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

January 7, 2014

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 of the 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 some facilities like summary, describe (Hmisc), stat.desc (library pastecs), all of them missing some functionality or flexibility we would have expected.
Then there are quite a bit of commonly used functions, which curiously are not present in the *statistics* package, think e.g. of skewness, kurtosis but also gini-coefficent or Somers' delta. This led to a rank growth of libraries implementing just the 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 the variety concerning NA-handling, recycling rules and so on will be quite large.

So after completion of a project, where we had to describe a dataset under deadline pressure, we started to gather our newly created functions and put them together to the first version of "DescTools".

The collection has meanwhile grown to a considerably versatile toolset for descriptive statistics, providing rich univariate and bivariate descriptions of data without expecting you 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.
Taking into account the fact that most problems can be satisfactorily visualized with bar- and dotplots, still some more specific plot types are included in the library. Some of them are new, and some of them are based on types found scattered in the myriads of R-packages found out there.

This document describes quickly the essentials of the package DescTools.

Contents

Describing a full 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. Let's describe some variables out of the integrated dataset `d.pizza` (a data.frame) first.
The output can either be sent to the R-console or as well be redirected to a new MS-Word document.

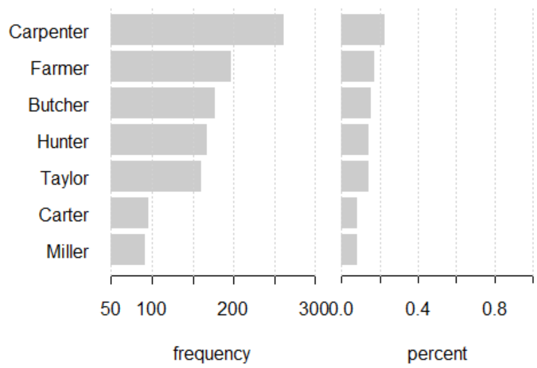
```
library(DescTools)

# we start a new word instance, where the results will be sent to
wrd <- GetNewWrd()
Desc(d.pizza[,c("driver","temperature","count","wine_ordered")], wrd=wrd)
```

```
'data.frame':	1209 obs. of  4 variables:
 1 $ driver      : Factor w/ 7 levels "Carpenter","Carter",..: 1 1 4 4 3 7 4 5 1 1 ...
 2 $ temperature : num  53.2 NA 53.5 38.3 31.3 ...
 3 $ count       : int  2 2 2 3 5 10 10 3 2 3 ...
 4 $ wine_ordered: int  0 0 0 0 1 1 0 0 0 0 ...
```

First a simple Str() of the data.frame is performed. Then every single column will be described according to the type of its class.

1 : driver (factor)

```
  length      n    NAs levels unique  dupes
   1'209  1'152     57      7      7      y


       level freq   perc cumfreq cumperc
1  Carpenter  262   .227     262    .227
2     Farmer  197   .171     459    .398
3    Butcher  177   .154     636    .552
4     Hunter  167   .145     803    .697
5     Taylor  161   .140     964    .837
6     Carter   96   .083    1060    .920
7     Miller   92   .080    1152   1.000
```

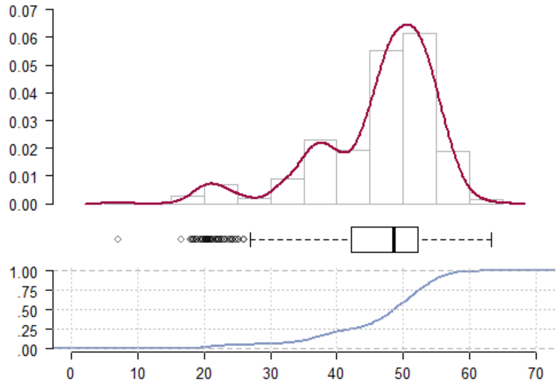


Synopsis

`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 more levels than unique values (but not the other way round)
`dupes`	y or n, reporting if there are any duplicate values in the vector. If "n" then there are only unique values.
`freq`	the 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

2 : temperature (numeric)

```
 length       n    NAs  unique      0s    mean  meanSE
  1'209   1'162     47     917       0  46.289   0.259

    .05     .10    .25  median     .75     .90     .95
 25.870  34.849 42.145  48.505  52.265  55.010  56.578

    rng      sd  vcoef     mad     IQR    skew    kurt
 56.320   8.839  0.191   6.605  10.120  -1.261   1.478

lowest : 6.97, 16.56, 17.92, 18.12, 18.28
highest: 60.49, 60.62, 61.37, 62.31, 63.29

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

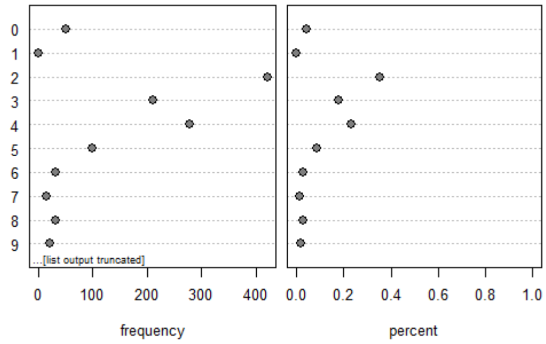


The first measures length, n, NAs, unique have the same meaning as above.

`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`.

3 : count (integer)

```
  length       n     NAs  unique      0s    mean  meanSE
   1'209   1'194      15      13      51   3.489   0.060

     .05     .10     .25  median     .75     .90     .95
       2       2       2       3       4       6       8

     rng      sd   vcoef     mad     IQR    skew    kurt
      12   2.063   0.591   1.483       2   1.384   2.337

Shapiro-Wilks normality test  p-value = < 2.22e-16

lowest : 0 (51), 1, 2 (420), 3 (210), 4 (279)
highest: 8 (32), 9 (21), 10 (29), 11 (2), 12 (2)
```

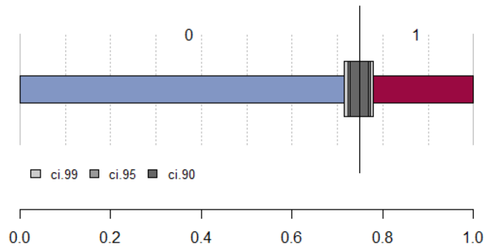


This is the description of a count variable, which is somewhere between numeric and factors as far as descriptive measures are concerned. In fact, if there are only just a few unique values, then the factor representation might be more appropriate than the numeric description with densities etc.. We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

`plot`	the plot is produced as (horizontal) dotchart. More than 10 unique values are truncated (a warning is placed in the plot area).

4 : wine_ordered (integer)

```
  length       n     NAs  unique
   1'209   1'198      11       2

  freq   perc  lci.95  uci.95¹
0  897   .749    .723    .772
1  301   .251    .228    .277

¹ 95%-CI Wilson
```



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 as `BinomCI` with the option “Wilson”.

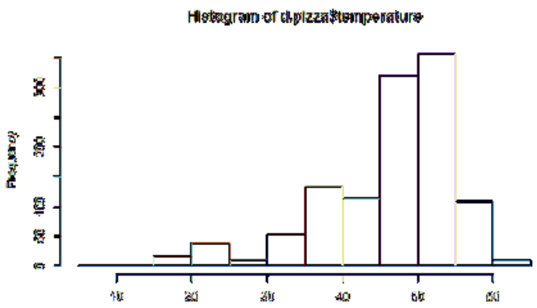
`plot`	this is basically a univariate barplot, with confidence intervals on the confidence levels of 0.90, 0.95 and 0.99. The vertical line denominates the point estimator.

Simple Frequencies

Get the frequencies and the percentages of a binned variable with the same logic as in `hist`.
The single and the cumulative values are reported.

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

```
     level freq  perc cumfreq  cumperc
1   [5,10]    1 0.001       1    0.001
2  (10,15]    0 0.000       1    0.001
3  (15,20]   16 0.014      17    0.015
4  (20,25]   39 0.034      56    0.048
5  (25,30]   10 0.009      66    0.057
6  (30,35]   53 0.046     119    0.102
7  (35,40]  134 0.115     253    0.218
8  (40,45]  113 0.097     366    0.315
9  (45,50]  321 0.276     687    0.591
10 (50,55]  357 0.307    1044    0.898
11 (55,60]  109 0.094    1153    0.992
12 (60,65]    9 0.008    1162    1.000
```



Percentage 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)
```

A) Frequencies and percentages

```
                 Zurich London  Paris    Sum

Carpenter freq      191      4      0    195
          perc     .180   .004   .000   .184
          p.col    .534   .009   .000   .184
Carter    freq        0    203      0    203
          perc     .000   .191   .000   .191
          p.col    .000   .477   .000   .191
Taylor    freq        0      0    142    142
          perc     .000   .000   .134   .134
          p.col    .000   .000   .511   .134
Butcher   freq        0    110      0    110
          perc     .000   .104   .000   .104
          p.col    .000   .258   .000   .104
Hunter    freq        0    109      0    109
          perc     .000   .103   .000   .103
          p.col    .000   .256   .000   .103
Miller    freq      167      0      1    168
          perc     .157   .000   .001   .158
          p.col    .466   .000   .004   .158
Farmer    freq        0      0    135    135
          perc     .000   .000   .127   .127
          p.col    .000   .000   .486   .127
Sum       freq      358    426    278   1062
          perc     .337   .401   .262  1.000
          p.col   1.000  1.000  1.000  1.000
```

B) Expected values and std. residuals

```
                  Zurich   London    Paris      Sum

Carpenter freq       191        4        0      195
          exp     65.734   78.220   51.045        .
          stdres  21.002  -12.002  -9.203         .
Carter    freq         0      203        0      203
          exp     68.431   81.429   53.139        .
          stdres -11.297  19.357   -9.434         .
Taylor    freq         0        0      142      142
          exp     47.868   56.960   37.171        .
          stdres  -9.130  -1 .478  21.500         .
Butcher   freq         0      110        0      110
          exp     37.081   44.124   28.795        .
          stdres  -7.899   13.535   -6.596        .
Hunter    freq         0      109        0      109
          exp     36.744   43.723   28.533        .
          stdres  -7.859   13.466   -6.563        .
Miller    freq       167        0        1      168
          exp     56.633   67.390   43.977        .
          stdres  19.633  -11.562  -8.221         .
Farmer    freq         0        0      135      135
          exp     45.508   54.153   35.339        .
          stdres  -8.868  -1 .178  2 .885         .
Sum       freq       358      426      278     1062
          exp          .        .        .        .
          stdres       .        .        .        .
```

Pairwise descriptions

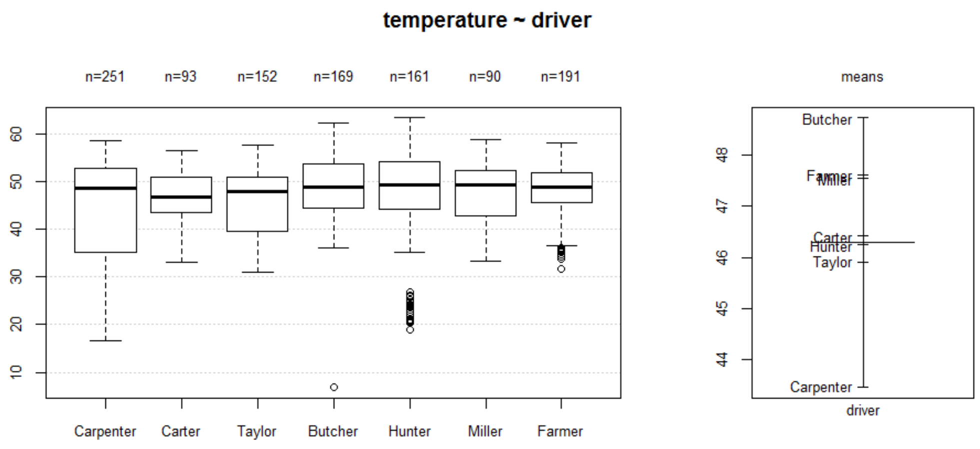
`Desc` implements a formula interface allowing to define bivariate descriptions straight forward.

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, if such exist.

```
Desc(temperature + operator ~ driver, d.pizza, digits=1, wrd=wrd)
```

temperature ~ driver (numeric ~ categorical)

```
Summary:
n pairs: 1'209, valid: 1'107 (92%), missings: 102 (8%), groups: 7


        Carpenter     Carter    Taylor    Butcher     Hunter    Miller    Farmer
mean        43.5¹      46.4      45.9      48.7²       46.2      47.5      47.6
median      48.5       46.7¹     48.0      48.8        49.3      49.3²     48.7
sd          11.6        5.4       6.5       6.6        11.6       6.9       5.8
IQR         17.5        7.4      11.5       9.4         9.9       9.3       6.2
n            251         93       152       169         161        90       191
np         0.227      0.084     0.137     0.153       0.145     0.081     0.173
NAs           11          3         9         8           6         2         6
0s             0          0         0         0           0         0         0

¹ min, ² max

Kruskal-Wallis rank sum test:
  Kruskal-Wallis chi-squared = 24.5591, df = 6, p-value = 0.000412

Warning:
  Grouping variable contains 57 NAs (4.71%).
```

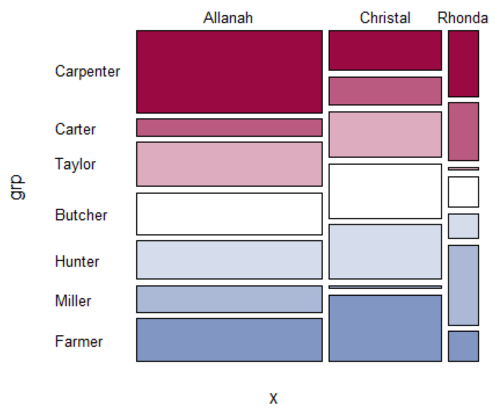


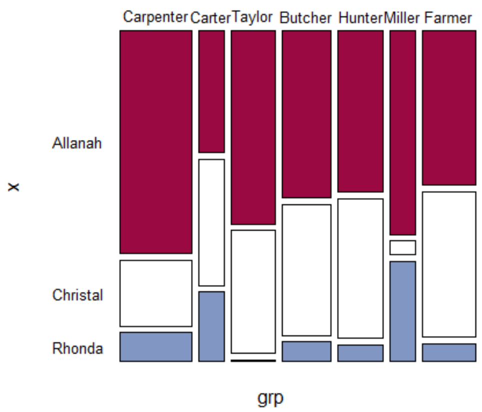
`plot`	a boxplot combined with a means-plot as used in anova.

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

operator ~ driver (categorical ~ categorical)

```
Summary:
n pairs: 1'209, valid: 1'152 (95%), missings: 57 (5%), nrow: 3, ncol: 7

Pearson's Chi-squared test:
  X-squared = 160.1972, df = 12, p-value < 2.2e-16

Phi-Coefficient   : 0.373
Contingency Coeff.: 0.349
Cramer's V        : 0.264


Abs. frequencies
          driver
operator   Carpenter Carter Taylor Butcher Hunter Miller Farmer  Sum
  Allanah        184     37     98      93     85     59     96  652
  Christal        54     38     62      73     73      4     90  394
  Rhonda          24     21      1      11      9     29     11  106
  Sum            262     96    161     177    167     92    197 1152

Rel. frequencies
          driver
operator   Carpenter Carter Taylor Butcher Hunter Miller Farmer
  Allanah       .160   .032   .085    .081   .074   .051   .083
  Christal      .047   .033   .054    .063   .063   .003   .078
  Rhonda        .021   .018   .001    .010   .008   .025   .010
```





The measures in detail:

`nrow`	the number of levels of the left-side variable (here: operator)
`ncol`	the number of levels of the right-side variable (here: driver)
`plot`	two mosaicplots, x ~ y and y ~ x.
`tests`	results of the Pearson's Chi-squared test
`Association measures`	Some association measures as Phi-coefficient, Contingency coefficient and Cramer's V
`abs. & rel. frequencies`	Absolute and relative frequencies

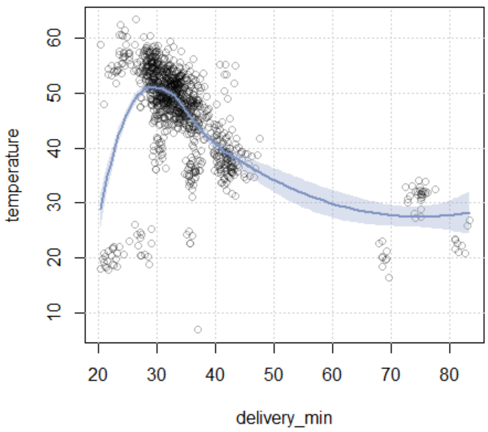
Two numerical variables have no obvious standard description as their relationship can have many forms. We report therefore only the simple correlation coefficients (Pearson, Spearman and Kendall).

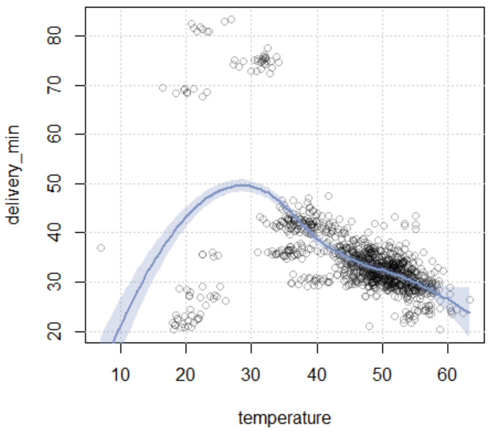
The variables are plotted as xy-scatterplots with interchanging mutual dependency, supplemented with a LOESS smoother.

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

temperature ~ delivery_min (numeric ~ numeric)

```
Summary:
n pairs: 1'209, valid: 1'129 (93%), missings: 80 (7%)


Pearson corr. : -0.519
Spearman corr.: -0.584
Kendall corr. : -0.457
```





Tables

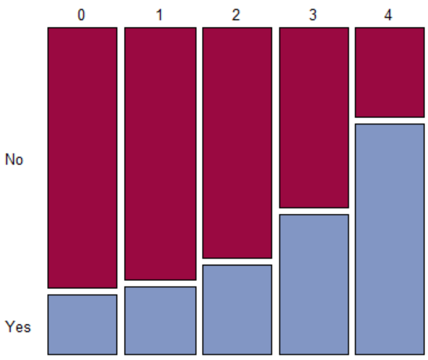
There are many suggestions for the description of tables out there. We use a mix between SAS- and SPSS-flavour here. Let's take a SAS-example[i] and describe the table with SPSS-verbosity:

```
pain <- as.table(matrix(c(26,26,23,18, 9,
                           6, 7, 9,14,23), nrow=2, byrow=TRUE))
Desc(pain, verb="hi", wrd=wrd)
```

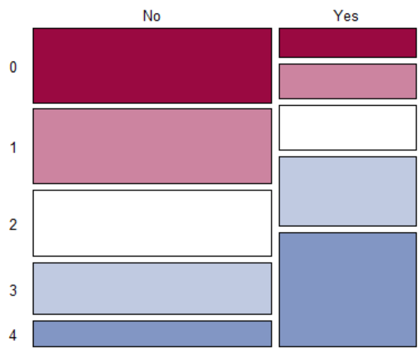
Clinical Trial for Treatment of Pain (2x5-table)

```
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	NA	NA
Contingency Coeff.	.3766	NA	NA
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	NA	NA



		0	1	2	3	4	Sum
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

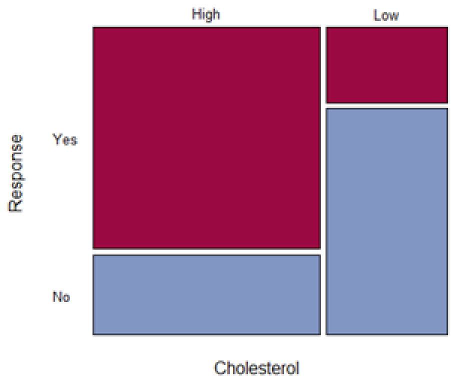


Let's have a look at a 2x2-table. We switch from ChiSquare-test to the exact Fisher test and report McNemar's symmetry test. The odds ratio and relative risk are displayed in addition by default. The verbosity can be set to “med”, “lo”, “hi” for medium, low and high.

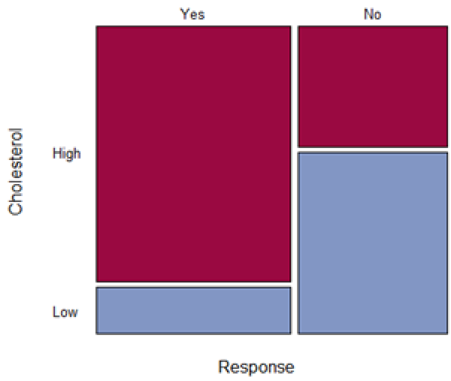
```
heart <- as.table(matrix(c(11,4,
                            2,6), nrow=2, byrow=TRUE,
                          dimnames=list(Cholesterol=c("High","Low"), Response=c("Yes","No"))) )

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
```



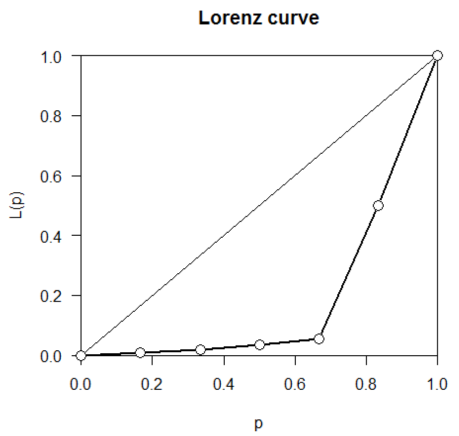

```
                  Response    Yes     No    Sum
Cholesterol
High        freq               11      4     15
            perc             .478   .174   .652
            p.row            .733   .267  1.000
            p.col            .846   .400   .652
Low         freq                2      6      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 those 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)
```



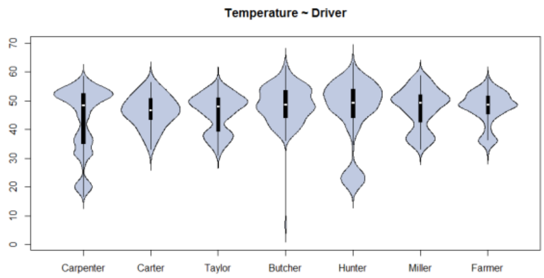

```
> Gini(x)
[1] 0.7535714

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

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

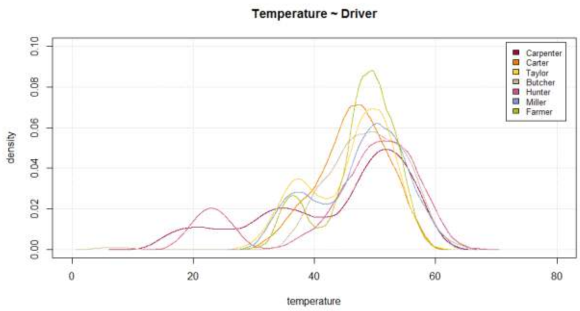
Comparing distributions: PlotViolin and PlotMultiDens

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 to take 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, will this be the most direct way to compare their distributions. (Note: For violins this limit lies much higher as they do not overlap and so hide mutually.)

```
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 )
```

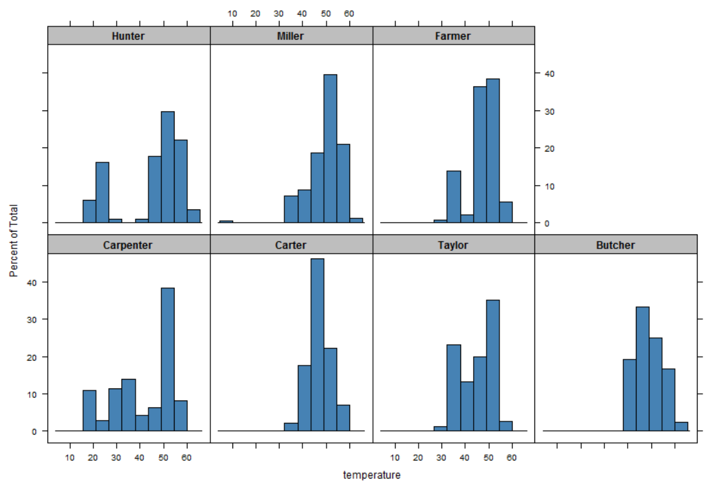




Several distributions with Trellis

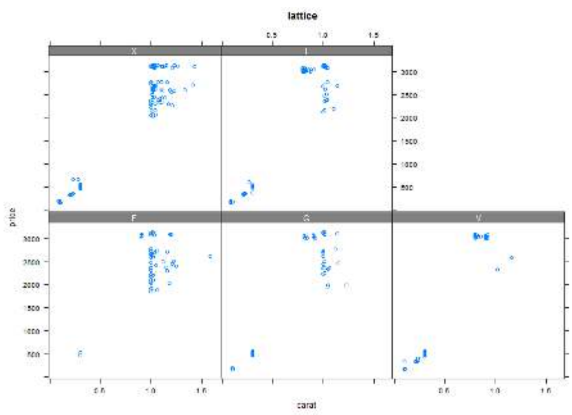
The classic way is to spend a full plot for every single variable. There's an interesting link, demonstrating this technique: http://www.statmethods.net/advgraphs/trellis.html

```
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.

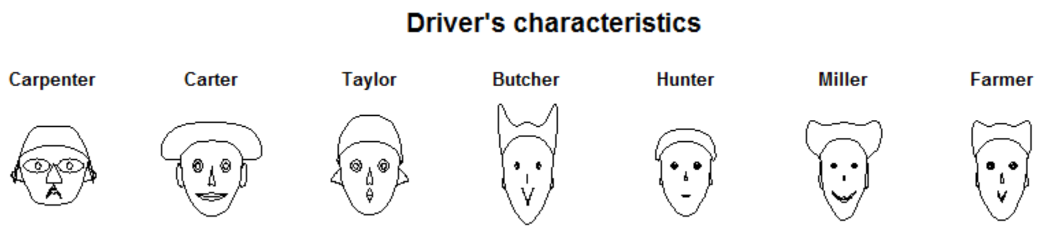
```
xyplot(price ~ carat | cut, d.diamonds, main='lattice')
```



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.

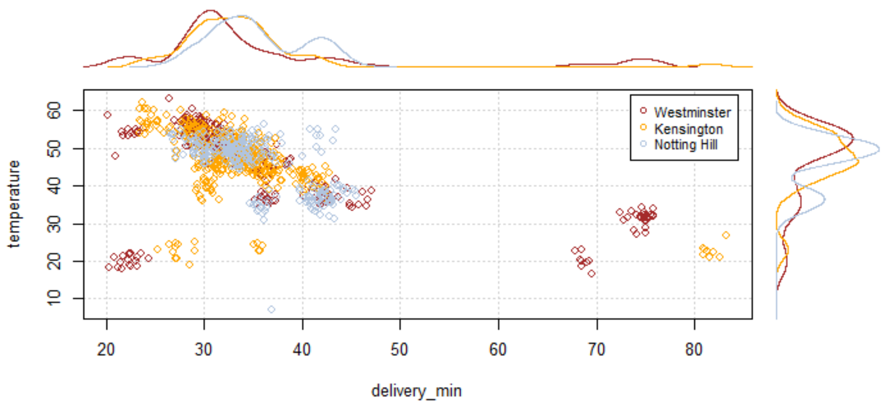
```
m <- data.frame( lapply( d.pizza[,c("temperature","price","delivery_min","wine_ordered","weekday")]
                        , tapply, d.pizza$driver, mean, na.rm=TRUE))

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



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.

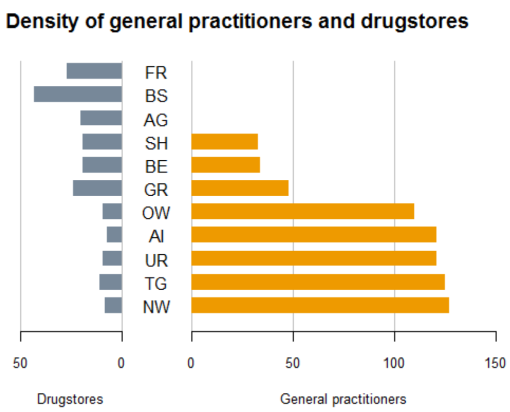
```
PlotMarDens(y=d.pizza$temperature, x=d.pizza$delivery_min, grp=d.pizza$area,
            xlab="delivery_min", ylab="temperature",
            col=c("brown","orange","lightsteelblue"), panel.first=grid(),
            main="temperature ~ delivery_min | city"  )
```



PlotPyramid

This function produces a “pyramid plot”, a simple back to back horizontal barplot.

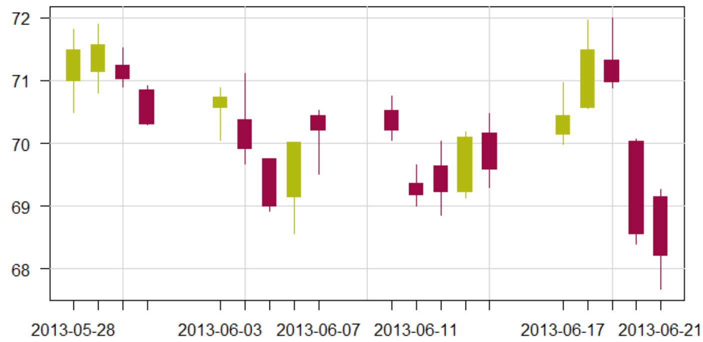
```
d.sda <- data.frame(
  kt_x  = c("NW","TG","UR","AI","OW","GR","BE","SH","AG","BS","FR"),
  apo_n = c(  8,  11,   9,   7,   9,  24,  19,  19,  20,  43,  27 ),
  sda_n = c(127, 125, 121, 121, 110,  48,  34,  33,   0,   0,   0 ))

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))
```



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.

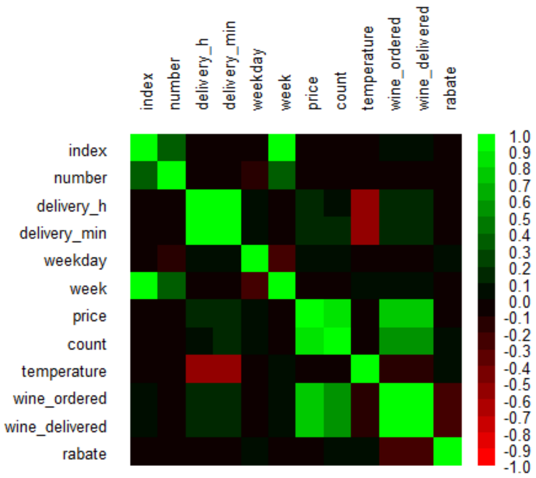
```
example(PlotCandlestick)
PlotCandlestick(x=as.Date(rownames(nov)), y=nov, border=NA, las=1, ylab="")
```

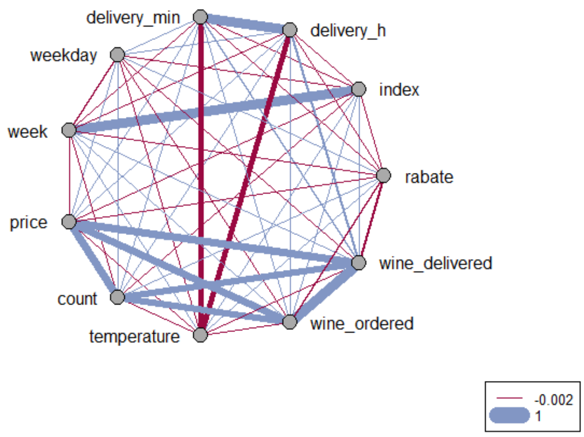


Correlations: PlotCorr and PlotWeb

This functions produce a graphical display of a correlation matrix. In the classic matrix representation the cells of the matrix can be shaded or colored 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.

```
m <- cor(d.pizza[,WhichNumerics(d.pizza)], use="pairwise.complete.obs")
PlotCorr(m, cols=colorRampPalette(c("red", "black", "green"), space = "rgb")(20))
PlotWeb(m, col=c(hred, hblue)
```

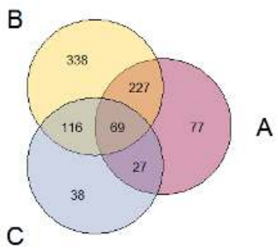




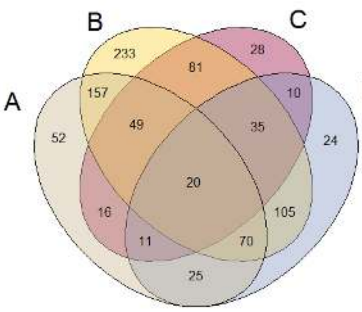
Venn plots

In rare cases one might want to plot a Venn diagram. This function does that for up to 5 datasets using the simple proposed geometric representatations.

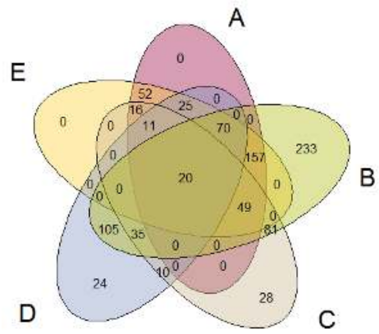
```
example(PlotVenn)
PlotVenn(x=x[1:3], col=SetAlpha(c(PalHelsana()[c(1,3,6)]), 0.4))
PlotVenn(x=x[1:4], col=SetAlpha(c(PalHelsana()[c(1,3,6,4)]), 0.4))
PlotVenn(x=x[1:5], col=SetAlpha(c(PalHelsana()[c(1,3,6,4,7)]), 0.4))
```



3 sets



4 sets



5 sets

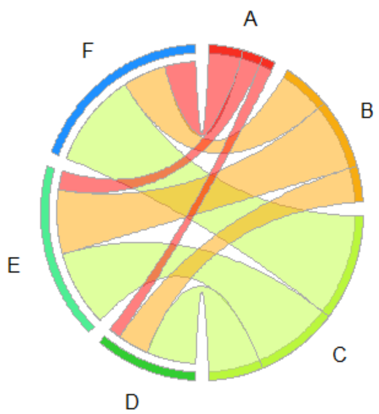
Associations with circular plots

This is, although working, still 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

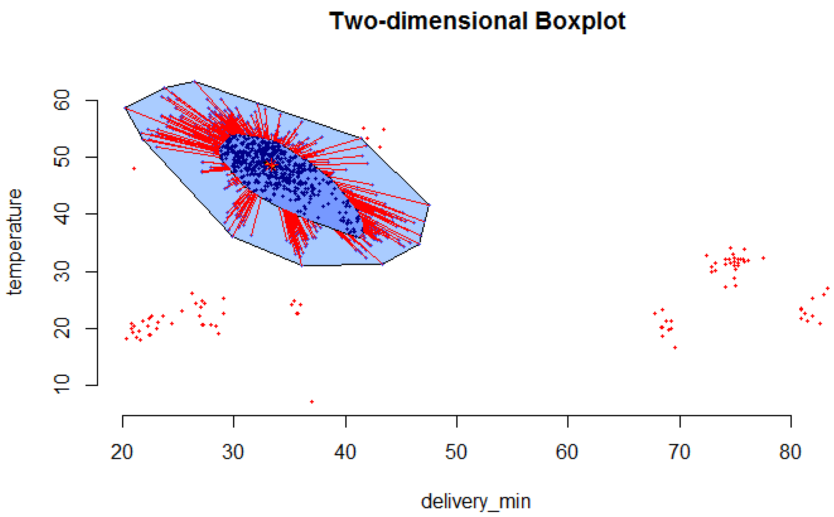
	D	E	F
A	2	3	5
B	5	10	7
C	8	12	15



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")
```



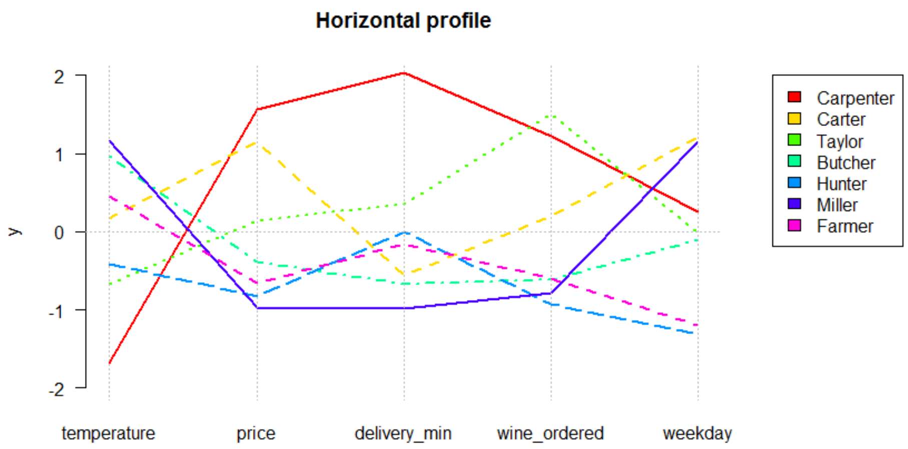
Lineplots

There are many flavours of lineplot, most (all?) of them handled by the function `matplot`.
We desist from defining own plot functions, which would only set the suitable arguments for an existing function, as we fear we would run into a forest of functions, loosing overview.

Yet the parametrization of `matplot` can be a haunting experience and so we integrate some common examples here.

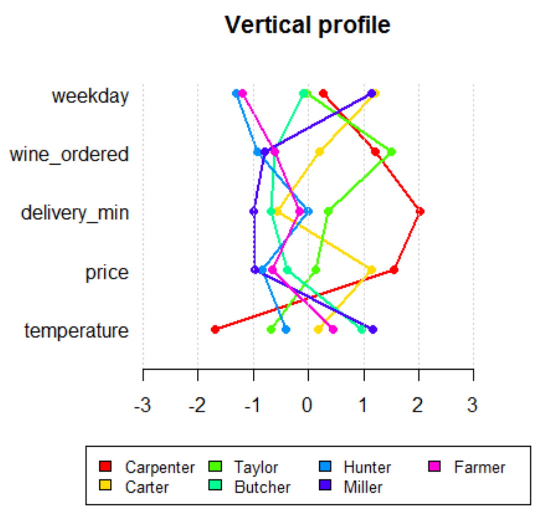
Let's for example have a horizontal profile of the driver's characteristics.

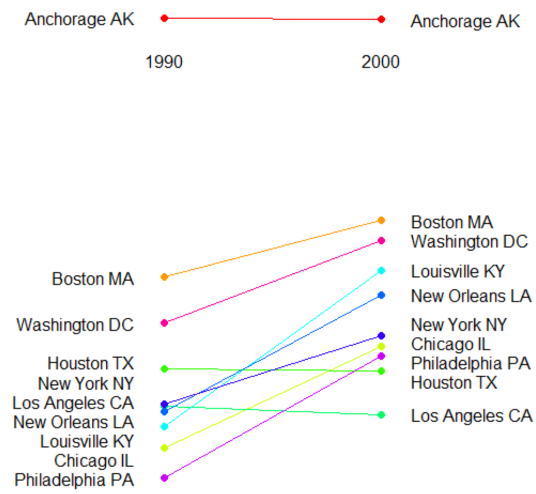
```
m <- data.frame(lapply(d.pizza[,c("temperature","price","delivery_min","wine_ordered","weekday")],
                tapply, d.pizza$driver, mean, na.rm=TRUE))
(ms <- data.frame(lapply(m, scale)))        # lets scale that

          temperature price delivery_min wine_ordered weekday
Carpenter       -1.68  1.56         2.03         1.22    0.26
Carter           0.18  1.15        -0.56         0.20    1.21
Taylor          -0.68  0.14         0.35         1.50   -0.02
Butcher          0.97 -0.39        -0.67        -0.62   -0.10
Hunter          -0.41 -0.83        -0.01        -0.92   -1.31
Miller           1.17 -0.98        -0.99        -0.78    1.15
Farmer           0.45 -0.66        -0.16        -0.60   -1.20

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"

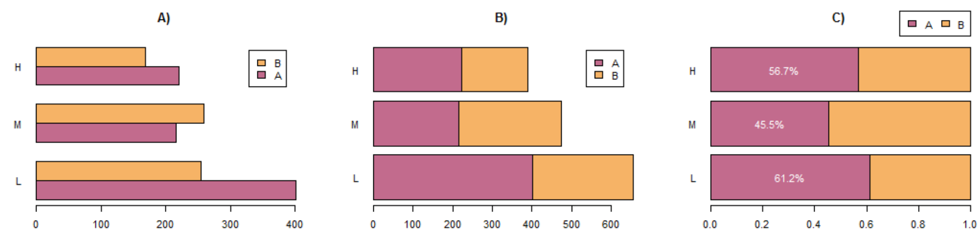
Plot B is sometimes called bumpchart (Jim Lemon).

```
# 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.

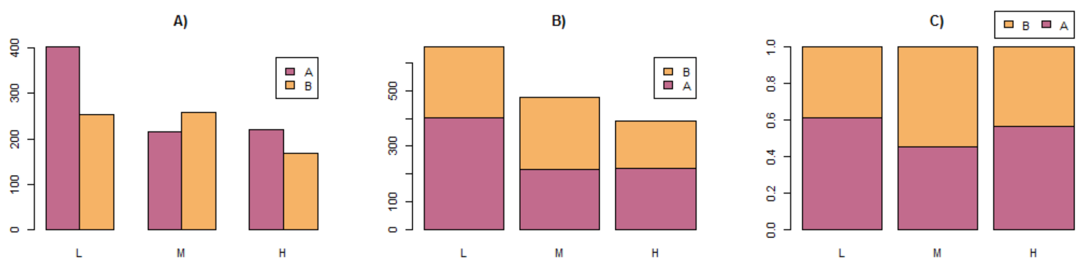
```
windows(height=3, width=11)
par(mfrow=c(1,3))

# 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.

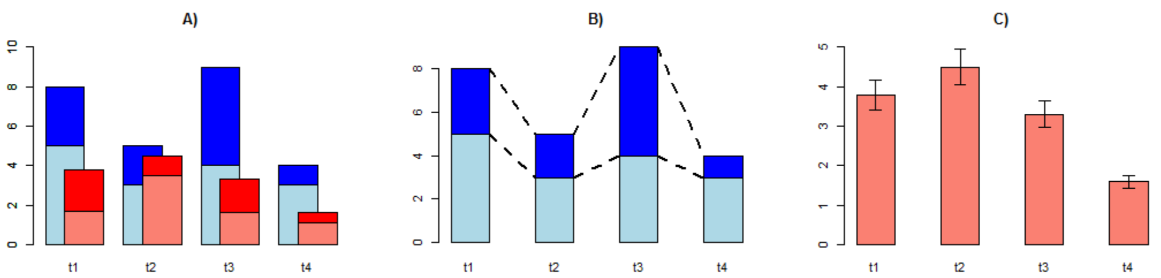
```
windows(height=3, width=11)
par(mfrow=c(1,3))

# 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(m.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)" )
arrows( x0=b, y0=cred * .90, y1 = cred * 1.1, angle=90, code=3, length=0.05 )
```



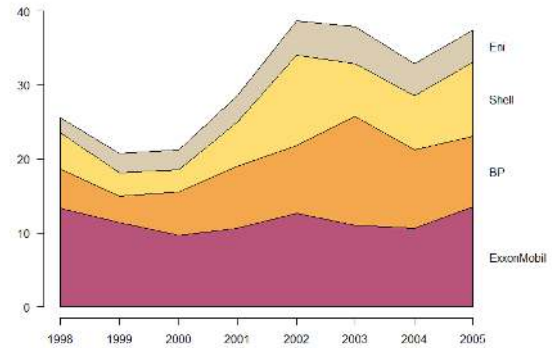
Areaplot

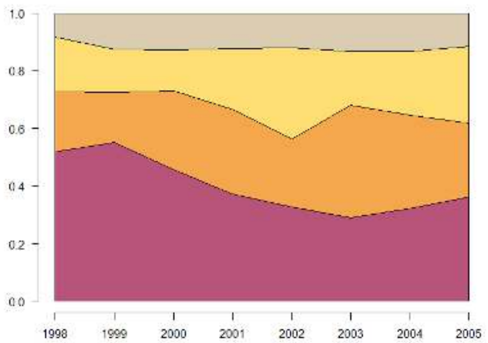
This function produces an areaplot.

```
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(mar=c(5,4,5,5))
PlotArea(t.oil, col = PalTibco()[4:7], 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 = PalTibco()[4:7], 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
```

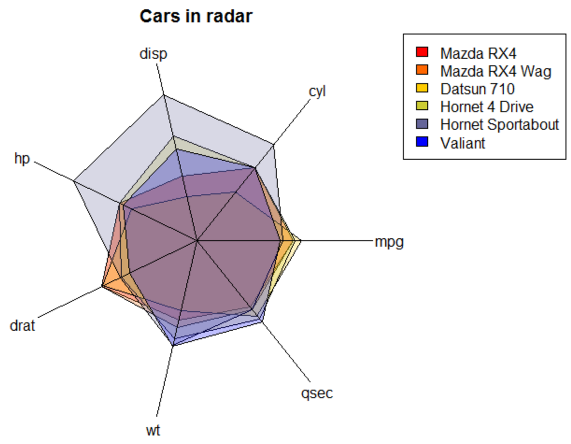




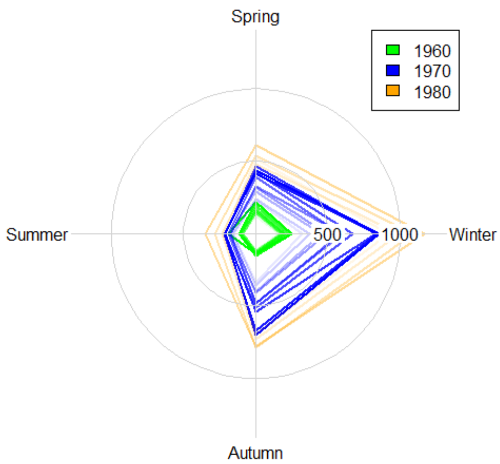
PlotPolar (Radarplot)

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))
```



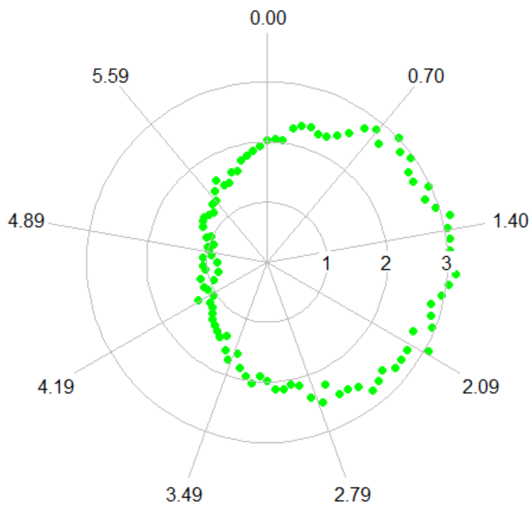
A)

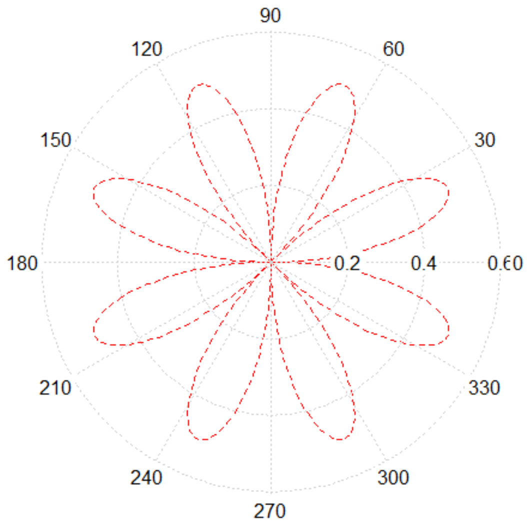


B)

```
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()
```

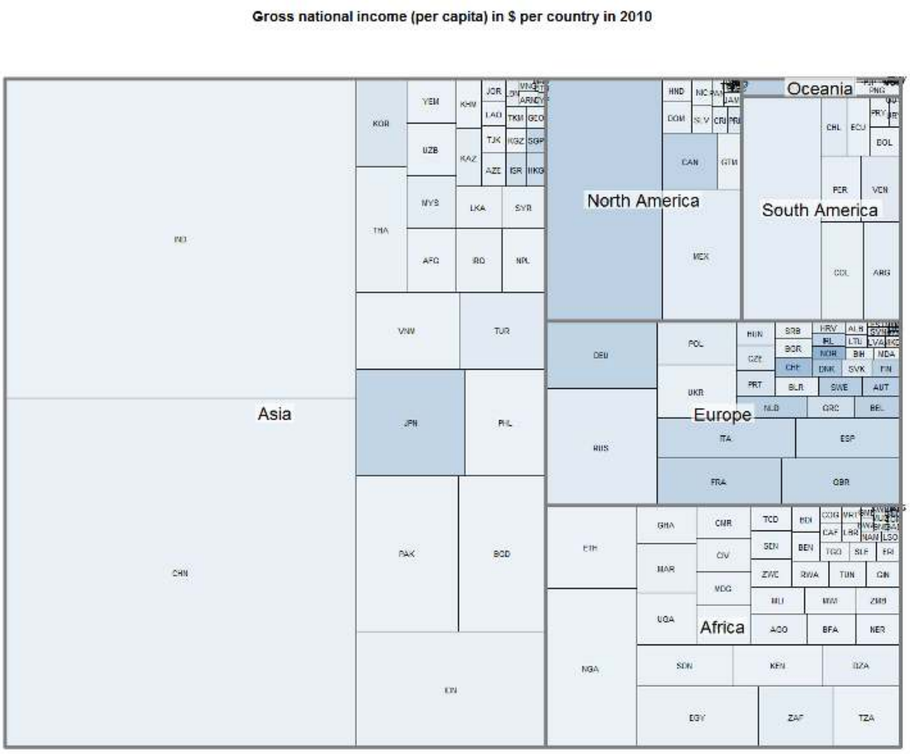




PlotTreemap

This function produces a treemap.

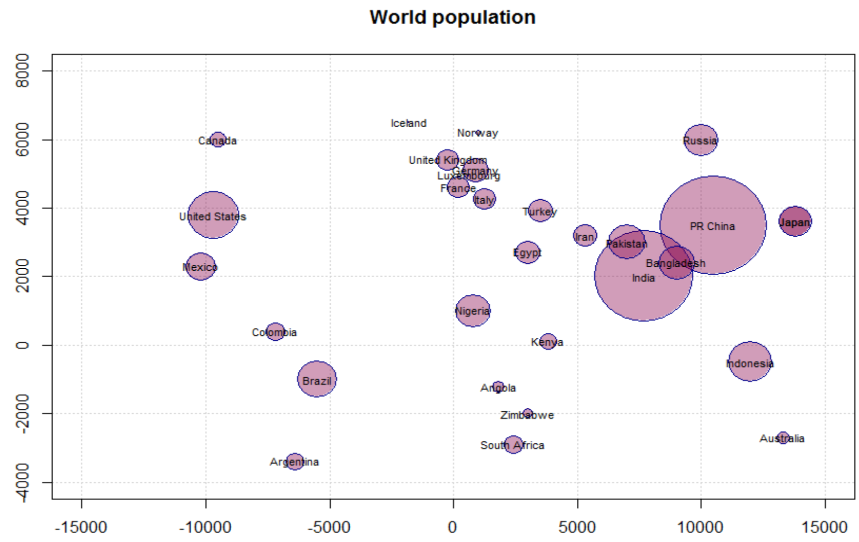
```
# get some data
data(GNI2010, package="treemap")
gn <- GNI2010[,c("iso3","population","continent","GNI")]
gn <- gn[gn$GNI!=0,]

# 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

Bubble plot of the world population.

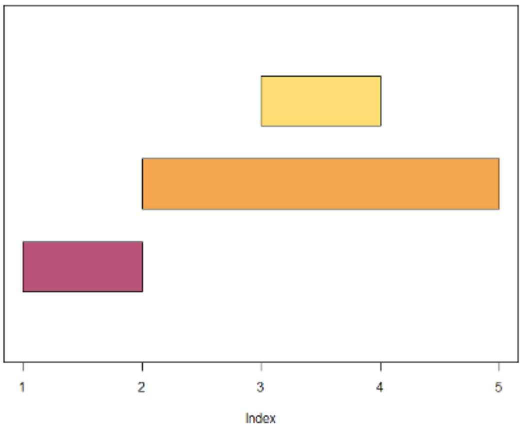
```
PlotBubble(d.world$x, d.world$y, area=d.world$pop/90, col=SetAlpha("deeppink4",0.4), border="darkblue",
          xlab="", ylab="", panel.first=grid(), main="World population")
text(d.world$x, d.world$y, labels=d.world$country, cex=0.7, adj=0.5)
```



PlotHorizBar

Simple implementation for plotting horizontal bars.

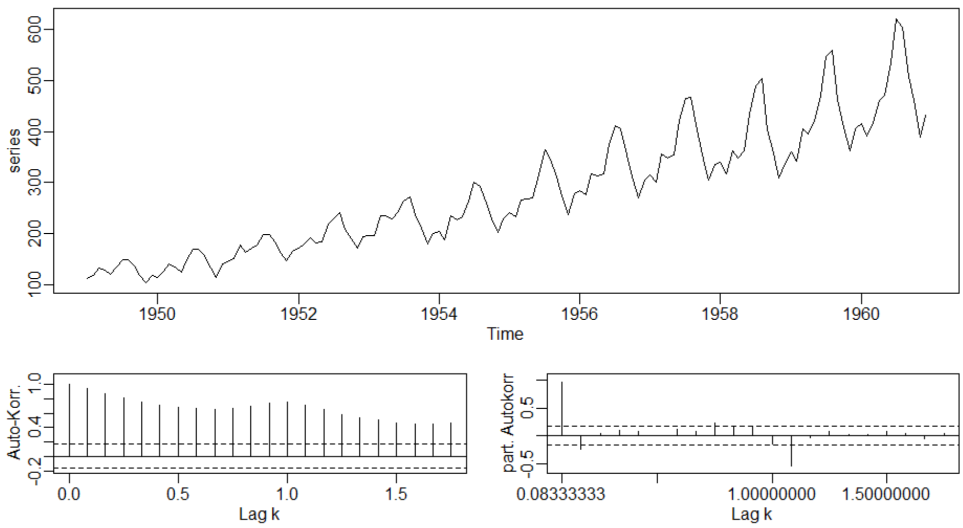
```
PlotHorizBar(from=c(1,2,3), to=c(2,5,4), grp=c(1,2,3), col=PalHelsana()[1:3])
```



PlotACF

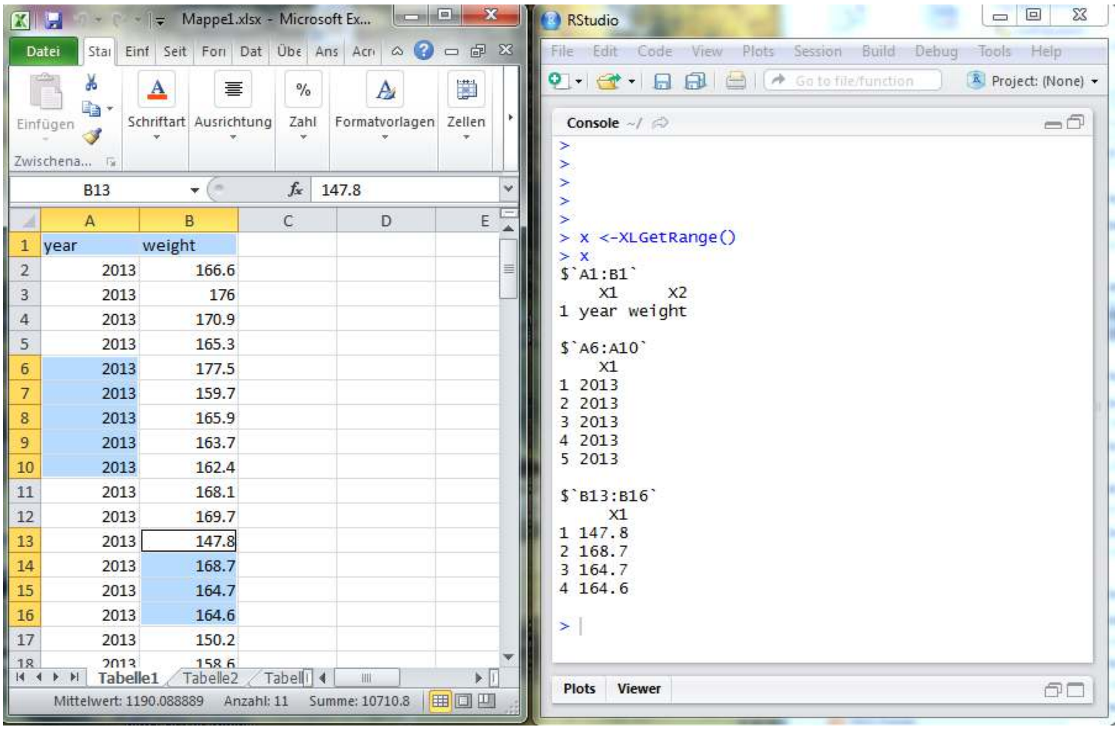
This produces a combined plot of a time series and its autocorrelation and partial autocorrelation

```
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.



---

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