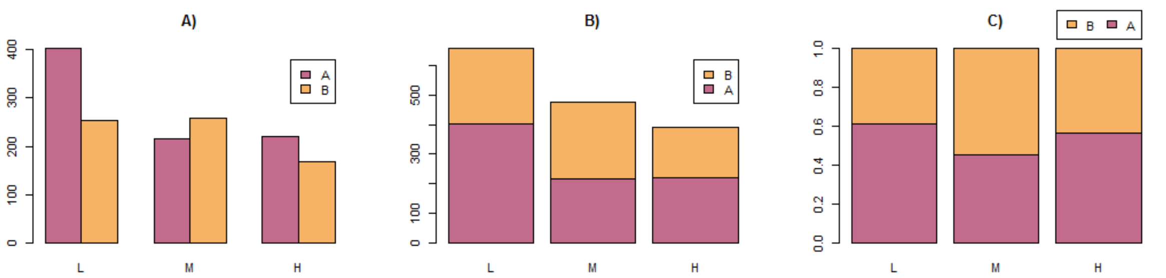
```
windows(height=3, width=11); par(mfrow=c(1,3))
col <- SetAlpha(PalHelsana(), 0.6)
tab <- matrix(c(401,216,221,254,259,169), nrow=2, byrow=TRUE,
       dimnames=list(wool=c("A","B"), tension=c("L","M","H")))
ptab <- prop.table(tab, 2)

# A)
barplot(tab, beside = TRUE, horiz=TRUE, main="A)",
        col = col[1:2], las = 1, legend = rownames(tab))
# B)
barplot(tab, beside = FALSE, horiz=TRUE, main="B)",
        col = col[1:2], las = 1, legend = rownames(tab))
# C)
b <- barplot(ptab, beside = FALSE, horiz=TRUE, main="C)",
             col = col[1:2], las = 1, legend.text = rownames(tab),
             args.legend = list(x=1, y=4.4, bg="white", ncol=2))

x <- t(apply(ptab, 2, Midx, incl.zero=TRUE, cumulate=TRUE))
text(Format(t(ptab), fmt="%", digits=1), x=x, y=b, col="white")
```




## 15.4 Barplot vertical

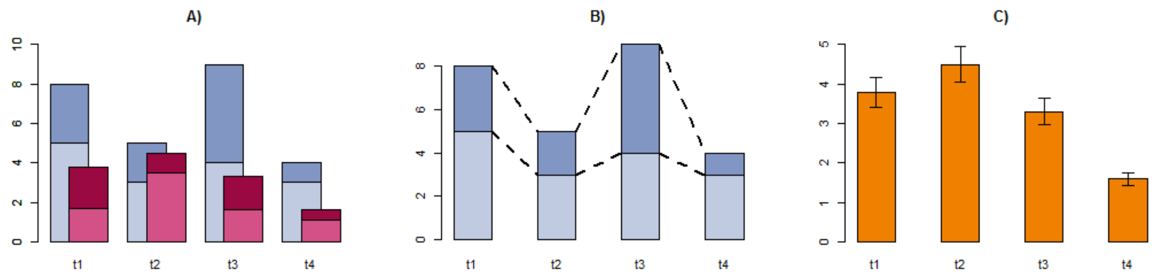
This same as above but with vertical bars.

```
# A)
barplot(tab, beside = TRUE, main="A)",
        col = col[1:2], legend = rownames(tab))
# B)
barplot(tab, beside = FALSE, main="B)",
        col = col,  legend = rownames(tab))
# C)
barplot(ptab, beside = FALSE, main="C)",
        col = col, legend.text = rownames(tab),
        args.legend = list(x=3.6, y=1.2, bg="white", ncol=2))
```



## 15.5 Barplot (specials)

Some specials like overlapping bars, connecting lines or error bars in combination with a barplot.

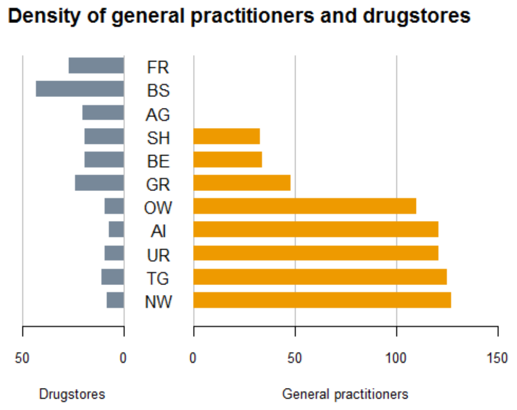



```
windows(height=3,11)
par(mfrow=c(1,3))

# A) Overlapping bars -----------------------------
blue <- rbind(c(5, 3, 4, 3),
              c(3, 2, 5, 1))
dimnames(blue) <- list(c("A","B"),c("t1","t2","t3","t4"))
red <- rbind(c(1.7,3.5,1.6,1.1),
              c(2.1,1.0,1.7,0.5))
dimnames(red) <- list(c("A","B"),c("t1","t2","t3","t4"))

# Set parameters
osp <- 0.5                # overlapping part in %
sp <- 1                   # spacing between the bars

nbars <- dim(blue)[2]   # how many bars do we have?

# Create first barplot
b <- barplot( blue, col=SetAlpha(hblue, c(0.5,1)), main="A)"
              , beside=FALSE, ylim=c(0,10), axisnames=FALSE
              , xlim=c(0, nbars*2-osp )       # enlarge x-Axis
              , space=c(0, rep(sp, nbars-1) ) # set spacing=1, starting with 0
             )
# Draw the red series
barplot( red, col= c(PalHelsana()[5], hred), beside=FALSE
         , space=c(1-osp, rep(1, nbars-1)) # shift to right by 1-osp
         , axisnames=FALSE, add=TRUE)

# Create axis separately, such that labels can be shifted to the left
axis(1, labels=colnames(red), at=b+(1-osp)/2, tick=FALSE, las=1)


# B) Connecting lines -----------------------------
barplot(blue, col= SetAlpha(hblue, c(0.5,1)), space=1.2, main="B)" )
ConnLines(blue, lwd=2, lty="dashed", space=1.2)


# C) Add error bars -------------------------------
cred <- apply(red, 2, sum)
b <- barplot(cred, col=horange, space=1.2, ylim=c(0,5), main="C)" )
ErrBars(from=cred * .90, to=cred * 1.1, pos=b)
```

## 15.6 PlotPyramid

A special kind of horizontal barplot is a "pyramid plot", where the bars are plotted back to back. This is sometimes needed, when your boss has specific and strict ideas how his presentation should look like.

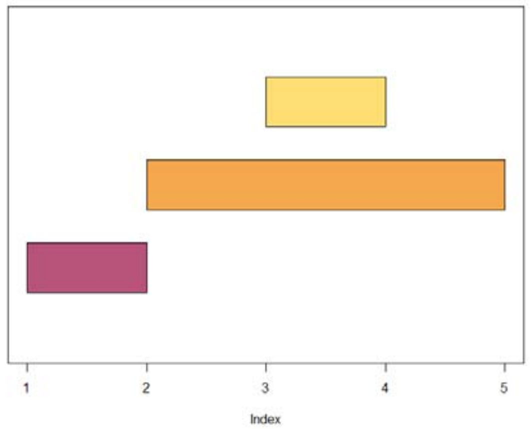
```
d.sda <- data.frame(
  kt_x  = c("NW","TG","UR","AI","OW","GR","BE","SH","AG","BS","FR"),
  apo_n = c(  8,  11,   9,   7,   9,  24,  19,  19,  20,  43,  27 ),
  sda_n = c(127, 125, 121, 121, 110,  48,  34,  33,   0,   0,   0 ))

PlotPyramid(lx=d.sda[,c("apo_n","sda_n")], ylab=d.sda$kt_x,
            col=c("lightslategray", "orange2"), border = NA, ylab.x=0, xlim=c(-110,250),
            gapwidth = NULL, cex.lab = 0.8, cex.axis=0.8, xaxt = TRUE,
            lxlab="Drugstores", rxlab="General practitioners",
            main="Density of general practitioners and drugstores",
            space=0.5, args.grid=list(lty=1))
```




## 15.7 PlotHorizBar

This is a simple function for plotting flowing horizontal or vertical bars.

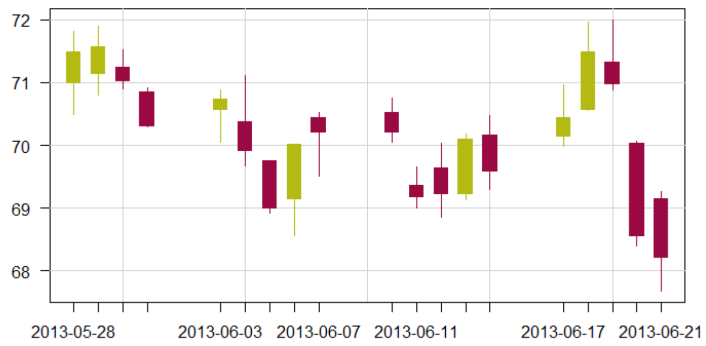
```
PlotHorizBar(from=c(1,2,3), to=c(2,5,4), grp=c(1,2,3), col=PalHelsana()[1:3])
```



## 15.8 PlotCandlestick

This plot is used primarily to describe price movements of a security, derivative or currency over time. Candlestick charts are a visual aid for decision making in stock, foreign exchange, commodity, and option trading.

```
example(PlotCandlestick)
PlotCandlestick(x=as.Date(rownames(nov)), y=nov, border=NA, las=1, ylab="")
```

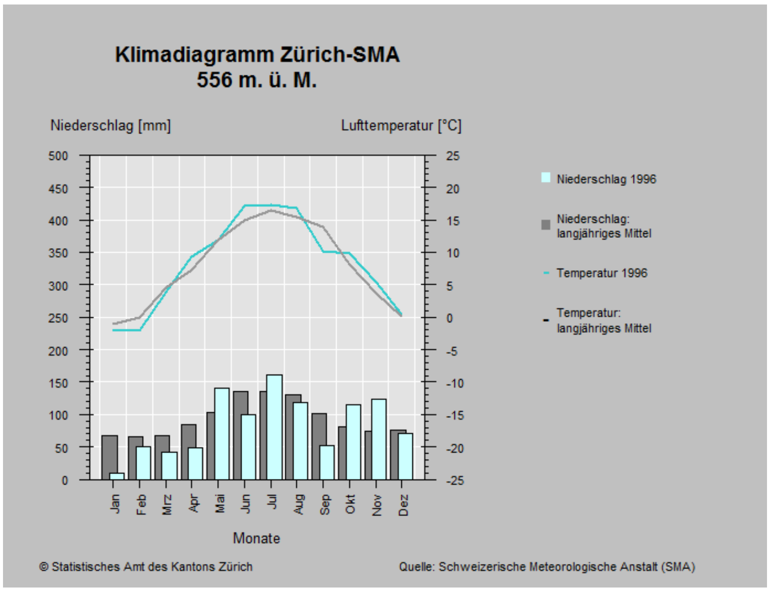



## 15.9 Combination of barplot and lineplot

It’s normally not recommended to use two axes, resp. combine two plots into one. However for displaying clima diagrams, consisting of a rain barplot and a temperature lineplot, this type is quite popular and often seen.
The used plot has a few special format features, that cost me much of time to find a solution. This includes the rug with positive and negative parts, the outside legend, the two axes with a suitable dimensions and the colouring of the background.

```
# get some data
d.temp <- data.frame(
  month=c("Jan","Feb", "Mrz", "Apr", "Mai", "Jun", "Jul", "Aug", "Sep", "Okt", "Nov", "Dez")
  ,nieder_96=c(9, 50, 41, 49, 141, 99, 161, 119, 52, 115, 123, 70)
  ,nieder_mittel=c(67, 65, 67, 85, 103, 135, 136, 130, 101, 81, 74, 76)
  ,temp_96=c(-1.9, -2.1, 3.8, 9.3, 11.8, 17.1, 17.3, 16.8, 10.2, 9.8, 5.4, 0.5)
  ,temp_mittel=c(-1, 0, 4.5, 7.3, 11.9, 15, 16.5, 15.5, 13.9, 8.1, 3.7, 0.2)
)

# define a few colors
hellblau <- rgb(red=204,green=255,blue=255, max=255)
dunkelblau <- rgb(red=51,green=204,blue=204, max=255)
dunkelgrau <- rgb(red=128,green=128,blue=128, max=255)
mittelgrau <- rgb(red=192,green=192,blue=192, max=255)
hellgrau <- rgb(red=227,green=227,blue=227, max=255)

# set the parameters
windows(width=7.2, height=5.5)
par(mar=c(5.1,4.1,7.1,16.1)) # set margins, default: c( 5.1, 4.1, 4.1, 2.1 )
par(bg=mittelgrau) # background color

# start plotting, we use barplot as basis
b <- barplot( t(d.temp[,c("nieder_mittel","nieder_96")])
              , col=c(dunkelgrau, hellblau)
              , beside=TRUE , xlab="Monate", cex.lab=0.8, mgp=c(2.2,0.7,0)
              , space=rep( c(0.3,-0.5), 12) # bars should overlap 50%
              , ylim=c(0,500), yaxt="n"
              , panel.first = {
                par(xpd=FALSE) # barplot paints over the whole figure region by default
                usr <- par("usr") # set background color lightgrey
                rect(xleft=usr[1], ybottom=usr[3], xright=usr[2], ytop=usr[4], col=hellgrau)
```

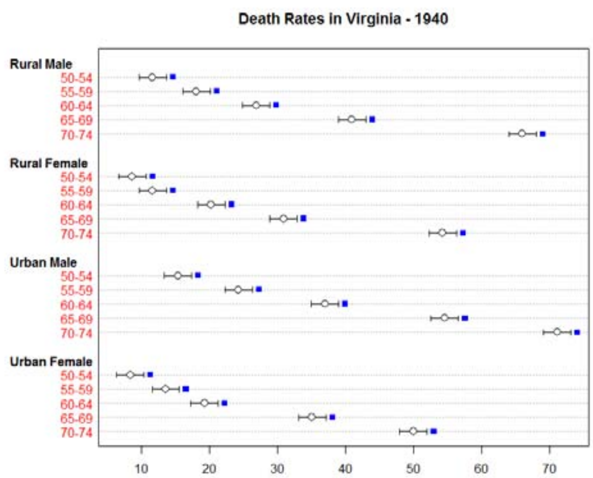
```
                grid(nx=NA, ny=10, col="white", lty="solid") # horiz grid only
                box()
              }
)

# find the centers of the bars and the gaps
barx <- apply(b, 2, FUN=mean)
run.mean <- filter( barx, filter=c(0.5,0.5))[-length(barx)]
gapx <- c(run.mean[1]-diff(barx)[1], run.mean, run.mean+diff(barx) )


# draw the vertikal gridlines
abline(v=gapx, col="white" )
box()

# design x-axis
axis(side=1, at=apply(b,2,FUN=mean), labels=d.temp$month, cex.axis=0.7
     , las=2, tck=-0.025  # no tickmarks for the x-axis
     , mgp=c(2.2,0.7,0) ) # decrease distance label to axis

# left y-axis
axis(side=2, at=seq(0,500,50), las=2, cex.axis=0.7)
rug( seq(0,500,10), side=2, ticksize=-0.01)
rug( seq(0,500,50), side=2, ticksize= 0.01)

# plot lines
par(new=TRUE)
matplot( x=barx, y=d.temp[,c("temp_96","temp_mittel")], col=c(dunkelblau,"grey60")
         , lwd=2, lty="solid", type="l", xaxt="n", yaxt="n", xlab="", ylab=""
         , xlim=par("usr")[1:2] # use the current xlim
         , ylim=c(-25, 25), xaxs="i", yaxs="i")

# design right axis
axis(side=4, labels=seq(-25,25,5), at=seq(-25,25,5), las=2, cex.axis=0.7)
rug( seq(-25,25,1), side=4, ticksize=-0.01)
rug( seq(-25,25,5), side=4, ticksize=0.01)

# write titles
mtext(text=c("Lufttemperatur [°C]","Niederschlag [mm]"), side=3, at=c(25,-3.2), adj=c(1,0)
      , las=1, line=1, cex=0.8 )
mtext(text="Klimadiagramm Zürich-SMA\n556 m. ü. M.", cex=1.2, font=2, side=3, line=3)

# plot legend
legend( x=30, y=27, xpd=TRUE
        , legend=c("Niederschlag 1996", "Niederschlag:\nlangjähriges Mittel", "Temperatur
1996", "Temperatur:\nlangjähriges Mittel" )
        , cex=0.7, bty="n", col=c(hellblau, dunkelgrau, dunkelblau, "black")
        , y.intersp=2.5, pt.cex=1.2, pch=c(15,15,45,45))

mtext("© Statistisches Amt des Kantons Zürich", side=1, line=3.5, at=-4, cex=0.7, las=1,
adj=0)
mtext("Quelle: Schweizerische Meteorologische Anstalt (SMA)", side=1, line=3.5, at=41,
cex=0.7, las=1, adj=1)
```




## 15.10 PlotDot

The base function dotchart has been improved but still has some potential for extensions. Especially an add argument is sometimes useful and returning the y-coordinates for the points would allow to add elements.
PlotDot implements these extensions and allows adding error bars. This is interesting, as the calculation of the x-limits should be done with respect to the bars and not only to the points.

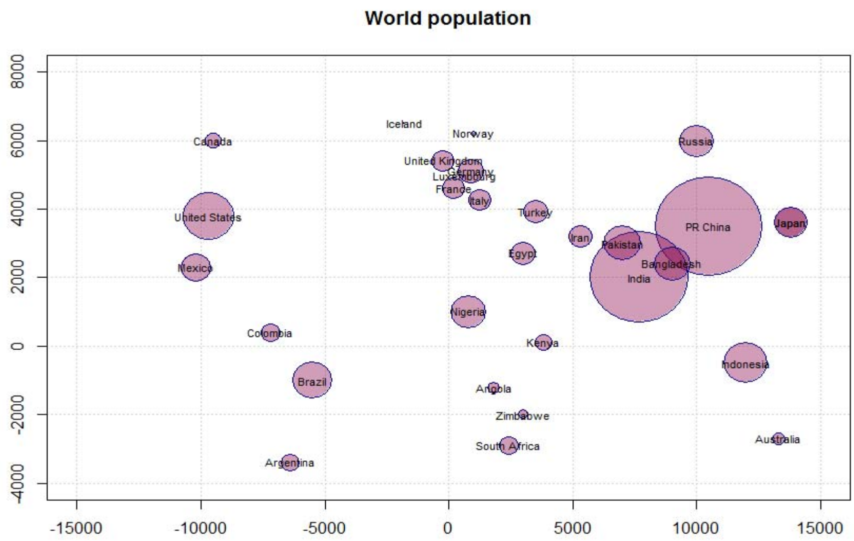
```
# add some error bars
PlotDot(VADeaths, main="Death Rates in Virginia - 1940", col="red", pch=NA,
        args.errbars = list(from=VADeaths-2, to=VADeaths+2, mid=VADeaths,
        pch=21, cex=1.4))

# add some other values
PlotDot(VADeaths+3, pch=15, col="blue", add=TRUE, labels=NA)
```



## 15.11 PlotBubble

Bubbles can actually easily be produced with the standard plot function. This function here helps you defining appropriate axis limits.

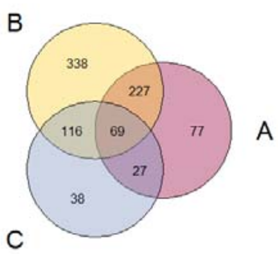
```
PlotBubble(d.world$x, d.world$y, area=d.world$pop/90, col=SetAlpha("deeppink4",0.4),
border="darkblue",
           xlab="", ylab="", panel.first=grid(), main="World population")
text(d.world$x, d.world$y, labels=d.world$country, cex=0.7, adj=0.5)
```



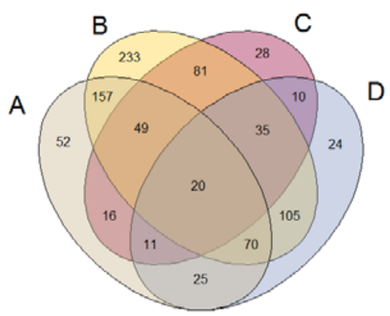

## 15.12 Venn plots

Now and then one might want to plot a Venn diagram. This function does this for up to 5 datasets using the simple proposed geometric representations.
(For more than 5 datasets the Venn representation loses its simplicity and other plot types become more adequate.)

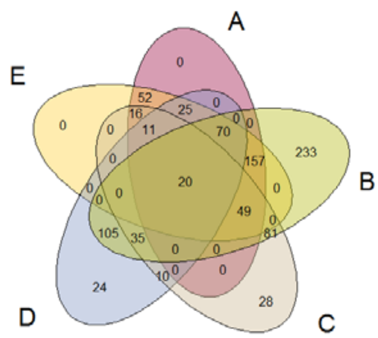
```
example(PlotVenn)
PlotVenn(x=x[1:3], col=SetAlpha(c(PalHelsana()[c(1,3,6)]), 0.4))
PlotVenn(x=x[1:4], col=SetAlpha(c(PalHelsana()[c(1,3,6,4)]), 0.4))
PlotVenn(x=x[1:5], col=SetAlpha(c(PalHelsana()[c(1,3,6,4,7)]), 0.4))
```




**3 sets**




**4 sets**




**5 sets**

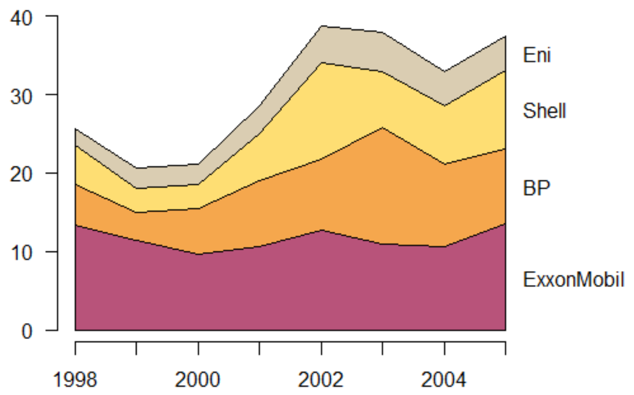
## 15.13 Areaplot

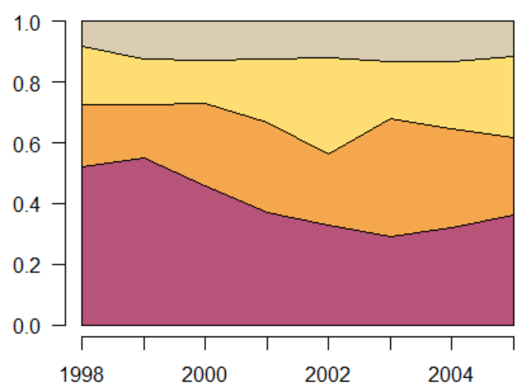
Areaplots have a high "ink factor"[1], say they use much ink to display the information and are therefore rarely the best way of representing data. But again, when your boss wants it this way, here's a function to produce it easily.

```
t.oil <- t(matrix(c(13.3,11.4, 9.7,10.6,12.7,11.0,10.6,13.5,
                   5.3, 3.6, 5.8, 8.4, 9.1,14.8,10.6, 9.6,
                   4.9, 3.1, 3.0, 6.0,12.2, 7.1, 7.3,10.0,
                   2.1, 2.6, 2.7, 3.5, 4.7, 5.0, 4.4, 4.3), nrow=4, byrow=TRUE,
   dimnames = list(c("ExxonMobil","BP","Shell","Eni"),
                     c("1998","1999","2000","2001","2002","2003","2004","2005"))))
t(t.oil)

par(mfrow=c(1,2), mar=c(5,4,5,5))
col <- SetAlpha(PalHelsana(), 0.7)
PlotArea(t.oil, col = col, las = 1, frame.plot=FALSE)
mtext(side=4, text=colnames(t.oil), las=1,
      at=Midx(tail(t.oil, 1)[,], incl.zero=TRUE, cumulate=TRUE))

PlotArea(prop.table(t.oil, 1), col = col, las = 1, frame.plot=FALSE)
```

```
tab (absolute values)
> t(t.oil)
           1998 1999 2000 2001 2002 2003 2004 2005
ExxonMobil 13.3 11.4  9.7 10.6 12.7 11.0 10.6 13.5
BP          5.3  3.6  5.8  8.4  9.1 14.8 10.6  9.6
Shell       4.9  3.1  3.0  6.0 12.2  7.1  7.3 10.0
Eni         2.1  2.6  2.7  3.5  4.7  5.0  4.4  4.3
```

```
ptab (relative values)
            1998  1999  2000  2001  2002  2003  2004  2005
ExxonMobil 0.520 0.551 0.458 0.372 0.328 0.290 0.322 0.361
BP         0.207 0.174 0.274 0.295 0.235 0.391 0.322 0.257
Shell      0.191 0.150 0.142 0.211 0.315 0.187 0.222 0.267
Eni        0.082 0.126 0.127 0.123 0.121 0.132 0.134 0.115
```



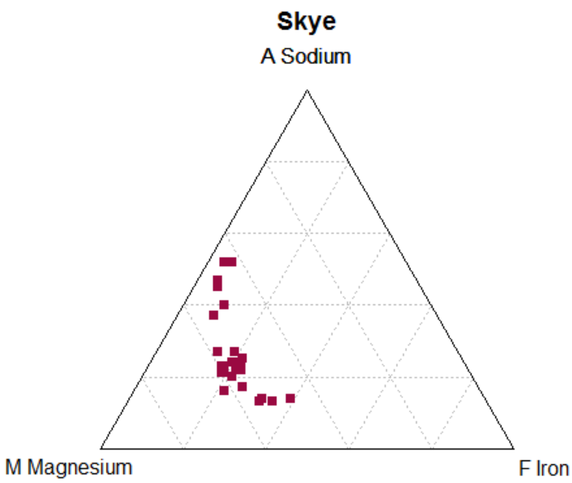




[1] Tufte, Edward R (2001) [1983], The Visual Display of Quantitative Information (2nd ed.), Cheshire, CT: Graphics Press, ISBN 0-9613921-4-2.

## 15.14 PlotTernary

This produces a ternary or triangular plot.

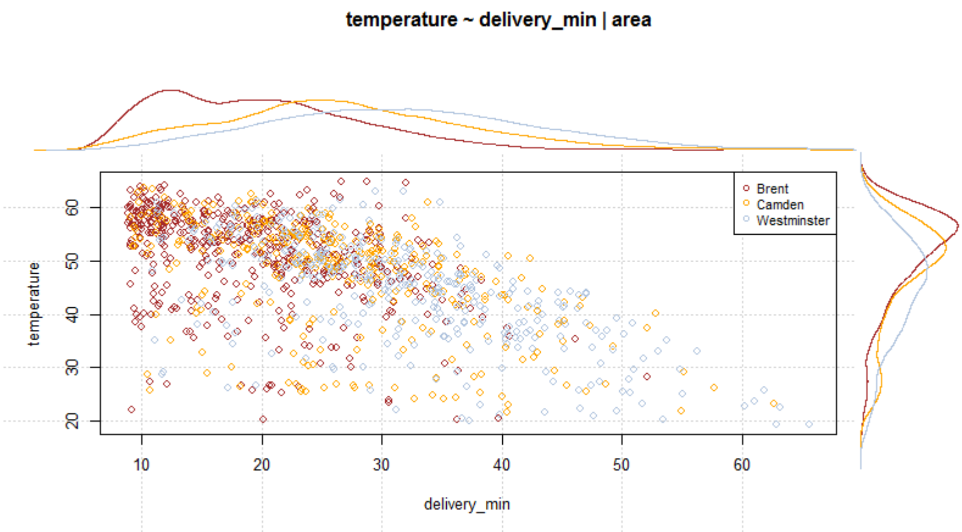
```
data(Skye, package="MASS")
PlotTernary(Skye[c(1,3,2)], pch=15, col=hred, main="Skye",
            lbl=c("A Sodium", "F Iron", "M Magnesium"))
```




## 15.15 PlotMarDens

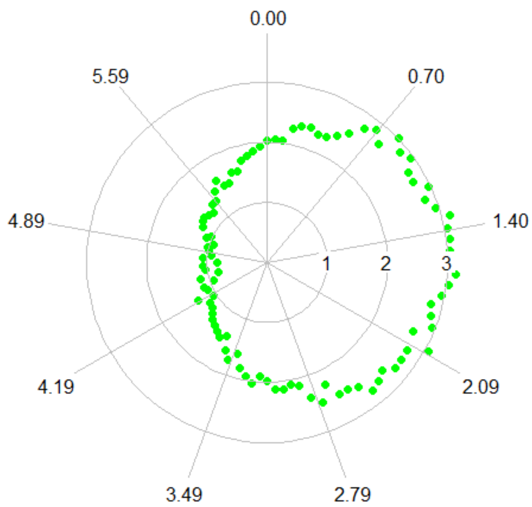
This plot shows a scatterplot of two numerical variables temperature and delivery_time, by area. On the margins the density curves of the specific variable are plotted, also stratified by area.

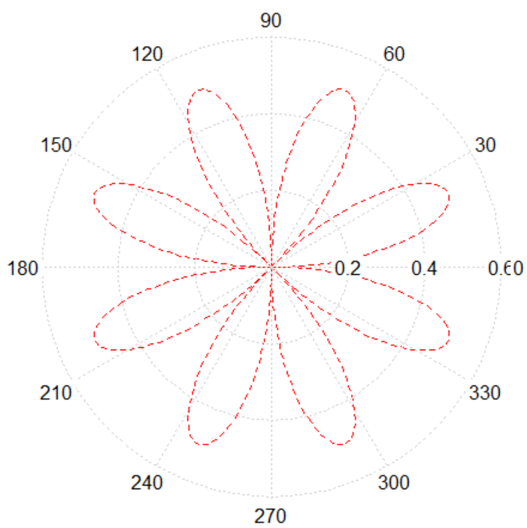
```
PlotMarDens(y=d.pizza$temperature, x=d.pizza$delivery_min, grp=d.pizza$area,
            xlab="delivery_min", ylab="temperature",
            col=c("brown","orange","lightsteelblue"), panel.first=grid(),
            main="temperature ~ delivery_min | area"  )
```



## 15.16 Polar plots

```
testlen <- c(sin(seq(0, 1.98*pi, length=100)) + 2 + rnorm(100)/10)
testpos <- seq(0, 1.98*pi, length=100)
# start at 12 o'clock and plot clockwise
PlotPolar(testlen, -(testpos - pi/2), type="p", main="Test Polygon", col="green", pch=16)

PolarGrid(ntheta = rev(seq(0, 2*pi, by=2*pi/9) + pi/2),
          alabels=Format(seq(0, 2*pi, by=2*pi/9),2)[-10], col="grey",
          lty="solid", lblradians=TRUE)

# just because of its beauty
t <- seq(0,2*pi,0.01)

PlotPolar(r=sin(2*t)*cos(2*t), theta=t, type="l", lty="dashed", col="red")
PolarGrid()
```



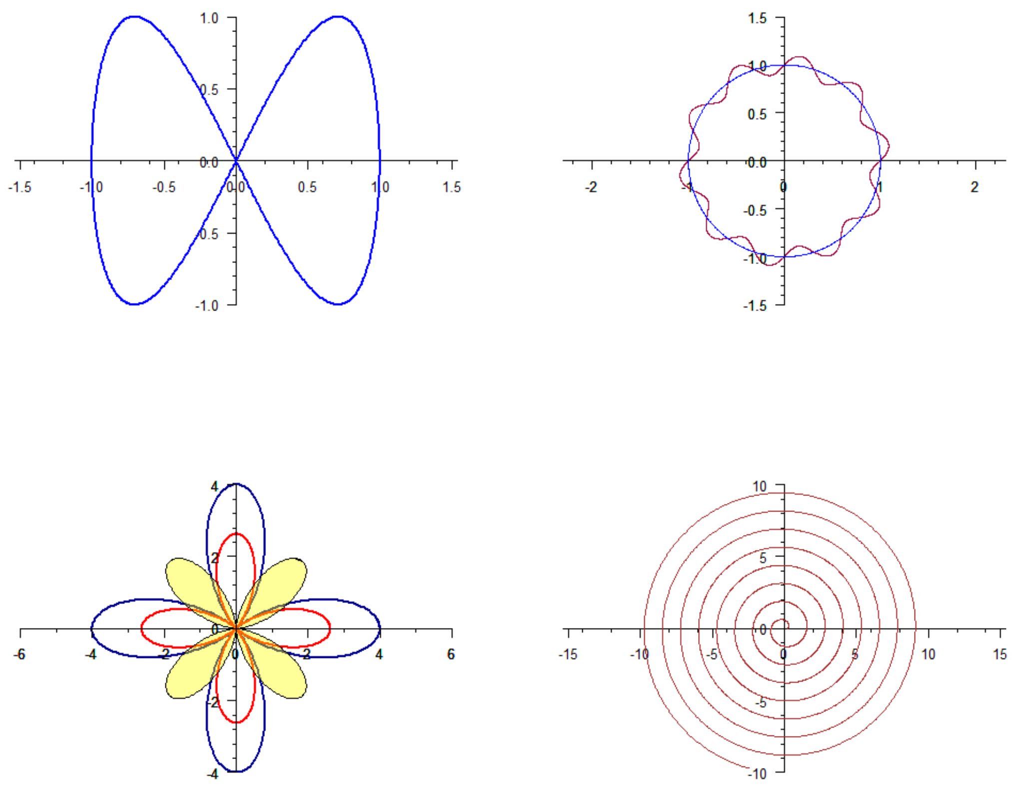

## 15.17 Plot Functions

Functions can be plotted a bit more comfortable by means of the function PlotFun. The idea behind it is to make use of the formula interface, for example x^2 ~ x, and let the function choose appropriate defaults for the rest. (This would be the best case scenario...;-).

There can as well be further parameters defined for plotting more than one function at once. Arguments as type="n" or add=TRUE are supported.
The function returns the calculated xy-coordinates as list. This can be used to modify the coordinates afterwards, e.g. rotate or translate them.

```
# get some data
par(mfrow=c(2,2))
PlotFun(sin(2*t) ~ sin(t), from=0, to=2*pi, by=0.01, col="blue", lwd=2)

PlotFun(1+ 1/10 * sin(10*x) ~ x, polar=TRUE, from=0, to=2*pi, by=0.001, col=hred)
# add a second curve with add=TRUE
PlotFun(sin(x) ~ cos(x), polar=FALSE, from=0, to=2*pi, by=0.01, add=TRUE, col="blue")


# lemniscate of Bernoulli
PlotFun((2*a^2*cos(2*t))^2 ~ t, args=list(a=1), polar=TRUE, from=0, to=2*pi+0.1, by=0.01,
        col="darkblue", lwd=2)
# add the second curve in red
PlotFun((2*a^2*cos(2*t))^2 ~ t, args=list(a=0.9), polar=TRUE, from=0, to=2*pi+0.1, by=0.01,
```

```
        col="red", lwd=2, add=TRUE)
# calculate points for a third curve, but do not yet plot it
z <- PlotFun((2*a^2*cos(2*t))^2 ~ t, args=list(a=0.9), polar=TRUE, from=0, to=2*pi+0.1,
              by=0.01, add=TRUE, type="n")
# rotate the structure by pi/4
zz <- Rotate(z$x, z$y, theta=pi/4)
# add a polygon for being able to fill it
polygon(x = zz$x, y=zz$y, col=SetAlpha("yellow", 0.4))

# evolving circle
PlotFun(a*(sin(t) - t*cos(t)) ~ a*(cos(t) + t*sin(t)), args=list(a=0.2), from=0, to=50,
        by=0.01, col="brown")
```



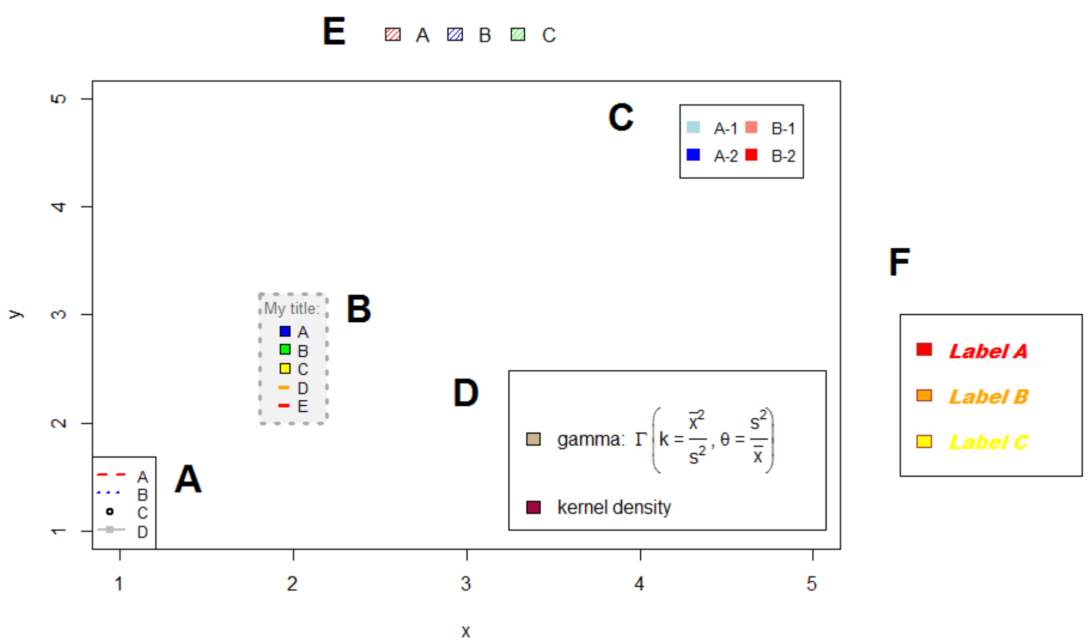
## 15.18 Legends and colour strips

The details of a legend can be challenging to define, respectively to find how to control. Think as well of the locator(), when a position should be placed by pointing and clicking.

Here are some examples of maybe nontrivial legends.

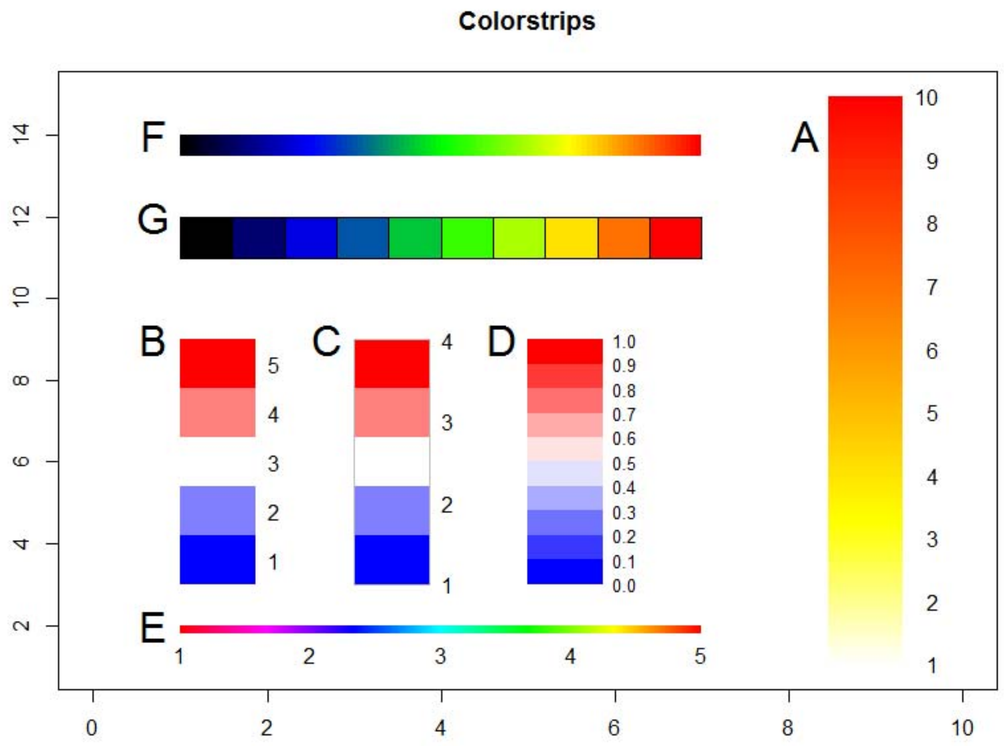
```
par(mar=c(5.1,4.1,4.1,11.1))
plot( x=1:5, y=1:5, type="n", xlab="x", ylab="y" )

# A) Combine lines and point characters **********
legend( x="bottomleft", inset=0.02, legend=c("A","B","C","D")
  , lty=c("dashed","dotted",NA,"solid"), lwd=2, cex=0.8
  , pch=c(NA,NA,21,15)
  , col=c("red","blue","black","grey"), bg="white" )

# B) Combine colours and lines **********
```

```
legend( x=2, y=2, xjust=0.5, yjust=0
  , title=" My title:", title.col="grey40", title.adj=0
  , legend=c("A","B","C","D","E")
  , pch=c(22,22,22,45,45), pt.cex=c(1.2,1.2,1.2,2,2)
  , col=c(rep("black",3),"orange","red")
  , pt.bg=c("blue","green","yellow")
  , bg="grey95", cex=0.8
  , box.col="darkgrey", box.lwd=3, box.lty="dotted" )

# C) 2 columns **********
legend("topright", inset=0.05, cex=0.8, bg="white"
  , legend=c("A-1","A-2","B-1", "B-2")
  , col=c("lightblue","blue","salmon","red"), pch=15, pt.cex=1.5
  , y.intersp=1.5, x.intersp=1.5 , ncol=2 )

# D) formula **********
legend(x="bottomright", inset=c(0.02, .04)
  , legend=c(expression(plain("gamma:  ") * Gamma * " " * bgroup("(", k * " = " *
      over(bar(x)^2, s^2) * " , " * theta * plain(" = ") * over(s^2, bar(x)), ")") ),
           "kernel density")
  , fill=c(hecru, getOption("col1", hred)), text.width = 1.5)

# E) outside the plot area **********
legend( x=2, y=6, legend=c("A","B","C")
  , fill=c("red","blue","green")
  , density=30, bty="n", horiz=TRUE
  , xpd=TRUE )

# F) change fonts **********
windowsFonts("sans2"="Arial Black")
usr <- par(font=4, family="sans2" )
legend( x=5.5, y=3, legend=c("Label A","Label B","Label C")
  , fill=c("red","orange","yellow")
  , border="brown"
  , y.intersp=2, text.width=strwidth("Make larger")
  , text.col=c("red","orange","yellow")
  , xpd=TRUE )
par(usr)
```



The function ColorLegend produces colour strips, which often are needed for colour coded maps.

```
plot(1:15,, xlim=c(0,10), type="n", xlab="", ylab="", main="Colorstrips")

# A
ColorLegend(x="right", inset=0.1, labels=c(1:10))

# B: Center the labels
ColorLegend(x=1, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(5), labels=1:5, cntrlbl = TRUE)

# C: Outer frame
ColorLegend(x=3, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(5), labels=1:4, frame="grey")

# D
ColorLegend(x=5, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(10), labels=sprintf("%.1f",seq(0,1,0.1)), cex=0.8)

# E: horizontal shape
ColorLegend(x=1, y=2, width=6, height=0.2, col=rainbow(500), labels=1:5,horiz=TRUE)

# F
ColorLegend(x=1, y=14, width=6, height=0.5, col=colorRampPalette(
  c("red","yellow","green","blue","black"), space = "rgb")(100), horiz=TRUE)

# G
ColorLegend(x=1, y=12, width=6, height=1, col=colorRampPalette(c("red","yellow",
  "green","blue","black"), space = "rgb")(10), horiz=TRUE, border="black")
```



# 16 Format, Strings and Date functions

## 16.1 Formatting numbers and dates

Number formatting can sometimes be a nightmare in base R. The function Format tries to concentrate as much as possible form the functionality of formatC, format, symbol, pval etc. into one simple interface.

The following example will use a space as big mark, align the numbers on the position of the "e", flip to scientific notation for numbers < $10^{-2}$ and for such > $10^{4}$ and use 3 fixed digits for all numbers.

```
x <- pi * 10^(-5:7)
cbind(Format(x, big.mark=" ", align="e", sci=c(5,2), digits=3))

##       [,1]
##  [1,] "     3.142e-05"
##  [2,] "     3.142e-04"
##  [3,] "     3.142e-03"
##  [4,] "     0.031    "
##  [5,] "     0.314    "
##  [6,] "     3.142    "
##  [7,] "    31.416    "
##  [8,] "   314.159    "
##  [9,] " 3 141.593    "
## [10,] "31 415.927    "
## [11,] "     3.142e+05"
## [12,] "     3.142e+06"
## [13,] "     3.142e+07"
```

Engineering format, set with fmt = "eng", will snap to powers of multiples of 3 when using scientific notation.

```
Format(x, fmt="eng", leading="00", digits=2)

##  [1] "31.42e-06"  "314.16e-06" "03.14e-03"  "31.42e-03"  "314.16e-03" "03.14e+00"
##  [7] "31.42e+00"  "314.16e+00" "03.14e+03"  "31.42e+03"  "314.16e+03" "03.14e+06"
## [13] "31.42e+06"
```

Formatting dates use format codes "d" for days, "m" for months etc.

```
Format(as.Date(c("2014-11-28", "2014-1-2")), fmt="ddd, d mmmm yyyy")
## [1] "Fri, 28 November 2014" "Thu, 2 January 2014"

Format(Today(), fmt="dddd, dd.mm.yyyy")
## [1] "Thursday, 26.05.2016"

Format(Today(), fmt="dddd, yy/mm/dd")
## [1] "Thursday, 16/05/26"

Format(Today(), fmt="dddd, yy/mm/dd", lang="loc")   # with local language
## [1] "Donnerstag, 16/05/26"
```

The format code "p" will produce formatted p-values and is a simple wrapper for format.pval.

```
Format(c(0.442, 0.02125, 4e-21), fmt="p")
## [1] "0.44200"   "0.02125"   "< 2.2e-16"
```

Significance stars mimics the function symnum.

```
Format(c(0.4, 0.02, 0.0004), fmt="*")
## [1] "    " "*   " "***"
```

When formatting percentages the function Format will multiply the numbers with 100, round them to the given number of fixed digits and append a "%"sign.
A sometimes suitable alternative format could be to drop the leading zeros.

```
Format(c(0.24534, 0.4512345, 1.347), fmt="%", digits=2)
## [1] "24.53%"  "45.12%"  "134.70%"

Format(c(0.24534, 0.4512345, 1.347), leading="drop", digits=2)
## [1] ".25"  ".45"  "1.35"
```

NAs and zeros must sometimes be formatted specially. Think eg. of sparse matrices, where one would like the 0s being displayed as "." or maybe even not at all "".

```
Format(c(3.45, 451.2345, 0, NA), digits=2, na.form="<NULL>", zero.form="-")
## [1] "3.45"   "451.23" "-"      "<NULL>"
```

Alignment can be done directly within the function. There are 3 special codes supported, left alignment with "\\l", centered with "\\c" and right with "\\r".

```
cbind(Format(cumsum(10^(0:6)), align="\\c", digits=0))
     [,1]
[1,] "   1   "
[2,] "  11   "
[3,] "  111  "
[4,] " 1111  "
[5,] " 11111 "
[6,] "111111 "
[7,] "1111111"
```

## 16.2 Date functions

Many date functions are presumably thought to be reached via format and some subsequent cast in base R. However in the analyst's daily life it's often convenient to be able to directly extract parts of a date. So DescTools contains the following ones:

| | |
|---|---|
| day.name, day.abb | Defined names of the days |
| AddMonths, AddMonthsYM | Add a number of months to a given date |
| IsDate | Check whether x is a date object |
| IsWeekend | Check whether x falls on a weekend |
| IsLeapYear | Check whether x is a leap year |
| LastDayOfMonth | Return the last day of the month of the date x |
| DiffDays360 | Calculate the difference of two dates using the 360-days system |
| Date | Create a date from numeric representation of year, month, day |
| Day, Month, Year | Extract part of a date |
| Hour, Minute, Second | Extract part of time |
| Week, Weekday | Returns ISO week and weekday of a date |
| Quarter | Quarter of a date |
| YearDay, YearMonth | The day in the year of a date |
| Now, Today | Get current date or date-time |
| HmsToSec, SecToHms | Convert h:m:s times to seconds and vice versa |
| Zodiac | The zodiac sign of a date :-) |

## 16.3 Strings

String functions are scattered in base R and the solution for some daily tasks are sometimes hard to find. Experts will solve most of their daily life string manipulation with regular expressions. But beginners and a big part of advanced users are supposed to profit by a set of basic string functions.

| | | |
|---|---|---|
| nchar | the length of a string, say the number of characters | Base |
| tolower | convert to lower case | base |
| toupper | convert to upper case | base |
| StrCap | capitalize the first letter of a string | DescTools |
| StrAbbr | abbreviates a string | DescTools |
| abbreviate | abbreviation | base |
| StrTrunc | truncate string on a given length and add ellipses if it really was truncated | DescTools |
| StrTrim | delete white spaces from a string | DescTools |
| StrPad | fill a string with defined characters to fit a given length | DescTools |
| StrRev | reverse a string | DescTools |
| paste | concatenate strings | base |
| strrep | repeat the elements of a character vector | base |
| strwrap | wrap character strings to format paragraphs | base |
| chartr | character translation | base |
| substr | extract or replace substrings in a character vector, but only with position indices, not with regexp patterns | base |
| (substring) | (substring is compatible with S Plus) | |
| StrChop | split a string by a fixed number of characters. | DescTools |
| strsplit | splitting regex matches split vector according to matches | base |
| StrCountW | count the words in a string | DescTools |
| StrVal | extract numeric values from a string | DescTools |
| StrPos | find position of first occurrence of a string in another one | DescTools |
| StrIsNumeric | check whether a string does only contain numeric data | DescTools |
| FixToTab | create table out of a running text, by using columns of spaces as delimiter | DescTools |
| StrDist | compute Levenshtein or Hamming distance between strings | DescTools |
| grep | regex matches which elements are matched and returns the index or value (argument value=TRUE) | base |
| grepl | same, but returns logical vector (TRUE & FALSE) | base |
| regexpr | regex matches positions of **the first** match | base |
| gregexpr | same for **all** matches | base |
| (regexec) | (regex matches hybrid of regexpr and gregexpr) | base |
| sub | replacing regex matches **only first** match is replaced | base |
| gsub | same, but **all** matches are replaced | base |
| strwidth | compute the width of the given string on the current plotting device | graphics |
| strheight | same with height | Graphics |
| noquote | | |

| | |
|---|---|
| cat, print | |
| “replace” | x <- c("may", "the", "force", "be", "with", "you")<br>substr(x, 2, 2) <- "#" |
| StrExtract | extracts found matches |

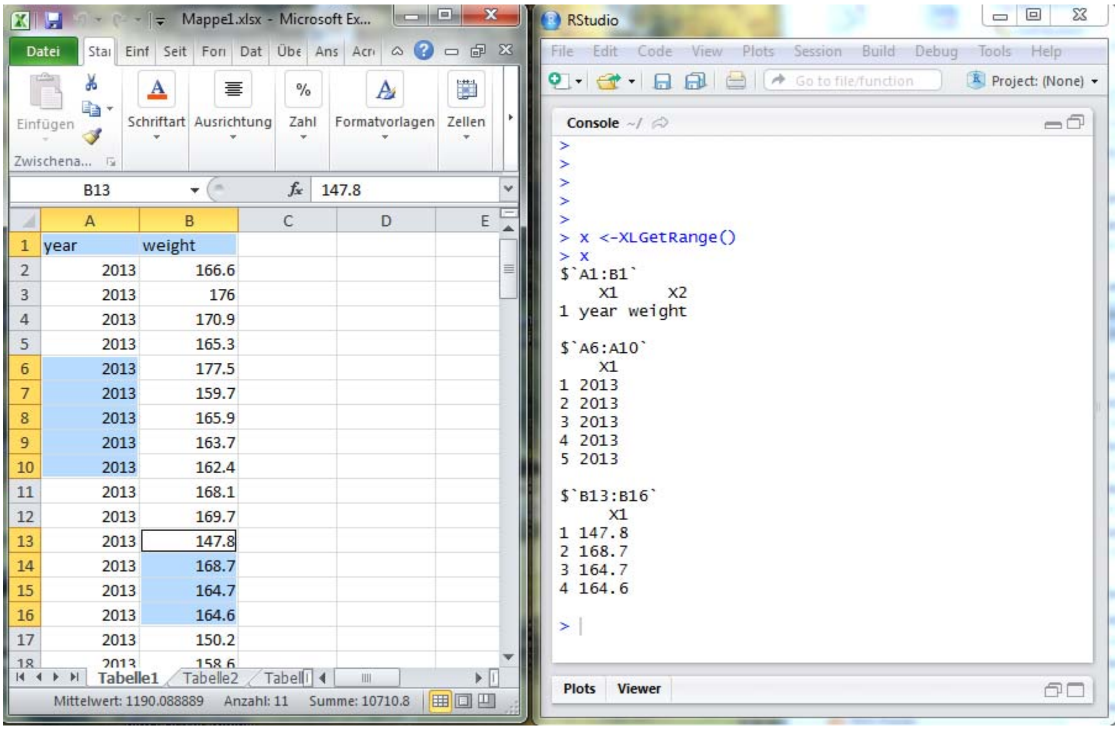
HowTo

| Question | Answer |
|---|---|
| Does y contain x? | `grepl(x, y)` |
| Extract “ab..” from c(“uiabex”,”ummjeabxy”) | `StrExtract(x=c("uiabexz","umeabxymj"), pattern="ab..")`<br>`## "abex", "abxy"` |

# 17 Import – Export

## 17.1 Import data via Excel

The function XLGetRange allows a quick import of data from an Excel-Sheet. The user can either specify a number of cell-references (including a path- and filename) or just select the regions which are to be imported.
The following command will return a list with the contents of the selected cell ranges.




XLView(d.frm) can be used to view a data.frame d.frm in Excel.

### 17.2 Import SAS datalines

The function ParseSASDatalines can be used to import the SAS data like the following:

```
sas <- "
  data FatComp;
  input Exposure Response Count;
  label Response='Heart Disease';
  datalines;
    0 0  6
    0 1  2
    1 0  4
    1 1 11
 ;"

(FatComp <- ParseSASDatalines(sas))

  Exposure Response Count
1        0        0     6
2        0        1     2
3        1        0     4
4        1        1    11
```

# 18 DescToolsOptions

There are a few options for the graphical or textual output that can be set. `DescToolsOptions()` displays the currently defined options.

```
$col
    hblue      hred    hgreen
"#8296C4" "#9A0941" "#B3BA12"

$digits
[1] 3

$fixedfont
$name
[1] "Consolas"

$size
[1] 7

attr(,"class")
[1] "font"

$fmt
$fmt$abs
Format name:    abs
 Description:   Number format for counts
 Definition:    digits=0, big.mark="'"
 Example:       314'159

$fmt$num
Format name:    num
 Description:   Number format for floats
 Definition:    digits=3, big.mark="'"
 Example:       314'159.265


$footnote
[1] "¹" "²" "³"

$lang
[1] "engl"

$plotit
[1] TRUE
```

```
$stamp
expression(gettextf("%s/%s", Sys.getenv("USERNAME"), Format(Today(),
    fmt = "yyyy-mm-dd")))

$lastWrd
NULL

$lastXL
NULL

$lastPP
NULL
```

Invoking `DescToolsOptions()` with no arguments returns a list with the current values of the options. Note that not all options listed below are set initially. To access the value of a single option, one can simply use `DescToolsOptions("plotit")`.

To set a new value use the same rationale as with the R options:
`DescToolsOptions(plotit=FALSE)`

`col:`
: a vector of colours, defined as names or as RGB-longs ("#RRGGBB"). By now three colors are used in several plots as defaults. By default they're set to hred, hblue and horange. Change the values by defining DescToolsOptions(col=c("pink", "blue", "yellow")). Any color definition can be used here.

`digits:`
: the number of FIXED digits, used throughout the print functions.

`fixedfont:`
: this font will be used by default, when `Desc` writes to a Word document. Must be defined as a font object, say enumerating name, face and size of the font and setting the class font, e.g. `structure(list(name="Courier New", size=7), class="font")`.

`fmt:`
: Three number format definitions are currently used in the `Desc` routines. The format used for integer values is named `"abs"`, for percentages `"perc"` and for floating point numeric values `"num"`. The format definitions must be of class `"fmt"` and may contain any argument used in the function Format.
Use `Fmt` to access and update formats (as they are organised in a nested list). See the current definitions with:

```
Format(pi*1000, fmt=Fmt("abs"))
# [1] "3'142"
Format(pi*.1, fmt=Fmt("per"))
# [1] "31.4%"
Format(pi*1000, fmt=Fmt("num"))
# [1] "3'141.593"
```

`footnote:`
: a character vector, containing characters to be used as footnote signs. Any character can be defined here. This is currently used by `TOne`.
The author's favorites: `DescToolsOptions("footnote"=c("¹","²","³"))`

`lang:`
: either `"engl"` or `"local"`, defining the language to be used for the names of weekdays and months when using Format.

`plotit:`

logical, defining whether the Desc-procedures should produce plots by default. This is usually a good thing, but it may clutter up your desktop, if you're not using RStudio. Therefore it can be turned off.

`stamp:`

text or expression to be placed in the right bottom corner of the DescTools plots. This can be useful, if some author or date information should be inserted by default. The default would use an expression as <username>/<date>. See defaults below.

Calling `DescToolsOptions(reset=TRUE)` will reset the options to these defaults:

```
options(DescTools = list(
  col       = c(hblue="#8296C4", hred="#9A0941", hgreen="#B3BA12"),
  digits    = 3,
  fixedfont = structure(list(name = "Consolas", size = 7), class = "font"),
  fmt       = list(abs = structure(list(digits = 0, big.mark = "'"),
                     name = "abs", label = "Number format for counts", default = TRUE,
                     class = "fmt"),
                   per = structure(list(digits = 1, big.mark = "%"),
                     name = "per", label = "Percentage number format", default = TRUE,
                     class = "fmt"),
                   num = structure(list(digits = 3, big.mark = "'"),
                     name = "num", label = "Number format for floats", default = TRUE,
                     class = "fmt")
              ),
  footnote  = c("'", "\"", "\"\""),
  lang      = "engl",
  plotit    = TRUE,
  stamp     = expression(gettextf("%s/%s", Sys.getenv("USERNAME"),
                                  Format(Today(), fmt = "yyyy-mm-dd")))
))
```

This code can as well be copied and pasted to the users' RProfile file, in order to have the options permanently available.

# 19 References

http://cran.r-project.org/web/packages/vcdExtra/vignettes/vcd-tutorial.pdf

http://www.stattutorials.com/SAS/TUTORIAL-PROC-FREQ-1.htm

Vittinghoff, E., Glidden, D. V., Shiboski, S. C., and Mcculloch, C. E. (2005) Regression Methods in Biostatistics: Linear, Logistic, Survival, and Repeated Measures Models. Springer, New York

http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf

http://www.stat.ufl.edu/~presnell/Courses/sta4504-2011sp/Notes/icda-notes-3x2.pdf

Agresti, A. (2002) Categorical Data Analysis. John Wiley & Sons.

Dalgaard, P. (2008) Introductory Statistics with R (2. Aufl.), London, UK: Springer.

Rajul Parikh et. al. Understanding and using sensitivity, specificity and predictive values, 2008 Indian J Ophthalmol. Jan-Feb; 56(1): 45–50.

Wollschläger, D. (2010, 2012) Grundlagen der Datenanalyse mit R, Springer, Berlin.

Tufte, Edward R (2001) [1983], The Visual Display of Quantitative Information (2nd ed.), Cheshire, CT: Graphics Press