Christophe Lalanne

A Visual Guide to R Graphics and Data Munging

With 65 illustrations

9a3f538
2012/02/16

The greatest value of a picture is when it forces us to notice what we never expected to see. — John Tukey

At their best, graphics are instruments for reasoning. — Edward Tufte

Contents

CHAPTER 1

Getting started with R graphics

1.1 Why R?

In the “GNU world”, most of the plotting program expect data from text file (tab-delimited or csv) arranged by columns, with extra grouping levels denoted by string or integer codes. This is the case with gnuplot[1] (`http://www.gnuplot.info/`) or plotutils (`http://www.gnu.org/software/plotutils/`), for example.

Here is how we could create an histogram in gnuplot, from a series of 500 gaussian variates generate using R as follows:

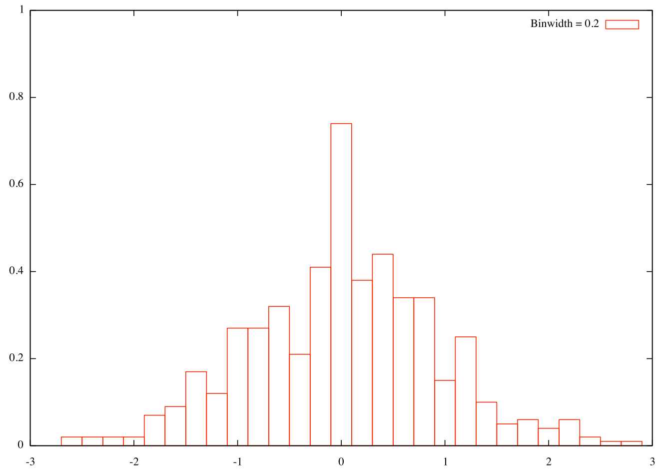
```
$ Rscript -e 'cat(rnorm(500), sep="\\n")' > rnd.dat
```

Then, in gnuplot, we can run

```
bw=0.2
n=500
bin(x,width)=width*int(x/width)
tstr(n)=sprintf("Binwidth = %1.1f\n", n)
set xrange [-3:3]
set yrange [0:1]
set boxwidth bw
plot 'rnd.dat' using (bin($1,bw)):(1./(bw*n)) smooth frequency \
  with boxes title tstr(bw)
```

to get the picture shown below. (Note that we didn’t try to customize anything, except the title.)

[1] **janert09**.



The above example shows one important aspect of using a dedicated statistical package: Gnuplot has no function to draw an histogram, and we have to write some code to perform additional tasks, like binning in this case. The same applies for plotutils. We could make use of external programs to do that, like the GSL library (see example 22.11 from the manual, `http://www.gnu.org/software/gsl/manual/`), but this two-stage approach is rather likely to be cumbersome for repetitive tasks.

The author found that Stata is one of the only great alternative to R, but it has its cost. In fact, this textbook is largely inspired from one of Stata Press book on Stata graphical capabilities.[2]

1.2 The R graphical model

1.3 Base vs. `lattice` graphics

1.4 The grammar of graphics

[2]**mitchell08**.

CHAPTER 2

Data management

2.1 Structuring data

In R, the most common structure used to store a data set with mixed-type variables is a `data.frame`. Such an `R` object presents several characteristics that makes it most appropriate for managing statistical data structure, with few exceptions (e.g., when one only has to work with aggregated data or two-way tables). It should be noted that other data structures might be more appropriate, for example when one is interested in time series analysis, but see the `zoo` package.[1]

Many `R` functions accept `data.frame` as input, and further allow to subset or index it for computation or visualization purpose. In addition to receiving a `data.frame`, some `R` commands allow to use a *formula* notation, where the right and left-hand side are separated by the ~ (tilde) operator. The use of together with a `data.frame` simplify the accession of variable in a given environment. This is especially true when using the `lattice` package which is entirely based on formula, even if this is not apparent at first sight.

2.2 Managing data

Consider, for example, the *low birth study* which is discussed at length in Hosmer and Lemeshow's textbook on logistic regression.[2] A quick look at the variables should make it clear that they won't be treated the way we like them to be considered: mother's ethnicity status (`race`) takes three integer values, without any explicit meaning.

```
data(birthwt, package="MASS")
str(birthwt)
```

[1] **zeilis05**.
[2] **hosmer89**.

Instead, we might want to recode all categorical predictors as R factors, like this:

```
birthwt <- within(birthwt, {
  low <- factor(low, labels=c("No","Yes"))
  race <- factor(race, labels=c("White","Black","Other"))
  smoke <- factor(smoke, labels=c("No","Yes"))
  ui <- factor(ui, labels=c("No","Yes"))
  ht <- factor(ht, labels=c("No","Yes"))
})
```

In case we would like to consider one of the above factors as a numerical variable, we can now use `as.numeric` and R will take care of attributing the lowest integer score to the baseline category. Of course, there might be occasion where we would like to change that reference level; or, we might want to collapse two discrete categories. Again, there are simple commands to do that, for example:

```
birthwt$low <- relevel(birthwt$low, "Yes")
levels(birthwt$race)[2:3] <- "Black+Other"
```

Another common task consists in transforming some predictors, either for visualization purpose or when building an explantory or predictive model. As a simple example, we can imagine centering some of the predictors of interest, like age, in the above example. The `within` or `transform` command can be used to append the centered variable to the list of variables present in the `data.frame`:

```
birthwt <- transform(birthwt, age.c=scale(age, scale=FALSE))
```

Likewise, we may want to recode previous premature labours (`ptl`) as yes/no and number of physician visits during the first trimester (`ftv`) as one/more than one, like shown below (we show two different syntax that basically perform the same task by relying on R indexing):

```
birthwt <- transform(birthwt, ptl.yn=factor(ptl > 0, labels=c("No","Yes")),
                     ftv.c=factor(ifelse(ftv < 2, "1", "2+")))
```

If there is some reason to treat `ftv` as an ordered factor, a command like

```
as.ordered(cut2(birthwt$ftv, c(0, 1, 2, 6)))
```

might do the job. (This would also be possible with the base `cut` function, but the one in Hmisc[3] has better default options and it is more flexible.)

2.3 Indexing, subsetting, conditioning

A lot of statistical operations that practictioners use to apply on a given dataset are mostly variations around the idea of indexing or subsetting.

We have already seen an example of indexing when we recoded factor levels of two explanatory variables in the low birth dataset.

[3] **harrell11**.

2.4 Summarizing data

Statisticians generally spend a great part of their time in data cleansing, data transformation or re-expression,[4] and data visualization. The `Hmisc` package includes numerous `R` functions that will facilitate the task of data checking (`describe` provides "codebook" facilities), summarizing (`summary.formula`) or aggregating data, (`summary.formula`).

The `plyr` package[5] offers a general solution to those kind of tasks.

[4] **hoaglin83**.
[5] **wickham11**.

CHAPTER 3

Univariate distributions

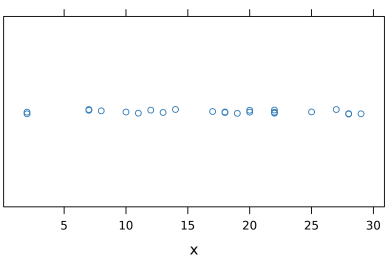
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.1 Stripchart

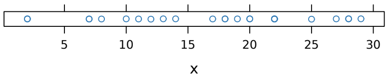
Stripchart aims at showing the distribution of a series of continuous measurements (much like scatterplot for 2D data discussed in § 4.2). They are useful for small to moderate dataset. With large N it is proably better to switch to alternative displays, see next sections.

```
x <- sample(1:30, 25, replace=TRUE)
stripplot(~ x, jitter.data=TRUE, factor=.8, aspect=.5)
```

With this synthetic dataset where several observations can take the same value, jittering point locations on the horizontal and vertical axes ensures a better representation.

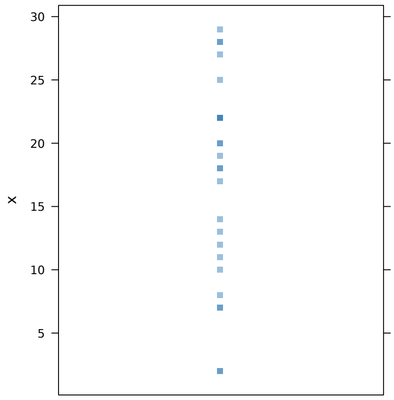


```
stripplot(~ x, jitter.data=TRUE, factor=.8, aspect="xy")
```



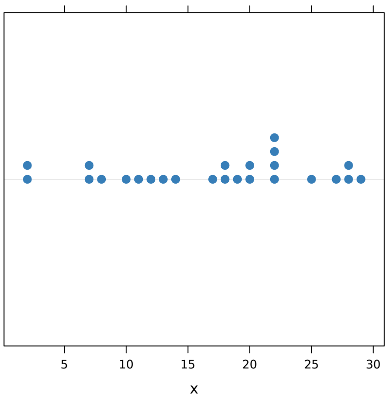
A better way of flattening the display is to use an "xy" aspect.

```
stripplot(x ~ 1, horizontal=FALSE, jitter.data=TRUE, aspect=1.2,
          scales=list(x=list(draw=F)), xlab="", pch=15, alpha=.5)
```



This is basically the same picture but the x and y axis have been transposed. We used a different symbol and transparency to highlight where replication occurs in the data. Obviously, that won't work so nicely with larger sample size or a higher density of replication.

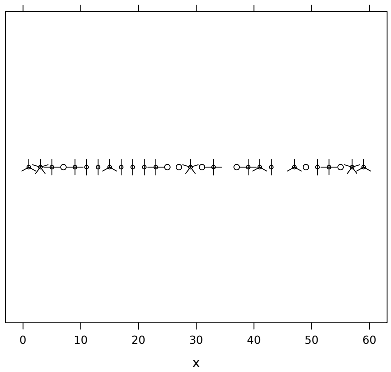
```
stripplot(~ x, panel=HH::panel.dotplot.tb, cex=1.2, factor=.2)
```



In contrast to the base `stripchart` function, there is no way of imposing a stacked display in `lattice`. However, there is some convenient panel function in the `HH` package.

```
x <- sample(seq(1, 60, by=2), 75, replace=TRUE)
stripplot(1 ~ x, panel=panel.sunflowerplot, col="black",
          seg.col="black", seg.lwd=1, size=.08)
```

With possible replicates, it is also interesting to use "sunflowers" where multiple leaves are used for each duplicate. The custom panel function mimics the base `sunflowerplot` function. For an alternative way of embedding "sunflowers" into a `lattice` display, see the following thread on R-help: `http://bit.ly/Ig4RTq`. Note that we remove y-axis annotation using commands presented before (i.e., using `scales=`).

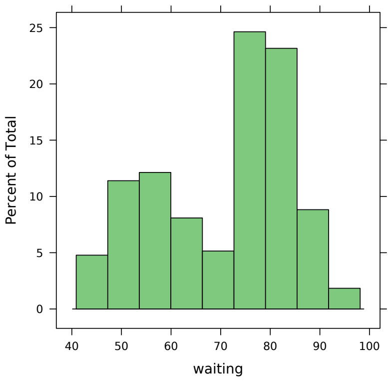


3.2 Histograms

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

```
histogram(~ waiting, data=faithful)
```

A simple histogram of waiting time expressed as density. Note that forgetting the ~ operator will raise an error message.

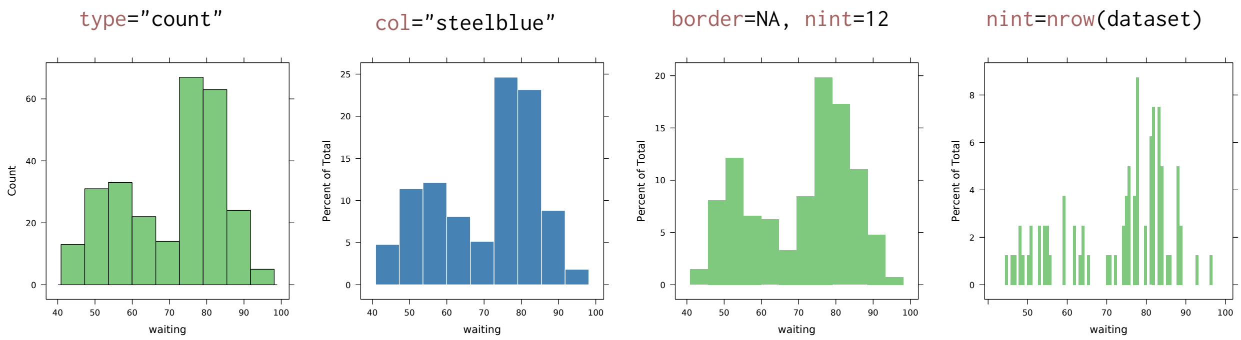


faithful. Waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

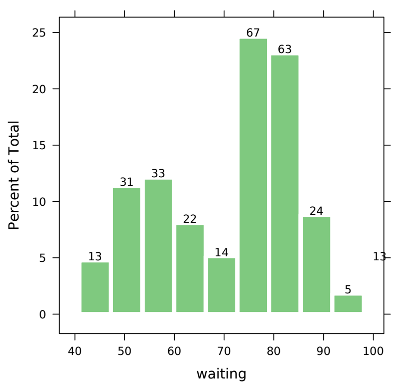
Box 3.1 shows some custom settings with the `faithful` dataset. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Box 3.1

Common options for `histogram` include displaying counts instead of percents, or varying bar color. It is also possible to change default bin size. When `nint=nrow(dataset)`, we have a so-called high-density vertical lines, much like when using `plot(..., type="h")`.



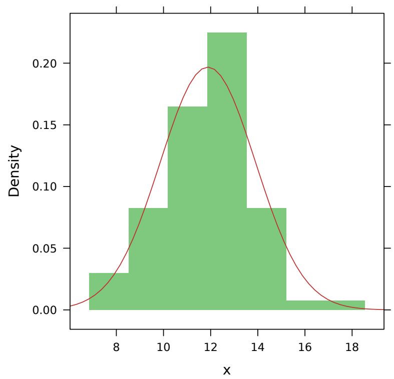

```
p <- histogram(~ waiting, data=faithful, lwd=5, type="percent",
                border="white")
x.breaks <- p$panel.args.common$breaks
y.values <- table(cut(faithful$waiting, breaks=x.breaks))
update(p, panel=function(...) {
  panel.histogram(...)
  panel.text(x.breaks+diff(x.breaks)[1]/2,
             y.values/nrow(faithful)*100,
             as.character(y.values), cex=.8, adj=c(.5,0))
})
```



The following example demonstrates how a default histogram displaying percent data can be annotated with counts data. This is intended to show how we can steal away default setting to display the distribution of discrete values.)

```
x <- rnorm(80, mean=12, sd=2)
histogram(~ x, type="density", border=NA,
          panel=function(x, ...) {
            panel.histogram(x, ...)
            panel.mathdensity(dmath=dnorm, col="#BF3030",
                              args=list(mean=mean(x),sd=sd(x)))
          })
```

An example where we superimposed a normal density with parameters estimated from the sample.)



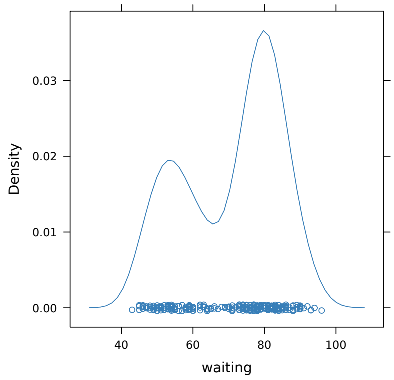
3.3 Density plots

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

.[1]

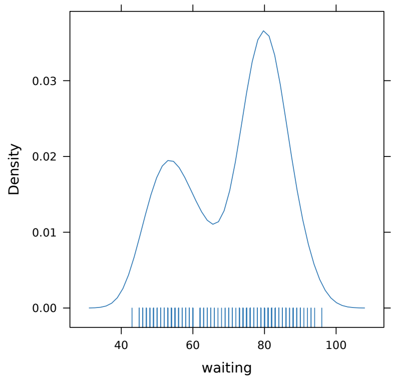
[1] **silverman86**.

```
densityplot(~ waiting, data=faithful)
```



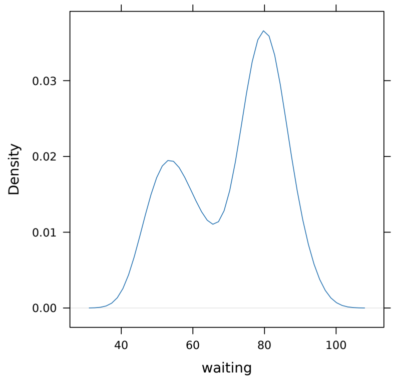
The default panel relies on R's `density` function. As such, the default kernel is `gaussian` with $n = 512$ equally spaced at which the density is estimated. The choice of the bandwith follows Silverman's rule of thumb, namely $\min(0.9\mathrm{SD}, \mathrm{IQR}/1.34n)$. An alternative bandwith can be selected using `bw="SJ"`.

```
densityplot(~ waiting, data=faithful, plot.points="rug")
```



Instead of a mini stripchart displayed at $y = 0$, a "rugplot" can be preferred. It might help spotting possible local concentration of data points, compared to simple jittered points.

```
densityplot(~ waiting, data=faithful, plot.points=FALSE, ref=TRUE)
```



Sometimes, adding a reference line crossing at $y = 0$ may be informative, especially for multimodal distributions. It is advised to avoid plotting individual observations like was done in the preceding graphics.

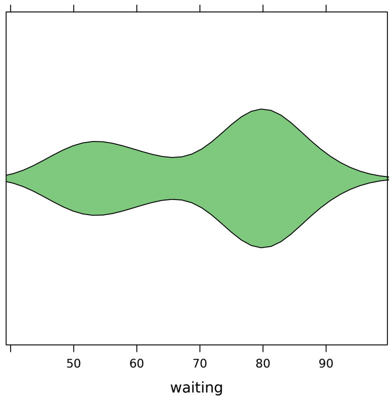
An alternative way of presenting density plots are so-called "violin plots"[2] which feature the main components of a boxplot (§ 3.5) and a kernel density estimation, and "bean

[2] **hintze98**.

plots"[3] where density trace are mirrored to form a polygon shape. The latter presents the advantage of allowing asymmetrical plotting depending on a grouping factor.

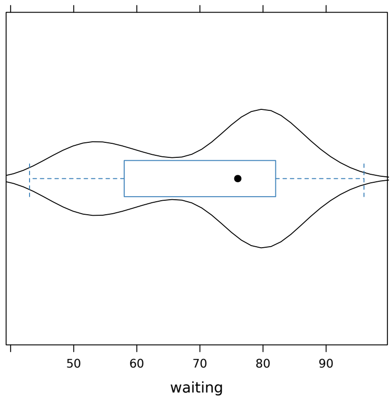
```
bwplot(~ waiting, data=faithful, panel=panel.violin)
```

A simple "violin" panel.




```
bwplot(~ waiting, data=faithful,
       panel=function(..., box.ratio) {
         panel.violin(..., col="transparent", varwidth=FALSE,
                      box.ratio=box.ratio)
         panel.bwplot(..., fill=NULL, box.ratio=.15)})
```

A simple "violin" panel.



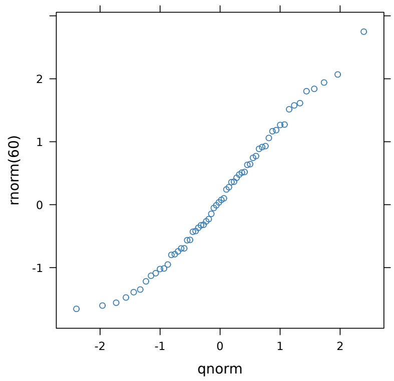
3.4 Quantile and related probability plots

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est,

[3] **kampstra08**.

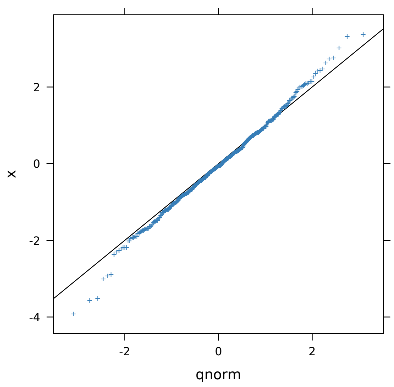
iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

```
qqmath(~ rnorm(60))
```



A simple quantile plot of 60 gaussian variates. The theoretical quantiles of an $\mathcal{N}(0;1)$ are shown on the x-axis.

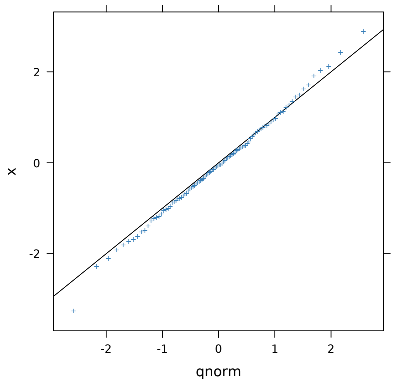
```
x <- rt(500, df=20)
qqmath(~ x, pch="+", abline=c(0,1), dist=qnorm)
```



This is basically asking to draw the same QQ-plot, but with a higher number of data points coming from a different distribution (Student $t(20)$). A reference line is added to facilitate comparison. This basically shows how closely the t-distribution can be approximated by an $\mathcal{N}(0;1)$ when n gets very large.

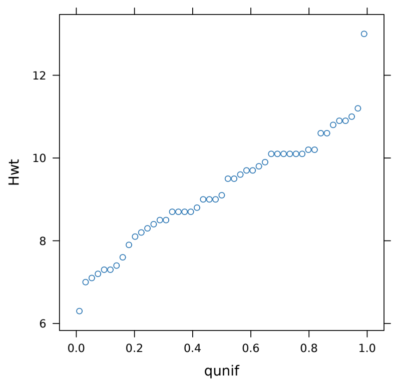
```
qqmath(~ x, pch="+", abline=c(0,1), dist=qnorm,
       f.value=ppoints(100))
```

Same as above but subsampling data points.




```
qqmath(~ Hwt, data=cats, subset=Sex == "F", dist=qunif)
```

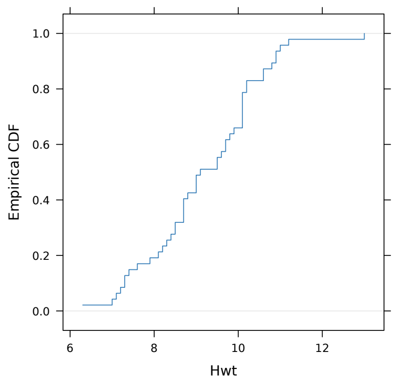
Here is one way to show the cumulative distribution function (CDF) of some sample dataset. Note that we need to explicitly need to ask `qqmath` to use the Uniform distribution as a reference.



cats. The heart and body weights of samples of male and female cats used for *digitalis* experiments. The cats were all adult, over 2 kg body weight.

```
ecdfplot(~ Hwt, data=cats, subset=Sex == "F")
```

An alternative way of plotting the empirical CDF is to rely on the `ecdfplot` function from the `latticeExtra` package.

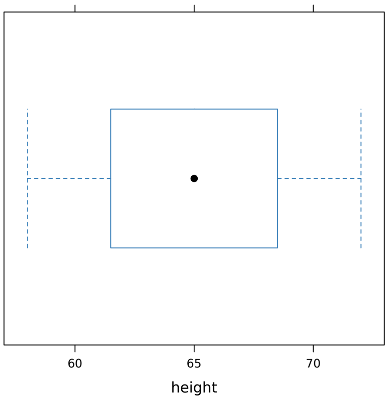


3.5 Boxplots

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

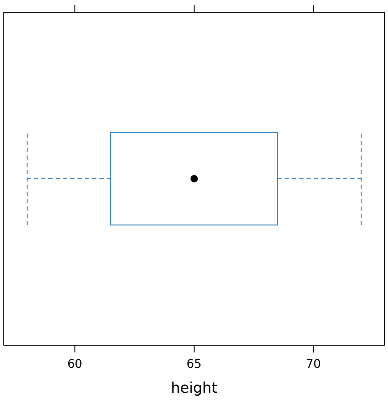
```
bwplot(~ height, data=women)
```

women. This data set gives the average heights and weights for American women aged 30-39.



Boxplot are shown in horizontal mode by default. Internally, the `boxplot.stats` function is used so that hinges corresponds to the first and third quartile while notches extends to $\pm 1.58 \mathrm{IQR}/\sqrt{n}$. It provides a visual summary analogous to Tukey's five number (see `fivenum`).

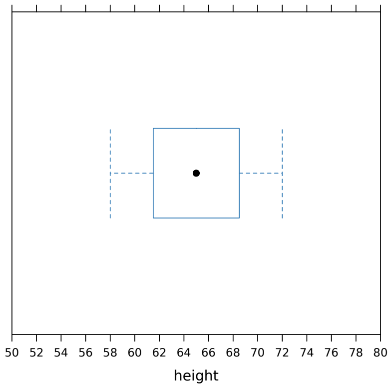
```
bwplot(~ height, data=women, box.ratio=.5)
```



This is the same picture but with a thinner box. When there are several boxplots to draw side by side, this might be useful.

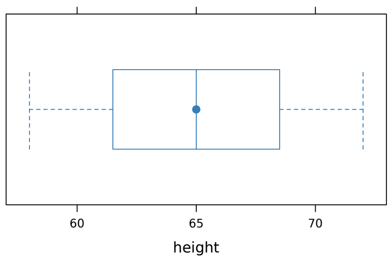
```
bwplot(~ height, data=women, box.ratio=.5,
       scales=list(x=list(limits=c(50, 80), at=seq(50, 80, by=2))))
```

A more detailed x-scale has been created (without prejudice to its usefulness) by simply updating the scales= argument.




```
bwplot(~ height, data=women, aspect=.5,
       panel=function(x, ...) {
         panel.bwplot(x, pch="|", ...)
         panel.points(mean(x), 1, pch=19, cex=1)
       })
```

It is possible to alter the way boxplot are drawn, but also the statistical summaries that are displayed. For instance, in the above code, we computed the mean (drawn as a vertical bar inside the box) in addition to the median. Of note, if there are missing values, we should add na.rm=TRUE when calling mean(x).

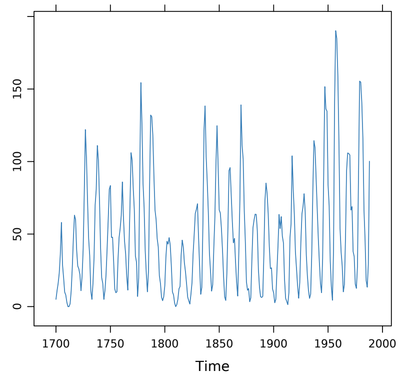


3.6 Time series

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

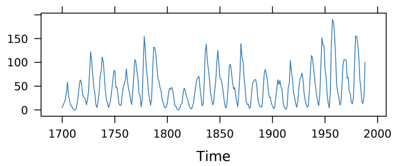
```
xyplot(sunspot.year)
```

sunspot.year. Yearly numbers of sunspots from 1700 to 1988.



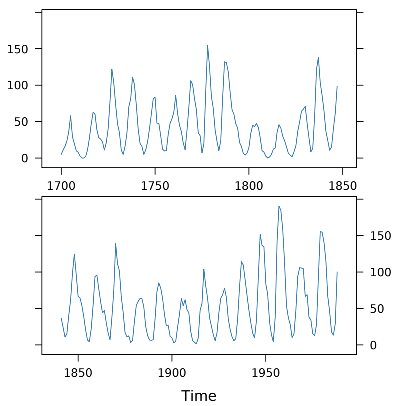
A simple time-series is displayed as a lineplot, but taking care of arranging the x-axis for time measurements.

```
xyplot(sunspot.year, aspect=.3, scales=list(y=list(rot=0)))
```



A more comprehensive picture after aspect ratio has been lowered so as to better highlight the cyclic component.

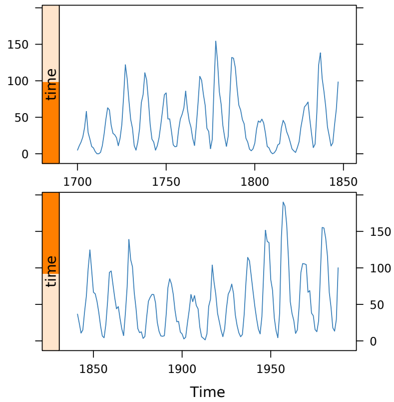
```
xyplot(sunspot.year, strip=FALSE, cut=list(number=2, overlap=.05))
```



The same time series cut into two pieces with 5% of overlap.

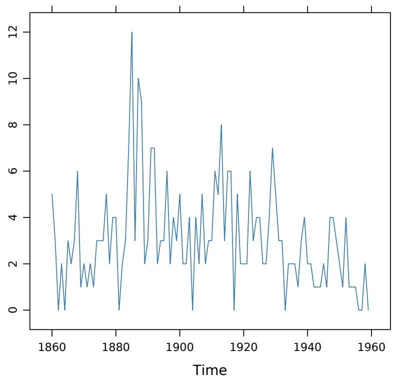
```
xyplot(sunspot.year, strip=FALSE, strip.left=TRUE,
       cut=list(number=2, overlap=.05))
```

Now we highlight explicitely that the two series of measurements are related by adding a colored ribbon denoting time period.




```
xyplot(zoo(discoveries))
```

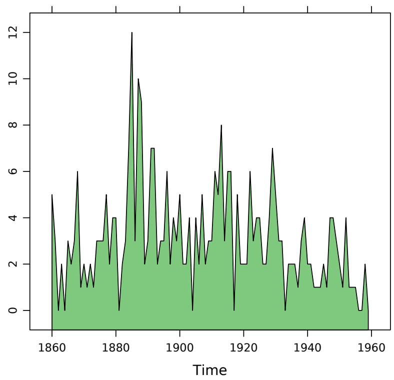
The zoo has to be loaded before using the above command. Briefly, it takes care of handling time-series data correctly, and it is interfaced to lattice's xyplot as an S3 method (see xyplot.zoo).



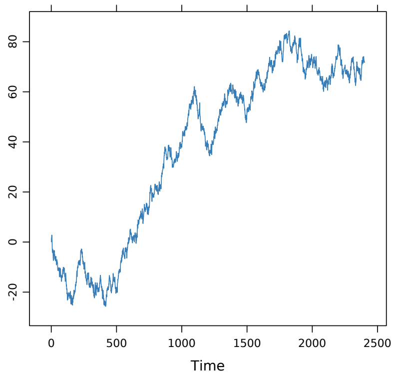
discoveries. The numbers of "great" inventions and scientific discoveries in each year from 1860 to 1959.

```
xyplot(zoo(discoveries), panel=panel.xyarea)
```

It is possible to add a shaded area by using a specific panel function from the latticeExtra package.



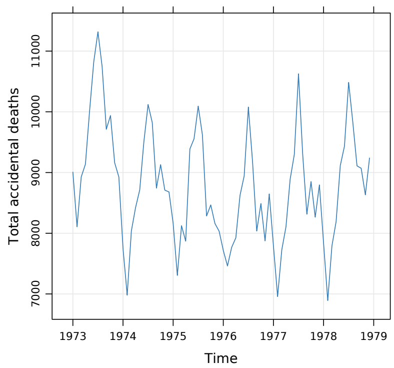
```
xt <- ts(cumsum(rnorm(200 * 12)))
p <- xyplot(xt)
```



The `lattice` package works with `ts` objetcs too.

```
xt <- zoo(accdeaths)
xyplot(xt, type=c("l","g"), ylab="Total accidental deaths")
```

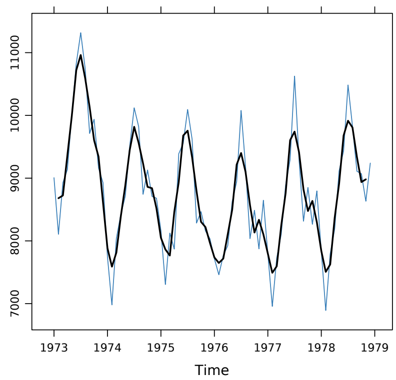
accdeaths. A regular time series giving the monthly totals of accidental deaths in the USA.



Another regular time series is.

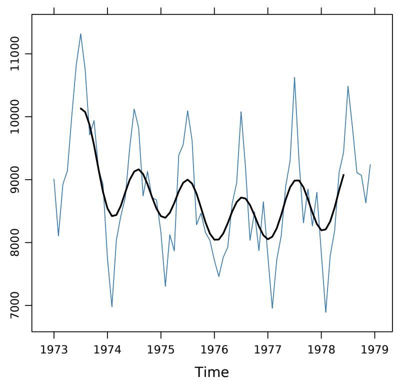
```
xyplot(xt,
       panel=function(x, y, ...) {
         panel.xyplot(x, y, ...)
         panel.lines(rollmean(zoo(y, x), 3), lwd=2, col=1)
       })
```

The same dataset with a rolling mean (of width 3).



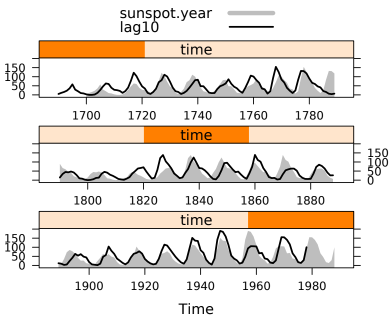

```
xyplot(xt) +
  layer(panel.tskernel(x, y, c=3, col=1, lwd=2))
```

Discrete symmetric smoothing kernels, available in `latticeExtra`, can be used instead of a rolling mean. Here an approximate gaussian filter was used to highlight the seasonal component.



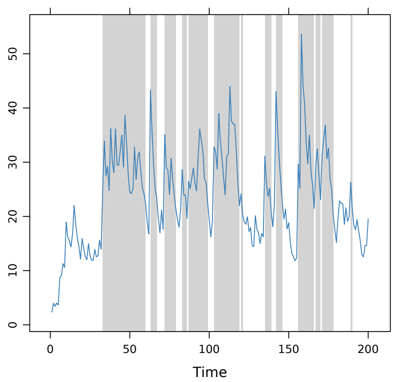
```
xyplot(ts.union(sunspot.year, lag10=lag(sunspot.year, 10)),
       superpose=TRUE, panel=panel.superpose,
       panel.groups=function(..., group.number) {
         if (group.number == 1) panel.xyarea(...)
         else panel.xyplot(...)
       }, border=NA, cut=list(n=3, overlap=0), aspect="xy",
       par.settings=simpleTheme(col=c("grey","black"), lwd=c(5,2)))
```

sunspot.year. Yearly numbers of sunspots between 1700 and 1988.



More complex arrangement can be done, again with the `latticeExtra` panel function. Here, yearly numbers of sunspots are shown together with a lagged version (10 years).

```
flow <- ts(filter(rlnorm(200, mean = 1), 0.8, method = "r"))
xyplot(flow,
       panel=function(x, y, ...) {
         panel.xblocks(x, y > mean(y), col="lightgray")
         panel.xyplot(x, y, ...)
       })
```



blabla blabla

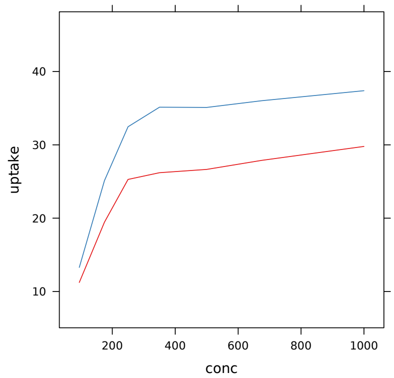
CHAPTER 4

Two-way graphics

This chapter covers graphical displays for two-way relationships, possibly by considering additional variables (numerical or categorical) to highlight ternary relationships. Two-way graphics are not limited to numerical variables as we may be interested in showing the relationships between two ordered categorical variables, two unordered or "nominal" variables. Moreover, as stated above, categorical or discretized variables can be used to provide additional information on top of a line- or scatter-plot by simply varying point size, point or line colors, and so on. Of course, we could extend this idea to the point of displaying sixth dimensions in a single graph (e.g., using symbol with varying length and width, color, and shading pattern). But, such a complex graph would likely be poorly readable and uninformative in the end. So, this chapter basically provides necessary `R` command to create line-plot, scatter-plot, bar-plot, dot-plot

4.1 Lineplots

```
xyplot(uptake ~ conc, data=CO2, groups=Treatment, type="a")
```



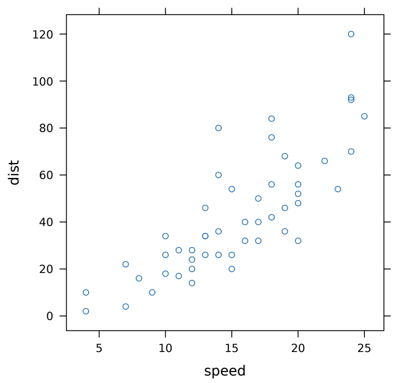
Automatic averaging.

4.2 Scatterplots

The basic R command for displaying a two-way scatterplot is `xyplot`. A command like `xyplot(y ~ x)` will produce a 2D plot almost identical to what would be obtained using base graphics, `plot(x, y)`. However, the default layout is generally better and it looks more pretty.

```
xyplot(dist ~ speed, data=cars)
```

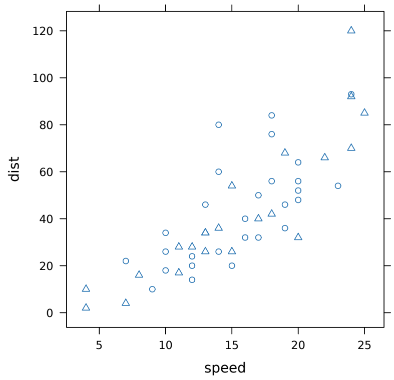
cars. The data give the speed of cars and the distances taken to stop. Note that the data were recorded in the 1920s.



This basic scatterplot show default options when calling the `xyplot` command. The formula interface is used to plot `dist` (y-axis) as a function of speed (x-axis), with automatic determination of axis units.

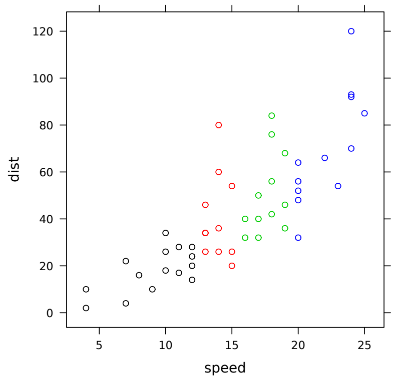
```
xyplot(dist ~ speed, data=cars, pch=rbinom(nrow(cars), 1, .5)+1)
```

We pick a random symbol (○ = 1, △ = 2) for each observation, using the pch= argument. In fact, this argument is transferred to the panel.xyplot function that acts as the default panel function. Note that the vector of symbols should have the same length as the x and y components, otherwise recycling occurs.



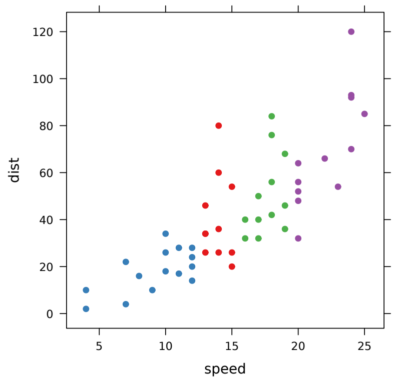

```
xyplot(dist ~ speed, data=cars,
       col=with(cars, cut(speed, breaks=quantile(speed),
                          include.lowest=TRUE)))
```

Color (col=) of each observation depends of the quartile they belong to. Note that passing colors this way will override default theming options.

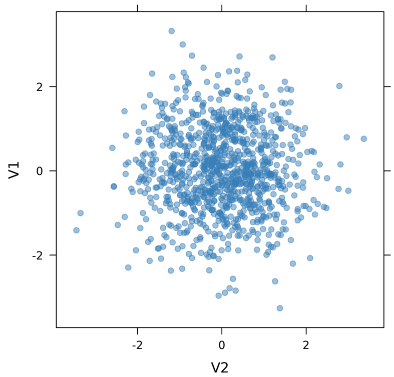



```
xyplot(dist ~ speed, data=cars, pch=19,
       groups=with(cars, cut(speed, breaks=quantile(speed),
                             include.lowest=TRUE)))
```

This is basically the same code as previously shown except that we replaced the col= argument by groups=. This has the advantage of observing the current theme, and this will further facilitate the insertion of an automatic legend.

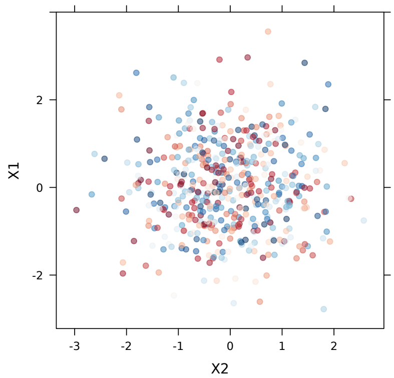


```
xy <- as.data.frame(replicate(2, rnorm(1000)))
xyplot(V1 ~ V2, data=xy, pch=19, alpha=.5)
```



When there are a high proportion of points that overlap, using transparent color may be useful. We replaced the default symbol with its filled counterpart. An equivalent way of specifying transparent color would be to use `rgb(.22, .49, .72, alpha=.5)`.

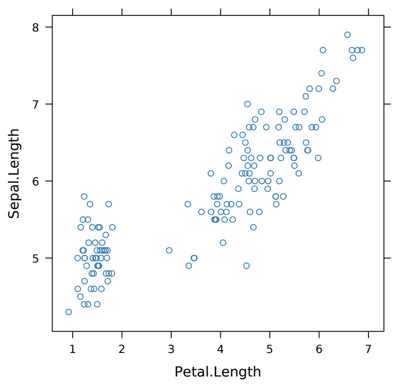
```
dat <- data.frame(replicate(2, rnorm(500)), z=sample(0:40, 500, T))
cols <- colorRampPalette(brewer.pal(11, "RdBu"))(diff(range(dat$z)))
xyplot(X1 ~ X2, data=dat, col=cols[dat$z], pch=19, alpha=.5)
```



Alpha-blending and color palette might be combined as well. Here, we used a pre-defined color scheme (Red to Blue) from the `RColorBrewer` package. As the selected palette has only 11 different colors, whereas the grouping factor, z, has 40 levels, we use linear interpolation to increase the number of available colors.

```
xyplot(Sepal.Length ~ Petal.Length, data=iris, jitter.x=TRUE,
       amount=.2)
```

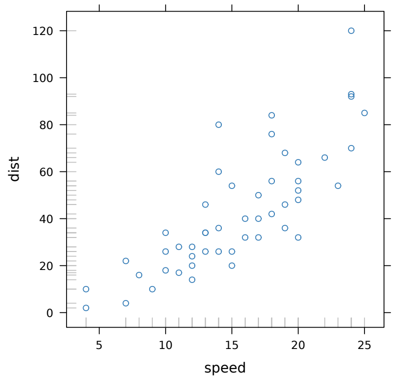
iris. This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*.



As an alternative to transparent colors, one may resort on "jittering". This is also useful when not so many points are available but show few variations on one dimension. The `panel.xyplot` function uses `jitter.x=` and `jitter.y=` to vary x and y coordinates by adding a random shift drawn from a uniform distribution, $\mathscr{U}(-a, a)$, where a stands for the `amount=` parameter.

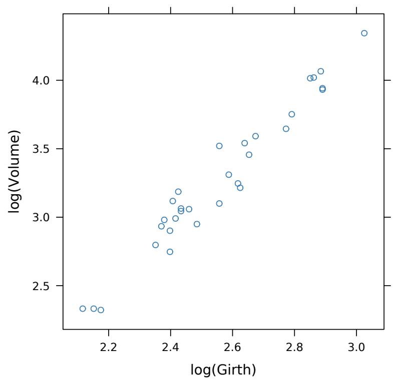
```
xyplot(dist ~ speed, data=cars,
       panel=function(x, y, ...) {
         panel.xyplot(x, y, ...)
         panel.rug(x, y, ...)
       })
```

It is possible to superimpose the univariate distribution of both series of measurement using "rug" plots. Usually, they remain quite discreet (read *non-invasive*) but provide additional information to spot possible asymmetry. We need to ask explicitly for a custom panel, though.



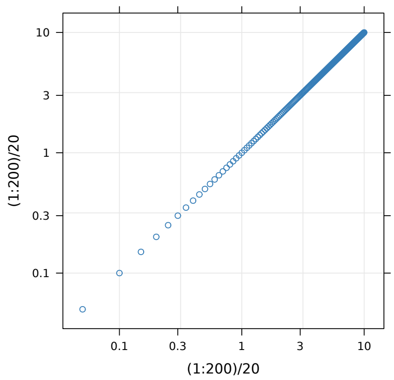

```
xyplot(log(Volume) ~ log(Girth), data=trees)
```

A simple log-log plot. Note that we would have to manually update the scales= component to provide more suitable annotations for the x and y-axis.



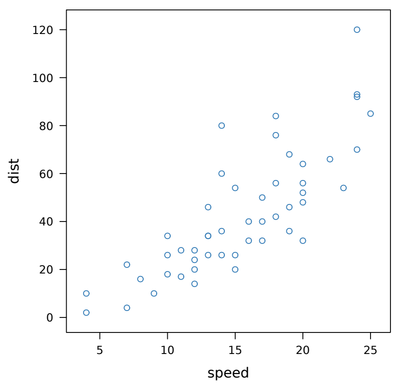
trees. This data set provides measurements of the girth, height and volume of timber in 31 felled black cherry trees. Note that girth is the diameter of the tree (in inches) measured at 4 ft 6 in above the ground.

```
xyplot((1:200)/20 ~ (1:200)/20, type=c("p", "g"),
       scales=list(x=list(log=10), y=list(log=10)),
       xscale.components=xscale.components.log10.3,
       yscale.components=yscale.components.log10.3)
```



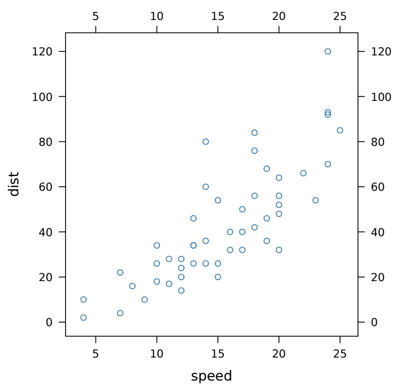
Instead of transforming variables in the formula, it is easier and safer to do this through the scales= parameter. The [x|y]scale.components are convenient functions that help to annotate axes with correct units and tick marks spacing.

```
xyplot(dist ~ speed, data=cars, scales=list(tck=c(1,0)))
```



To get ride of ticks on opposite axes, we can change default values for tck= in the scales= component. The tck= parameter controls the length of tick marks; however, with a vector of length 2 it can be used to deal with left/bottom and right/top axis separately.

```
xyplot(dist ~ speed, data=cars, scales=list(alternating=3))
```

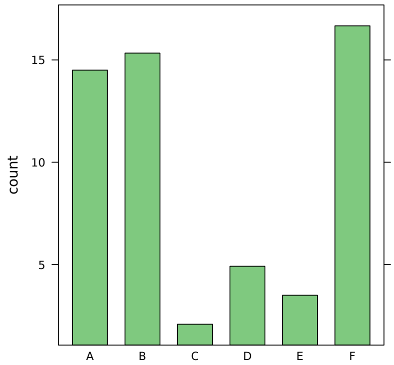


To annotate both axes, we can alter the alternating= parameter. In most case, however, adding grid lines in the background should provide enough information. Using alternating=2 would reverse the annotation of axis (right/top instead of left/bottom).

4.3 Barcharts

```
spray.df <- aggregate(count ~ spray, data=InsectSprays, FUN=mean)
barchart(count ~ spray, data=spray.df)
```

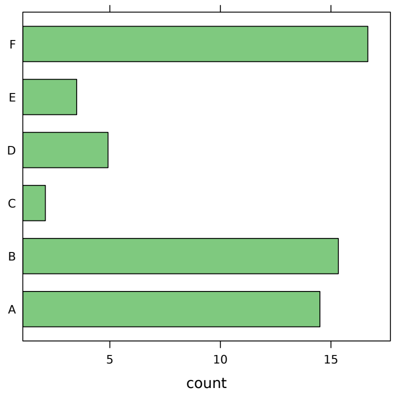
Before using barchart with one continuous and one categorical variable, we need to consider how to aggregate data, in other words what summary measure to consider.



InsectSprays. The counts of insects in agricultural experimental units treated with different insecticides.

```
barchart(spray ~ count, data=spray.df)
```

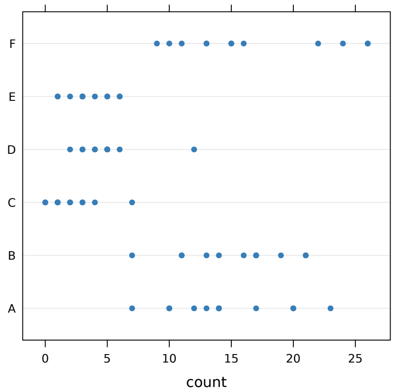
To reverse x and y axis, we just need to use the exchange the right-hand and left-hand side of the preceding formula.



4.4 Dotcharts

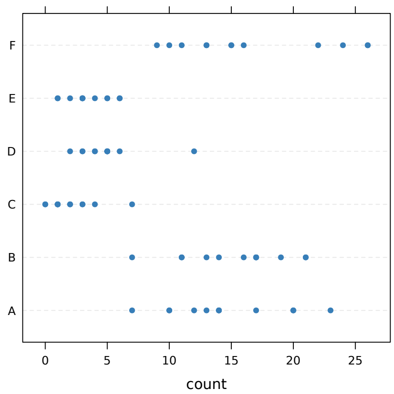
low ink-ratio

```
dotplot(spray ~ count, data=InsectSprays)
```



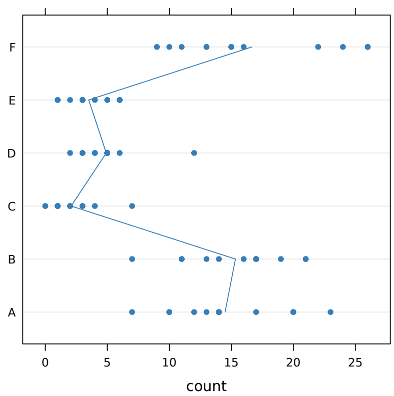
Instead of barcharts, it is usually more easy to use Cleveland's dotcharts as they allow to show individual (within level) variations.

```
dotplot(spray ~ count, data=InsectSprays, lty=2)
```



The main panel can be customized easily. For example, we can change the way horizontal lines are drawn.

```
dotplot(spray ~ count, data=InsectSprays, type=c("p","a"))
```



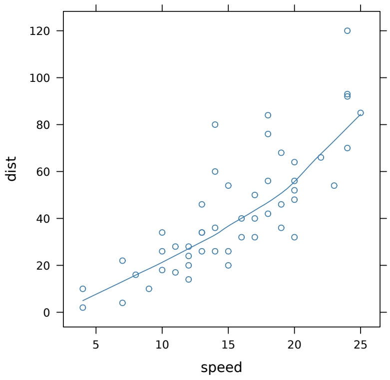
It is also possible to show aggregated data, like average count per level, using `panel.average`. The default `fun=` argument is `mean`, but we could use `median` instead.

4.5 Line fits

In this section, we discuss the addition of model fit to existing two-way graphics. For example, it may be interesting to show a regression or lowess[1] line when using xyplot.

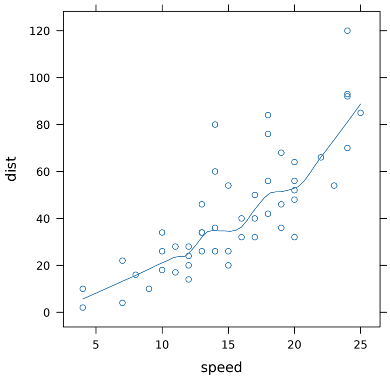
```
xyplot(dist ~ speed, data=cars, type=c("p","smooth"))
```

An adaptive loess smoother superimposed on top of a standard scatterplot.



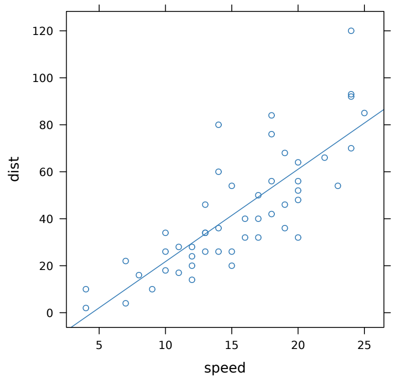

```
xyplot(dist ~ speed, data=cars, type=c("p","smooth"), span=1/3)
```

Window span can be controlled using the span= argument, where lower value means more sentivity to local variations.



[1]**cleveland79**.

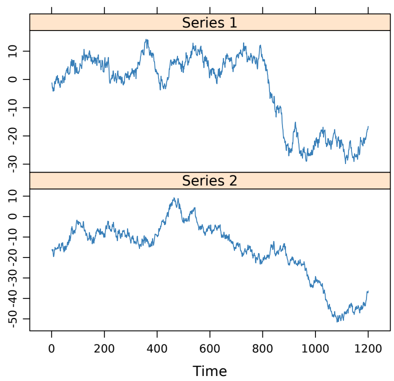
```
xyplot(dist ~ speed, data=cars, type=c("p","r"))
```



Regression fit can be shown using the same idea, through the `type=` argument.

4.6 Time series

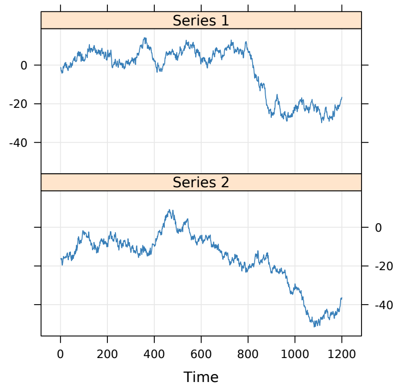
```
xt <- ts(matrix(cumsum(rnorm(200 * 12)), ncol=2))
xyplot(xt)
```



blabla blabla.

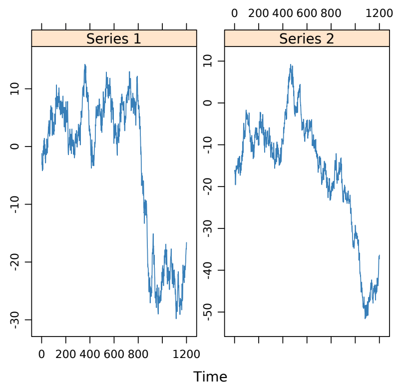
```
xyplot(xt, scales=list(y="same"), type=c("l","g"))
```

blabla blabla.



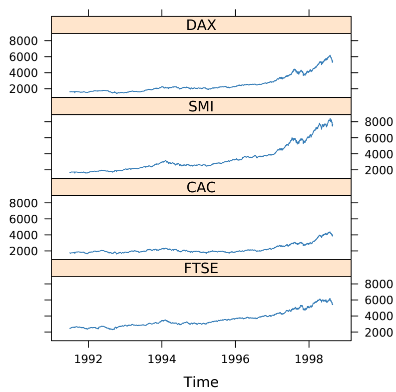

```
xyplot(xt, layout=c(2,1))
```

blabla blabla.



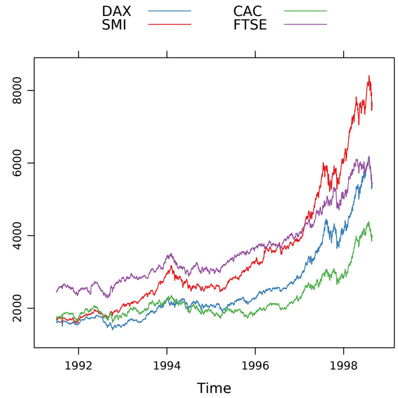

```
xyplot(EuStockMarkets, scales=list(y="same"))
```

blabla blabla.



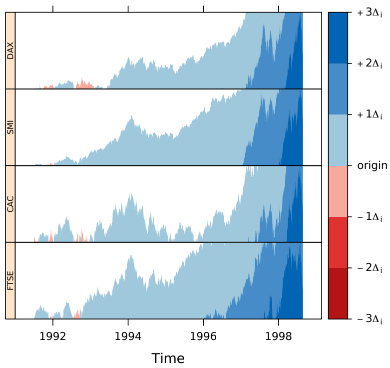
EuStockMarkets. Contains the daily closing prices of major European stock indices: Germany DAX (Ibis), Switzerland SMI, France CAC, and UK FTSE. The data are sampled in business time, i.e., weekends and holidays are omitted.

```
xyplot(EuStockMarkets, superpose=TRUE, auto.key=list(columns=2))
```



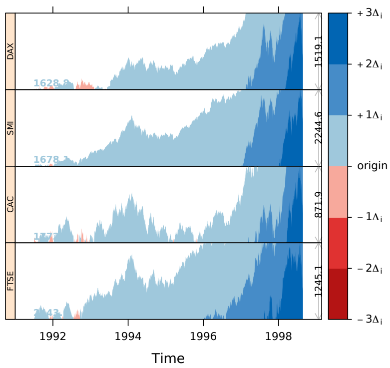
blabla blabla.

```
horizonplot(EuStockMarkets, colorkey=TRUE)
```



blabla blabla.

```
horizonplot(EuStockMarkets, colorkey=TRUE) + infolayers
```

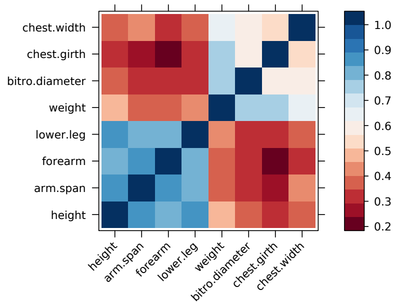


blabla blabla.

4.7 Level plot

```
data(Harman23.cor)
levelplot(Harman23.cor$cov, scales=list(x=list(rot=45)),
          xlab="", ylab="")
```

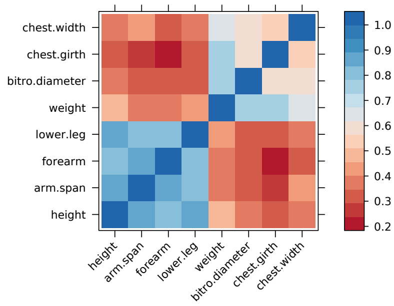
Level plots can be used to display correlation matrix in a concise way (not unlike `symnum`).



Harman23.cor. A correlation matrix of eight physical measurements on 305 girls between ages seven and seventeen. **harman76**

```
cols <- colorRampPalette(brewer.pal(8, "RdBu"))
levelplot(Harman23.cor$cov, scales=list(x=list(rot=45)),
          xlab="", ylab="", col.regions=cols)
```

The default color scheme can be changed easily. Here, we are using a Red-Blue color palette, from the `RColorBrewer` package. (Note that it introduces little changes wompared to the preceding plot because a custom theme is currently in use for the whole textbook.



CHAPTER 5

Multi-way graphics

This chapter focus on multi-variable displays, where usually two-way graphics are conditioned on values taken by one or more variables, or a combination thereof. These so-called " displays" are very good at conveying information about trend or variation between two numerical variables across the levels of a third factor.

5.1 Parallel displays

5.2 Scatterplot matrix

5.3 Three-way tabular data

5.4 N-way data

CHAPTER 6

Customizing theme and panels

CHAPTER 7

Interactive and dynamic displays

7.1 Exploratory data analysis

7.2 Brushing and linking

7.3 The ggobi toolbox