DescTools

A Hardworking Assistant for Describing Data

by Andri Signorell
Helsana Versicherungen AG, Health Sciences, Zurich
HWZ University of Applied Sciences in Business Administration, Zurich
andri@signorell.net

September 21, 2014

Abstract

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. There are facilities like summary, describe (Hmisc), stat.desc (library pastecs), but all of them are lacking some functionality or flexibility we would have expected.
Another point is, that there are quite a bit of commonly used functions, which curiously are not present in the *stats* package, think e.g. of skewness, kurtosis but also the Gini-coefficent 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 such 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 an office of an insurance 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).

This document describes quickly the essentials of the package DescTools.

Content

Describing a data.frame

The function `Desc` is designed to describe all the variables of a data.frame with some reasonable statistic measures and an adequate graphic representation. It includes code for describing

- factors, ordered and unordered
- numeric variables
- integer variables (typically counts)
- dates
- logical variables
- tables and matrices

A formula interface is implemented allowing to describe variables by 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 plot=TRUE argument to have the full results in the console.

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)
```

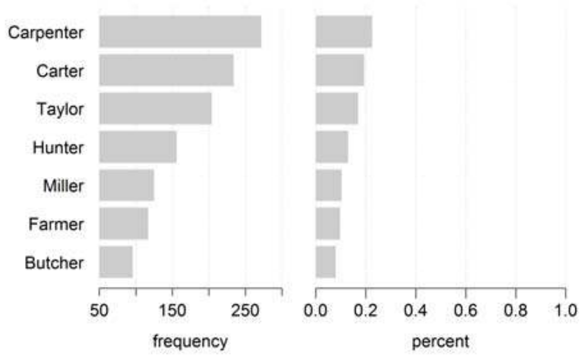
```
'data.frame':   1209 obs. of  4 variables:
 1 $ driver      : Factor w/ 7 levels "Butcher","Carpenter",..: 7 1 1 7 3 7 7 7 7 3 ...
 2 $ temperature : num  53 56.4 36.5 NA 50 27 33.9 54.8 48 54.4 ...
 3 $ count       : int  5 2 3 2 5 1 4 NA 3 6 ...
 4 $ weekday     : num  6 6 6 6 6 6 6 6 6 6 ...
 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, but with enumerated variables.

After that, every single variable will be described according to the type of its class.
The first variable is an unordered `factor`. Factors are typically best described 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","asc","name","level"`.
The frequency table is truncated in the case that there are more than a dozen values (this can be avoided by setting the argument `maxrows=NA`, see: `?Desc.factor` for more details).

1 : driver (factor)

```
  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    1108    .920
7   Butcher   96 .080    1204   1.000
```



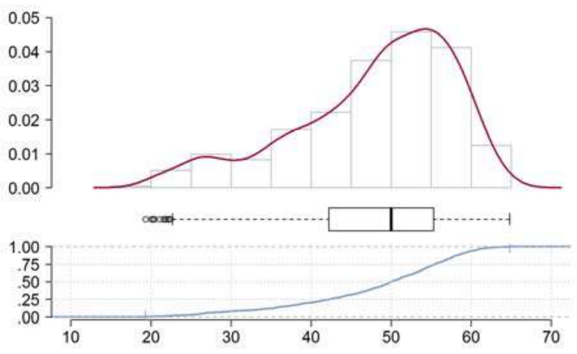
Synopsis

`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 values. Note that this is not the same as levels, as there might be empty levels. Thus the number of levels might be higher than the number of unique values (but not the other way round).
`dupes`	y(es) or n(o), reporting if there are any duplicate values in the vector. If "n" then there are only unique values in the variable.
`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 next variable, the temperature of the delivered pizza, is numeric. Numeric variables are described by the most usual statistical measures for location, variation and shape.

2 : temperature (numeric)

```
 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

    rng     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

Shapiro-Wilks normality test  p.value : < 2.22e-16
```



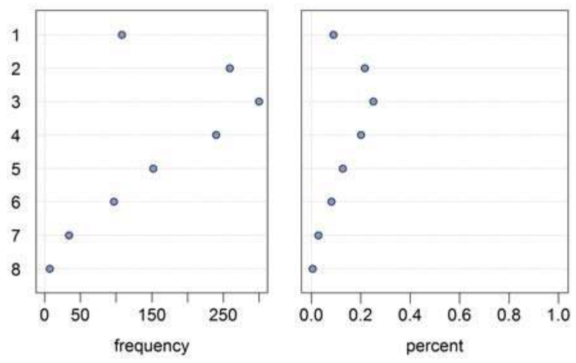
The first measures length, n, NAs, unique have the same meaning as above.

`0s`	total number of 0s, say zero values.
`mean`	the mean of the vector, NAs are silently removed.
`meanSE`	standard error of the mean, sd(x) / sqrt(n). 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)`. (See also: function `MeanCI(…)`)
`.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. Note that, if there are bindings here, the frequency of each value will be reported in brackets.
`highest`	same as lowest, but on the other end
`Shapiro-Wilks`	performs a Shapiro-Wilks normality test and reports its p-value. Shapiro-Wilks will be replaced by the Anderson-Darling-Test, if `length(x) > 5000`.
`plot`	the plot combines a histogram with a density plot, a boxplot and the ecdf-plot, as produced by the function `PlotFdist`.

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.

3 : count (integer)

```
  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

Shapiro-Wilks normality test  p.value : < 2.22e-16


  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
```



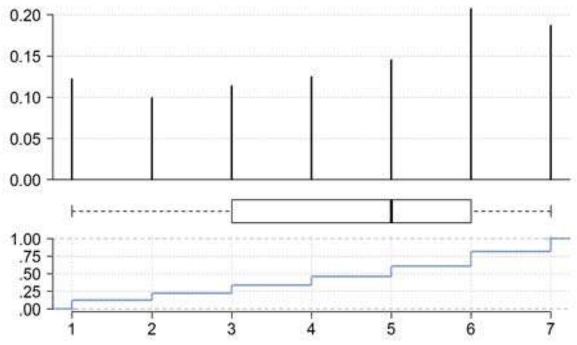
`plot`	the plot is produced as a (horizontal) dotchart. More than 12 unique values are truncated (a warning is placed in the plot area). Use maxrows argument to override this default (NA for all).

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 (...).

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. Therefore we change the graphic representation from a histogram to a histogram like h-type plot without density curve.

4 : weekday (numeric)

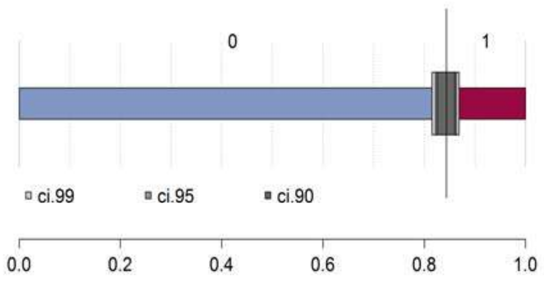
```
  length       n    NAs  unique     0s    mean  meanSE
   1'209   1'177     32       7      0   4.441   0.059

     .05     .10    .25  median    .75     .90     .95
       1       1      3       5      6       7       7

     rng      sd  vcoef     mad    IQR    skew    kurt
       6   2.019  0.455   2.965      3  -0.345  -1.170

lowest : 1 (144), 2 (117), 3 (134), 4 (147), 5 (171)
highest: 3 (134), 4 (147), 5 (171), 6 (244), 7 (220)

Shapiro-Wilks normality test  p.value : < 2.22e-16
```



The variable `wine_ordered` contains only two values, 0 and 1. Dichotomous variables do not have real dense (univariate) information. But still it is 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`".

5 : wine_ordered (integer)

```
  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
```

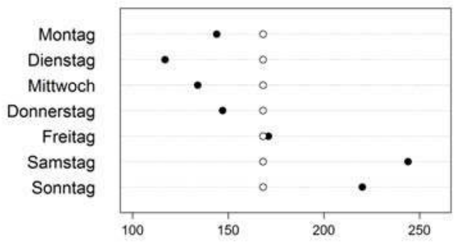


`plot`	this is basically a univariate horizontal stacked barplot, with confidence intervals on the confidence levels of 0.90, 0.95 and 0.99. The vertical line denominates the point estimator.

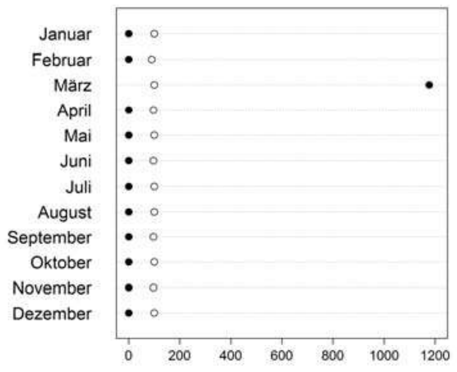
A date variable is harder to describe. What characteristics would one want to know from a date? We would normally choose a description similar to numeric values, supplemented by an analysis of the weekday and month.

6 : date (Date)

```
  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
```

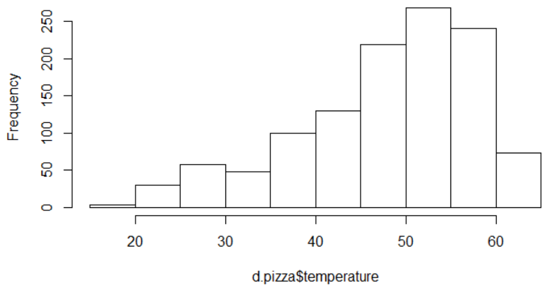


Simple Frequencies

There are as well more atomic functions in the package. The function `Freq` is designed to give the numeric representation of a histogram, produced by `hist`. It displays the frequencies and the percentages of a binned variable with the same logic as in `hist`. The single and cumulative frequencies values are reported.

```
Freq(d.pizza$temperature)
hist(d.pizza$temperature)
```

```
     level  freq  perc  cumfreq  cumperc
1  [15,20]     3 0.003        3    0.003
2  (20,25]    30 0.026       33    0.028
3  (25,30]    58 0.050       91    0.078
4  (30,35]    48 0.041      139    0.119
5  (35,40]   100 0.085      239    0.204
6  (40,45]   130 0.111      369    0.315
7  (45,50]   219 0.187      588    0.503
8  (50,55]   268 0.229      856    0.732
9  (55,60]   241 0.206     1097    0.938
10 (60,65]    73 0.062     1170    1.000
```



Pairwise descriptions

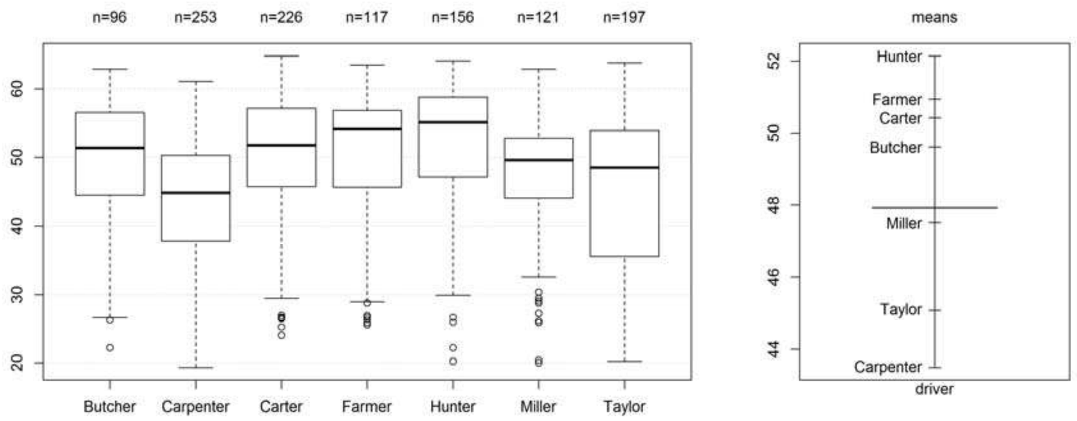
`Desc` implements a formula interface allowing to define bivariate descriptions straight forward.

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)
```

temperature ~ driver(numeric ~ categorical)

```
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%).
```



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

Two categorical variables are described by a contingency table. Again the total pairs, the valid pairs and the missings are again reported first.

Two numerical variables have no obvious standard description as their relationship can have many forms. We report therefore only the simple correlation coefficients (Pearson, Spearman and Kendall).

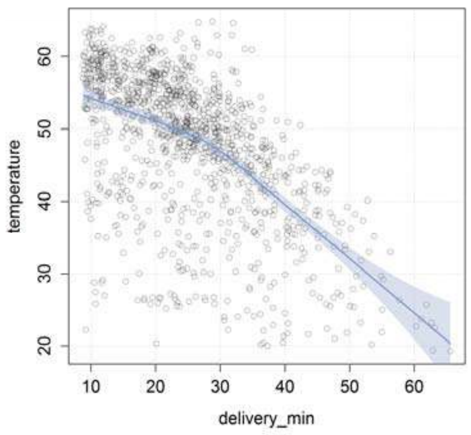
The variables are plotted as xy-scatterplots with interchanging mutual dependency, supplemented with a LOESS smoother.

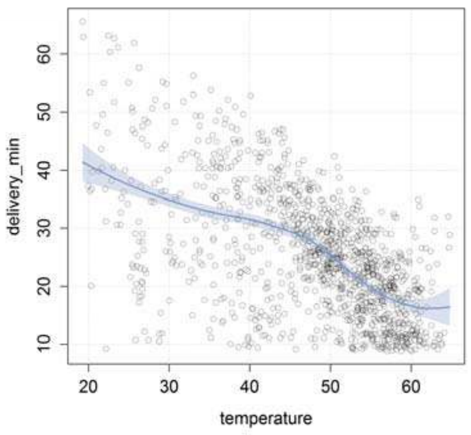
```
Desc(temperature ~ delivery_min, d.pizza, plotit=TRUE)
```

temperature ~ delivery_min (numeric ~ numeric)

```
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:





Two categorical variables are described by a contingency table, as shown in the next chapter.

Tables

Get the frequencies and the percentages of a RxC-dimensional contingency table and output a flat table. Expected values and standardized residuals can be computed.

```
# A)
PercTable(d.pizza$driver, d.pizza$city, margins=c(1,2), rfrq="101")
# B)
PercTable(d.pizza$driver, d.pizza$city, margins=c(1,2), rfrq="000", expected=TRUE, stdres=TRUE,
          digits=1)
```

A) Frequencies and percentages

```
                Brent  Camden Westminster    Sum

Butcher   freq     72       1          22     95
          perc   .060    .001        .018   .080
          p.col  .152    .003        .058   .080
Carpenter freq     29      19         221    269
          perc   .024    .016        .185   .225
          p.col  .061    .056        .582   .225
Carter    freq    177      47           5    229
          perc   .148    .039        .004   .192
          p.col  .374    .138        .013   .192
Farmer    freq     19      87          11    117
          perc   .016    .073        .009   .098
          p.col  .040    .255        .029   .098
Hunter    freq    128       4          24    156
          perc   .107    .003        .020   .131
          p.col  .271    .012        .063   .131
Miller    freq      6      41          77    124
          perc   .005    .034        .064   .104
          p.col  .013    .120        .203   .104
Taylor    freq     42     142          20    204
          perc   .035    .119        .017   .171
          p.col  .089    .416        .053   .171
Sum       freq    473     341         380   1194
          perc   .396    .286        .318  1.000
          p.col 1.000   1.000       1.000  1.000
```

B) Expected values and std. residuals

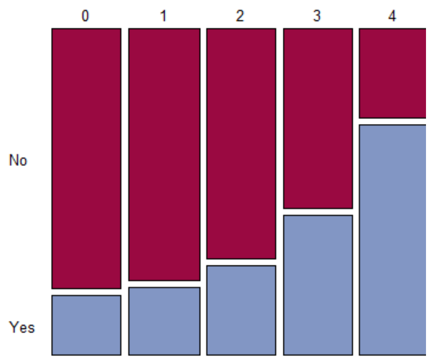
```
                Brent  Camden Westminster   Sum

Butcher   freq     72       1          22    95
          exp    37.6    27.1        30.2     .
          stdres  7.5    -6.2        -1.9     .
Carpenter freq     29      19         221   269
          exp   106.6    76.8        85.6     .
          stdres -11.0   -8.9        20.1     .
Carter    freq    177      47           5   229
          exp    90.7    65.4        72.9     .
          stdres 13.0    -3.0       -10.7     .
Farmer    freq     19      87          11   117
          exp    46.3    33.4        37.2     .
          stdres -5.4    11.5        -5.5     .
Hunter    freq    128       4          24   156
          exp    61.8    44.6        49.6     .
          stdres 11.6    -7.7        -4.7     .
Miller    freq      6      41          77   124
          exp    49.1    35.4        39.5     .
          stdres -8.4     1.2         7.6     .
Taylor    freq     42     142          20   204
          exp    80.8    58.3        64.9     .
          stdres -6.1    14.3        -7.4     .
Sum       freq    473     341         380  1194
          exp       .       .           .     .
          stdres    .       .           .     .
```

There are quite a few suggestions for the description of tables out there. We use a mix between SAS- and SPSS-flavour here. Let's take a SAS-example[1] and describe the table with SPSS-verbosity:

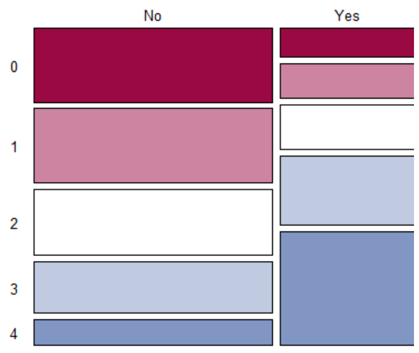
```
pain <- as.table(matrix(c(26,26,23,18, 9,
                           6, 7, 9,14,23), nrow=2, byrow=TRUE,
                        dimnames=list(treat=c("No","Yes"), strength=0:4) )
Desc(pain, verb="high", wrd=wrd)
```

Clinical Trial for Treatment of Pain (2x5-table)

```
Summary:
n: 161, rows: 2, columns: 5

Pearson's Chi-squared test:
  X-squared = 26.6025, df = 4, p-value = 2.392e-05
Likelihood Ratio:
  X-squared = 26.6689, df = 4, p-value = 2.319e-05
Mantel-Haenszel Chi-squared:
  X-squared = 22.8188, df = 1, p-value = 1.78e-06

                       estimate lwr.ci upr.ci
Phi Coeff.                .4065      -      -
Contingency Coeff.        .3766      -      -
Cramer V                  .4065  .2716  .5636
Goodman Kruskal Gamma     .5313  .3480  .7146
Kendall Tau-b             .3373  .2114  .4631
Stuart Tau-c              .4111  .2547  .5675
Somers D C|R              .4427  .2786  .6068
Somers D R|C              .2569  .1593  .3546
Pearson Correlation       .3776  .2368  .5029
Spearman Correlation      .3771  .2362  .5024
Lambda C|R                .1250  .0000  .2547
Lambda R|C                .2373  .0732  .4014
Lambda sym                .1604  .0388  .2821
Uncertainty Coeff. C|R    .0515  .0140  .0890
Uncertainty Coeff. R|C    .1261  .0346  .2175
Uncertainty Coeff. sym    .0731  .0199  .1262
Mutual Information        .1195      -      -
```



```
            strength      0      1      2      3      4    Sum
treat
No     freq              26     26     23     18      9    102
       perc            .161   .161   .143   .112   .056   .634
       p.row           .255   .255   .225   .176   .088  1.000
       p.col           .812   .788   .719   .562   .281   .634
Yes    freq               6      7      9     14     23     59
       perc            .037   .043   .056   .087   .143   .366
       p.row           .102   .119   .153   .237   .390  1.000
       p.col           .188   .212   .281   .438   .719   .366
Sum    freq              32     33     32     32     32    161
       perc            .199   .205   .199   .199   .199  1.000
       p.row           .199   .205   .199   .199   .199  1.000
       p.col          1.000  1.000  1.000  1.000  1.000  1.000
```



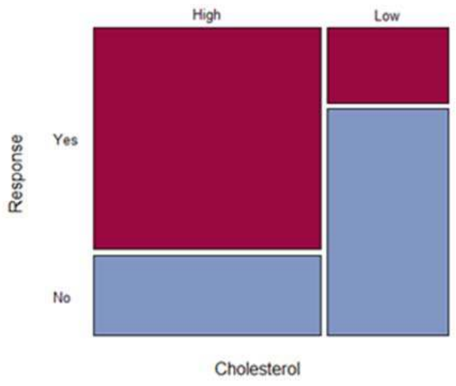
1 http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf, S. 1821

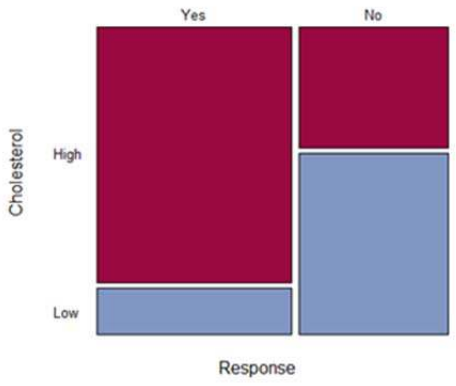
Let's have a look at a 2x2-table. We replace the ChiSquare-test by the exact Fisher test and report McNemar's symmetry test. The odds ratio and relative risk are displayed in addition by default. The verbosity can be set to “medium”, “low” and “high”.

```
heart <- as.table(matrix(c(11,4,
                            2,6), nrow=2, byrow=TRUE,
                         dimnames=list(Cholesterol=c("High","Low"), Response=c("Yes","No"))) )

Desc(heart, wrd=wrd, horiz=FALSE)
```

Heart (2x2-table)

```
Summary:
n: 23, rows: 2, columns: 2

Fisher's exact test p-value = 0.03931
McNemar's chi-squared = 0.1667, df = 1, p-value = 0.6831

                    estimate lwr.ci upr.ci

odds ratio             8.250  1.154 59.003
rel. risk (col1)       2.933  0.850 10.120
rel. risk (col2)       0.356  0.140  0.901

Phi-Coefficient        0.464
Contingency Coeff.     0.421
Cramer's V             0.464

                 Response    Yes     No    Sum
Cholesterol
High         freq             11      4     15
             perc           .478   .174   .652
             p.row          .733   .267  1.000
             p.col          .846   .400   .652
Low          freq              2      6      8
             perc           .087   .261   .348
             p.row          .250   .750  1.000
             p.col          .154   .600   .348
Sum          freq             13     10     23
             perc           .565   .435  1.000
             p.row          .565   .435  1.000
             p.col         1.000  1.000  1.000
```

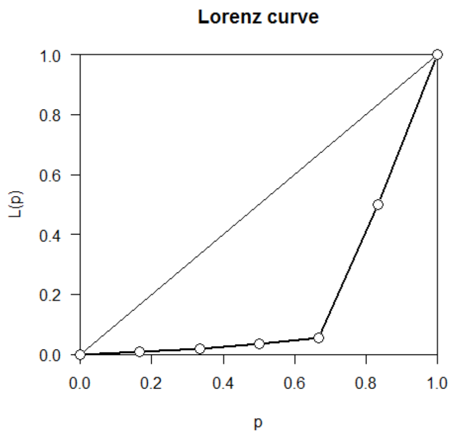




Lorenz curves

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
```

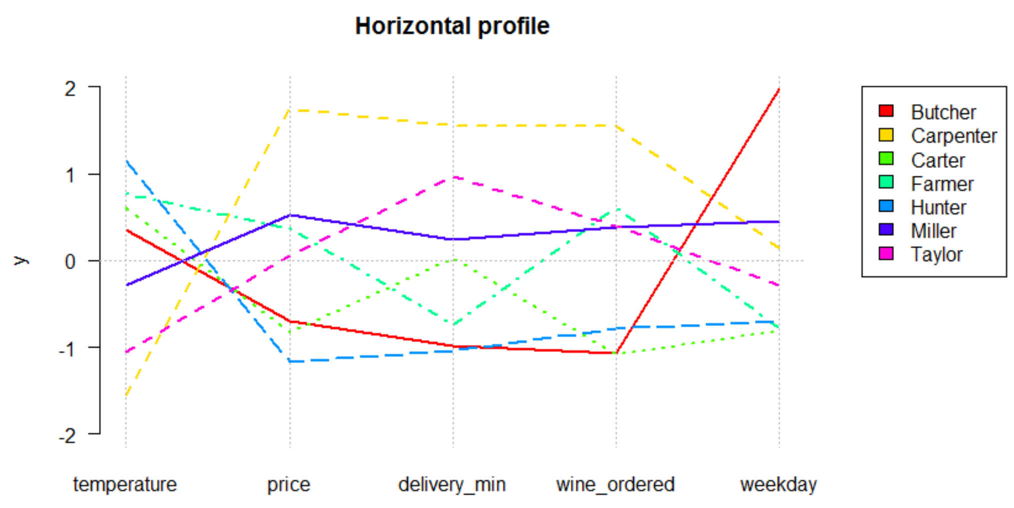
Lineplots

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.

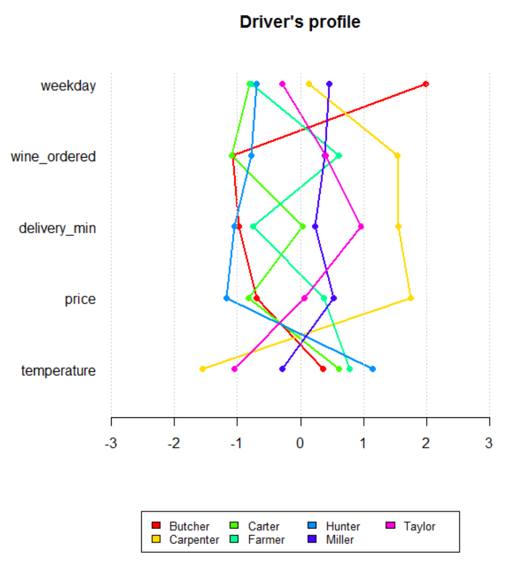
Let's for example have a horizontal profile of the driver's characteristics.

```
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")
```

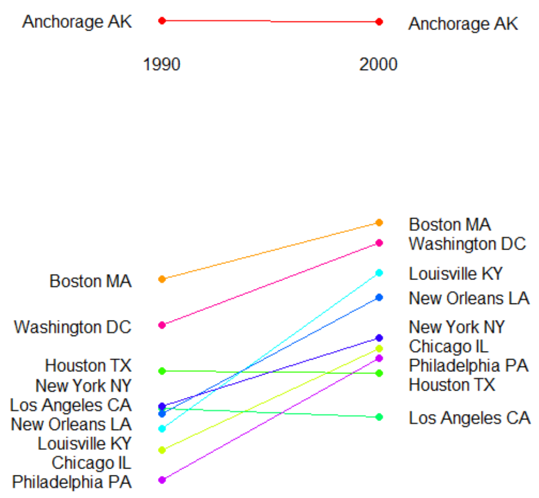


And the same, but on the vertical axis. (A)

```
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)

“Bumpchart”

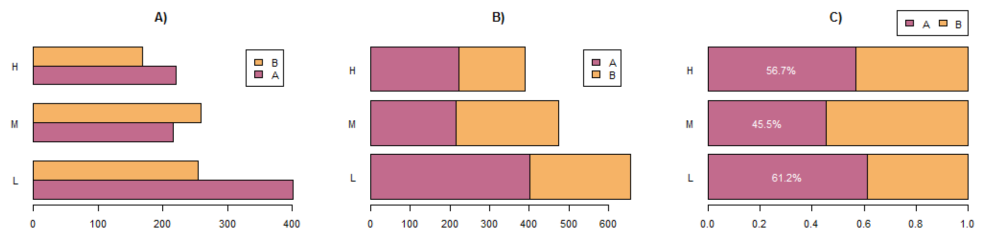
Plot B is sometimes called bumpchart (Jim Lemon).

```
# 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 )
```

Barplot horizontal

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

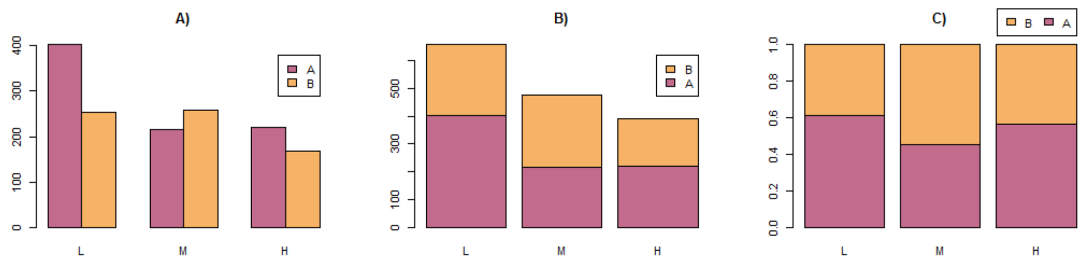
```
windows(height=3, width=11)
par(mfrow=c(1,3))

# 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))
text(paste(round(ptab[1,],3) * 100, "%",sep=""), x=ptab[1,]/2, y=b, col="white")
```



Barplot vertical

This same as above but with vertical bars.

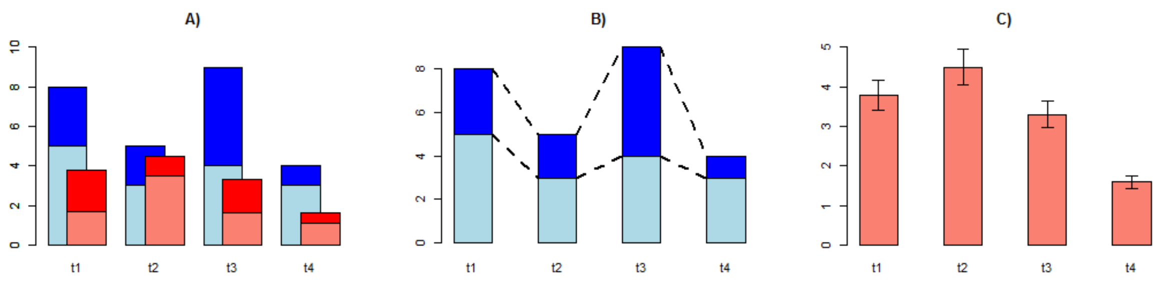
```
windows(height=3, width=11)
par(mfrow=c(1,3))

# 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))
```



Barplot (specials)

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

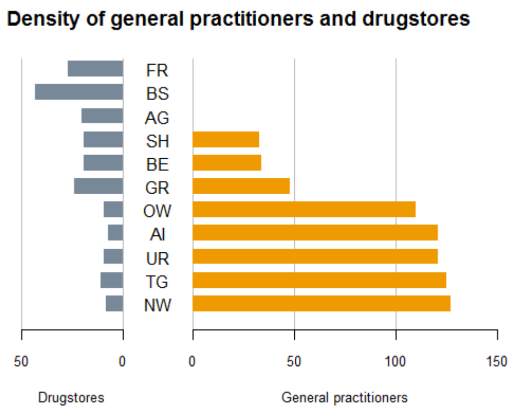
```
windows(height=3,11)
par(mfrow=c(1,3))

# 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=c("lightblue","blue"), 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("salmon","red"), 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=c("lightblue","blue"), space=1.2, main="B)" )
AddConnLines(blue, lwd=2, lty="dashed", space=1.2)


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



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))
```



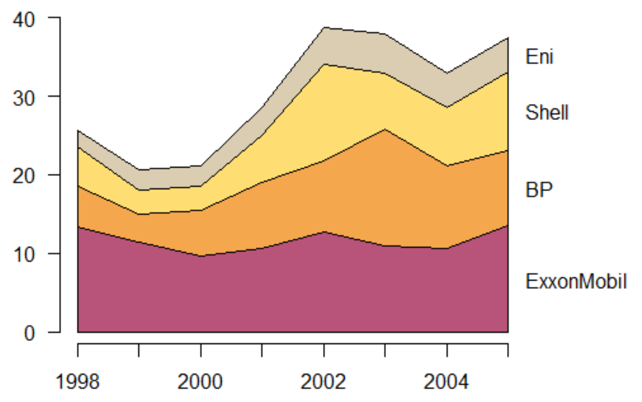
Areaplot

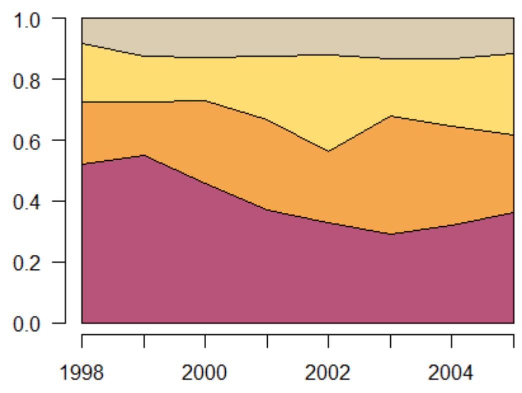
Areaplots have a high "ink factor"[2], say they use much ink to display the information and are therefore rarely the best way of representing data. But again, when your boss wants it this way, here's a function to produce it easily.

```
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), at=Midx(tail(t.oil, 1), 0), las=1 )

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
```





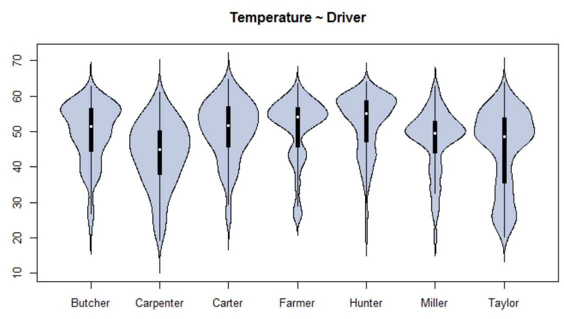
[2] Tufte, Edward R (2001) [1983], The Visual Display of Quantitative Information (2nd ed.), Cheshire, CT: Graphics Press, ISBN 0-9613921-4-2.

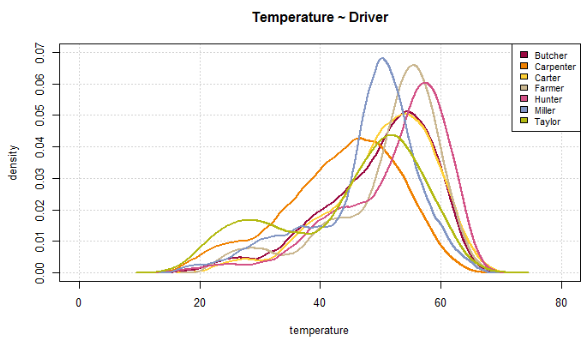
Comparing distributions

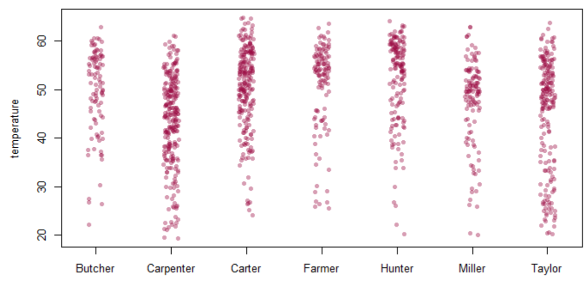
How should we compare distributions graphically, moving beyond a simple boxplot? PlotViolin serves the same utility as a side-by-side boxplot, but provides more detail about the single distribution. We started with John Verzani's Violinplot and rewrote it so that it takes exactly the same parameters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

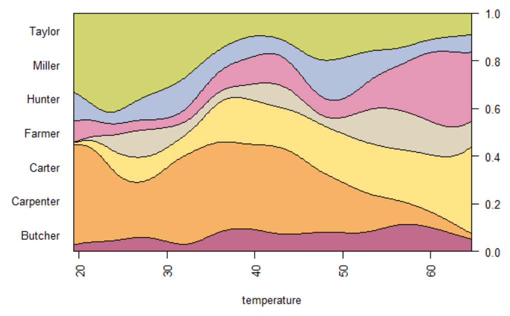
Another idea is to plot several densities within the same plot. PlotMultiDens does this while setting the xlim- and ylim-values to an appropriate value, ensuring all density lines are fully visible. For a smaller number of variables, say up to two handfuls, this will be the most direct way to compare 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 )
```









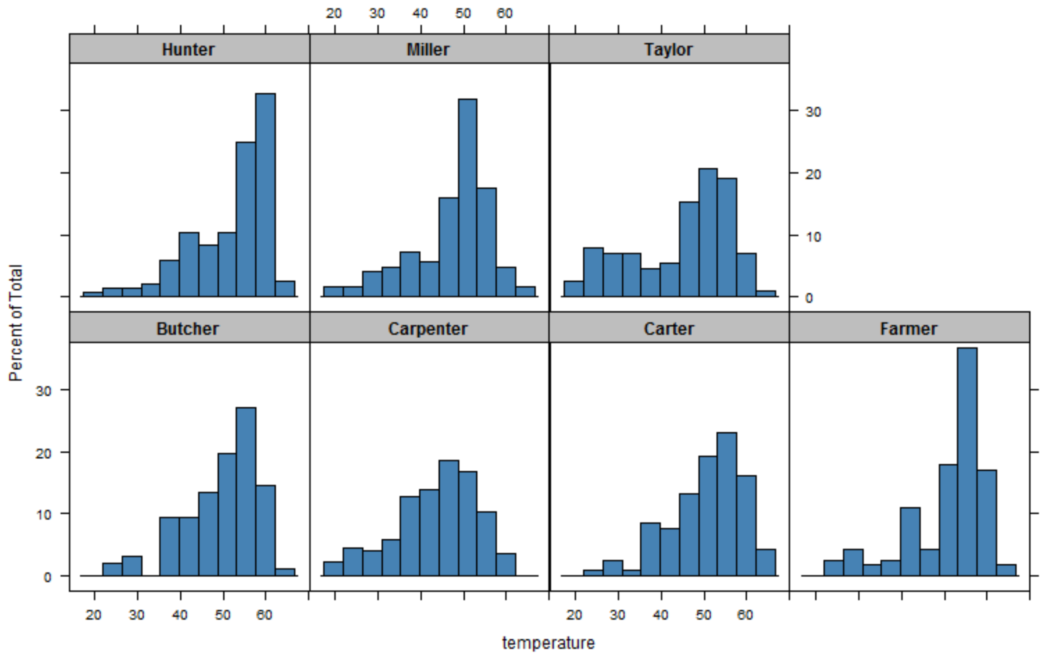
For small datasets a stripchart might be the best way to plot the data.
The conditional density-plot at the right allows to grasp the proportions within the total density.

```
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))
```

Several distributions with Trellis

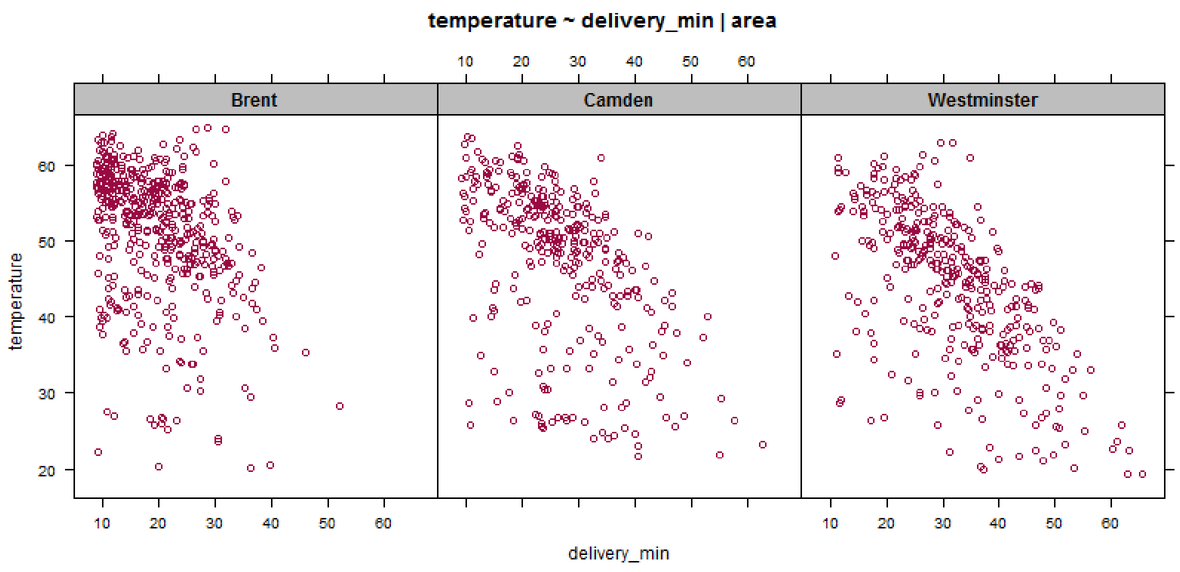
The classic way is to spend a full plot for every single variable. There's an interesting link, demonstrating this technique: http://www.statmethods.net/advgraphs/trellis.html

```
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.

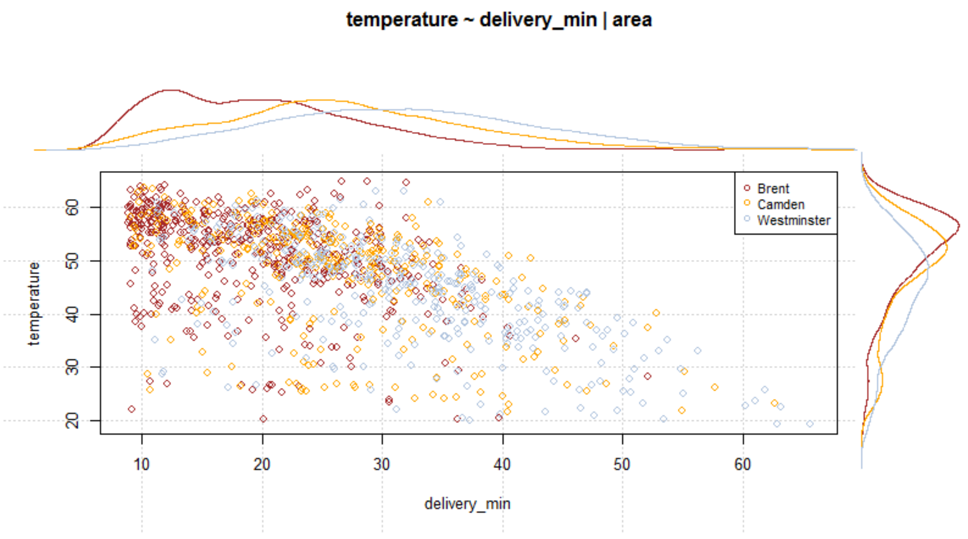
```
xplot(temperature ~ delivery_min | area, d.pizza, main='temperature ~ delivery_min | area',
       col=hred, strip=myStripStyle)
```



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.

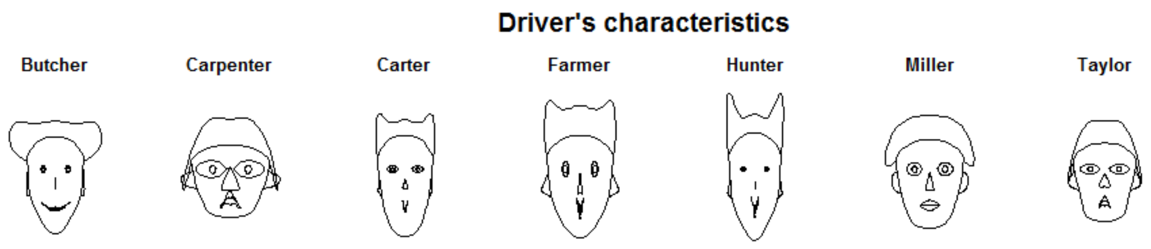
```
PlotMarDens(y=d.pizza$temperature, x=d.pizza$delivery_min, grp=d.pizza$area,
         xlab="delivery_min", ylab="temperature",
         col=c("brown","orange","lightsteelblue"), panel.first=grid(),
         main="temperature ~ delivery_min | city"  )
```



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.

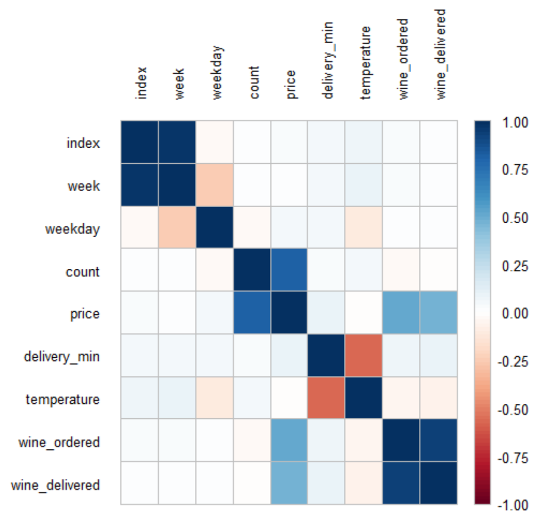
```
m <- data.frame( lapply( d.pizza[,c("temperature","price","delivery_min","wine_ordered","weekday")]
                        , tapply, d.pizza$driver, mean, na.rm=TRUE))

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

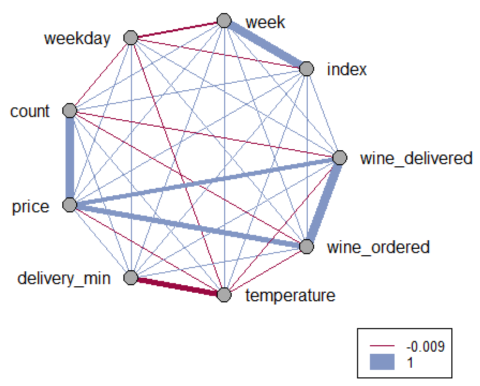


Correlations: PlotCorr and PlotWeb

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[, WhichNumerics(d.pizza)], use="pairwise.complete.obs")

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

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



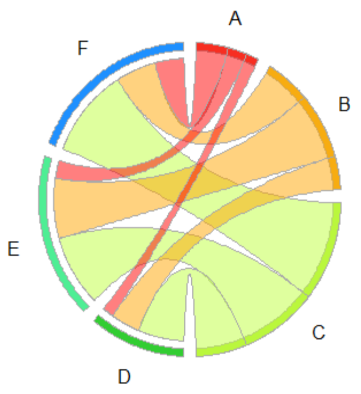


Associations with circular plots

Saw this lately and considered it a promising idea. Still this is, although working, rather experimental code.

```
tab <- matrix(c(2,5,8,3,10,12,5,7,15), nrow=3, byrow=FALSE)
dimnames(tab) <- list(c("A","B","C"), c("D","E","F"))
WrdText(tab)

PlotCirc( tab,
   acol = c("dodgerblue","seagreen2","limegreen","olivedrab2","goldenrod2","tomato2"),
   rcol = SetAlpha(c("red","orange","olivedrab1"), 0.5)
 )
```

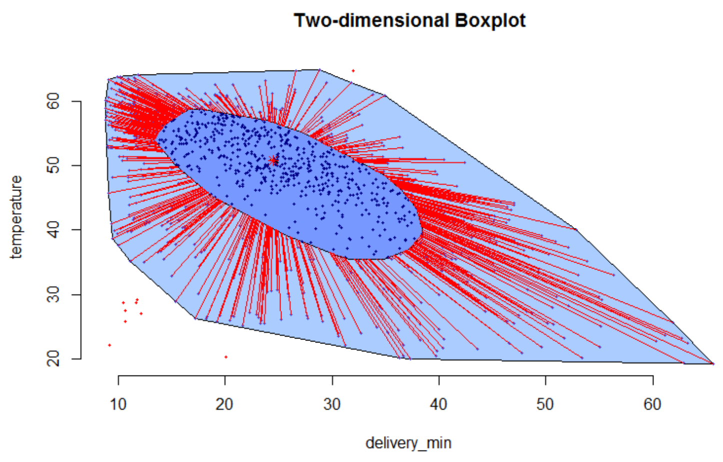
```
The table

   D  E  F
A  2  3  5
B  5 10  7
C  8 12 15
```



Boxplot on 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.

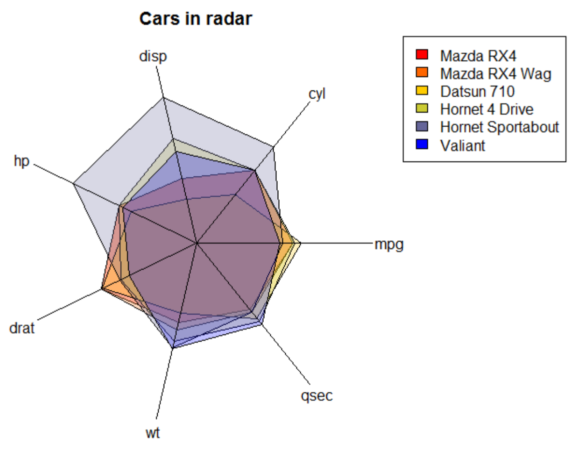
```
d.frm <- d.pizza[complete.cases(d.pizza[,c("temperature","delivery_min")]),]

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



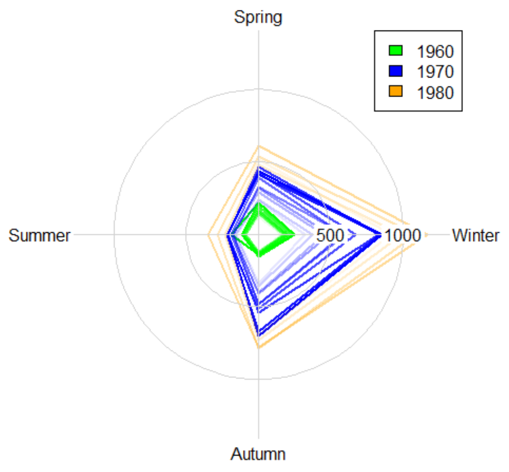
PlotPolar (Radarplot)

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))
```



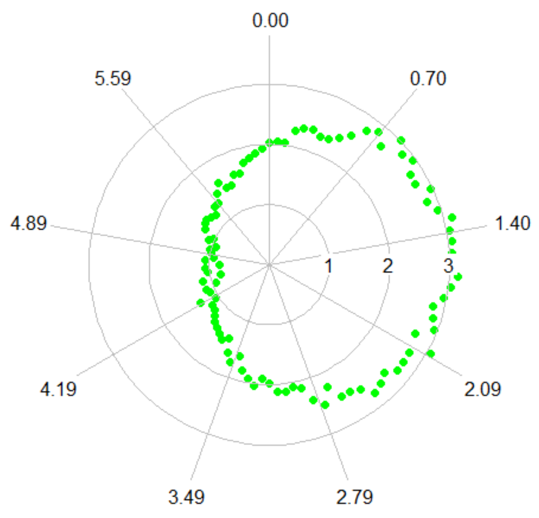
A)

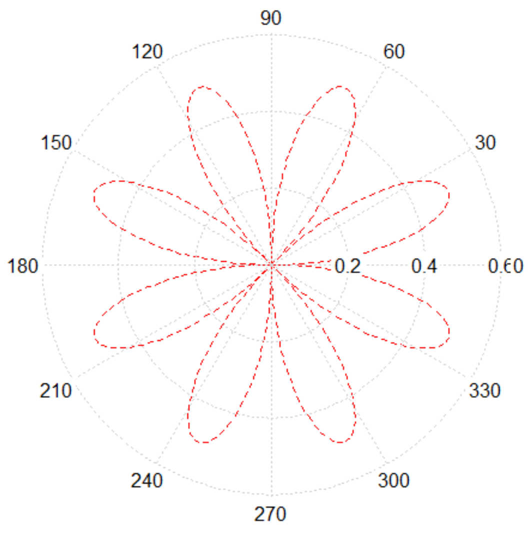


B)

```
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"))
```

```
testlen <- c(sin(seq(0, 1.98*pi, length=100)) + 2 + rnorm(100)/10)
# 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=FormatFix(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()
```

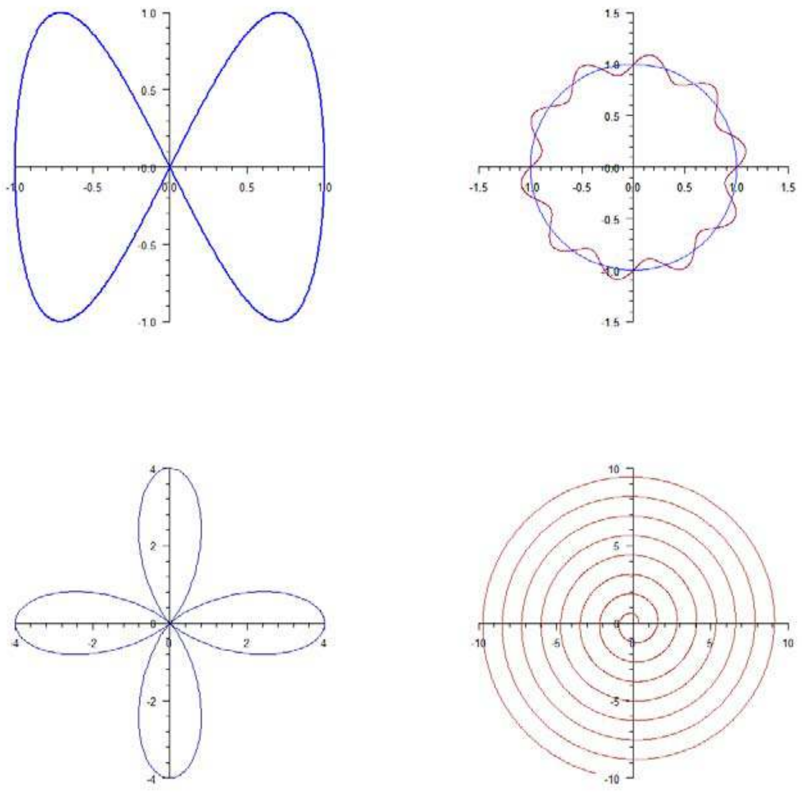




Plot Functions

Functions can be plotted a bit more comfortable by means of the function PlotFct. The idea is to be able to use 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.

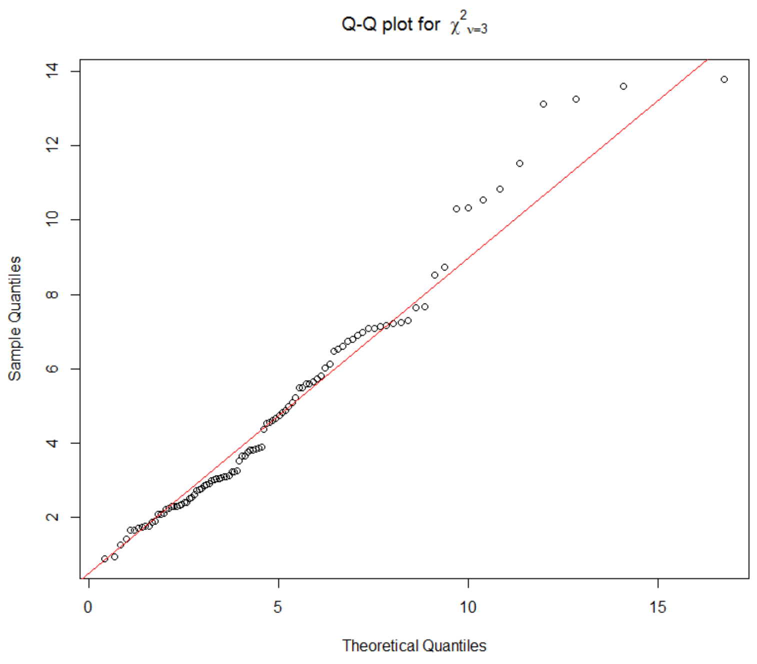
```
# get some data
par(mfrow=c(2,2))
PlotFct(sin(2*t) ~ sin(t), from=0, to=2*pi, by=0.01, col="blue", lwd=2)

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

# lemniscate of Bernoulli
PlotFct((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")

# evolving circle
PlotFct(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")
```



PlotQQ

This is a short wrapper for plotting QQ-plots with other than normal distributions. A qqline is inserted.

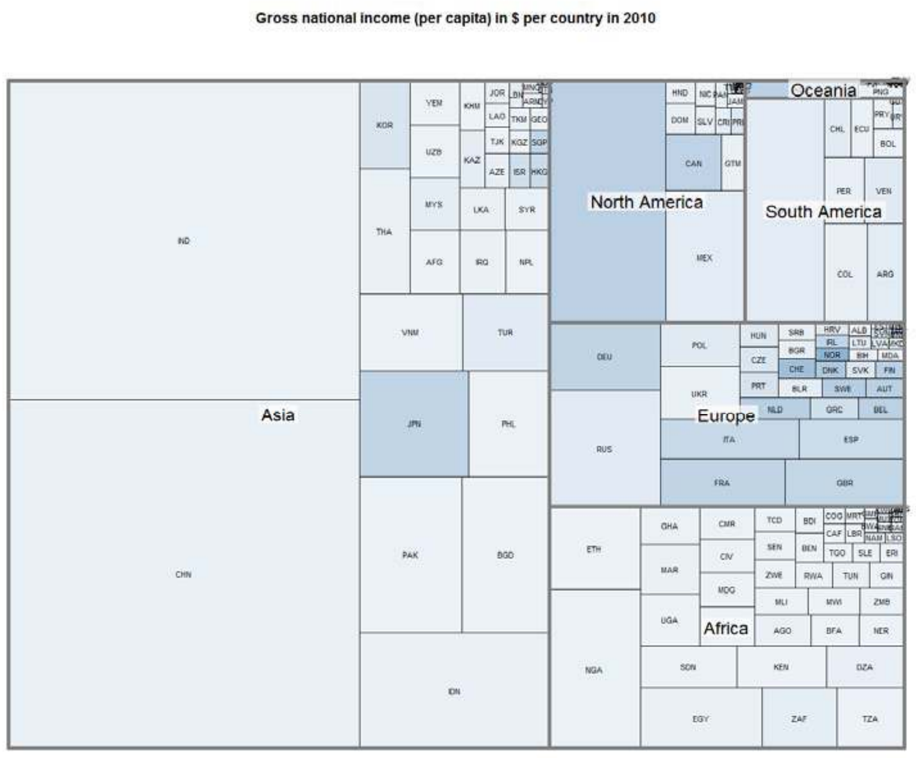
```
# get some data
PlotQQ(z, function(p) qchisq(p, df=5), args.qqline=list(col=2, probs=c(0.1,0.6)),
       main=expression("Q-Q plot for" ~~ {chi^2}[nu == 3]))
```



PlotTreemap

This function produces a treemap.

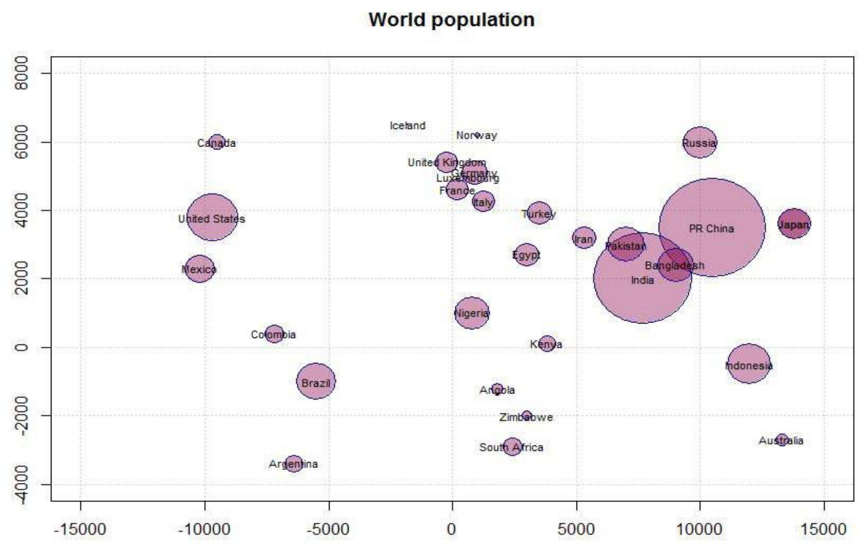
```
# get some data
data(GNI2010, package="treemap")
gn <- GNI2010[,c("iso3","population","continent","GNI")]
gn <- gn[gn$GNI!=0,]

# 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) )
```



PlotBubble

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

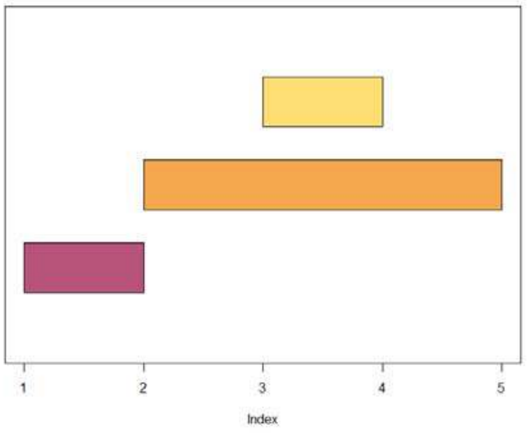
```
PlotBubble(d.world$x, d.world$y, area=d.world$pop/90, col=SetAlpha("deeppink4",0.4), border="darkblue",
          xlab="", ylab="", panel.first=grid(), main="World population")
text(d.world$x, d.world$y, labels=d.world$country, cex=0.7, adj=0.5)
```



PlotHorizBar

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

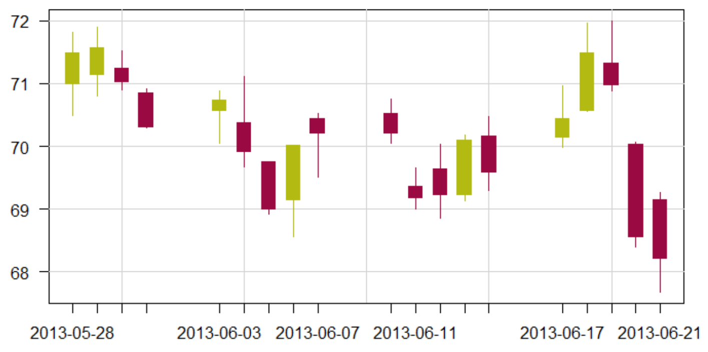
```
PlotHorizBar(from=c(1,2,3), to=c(2,5,4), grp=c(1,2,3), col=PalHelsana()[1:3])
```



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.

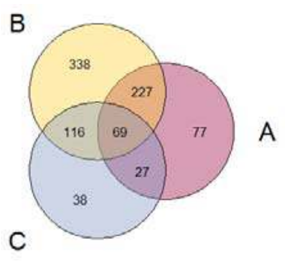
```
example(PlotCandlestick)
PlotCandlestick(x=as.Date(rownames(nov)), y=nov, border=NA, las=1, ylab="")
```



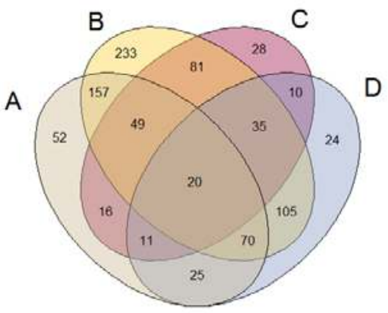
Venn plots

In rare cases 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.)

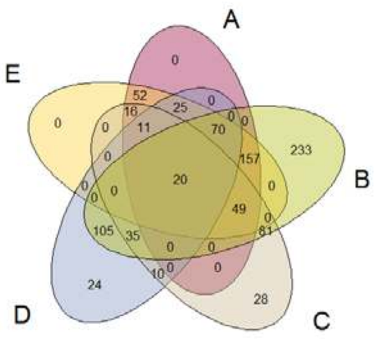
```
example(PlotVenn)
PlotVenn(x=x[1:3], col=SetAlpha(c(PalHelsana()[c(1,3,6)]), 0.4))
PlotVenn(x=x[1:4], col=SetAlpha(c(PalHelsana()[c(1,3,6,4)]), 0.4))
PlotVenn(x=x[1:5], col=SetAlpha(c(PalHelsana()[c(1,3,6,4,7)]), 0.4))
```



3 sets



4 sets

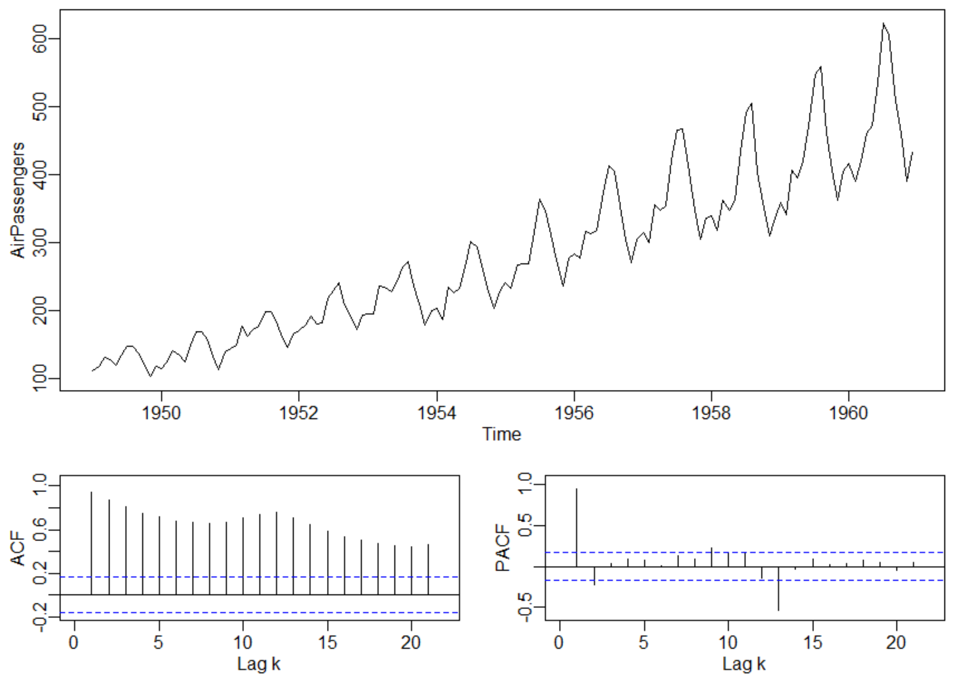


5 sets

PlotACF

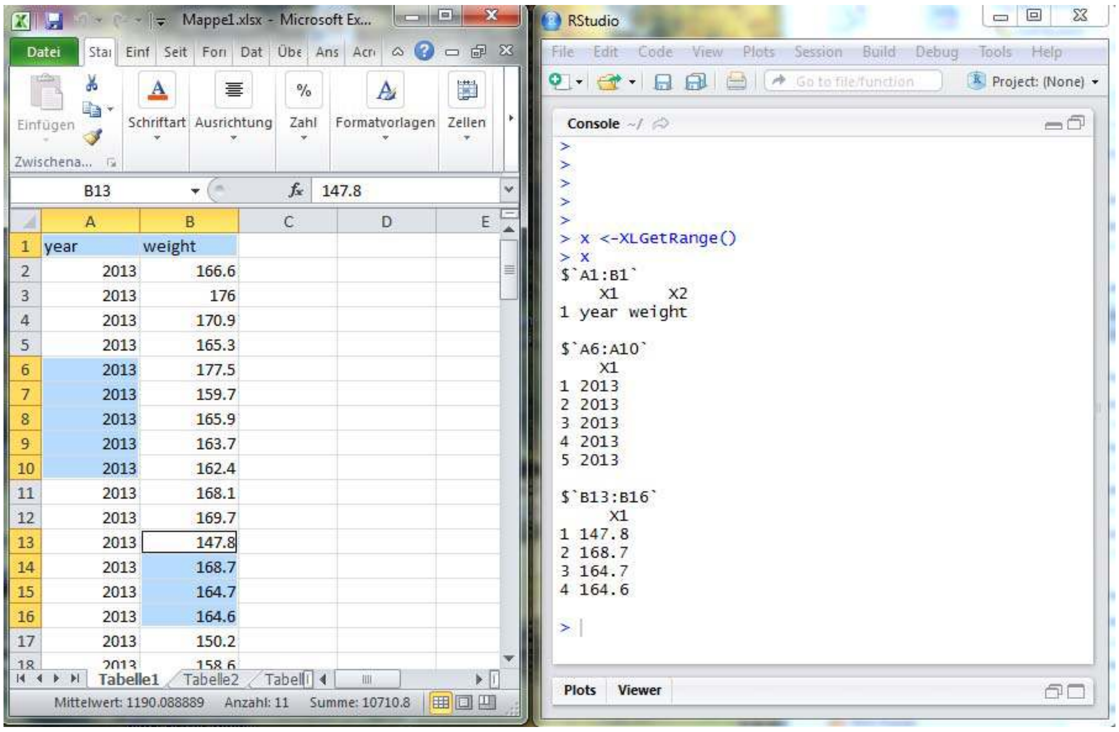
This produces a combined plot of a time series and its autocorrelation and partial autocorrelation

```
PlotACF(AirPassengers)
```



Import data via Excel

The function XLGetRange allows to quickly importing 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.



B13 147.8

	A	B	C	D	E
1	year	weight			
2	2013	166.6			
3	2013	176			
4	2013	170.9			
5	2013	165.3			
6	2013	177.5			
7	2013	159.7			
8	2013	165.9			
9	2013	163.7			
10	2013	162.4			
11	2013	168.1			
12	2013	169.7			
13	2013	147.8			
14	2013	168.7			
15	2013	164.7			
16	2013	164.6			
17	2013	150.2			
18	2013	158.6			

Tabelle1 Tabelle2 Tabell

Mittelwert: 1190.088889 Anzahl: 11 Summe: 10710.8

```
>
>
>
>
>
> x <-XLGetRange()
> x
$`A1:B1`
    X1     X2
1 year weight

$`A6:A10`
    X1
1 2013
2 2013
3 2013
4 2013
5 2013

$`B13:B16`
     X1
1 147.8
2 168.7
3 164.7
4 164.6

> |
```

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

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
```

String functions

As the string functions are highly scattered, they sometimes can hardly be found right from the start. This overview might be helpful:

`cat`	Print strings
`paste, %c%`	Concatenate strings
`nchar`	Number of characters in a string
`substr`	Extract or replace substrings in a character vector
`match, pmatch`	Searching parts of one string in another
`charmatch`	Partial string matching charmatch seeks matches for the elements of its first argument among those of its second.
`grep, grepl, regexpr`	Searching values in a character vector by means of regular expressions.
`sub, gsub`	Substitute a part of the string with another
`StrVal`	Extract numeric values out of a string
`StrPos`	Finds the first position of one string in another string
`StrTrim`	Delete white spaces from a string
`StrTrunc`	Truncate string on a given length and add ellipses if it really was truncated
`StrDist, adist`	Compute distances between strings
`StrChop`	Split a string by a fixed number of characters.
`strsplit`	Split the elements of a character vector x into substrings according to the matches to substring split within them.
`AscToChar, CharToAsc`	Converts ASCII codes to characters and vice versa
`abbr, StrAbbr`	Abbreviates a string
`StrCap`	Capitalize the first letter of a string
`tolower, toupper`	convert upper-case characters in a character vector to lower-case, or vice versa
`StrRev`	Reverse a String
`StrCountW`	Count the words in a string
`chartr`	Translate characters in character vectors, in particular from upper to lower case or vice versa.
`strtrim`	Trim character strings to specified display widths.
`StrIsNumeric`	Checks whether a string contains only numeric data
`sprintf`	Format a numeric value as a string

References

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

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

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