# Datasets and Familiarisation Exercises: 'Modern Regression, Classification and Multivariate Exploration with R' 

John Maindonald

August 29, 2011

The following provides guidance in gaining familiarity with selected datasets that are used in the examples in the notes. At the same time, it suggests ways to start graphical exploration of data sets. This is a good way to gain familiarity with code that can be used for producing graphs in R.

The first obvious step, in each case, is to look through the help page for the dataset. The str() function will give summary information about the dataset. After that, you might like to try the plots that are suggested.

## 1 A Brief Overview of R Graphics

## Base Graphics (mostly 2-D):

Base graphics implements a relatively "traditional" style of graphics

| Functions | plot(), points(), lines(), text(), mtext(), axis(), <br> identify() etc. form a suite (plot points, lines, text, etc.) |
| :--- | :--- |
| Plot y vs x | with(women, plot(height, weight)) \# Older syntax <br> plot(weight $\sim$ height, data=women) \# Graphics formula syntax |
| Caveat $\quad$Some base graphics functions do not take a data parameter |  |

## Other Graphics

(i) lattice (trellis) graphics, using the lattice package,
(ii) the low-level grid package on which lattice is built.
(iii) ggplot2, which implements Wilkinson's Grammar of Graphics
(iv) For 3-D graphics, note rgl, misc3d and tkrplot

### 1.1 Base graphics - plot () and allied base graphics functions

The following are alternative ways to plot y against x (obviously x and y must be the same length):

```
> plot(y ~ x) # Use a formula to specify the graph
> plot(x, y) # Horizontal ordinate, then vertical
    Try
> plot((0:20)*pi/10, sin((0:20)*pi/10))
> plot((1:30)*0.92, sin((1:30)*0.92))
```

Is it obvious that these points lie on a sine curve? (To make this obvious, place the cursor over the lower border of the graph sheet, until it becomes a double-sided arror. Drag the border in towards the top border, making the graph sheet short and wide.)

The following plots cons (consumption) against temp (temperature), for data in the dataset Icecream, from the Ecdat package.

```
> ## Code used for the plot
```

> library(Ecdat)
> data(Icecream)
> plot(cons ~ temp, data=Icecream)


Figure 1: Plot of cons (consumption) against temp (temperature). Data are from the dataset Icecream in the Ecdat package.
> \#\# Code used for the plot
> library(Ecdat)
> data(Icecream)
> plot(cons ~ temp, data=Icecream)
> NA
> \#\# The following is an alternative:
> with(Icecream, plot(temp, cons))

The points() function adds points to a plot. The lines() function adds lines to a plot ${ }^{1}$. The text () function adds text at specified locations. The mtext () function places text in one of the margins. The axis() function gives fine control over axis ticks and labels.

## Newer plot methods

Above, I described the default plot method. The plot function is a generic function that has special methods for "plotting" various different classes of object. For example, plotting an lm object (created by the use of the $\operatorname{lm}()$ modeling function) gives diagnostic and other information that can help in the interpretation of regression results.

Use of plot() with a data frame gives a scatterplot matrix, in which every column is plotted against every other column. The plot method for a data frame is the function pairs(). The request for a plot is passed to pairs(), which is the function that is finally responsible for plotting the scatterplot matrix. Figure 2 is an example.

[^0]

Figure 2: Scatterplot matrix for the four columns of the Icecream data, as obtained using the default plot() method for data frames.

```
> ## Code used for the plot
> plot(Icecream)
> # Calls pairs(Icecream)
```


## Interpereting Scatterplot Matrices:

For identifying the axes for each panel

- look across the row to the diagonal to identify the variable on the vertical axis.
- look up or down the column to the diagonal to identify the variable on the horizontal axis.

Each below diagonal panel is the mirror image of the corresponding above diagonal panel.
The function scatterplotMatrix() (alias spm() in the car package offers enhanced scatterplots. This will be introduced below.

### 1.2 Lattice graphics

```
Lattice Graphics:
    Lattice Lattice is a flavour of trellis graphics
        (the S-PLUS flavour was the original implementation)
    Grid grid is a low-level graphics system. It was used to build lattice.
        For grid, see Part II of Paul Murrell's R Graphics
    Lattice Lattice is more structured, automated and stylized.
    vs base Much is done automatically, without user intervention.
        Changes to the default style are harder than for base.
    Lattice Lattice syntax is consistent and tightly regulated
    syntax For lattice, graphics formulae are, except in a few special cases, mandatory.
```

Lattice (trellis) graphics functions allow the use of the layout on the page to reflect meaningful aspects of data structure. Different levels of a factor may appear in different panels. Or they may appear in the same panel, distinguished by color and/or symbol. If lines or smooth curves are added, there is a different line or curve for each different group.

Using lattice graphics, the equivalent of plot (cons temp, data=Icecream) is:

```
> library(lattice)
> gph <- xyplot(cons ~ temp,
    data = Icecream)
> # gph is then a trellis object
> plot(gph)
```

Figure 3 shows the result:


Figure 3: Lattice equivalent of Figure 1, obtained using the function xyplot().

```
> ## Code used for the plot
> library(lattice)
> gph <- xyplot(cons ~ temp,
data = Icecream)
> # gph is then a trellis object
> plot(gph)
> NA
```

Plotting lattice objects: Lattice functions return trellis objects. If returned to the command line, the command plot() is invoked, and the graph is plotted. Here, we first created the graphics object gph, then used plot (gph) to obtain the graph, in a separate step.

NB: An alternative to plot (gph) is print (gph); the result is the same.

The function trellis.device() can be used to open a new texttttrellis graphics device. The function trellis.par.set() can be used to control stylistic features. (color, plot characters, line type, etc.).

Trellis objects can be created even if no device is open. Such objects can be updated. Objects are plotted (by this time, a device must be open), either when output from a lattice function goes to the command line (thus implicitly invoking the print() command), or by the explicit use of print().

By successively updating a trellis graphics object, it can be built up and/or modified in steps. Additionally, it is possible to add to a 'printed" or displayed graphics page.

The lattice equivalent of pairs() is the function splom(). For example:

```
> xyplot(~ Icecream, data = Icecream)
```

Remember, however. If you are sourcing a file that is designed to plot the graph, or plotting from inside a function, you must use some equivalent of:

```
> gph <- xyplot(~ Icecream, data = Icecream)
> plot(gph)
```

Lattice plots come into their own when plots are required that reflect groups in the data, or that show multiple variables side by side. Consider the dataset Computers (Ecdat). Here is summary information about the columns:

```
> library(Ecdat)
> data(Computers)
> str(Computers)
'data.frame': 6259 obs. of 10 variables:
    $ price : num 1499 1795 1595 1849 3295 ...
    $ speed : num 25 33 25 25 33 66 25 50 50 50 \ldots..
    $ hd : num 80 85 170 170 340 340 170 85 210 210 ...
    $ ram : num 4 24816 1642 8 4 ...
    $ screen : num 14 14 15 14 14 14 14 14 14 15 ...
    $ cd : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 2 1 1 1 ...
    $ multi : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 1 1 1 ....
    $ premium: Factor w/ 2 levels "no","yes": 2 2 2 1 2 2 2 2 2 2 ...
    $ ads : num 94 94 94 94 94 94 94 94 94 94 ...
    $ trend : num 1 1 1 1 1 1 1 1 1 1 ...
```

CD drive?), multi (is a multi-media kit included?), and premium (is the manufacturer a "premium" firm, i.e., IBM or COMPAQ?)

The following (Figure 4) plots price against hd (size of hard drive), for each combination of cd and multi. Within panels, points are distinguished by whether or not the machine is from a "premium" manufacturer.


Figure 4: Plot of price against hd (size of hard drive), for each combination of cd and multi. Within panels, points are distinguished by whether or not the machine is from a "premium" manufacturer.

Note how an initial basic graph was created, which was then updated to:

- add a key: auto.key=list(columns=2)
- use different symbols for the different groups: par.settings $=$ simpleTheme (pch=c $(1,3)$
- make points somewhat transparent (alpha=0.25)
- include the names of the conditioning columns as a prefix to the strip labels: strip=strip.custom(strip.names=

```
> ## Code used for the plot
> gph <- xyplot(price ~ hd | cd * multi,
    groups=premium,
    data=Computers)
> gph1 <- update(gph, auto.key=list(columns=2),
        par.settings = simpleTheme(pch=c(1,3), alpha=0.25),
        strip=strip.custom(strip.names=c(TRUE,TRUE)))
    plot(gph1)
> NA
```


## The graphics formula

In price $\simeq \mathrm{hd} \mid \mathrm{cd} *$ multi, the columns cd and multi are conditioning columns. The $\mid$ is the conditioning symbol; what follows specifies the column(s) on which the plot is to be conditioned.

The argument groups=premium specifies that points are to be distinguished within panels, according as to whether the machine was not (No) or was (Yes) from a premium manufacturer.

## 2 Useful types of graph, for initial exploration

### 2.1 Scatterplots

Before plotting any graphs, one wants to know what data the columns hold. Commonly, columns will be one of:

- numeric, with enough distinct values that the data can be treated as continuous
- numeric, with a small number of values that code for unordered or ordered categories
- character
- factor - which is a common way to store character data. What is stored are integers $1,2, \ldots$. Associated with the factor (as an "attribute") is a table that translates 1 to the first factor level, 2 to the second level, and so on.

Before we do the analyses that will be described, it is helpful to have basic information on the columns in the data, including information on relationships between explanatory variables. The rattle GUI is very helpful in this respect. If you load a data frame into rattle, it will display basic information on each column.

Basically, we'd like to ensure, if we can, that:

- all columns have a distribution that is reasonably well spread out over the whole range of values, i.e., we want to avoid having most values squashed together at one end of the range, with a small number of very small or very large values occupying the remaining part of the range
- relationships between columns (which, except for the relationship with the outcome variable we prefer to be weak) are roughly linear.
Where values are concentrated at one end of the range, the small number (perhaps one or two) of values that lie at the other end of the range will, in a straight line regression with that column as the only explanatory variable, be a leverage point. When it is one explanatory variable among several, those values will have an overly large say in determining the coefficient for that variable.

The commonest situation is where positive (or non-zero) values are squashed together in the lower part of the range, with a tail out to the right. The distribution is then described as skewed to the right. Often, in these circumstances, a logarithmic transformation will remove much or all of the skew. Where transformations can be used to ensure that values in all columns are reasonably spread out over the whole of their range, it will then often turn out that relationships between variables are approximately linear.

The dataset mammals MASS furnishes an extreme example. Figure 5A shows the scatterplot for the raw data, while Figure 5B shows the scatterplot for the logged data.

```
> ## Code used for graph
> library(MASS)
> opar <- par(fig=c(0, 0.5, 0,1), mar=c(3.6,3.6,2.1,1.6), mgp=c(2.5,.5,0))
> plot(brain ~ body, data=mammals, pty="s")
> mtext(side=3, line=0.5, adj=O, "A: Unlogged data")
> par(fig=c(0.5, 1, 0, 1), new=TRUE)
> plot(brain ~ body, data=mammals, log="xy", pty="s")
> mtext(side=3, line=0.5, adj=O, "B: Log scales on both axes")
> NA
```



Figure 5: Brain weight (g) versus Body weight (kg), for 62 species of mammal. Panel A shows the unlogged data, while Panel B uses log scales, for both axes. Notice that the scales are labeled in the original (unlogged) units.

### 2.2 Scatterplot matrices

The hills2000 data frame (DAAG) has four columns: dist: climb (total height gained, in feet), dist (distance, in miles on the map), time (record time, in hours, for males), and timef (record time, in hours, for females). This dataset is a good candidate for a scatterplot matrix, as in Figure 6.


The car package has a more sophisticated version of scatterplot matrix (Figure 7). The function is scatterplotMatrix(), which can be abbreviated to spm(). We will turn off the option to fit a line, and instead fit a curve.


### 2.3 Density plots

The function $\operatorname{spm}()$ showed density plots in the diagonal. The density is an extimate of the relative number (proportion) of points per unit interval. We can do the density plots separately from the scatterplot. A good function for this purpose is densityplot() from the lattice package:


Code is:

Figure 8: Density plots for the four columns of the hills2000 data, as obtained using the densityplot() function in the lattice package. The argument from $=0$ specifies a sharp cutoff at zero, desirable as values must be positive. The individual data values are shown along the $x$-axis.

```
> library(lattice)
> gph <- densityplot(~ dist + climb + time + timef,
    from=0,
    data=hills2000,
    scales="free",
    outer=TRUE)
```

> NA

Figure 9 shows the density plots for the logged data:


Figure 9: Density plots of the logarithms of the four columns of the hills2000 data.

Code is:

```
> library(lattice)
> gph <- densityplot(~ dist + climb + time + timef,
                    data=hills2000,
                        scales=list(x=list(relation="free", log=10),
                                    y="free"),
                    outer=TRUE)
```

$>N A$

Two alternatives to density plots are:

- dotplots, using the lattice function dotplot(). These show the points spread out along a line;
- boxplots, using the lattice function bwplot(). A box that marks off the limits between the lower and upper quartile has a line across it that marks the median. Whiskers extend out either side of the box, commonly chosen so that for a normal distribution $1 \%$ of points would on average lie outside of this range. Points that lie out beyond the whiskers are plotted individually.

Figures 10A and 10B show, respectively, dotplot and boxplot summaries of the data:
A: Dotplots

> \#\# Code for Panel A
> \#\# Code for Panel A
> library(latticeExtra)
> gdot <-
dotplot(~ dist + climb + time + timef,
data=hills2000, outer=TRUE,
scales=list(x=list(relation="free")),
layout=c (1,4), xlab="",
main=list(expression(plain("A: Dotplots")),
x=0.05, just="left"))
> plot(gdot)
> \#\# Code for Panel B
> gbw <-
bwplot (~ dist + climb + time + timef,
data=hills2000, outer=TRUE,
scales=list( $x=1$ ist(relation="free")),
layout=c (1,4), xlab="",
main=list(expression(plain("B: Boxplots")),
$x=0.05$, just="left"))
> plot(gbw)

Boxplots are helpful for showing skewness, or the presence of outliers. Here, the data are very clearly skewed to the right.

## worldRecords: $\boldsymbol{D} \boldsymbol{A} \boldsymbol{A} \boldsymbol{G}$

Enter help(worldRecords) to view the help page for this dataset. Hereafter, it will be taken for granted that you know to look at the help page.

In the following, type the code that follows the '>' prompt.

```
> library(DAAG)
> # NB: Datasets in the DAAG package are available once the package
> # has been attached.
> # Other packages, e.g., Ecdat, may require use of data() to make
> # a dataset available.
## Show summary information about the data
str(worldRecords)
'data.frame': 40 obs. of 5 variables:
$ Distance : num 0.1 0.15 0.2 0.3 0.4 0.5 0.6 0.8 1 1.5 ...
$ roadORtrack: Factor w/ 2 levels "road","track": 2 2 2 2 2 2 2 2 2 2 ...
$ Place : chr "Athens" "Cassino" "Atlanta" "Pretoria" ...
$ Time : num 0.163 0.247 0.322 0.514 0.72 ...
$ Date : Date, format: "2005-06-14" "1983-05-22" ...
> ## Plot data
> plot(Time ~ Distance, data=worldRecords)
```


## cricketer: $\boldsymbol{D A A G}$

Code will be given without output

```
> library(DAAG) ## Not needed, if you typed library(DAAG) earlier
> ## Show summary information about the data
> str(cricketer)
```


## nihills: $\boldsymbol{D A A G}$

This dataset has record times for Northern Ireland mountain races, for males and females separately.

```
> ## Check the contents of the various columns
> str(nihills)
'data.frame': 23 obs. of 4 variables:
    $ dist : num 7.5 4.2 5.9 6.8 5 4.8 4.3 3 2.5 12 ...
    $ climb: int 1740 1110 1210 3300 1200 950 1600 1500 1500 5080 ...
    $ time : num 0.858 0.467 0.703 1.039 0.541 ...
    $ timef: num 1.064 0.623 0.887 1.214 0.637 ...
> ## Scatterplot matrix -- Plot each column against each other column
> plot(nihills)
> ## Bells and whistles scatterplot matrix
> scatterplotMatrix(nihills, smooth=TRUE, reg.line=NA,
    col=c("black","gray40"))
```


## A note on scatterplot matrices

A scatterplot matrix, which plots every column against every other column and shows the result in the layout used for correlation matrices, is useful for an initial look at the data. The scatterplot matrix is a graphical counterpart of the correlation matrix.

For identifying the axes for each panel

- look along the row to the diagonal to identify the variable on the vertical axis.

| Sugar yield data |  |  |
| :---: | ---: | :--- |
|  | weight | trt |
| 1 | 82.00 | Control |
| 2 | 97.80 | Control |
| 3 | 69.90 | Control |
| 4 | 58.30 | A |

Table 1: The table has the first few lines of the data frame sugar.

- look up or down the column to the diagonal to identify the variable on the horizontal axis.

Note that the data are positively skewed, i.e., there is a long tail to the right, for all variables. For such data, a logarithmic transformation often gives more nearly linear relationships.

## roller: $D \boldsymbol{A} \boldsymbol{A} \boldsymbol{G}$

The data has lawn depression for various weights of lawn roller. Type help(roller) to see the help page for this dataset.

Here, code is shown without output.

```
> library(DAAG)
> ## Show summary information about the data
> str(roller)
> ## Plot depression against weight
> plot(depression ~ weight, data=roller)
```


## sugar: $D A A G$ package

The sugar data frame ( $D A A G$ package) compares the amount of sugar obtained from an unmodified wild type plant with the amounts from three different types of genetically modified plants. Table 1 shows the first few lines of data.

The code used to fit the model is:

```
library(DAAG) # sugar is in DAAG package
## Examine data
sugar
    weight trt
        82.0 Control
97.8 Control
3 69.9 Control
4 58.3 A
5 67.9 A
6 59.3 A
7 68.1 B
8 70.8 B
9 63.6 B
10 50.7 C
11 47.1 C
12 48.9 C
> ## Summary information about data
> str(sugar)
'data.frame': 12 obs. of 2 variables:
    $ weight: num 82 97.8 69.9 58.3 67.9 59.3 68.1 70.8 63.6 50.7 ...
    $ trt : Factor w/ 4 levels "Control","A",..: 1 1 1 2 2 2 3 3 3 4...
```


## cuckoos: $D A A G$ package

Type help(cuckoos) to see the help page for this dataset. A good plot for these data is:

```
> ## Get details of data
> str(cuckoos)
'data.frame': 120 obs. of 4 variables:
    $ length : num 21.7 22.6 20.9 21.6 22.2 22.5 22.2 24.3 22.3 22.6 ...
    $ breadth: num 16.1 17 16.2 16.2 16.9 16.9 17.3 16.8 16.8 17 \ldots..
    $ species: Factor w/ 6 levels "hedge.sparrow",..: 2 2 2 2 2 2 2 2 2 2 ...
    $ id : num 21 22 23 24 25 26 27 28 29 30 ...
> ## Plot data
> dotplot(species ~ length+breadth, data=cuckoos, outer=TRUE,
    scale=list(x=list(relation="free")))
```

The length+breadth part of the formula results in separate plots (the argument outer=TRUE ensures plots in separate panels) for each of length and breadth.

A note on factors: The names for the different values that a factor can take are the "levels".

```
> levels(cuckoos$species) # column 'species' from the data frame 'cuckoos'
[1] "hedge.sparrow" "meadow.pipit" "pied.wagtail" "robin"
[5] "tree.pipit" "wren"
```

Internally, factors are stored as integer values. The column species of the data frame cuckoos is a factor that has 6 levels. A lookup table is used to associate levels with these integer values.

## Electricity: Ecdat package

Here, and subsequently for the most part, code will be shown without output.
In the Ecdat package, datasets do not automatically become available when you use library (Ecdat) to attach the package. Hence the use of data(Electricity) in the code that follows:

```
> library(Ecdat)
> data(Electricity) # For datsets in the 'Ecdat' package, use
> # data() as required to make datasets available.
> ## Get details of columns in the data frame
> str(Electricity)
> ## Examine scatterplot matrix
> plot(Electricity)
```

An alternative that gives more information is:

```
> library(car)
> scatterplotMatrix(Electricity, smooth=TRUE, reg.line=NA,
    col=c("black", "gray40"))
```

Be sure to look at the help page for Electricity (help(Electricity)) to get details of the variables.

## Crime: Ecdat package

> library (Ecdat)
> data(Crime)
> str(Crime)
You can try
> plot (Crime)
Because however there are so many columns, this may not be satisfactory. Density plots for the columns that have continuous variables are however perfectly feasible:

```
> library(lattice)
> contnums <- (1:ncol(crime))[-c(1:2,11:12)]
> formCont <- formula(paste("~", paste(names(Crime)[contnums], collapse="+")))
> densityplot(formCont, data=Crime, outer=TRUE,
    scales=list(x=list(relation="free"), y=list(relation="free")))
```


## Wages: Ecdat package

Here, code is shown without output.

```
> library(Ecdat)
> data(Wages)
> str(Wages)
> library(lattice)
> splom(Wages[,c(1,2,10,12)], alpha=0.4)
```

Use splom() (lattice) rather than plot () because this makes it easier to adjust the transparency; the argument alpha does this. Set alpha to be any value between 0 (full transparancy) and 1 (totally opaque).

## bronchit: SMIR package

Again, code is shown without output.

```
> library(SMIR); data(bronchit)
> data(bronchit)
> str(bronchit)
> library(lattice)
> xyplot(poll ~ cig, groups=r, auto.key=list(columns=2),
    xlab="# cigarettes per day", ylab="Pollution",
    data=bronchit)
```


## nassCDS: $D A A G$ package

Code is shown without output.

```
> library(DAAG)
> str(nassCDS)
```


## Fair: : Ecdat package

```
> library(Ecdat)
> data(Fair)
> str(Fair)
```


## fgl: MASS

```
> library(MASS)
> # NB: Datasets in the MASS package are available once the package
> # has been attached.
## Show summary information about the data
str(fgl)
## Show scatterplot matrix
plot(fgl)
# See the note below on scatterplot matrices
```

Here is a more informative type of scatterplot matrix:

```
library(car)
scatterplotMatrix(fgl, smooth=TRUE, reg.line=NA,
                    col=c("black","gray40"))
## For versions of the car package prior to 2.0-0, specify
## scatterplot.matrix(fgl, smooth=TRUE, reg.line=NA,
## col=c("black","gray40"))
## The first colour is used for lines, and the second for points.
```

Note that scatterplotMatrix can be abbreviated to spm().
Try also a plot that uses separate colours and characters for different groups in the data. The default colour palette is not very satisfactory. Hence the alternative used here.

```
> library(lattice) # Makes available the seven lattice colours
> scatterplotMatrix(~ . | type, smooth=TRUE, reg.line=NA, data=fgl,
    col=trellis.par.get()$superpose.symbol$col)
```

The graphics formula ~ . I type causes all of the columns except type to be used for the rows and columns of the scatterplot matrix. Different colours and symbols are used for the different types.

The first colour is used for the lines. The second and subsequent colours are used for the points, i.e., for the six different types. With so many columns of data, this is not a very satisfactory plot.

## We can readily show all the distributions on one page

For this we use the lattice function densityplot():

```
> library(lattice)
> densityplot(~ RI+Na+Mg+Al+Si+K+Ca+Ba+Fe, groups=type, data=fgl, outer=TRUE,
    scales=list(x=list(relation="free"), y=list(relation="free")),
    auto.key=list(columns=3))
```


## diabetes: : mclust package

Code is shown without output.

```
> library(mclust)
> data(diabetes)
> str(diabetes)
> scatterplotMatrix(~ glucose +insulin+sspg | class, smooth=TRUE,
    reg.line=NA, data=diabetes,
    col=brewer.pal(n=4, name="Set1"))
```


## spam7: : $D \boldsymbol{A} \boldsymbol{A} \boldsymbol{G}$ package

```
> library(DAAG)
> str(spam7)
> bwplot(yesno ~ crl.tot + dollar + bang + money + n000 + make,
    outer=TRUE, data=spam7, scales=list(x=list(relation="free")))
> densityplot(~ crl.tot + dollar + bang + money + n000 + make,
    groups=yesno, outer=TRUE, data=spam7,
    scales=list(x=list(relation="free"), y=list(relation="free")))
> ## Try also (this is not a very satisfactory plot)
> spm(~ crl.tot + dollar + bang + money + n000 + make | yesno, data=spam7)
```

Because the data are so highly skew, boxplots are a much more satisfactory form of display than density plots. For the same reason, the scatterplot matrix is an unsatisfactory form of display.

## germandata: : nws package

Code is shown without output.

```
> library(nws)
> data(germandata)
> str(germandata)
> sapply(germandata, range) # Check range of values in each column
> scatterplotMatrix(~ X6 + X12 + jitter(X5) + jitter(X5.1) + X67 | X1.2,
    smooth=TRUE, reg.line=NA, data=germandata,
    col=brewer.pal(n=4, name="Set1"))
```

Further data sets may be added to the list later.


[^0]:    ${ }^{1}$ Actually these functions differ only in the default setting for the parameter type. The default setting for points() is type = "p", and for lines() is type = "l". Explicitly setting type = "p" causes either function to plot points, type = "l" gives lines.

