Data analysis with R

Lecture 6

Deeper in graphics

Jouni Junnila

par()

- *par*-function is the most important issue, when we want to modify our graphical representations.
- There are two different kind of options in *par*.
  1) A set of options which can be only called through the *par*-function
  2) Another set of options which can be called also through different graphical functions.
- Next we're going to go through several of these options, needed in modifying figures.

- *xlab*, *ylab:* Labels for the x-axis and the y-axis
- *main:* The main title.
- *sub:* The subtitle.
- *ylim*, *xlim:* Limiting coordinates for the two axes.
- *font:* Used, when bold or italic fonts are needed
  - 1 corresponds to plain text (the default),
  - 2 to bold,
  - 3 to italic and
  - 4 to bold italic

- *bg:* defines the color to be used for the background of the device region. (eg. Bg="lightblue")
- *cex:* gives a numerical value giving the amount by which plotting text and symbols should be magnified relative to the default (=1).
- *cex.axis:* The magnification to be used for axis annotation relative to the current setting of *cex*.
- *cex.lab:* The magnification to be used for x and y labels relative to the current setting of *cex*.
- *cex.main:* The magnification to be used for main titles relative to the current setting of *cex*.
- *cex.sub:* The magnification to be used for sub-titles relative to the current setting of *cex*.

- *col*: A specification for the default plotting color.*col.main*, *col.axis*, *col.lab* and *col.sub* change the color for corresponding texts.
- *family*: The name of a font family for drawing text. (eg. “serif”)
- *lty:* The line type. Line types can either be specified as an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as one of the corresponding characters.
- *lwd:* The line width, a *positive* number, defaulting to 1.
- *pch:* Either an integer specifying a symbol or a single character to be used as the default in plotting points.
- *mfcol, mfrow:* A vector of the form c(nr, nc). Subsequent figures will be drawn in an nr-by-nc array on the device by *columns* (mfcol), or *rows* (mfrow), respectively.

- *las:* numeric in {0,1,2,3}; the style of axis labels.
  - 0:always parallel to the axis [*default*],
  - 1:always horizontal,
  - 2:always perpendicular to the axis,
  - 3:always vertical.
- *mar:* A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot.
- *oma:* A vector of the form c(bottom, left, top, right) giving the size of the outer margins in lines of text.
- *fig:* A numerical vector of the form c(x1, x2, y1, y2) which gives the coordinates of the figure region in the display region of the device.

- *xlog, ylog*: A logical value. If TRUE, a logarithmic scale is in use.
- *xaxp:* A vector of the form c(x1, x2, n) giving the coordinates of the extreme tick marks and the number of intervals between tick-marks when *par("xlog")* is false.
- *xaxs:* The style of axis interval calculation to be used for the x-axis. Possible values are "r", "i", "e", "s", "d". Style "r" (regular) first extends the data range by 4 percent at each end and then finds an axis with pretty labels that fits within the extended range. Style "i" (internal) just finds an axis with pretty labels that fits within the original data range.

Type of plot

- There are several different types of plots we can draw. This is handled by an option *type*.
- Possible types are:
  - "p" for points,
  - "l" for lines,
  - "b" for both,
  - "h" for ‘histogram’ like (or ‘high-density’) vertical lines,
  - "s" for stair steps,
  - "n" for no plotting.
  - "r" for regression line (with *xyplot*)

Adding points

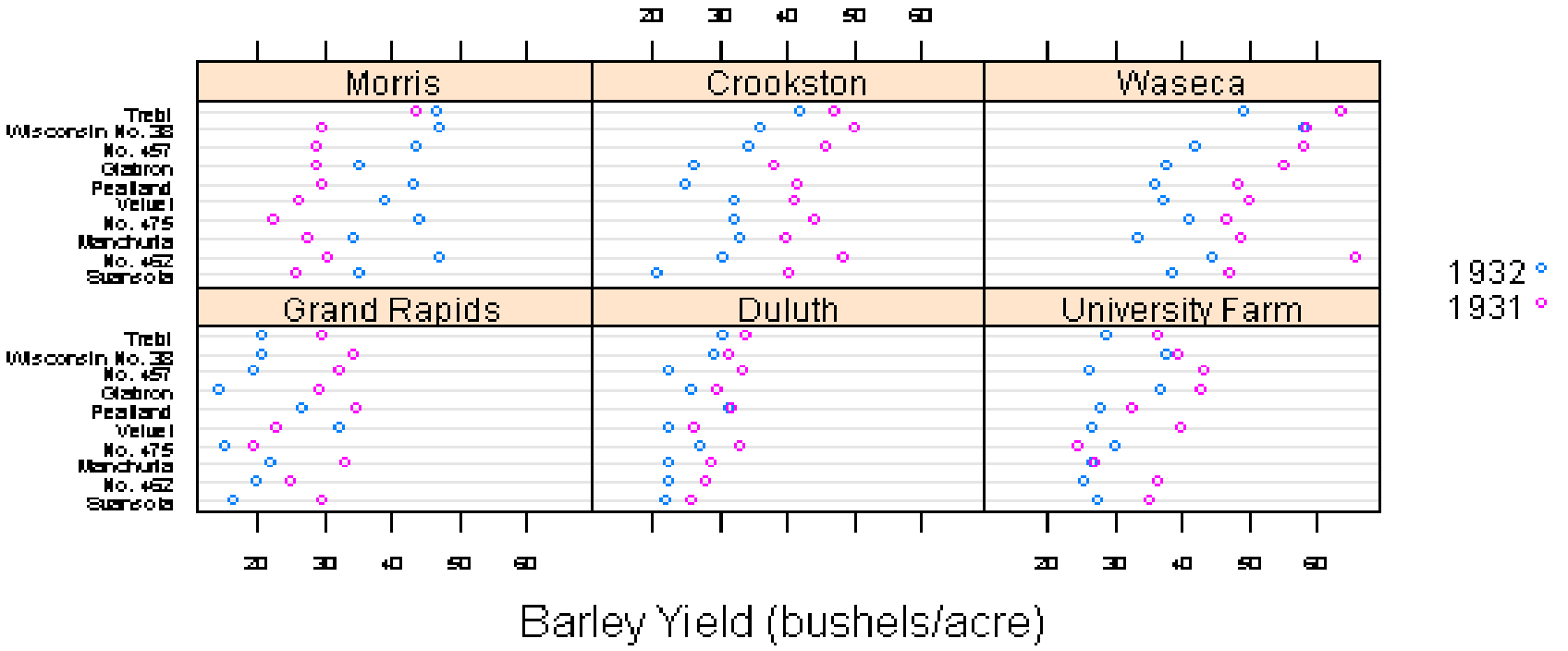
- With the function *points()* we can add points to the current graph.
- Actually *points()* has a type-option, so we can add for example lines to the plot as well, using type=”l”.
  - There is a special function for adding lines as well *lines()*
- This function is specially convenient when we want to overlay to different plots into the same graph

Example of *dotplot()*

- Let's consider an example of trellis.style dotplot.
- First we'll do some changes to the options and then draw the plot.

```
› trellis.par.set(list(fontsize=list(text=6),
  par.xlab.text=list(cex=2),
  add.text=list(cex=1.5),
  superpose.symbol=list(cex=0.5)))
› key <- simpleKey (levels(barley$year),space =
  "right")
› key$text$cex <- 1.5
›  dotplot(variety ~ yield | site, data = barley,
  groups = year,key = key,xlab = "Barley Yield
  (bushels/acre)",aspect=0.5,layout = c(3,2),
  ylab=NULL)
```

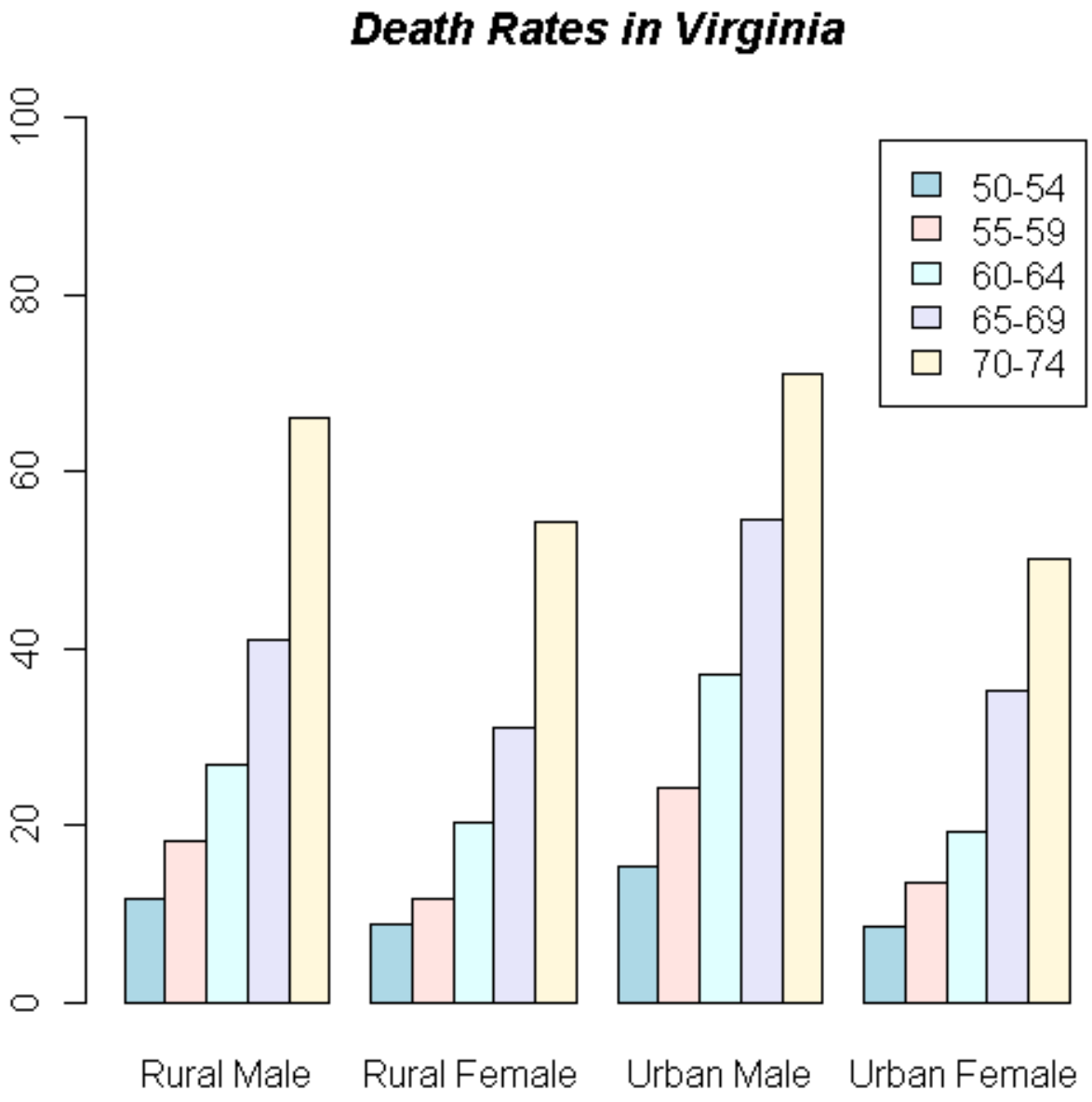
Beyond Your Data



barplots

- *barplot()* creates a bar plot with vertical or horizontal bars.
- We can also do stacked plots with this function.
- Example:
  - `barplot(VADeaths, beside = TRUE, col = c("lightblue", "mistyrose", "lightcyan", "lavender", "cornsilk"), legend = rownames(VADeaths), ylim = c(0, 100))`
  - `title(main = "Death Rates in Virginia", font.main = 4)`

Beyond Your Data



Maps

- R has a package for using geographical maps in figures. Let’s have an example about maps.
- *library(maps)*
- *par(mar=rep(0, 4))*
- *map("nz", fill=TRUE, col="grey80")*
- *points(174.75, -36.87, pch=16, cex=2)*
- *arrows(172, -36.87, 174, -36.87, lwd=3)*
- *text(172, -36.87, "Auckland ", adj=1, cex=2)*

Resulting map

Dendrograms

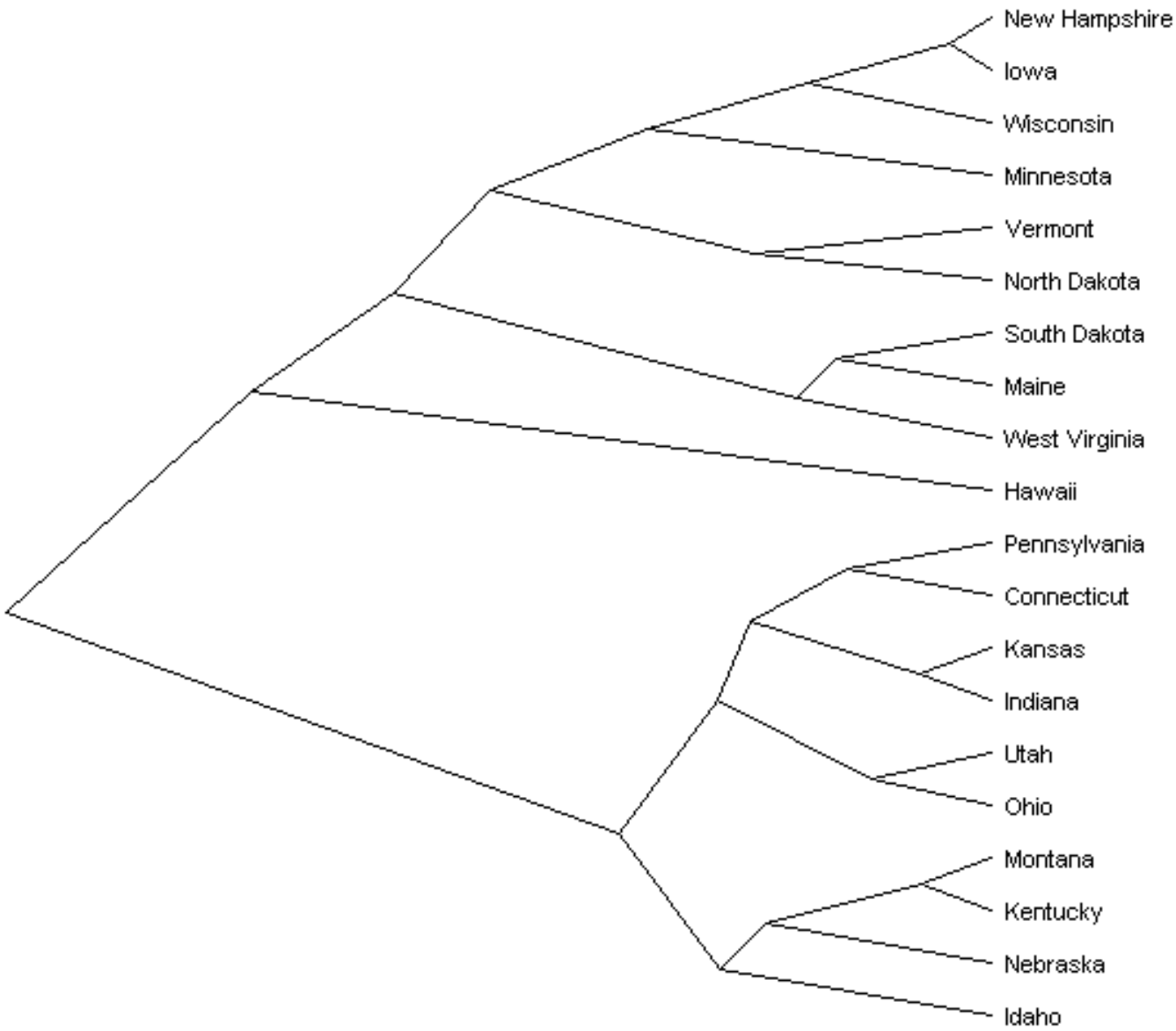
- Dendrograms are quite widely used in hierarchical cluster-analysis, where we want find different small clusters, and how are they forming bigger clusters etc. Example follows

```
› hc <- hclust(dist(USArrests), "ave")
› dend1 <- as.dendrogram(hc)
› dend2 <- cut(dend1, h=70)
```

Dendrogram; code

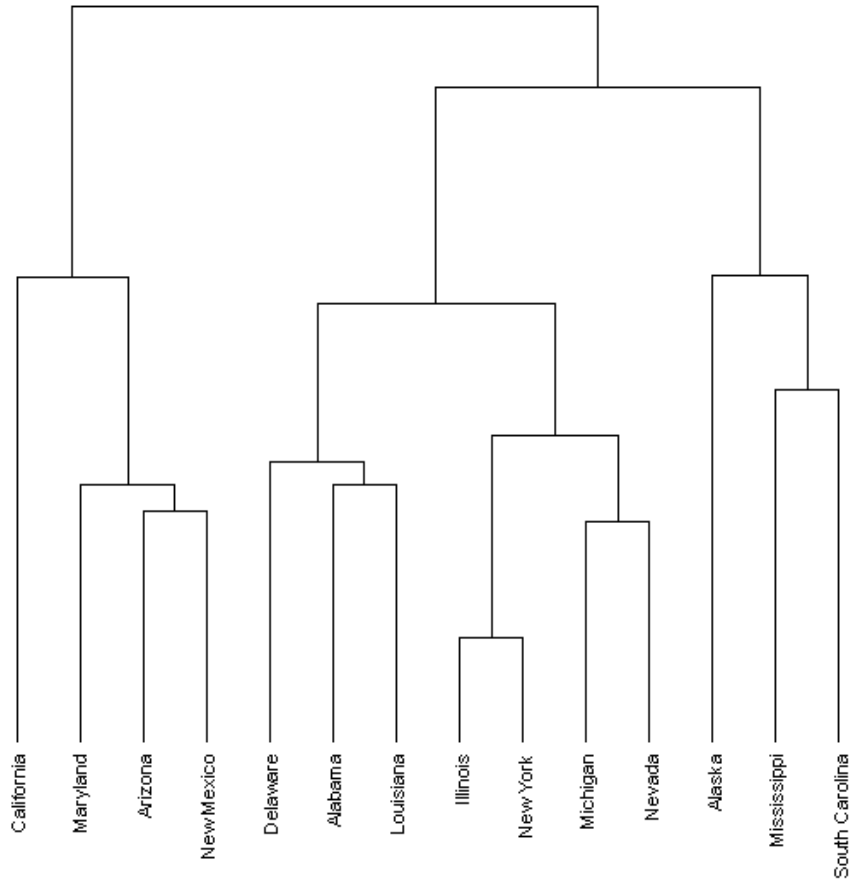
```
› par(cex=0.7)
› par(mar=c(1, 0, 2, 5.5))
› plot(dend2$lower[[3]],  horiz =
  TRUE, type = "tr", axes=FALSE,
  cex=0.8)
```

- Type=”tr” means tree-type plot


New Hampshire
Iowa
Wisconsin
Minnesota
Vermont
North Dakota
South Dakota
Maine
West Virginia
Hawaii
Pennsylvania
Connecticut
Kansas
Indiana
Utah
Ohio
Montana
Kentucky
Nebraska
Idaho

Dendrograms

- par(mar=c(6, 0, 2, 0))
- plot(dend2$lower[[2]], axes=FALSE, cex=0.8)

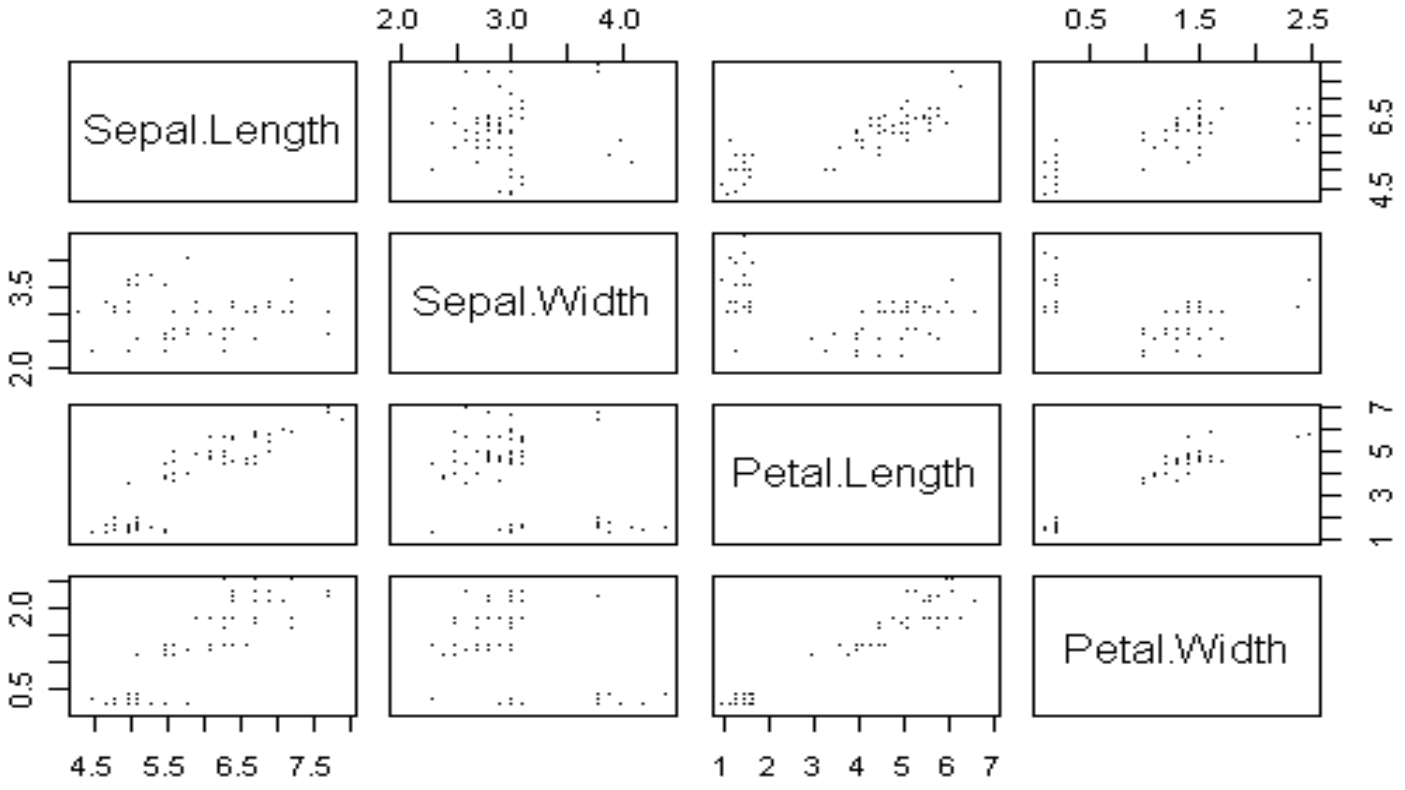


Plotting multivariate data

- If we want to look at relationships of more than two variables, we can eg. create matrix-plots.
  - Example of looking at four variables at the same time from the iris-data.

```
› par(cex=0.6) pairs(iris[1:4],
oma=c(18, 4, 4, 4),
panel=function(x, y, ...) {
points(x, y, lwd=0.1, pch = ".") })
par(cex=1)
```

Matrix plot; result



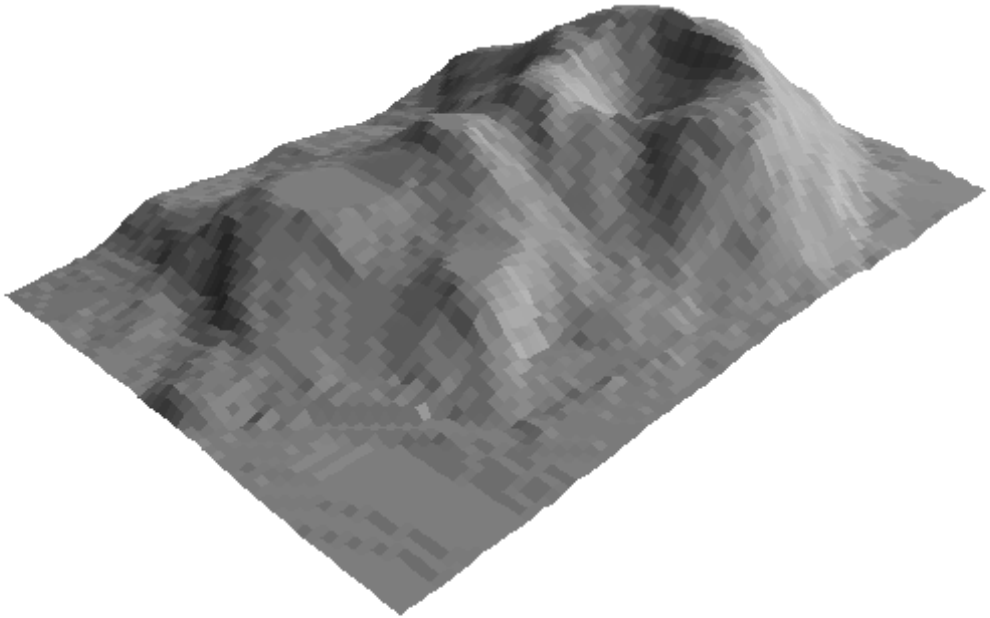
Plotting three variables

- When we look at three variable, there are special functions we can use for plotting the data. Let's have examples of the use of functions called
  - Persp()
  - Contour()
  - Image()

persp()

- This function draws perspective plots of surfaces over the x–y plane.
  - `z <- 2 * volcano # Exaggerate the relief`
  - `x <- 10 * (1:nrow(z)) # 10 meter spacing (S to N)`
  - `y <- 10 * (1:ncol(z)) # 10 meter spacing (E to W)`
  - `# Don't draw the grid lines : border = NA`
  - `par(mar=rep(0, 4))`
  - `persp(x, y, z, theta = 135, phi = 30, col = "light grey", scale = FALSE, ltheta = -120, shade = 0.75, border = NA, box = FALSE)`

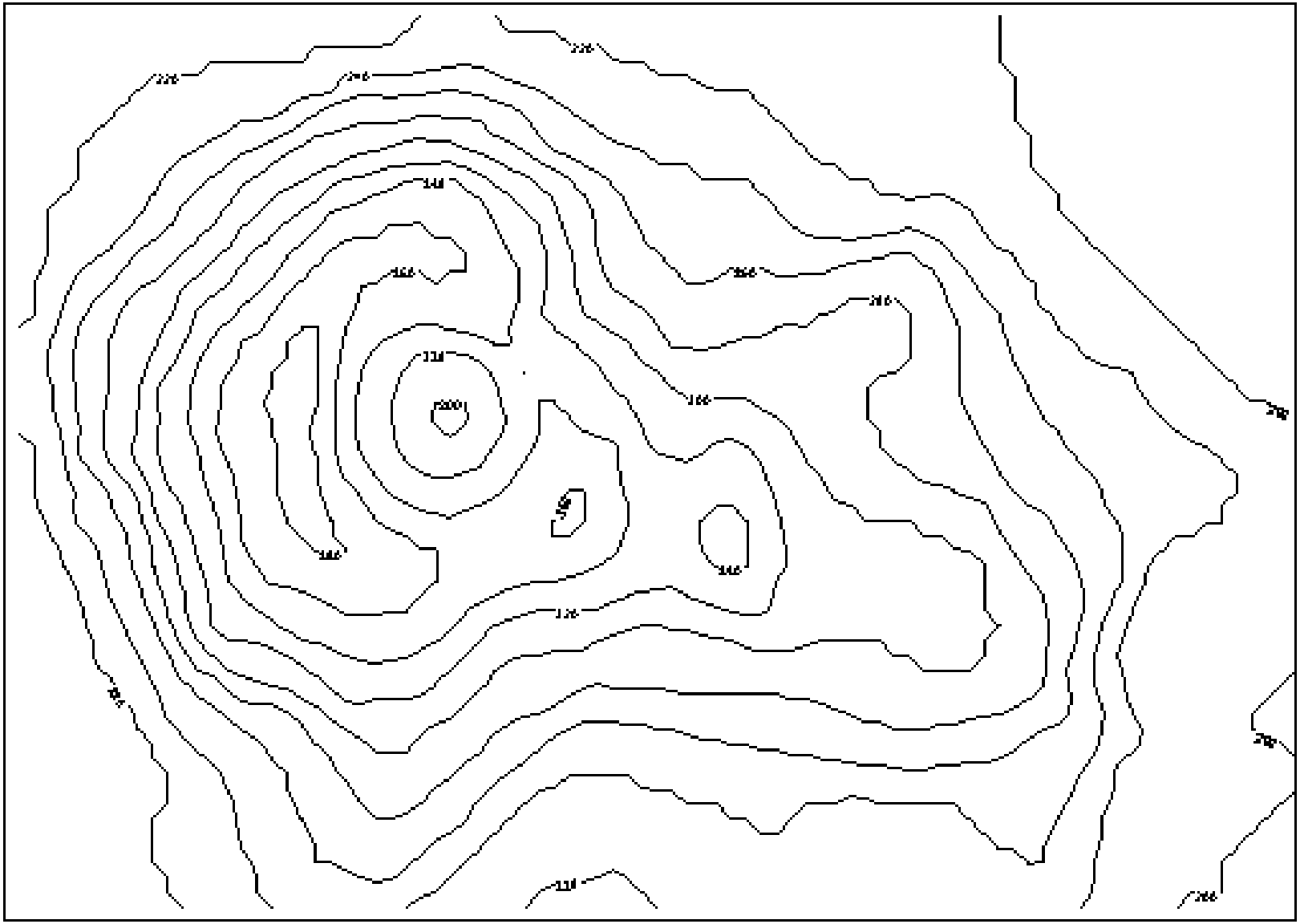
Persp() result



contour()

- Contour creates a contour plot, or add contour lines to an existing plot.
- Example with same data as previous
  - `par(mar=rep(0.5, 4))`
  - `contour(x, y, z, asp=1, labcex=0.35, axes=FALSE)`
  - `rect(0, 0, 870, 620)`

Beyond Your Data



Image()

- Image() creates a grid of colored or gray-scale rectangles with colors corresponding to the values in z.
  - `image(x, y, z, asp=1, col=grey(0.5 + 1:12/24), xlab="", ylab="", axes=FALSE)`
  - `rect(min(x)-5, min(y)-5, max(x)+5, max(y)+5)`

Beyond Your Data