POLS/CSSS 503: Advanced Quantitative Political Methodology

Regression & Graphics in R

Christopher Adolph

Department of Political Science
and
Center for Statistics and the Social Sciences
University of Washington, Seattle

Matrix Algebra in R

`det(a)`	Computes the determinant of matrix `a`
`solve(a)`	Computes the inverse of matrix `a`
`t(a)`	Takes the transpose of `a`
`a%*%b`	Matrix multiplication of `a` by `b`
`a*b`	Element by element multiplication

An R list is a basket containing many other variables

```
> x <- list(a=1, b=c(2,15), giraffe="hello")

> x$a
[1] 1

> x$b
[1]  2 15

> x$b[2]
[1] 15

> x$giraffe
[1] "hello"

> x[3]
$giraffe
[1] "hello"

> x[["giraffe"]]
[1] "hello"
```

R lists

Things to remember about lists

- Lists can contain any number of variables of any type
- Lists can contain other lists
- Contents of a list can be accessed by name or by position
- Allow us to move lots of variables in and out of functions
- Functions often return lists (only way to have multiple outputs)

`lm()` basics

```
# To run a regression
res <- lm(y~x1+x2+x3,
          data,                     # A dataframe containing
                                    # y, x1, x2, etc.
          na.action="")

# To print a summary
summary(res)

# To get the coefficients
res$coefficients

# or
coef(res)

#To get residuals
res$residuals

#or

resid(res)
```

`lm()` basics

```
# To get the variance-covariance matrix of the regressors
vcov(res)

# To get the standard errors
sqrt(diag(vcov(res)))

# To get the fitted values
predict(res)

# To get expected values for a new observation or dataset
predict(res,
        newdata,                    # a dataframe with same x vars
                                    # as data, but new values
        interval = "confidence",    # alternative: "prediction"
        level = 0.95
        )
```

R lists & Object Oriented Programming

A list object in R can be given a special "class" using the `class()` function

This is just a metatag telling other R functions that this list object conforms to a certain format

So when we run a linear regression like this:

```
res <- lm(y~x1+x2+x3, data, na.action="")
```

The result `res` is a list object of class `‘‘lm’’`

Other functions like `plot()` and `predict()` will react to `res` in a special way because of this class designation

Specifically, they will run functions called `plot.lm()` and `predict.lm()`

Object-oriented programming:
a function does different things depending on class of input object

An example: Party systems & Redistribution

Cross sectional data on industrial democracies:

povred	Percent of citizens lifted out of poverty by taxes and transfers
lnenp	Natural log of effective number of parties
maj	Majoritarian election system dummy
pr	Proportional representation dummy
unam	Unanimity government dummy (Switz)

Source of data & plot: Torben Iversen and David Soskice, 2002, "Why do some democracies redistribute more than others?" Harvard University.

An example: Party systems & Redistribution

```
# Clear memory of all objects
rm(list=ls())

# Load data
file <- "iver.csv";
data <- read.csv(file,header=TRUE);
attach(data)

lm.result <- lm(povred~lnenp)
print(summary(lm.result))
```

An example: Party systems & Redistribution

```
Call:
lm(formula = povred ~ lnenp)

Residuals:
    Min      1Q  Median      3Q     Max
-48.907  -4.115   8.377  11.873  18.101

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    21.80      16.15   1.349   0.2021
lnenp          24.17      12.75   1.896   0.0823 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 19.34 on 12 degrees of freedom
Multiple R-Squared: 0.2305,     Adjusted R-squared: 0.1664
F-statistic: 3.595 on 1 and 12 DF,  p-value: 0.08229
```

An example: Party systems & Redistribution

```
# A new model with multiple regressors
lm.result2 <- lm(povred~lnenp+maj+pr)
print(summary(lm.result2))
```

An example: Party systems & Redistribution

```
Call:
lm(formula = povred ~ lnenp + maj + pr)

Residuals:
     Min       1Q   Median       3Q      Max
-23.3843  -1.4903   0.6783   6.2687  13.9376

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -31.29      26.55  -1.179  0.26588
lnenp          26.69      14.15   1.886  0.08867 .
maj            48.95      17.86   2.740  0.02082 *
pr             58.17      13.52   4.302  0.00156 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.37 on 10 degrees of freedom
Multiple R-Squared: 0.7378,     Adjusted R-squared: 0.6592
F-statistic: 9.381 on 3 and 10 DF,  p-value: 0.002964
```

An example: Party systems & Redistribution

```
# A new model with multiple regressors and no constant
lm.result3 <- lm(povred~lnenp+maj+pr+unam-1)
print(summary(lm.result3))
```

An example: Party systems & Redistribution

```
Call:
lm(formula = povred ~ lnenp + maj + pr + unam - 1)

Residuals:
     Min       1Q   Median       3Q      Max
-23.3843  -1.4903   0.6783   6.2687  13.9376

Coefficients:
      Estimate Std. Error t value Pr(>|t|)
lnenp    26.69       14.15   1.886   0.0887 .
maj      17.66       12.69   1.392   0.1941
pr       26.88       21.18   1.269   0.2331
unam    -31.29       26.55  -1.179   0.2659
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.37 on 10 degrees of freedom
Multiple R-Squared: 0.9636,     Adjusted R-squared: 0.949
F-statistic: 66.13 on 4 and 10 DF,  p-value: 3.731e-07
```

An example: Party systems & Redistribution

```
# A model with an interaction term added
lm.result4 <- lm(povred~lnenp+maj+pr+lnenp:maj)
print(summary(lm.result4))
```

An example: Party systems & Redistribution

```
Call:
lm(formula = povred ~ lnenp + maj + pr + lnenp:maj)

Residuals:
      Min        1Q    Median        3Q       Max
-22.25124   0.06679   2.85314   4.73179  12.99480

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -14.83      31.42  -0.472  0.64813
lnenp          16.78      17.39   0.965  0.35994
maj            16.34      37.65   0.434  0.67445
pr             56.18      13.70   4.102  0.00267 **
lnenp:maj      29.55      30.02   0.984  0.35065
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.39 on 9 degrees of freedom
Multiple R-Squared: 0.7633,     Adjusted R-squared: 0.6581
F-statistic: 7.256 on 4 and 9 DF,  p-value: 0.006772
```

An example: Party systems & Redistribution

```
# A quicker way to add interactions
lm.result5 <- lm(povred~pr+lnenp*maj)
print(summary(lm.result5))
```

An example: Party systems & Redistribution

```
Call:
lm(formula = povred ~ pr + lnenp * maj)

Residuals:
     Min        1Q    Median        3Q       Max
-22.25124   0.06679   2.85314   4.73179  12.99480

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -14.83      31.42  -0.472  0.64813
pr             56.18      13.70   4.102  0.00267 **
lnenp          16.78      17.39   0.965  0.35994
maj            16.34      37.65   0.434  0.67445
lnenp:maj      29.55      30.02   0.984  0.35065
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.39 on 9 degrees of freedom
Multiple R-Squared: 0.7633,     Adjusted R-squared: 0.6581
F-statistic: 7.256 on 4 and 9 DF,  p-value: 0.006772
```

R Graphics

R has several graphics systems.

The `base` system

The `grid` system

(`grid` is more powerful, but has a steeper learning curve.
See Paul Murrel's book on R Graphics for an introduction.)

Focus here on `base`

R Graphics: Devices

Everything you draw in `R` must be drawn on a canvas

Must create the canvas before you draw anything

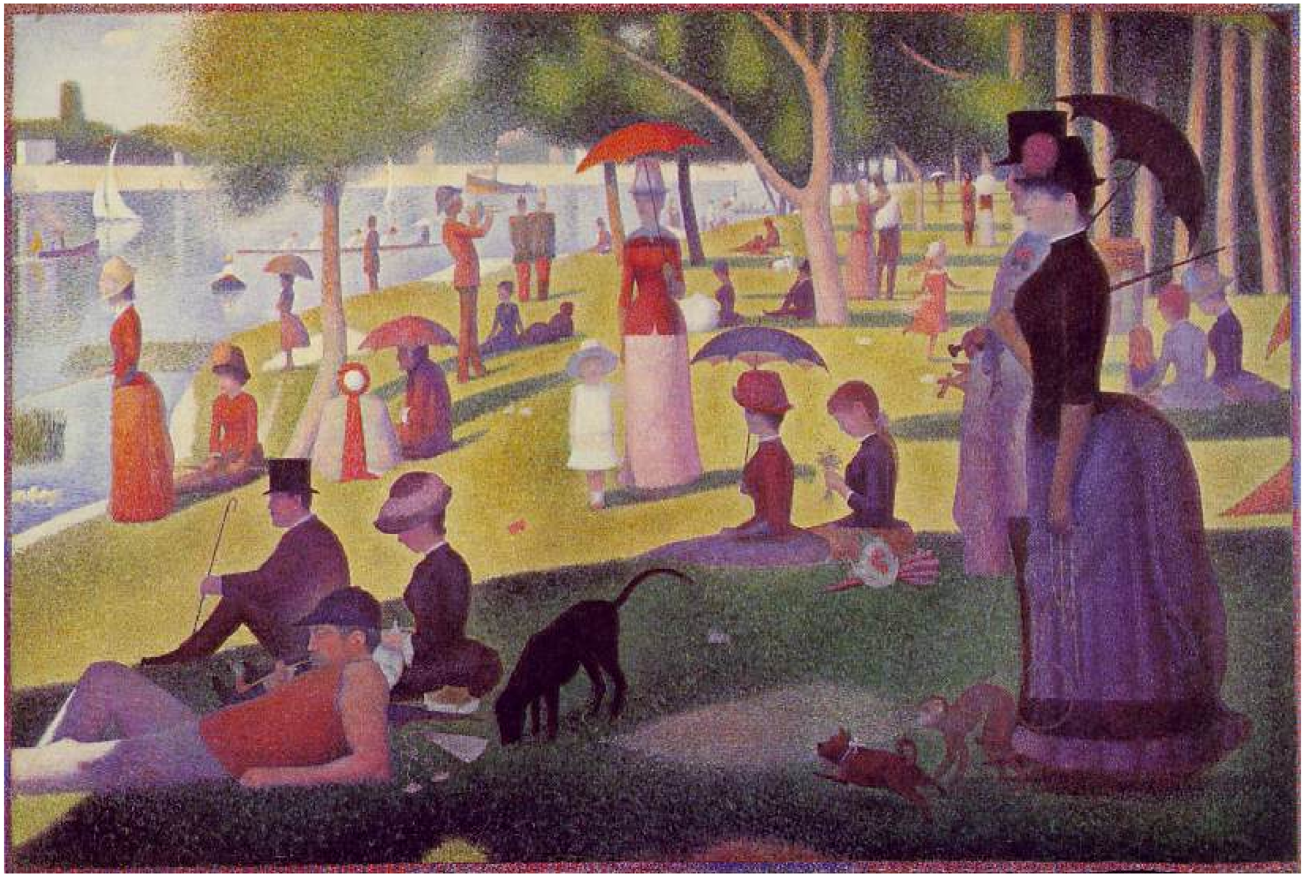
Computer canvasses are **devices** you draw to

Devices save graphical input in different ways

Sometimes to the disk, sometimes to the screen

Most important distinction: raster vs. vector devices

Vector vs. raster



Pointalism = raster graphics. Plot each pixel on an n by m grid.

Vector vs. raster

Pixel = Point = Raster

Good for pictures. Bad for drawings/graphics/cartoons.

(Puzzle: isn't everything raster? In display, yes. Not in storage)

Advantages of vector:

- Easily manipulable/modifiable groupings of objects
- Easy to scale objects larger or smaller/ Arbitrary precision
- Much smaller file sizes
- Can always convert to raster (but not the other way round, at least not well)

Disadvantages:

- A photograph would be really hard to show (and huge file size)
- Not web accessible. Convert to PNG or PDF.

Some common graphics file formats

	Lossy	Lossless
Raster	.gif, .jpeg	.wmf, .png, .bmp
Vector	—	.ps, .eps, .pdf, .ai, .wmf

Lossy means during file compression, some data is (intentionally) lost

Avoid lossy formats whenever possible

Some common graphics file formats

In `R`, have access to several formats:

win.metafile()	wmf, Windows media file
pdf()	pdf, Adobe portable data file
postscript()	postscript file (printer language)
windows()	opens a screen; PC only
quartz()	opens a screen; Mac only
x11()	opens a screen; works on all systems

Latex, Mac, and Unix users can't use wmf

`windows(record=TRUE)` let's you cycle thru old graphs with arrow keys

High-level graphics commands

In `R`, High level graphics commands:

- produce a standard graphic type
- fill in lots of details (axes, titles, annotation)
- have many configurable parameters
- have varied flexibility

You don't need to use HLCs to make `R` graphics.

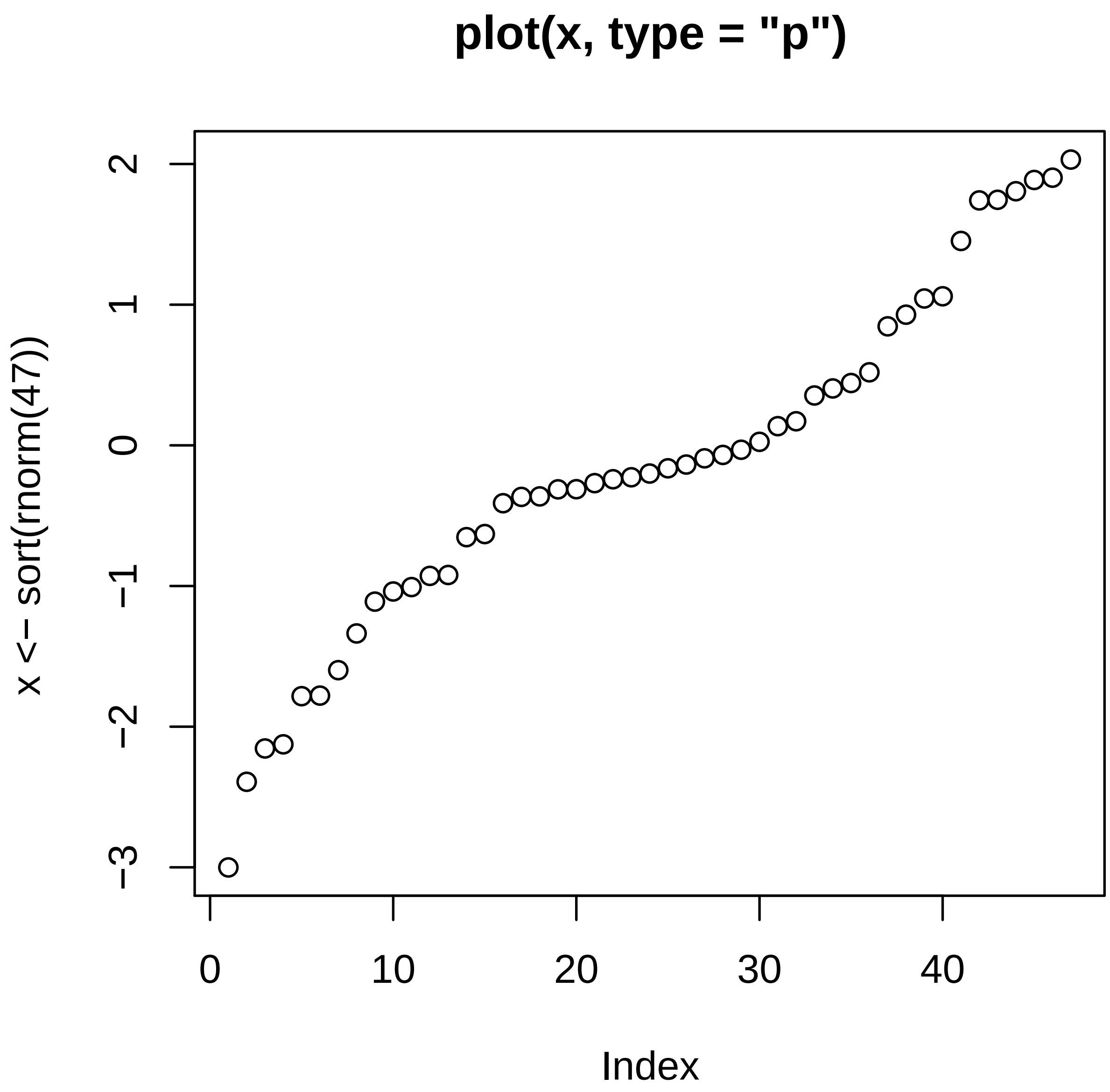
Could use primitive commands to do each task above

Using low levels commands gives more control but takes more time

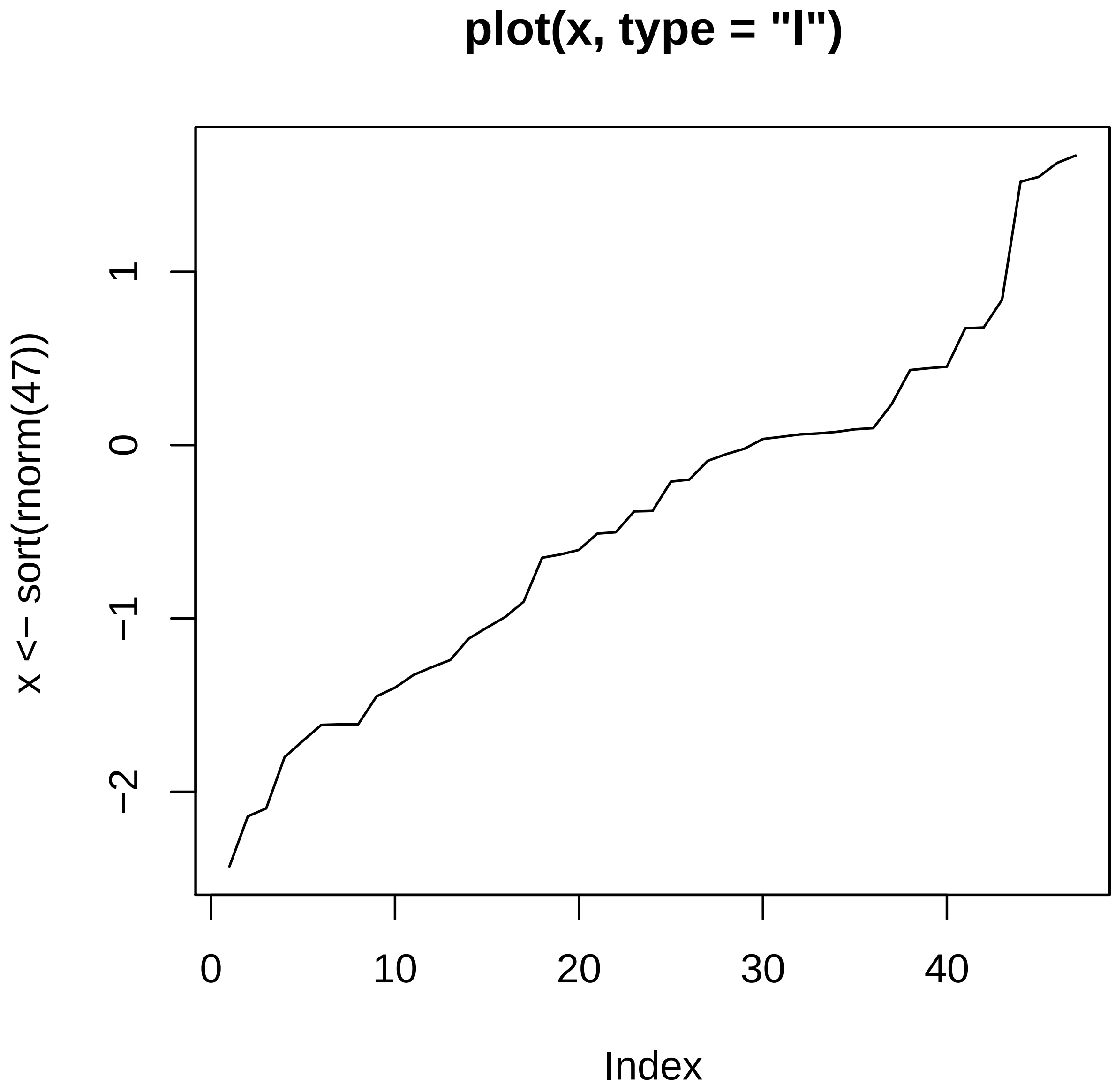
Some major high-level graphics commands

Graphic	Base command	Lattice command
scatterplot	plot()	xyplot()
line plot	plot(. . . ,type="l")	xyplot(. . . ,type="l")
Bar chart	barplot()	barchart()
Histogram	hist()	histogram()
Smoothed histograms	plot() after density()	densityplot()
boxplot	boxplot()	bwplot()
Dot plot	dotchart()	dotplot()
Contour plots	contour()	contourplot()
image plot	image()	levelplot()
3D surface	persp()	wireframe()
3D scatter	scatterplot3d()*	cloud()
conditional plots	coplot()	xyplot()
Scatterplot matrix		splom()
Parallel coordinates		parallel()
Star plot	stars()	
Stem-and-leaf plots	stem()	
ternary plot	ternaryplot() in vcd	
Fourfold plot	fourfoldplot() in vcd	
Mosaic plots	mosaicplot() in vcd	

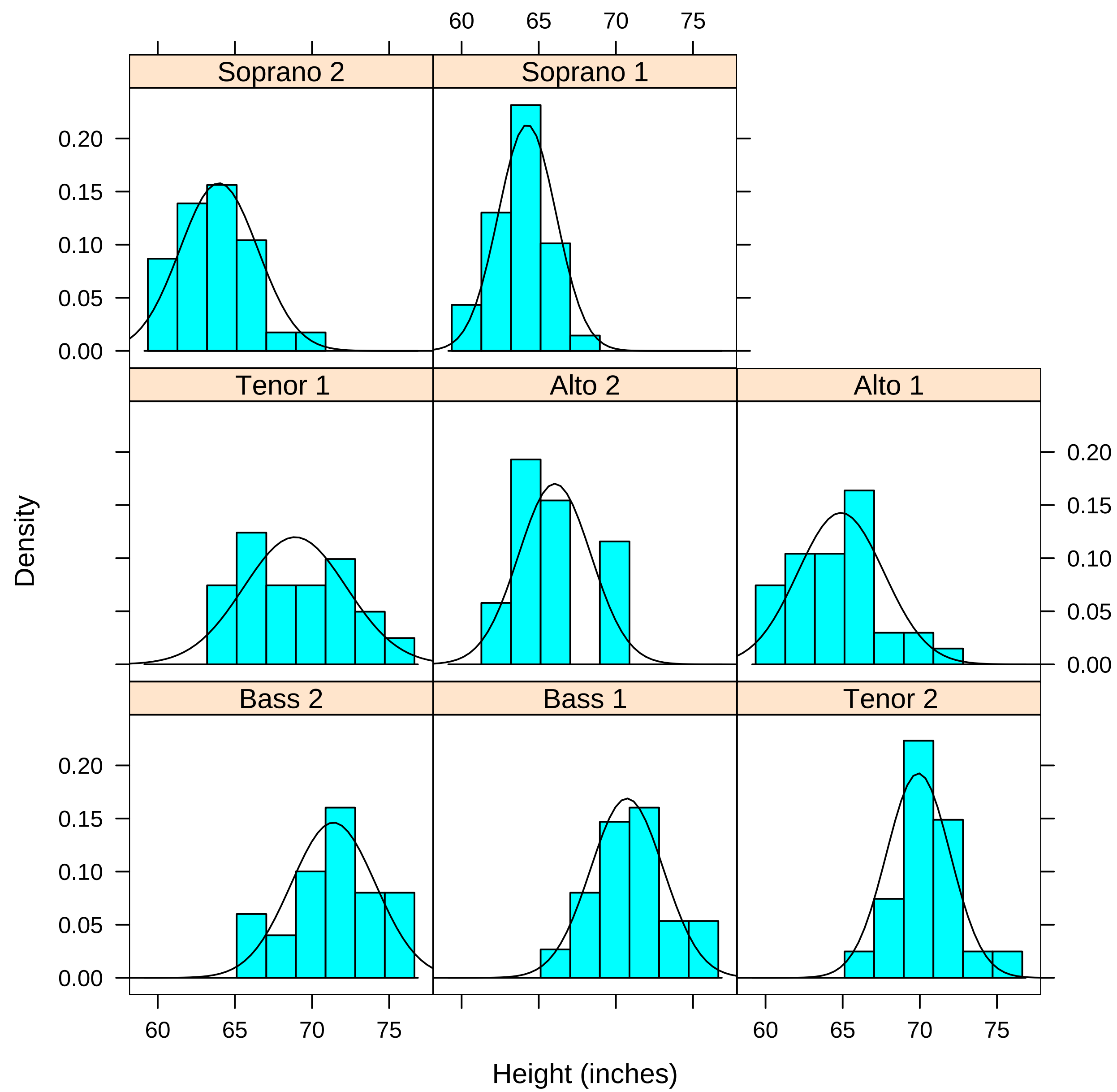
Scatterplot: `plot()`



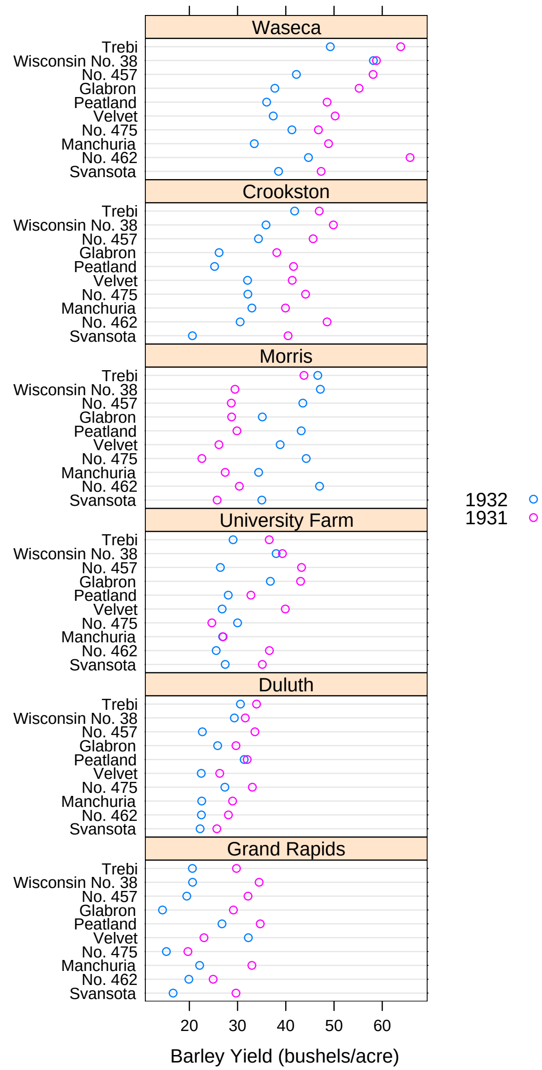
Line plot: `plot(...,type="l")`



(Smoothed) Histograms: `densityplot()` & others



Dot plot: `dotplot()`



Contour plot: `contour()`

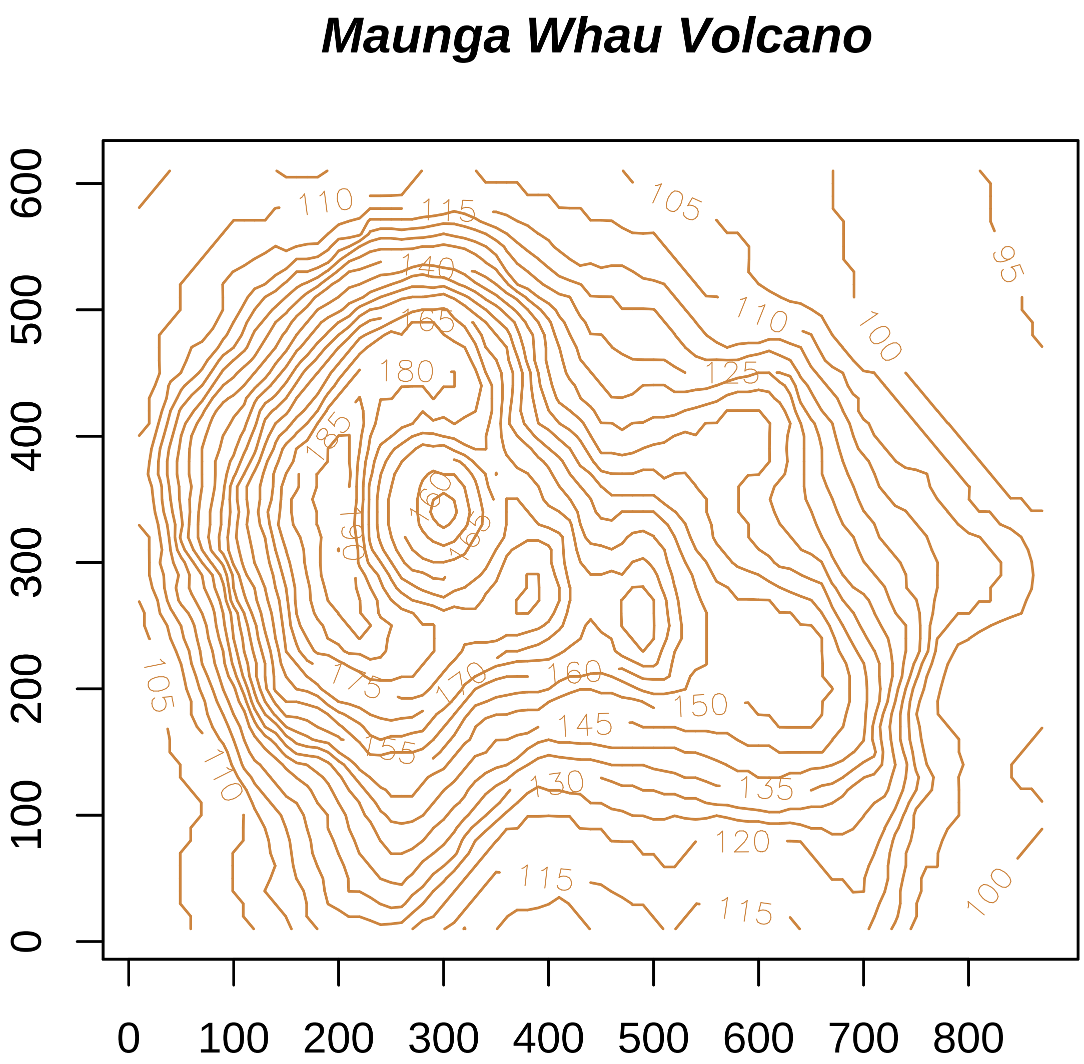


Image plot: `image()`

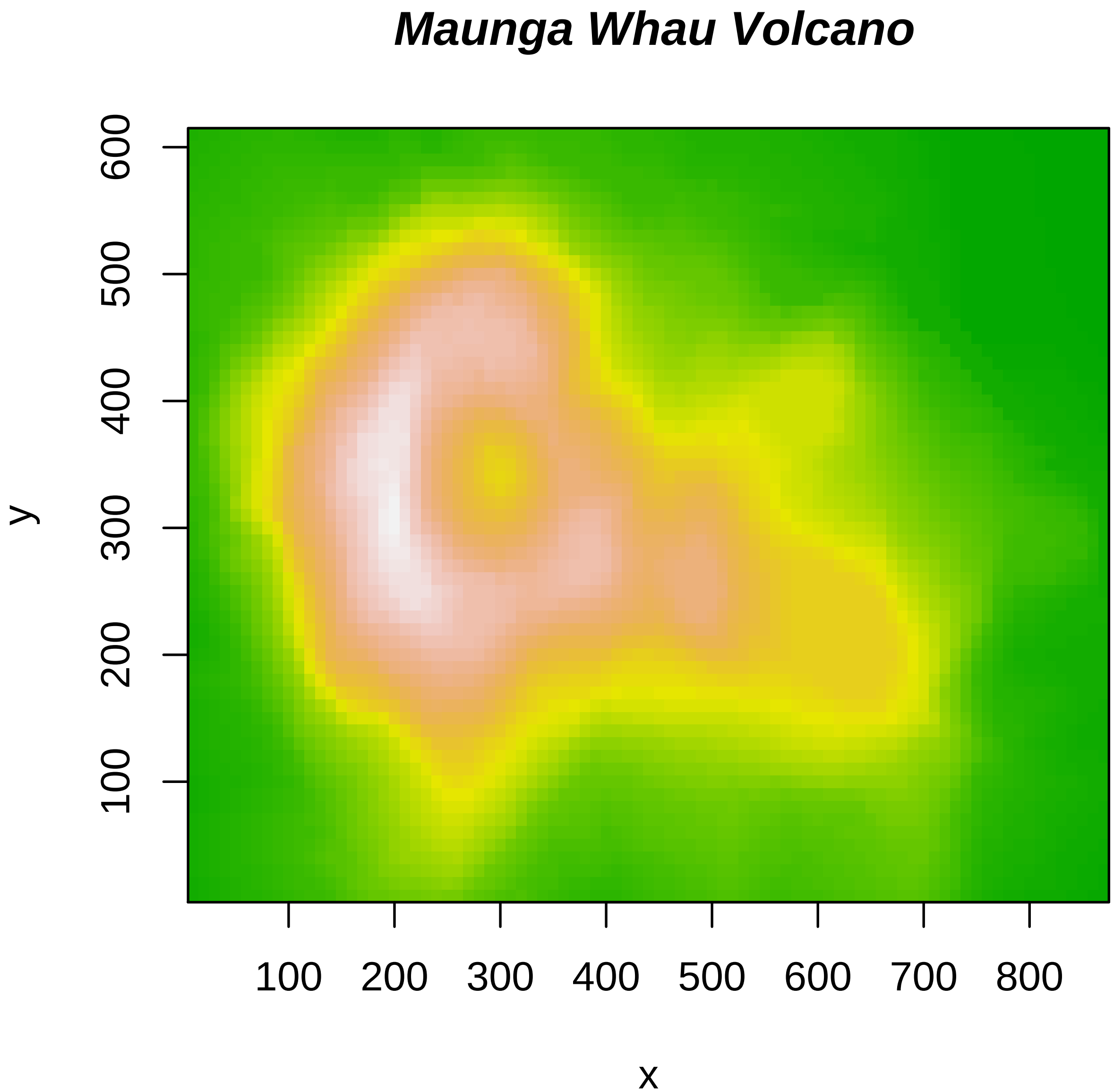
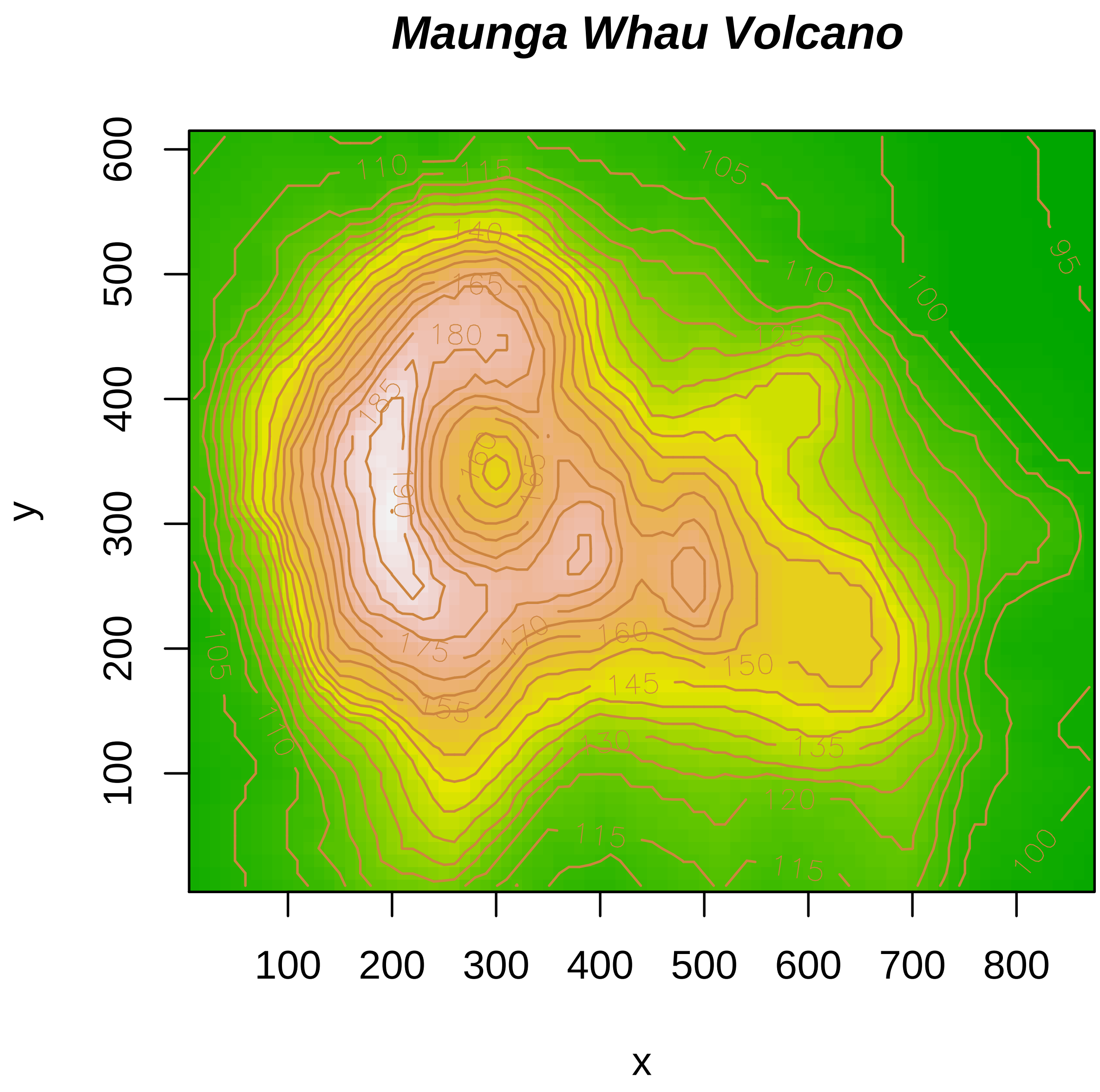
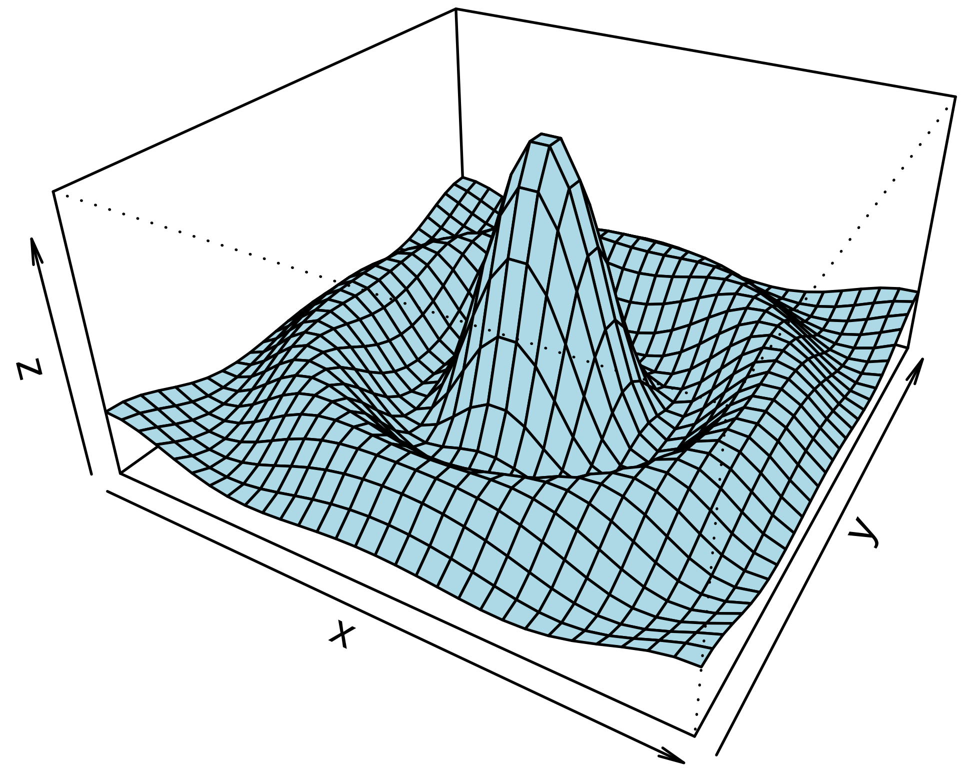


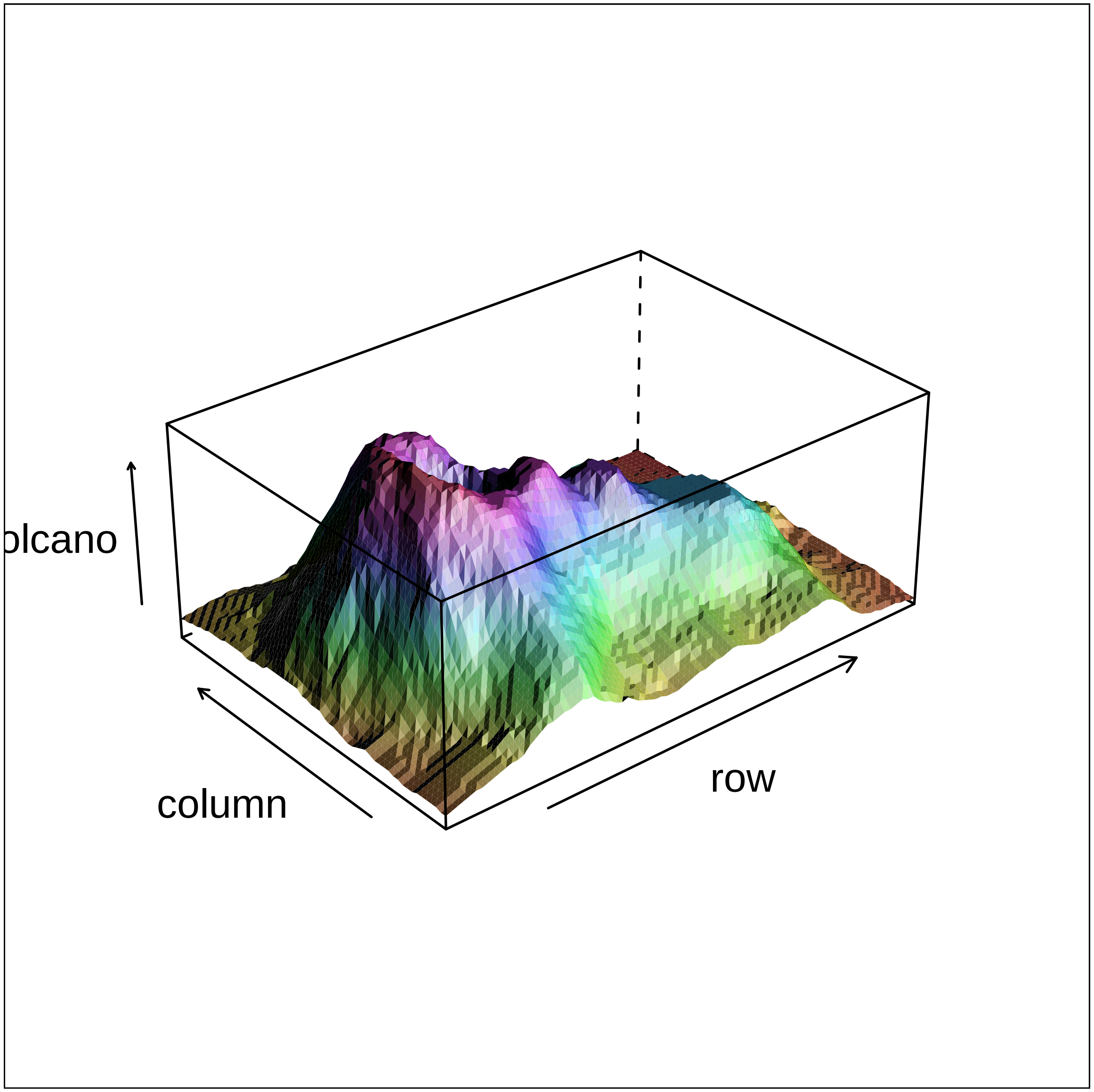
Image plot with contours: `contour(...,add=TRUE)`



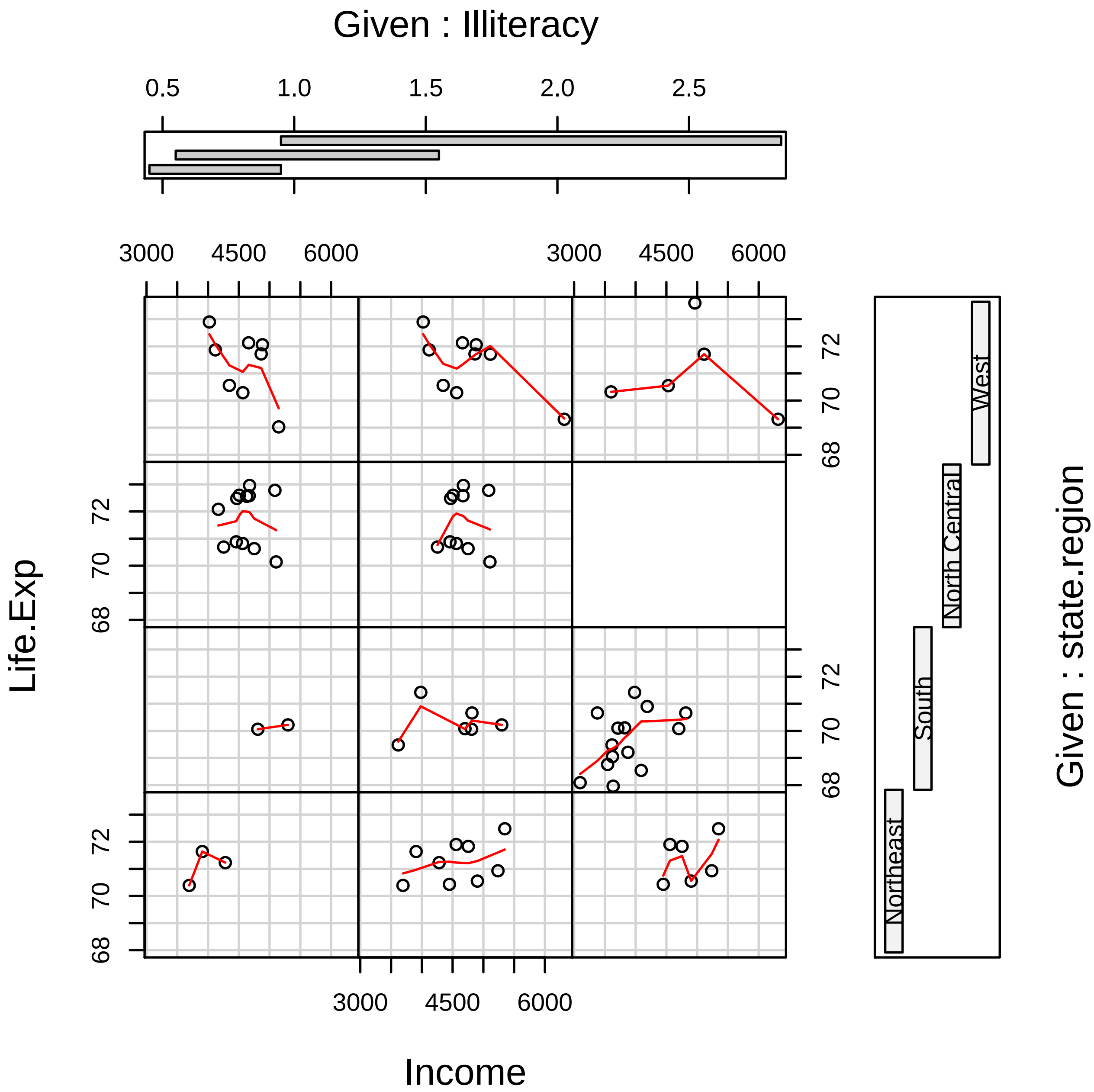
3D surface: `persp()`



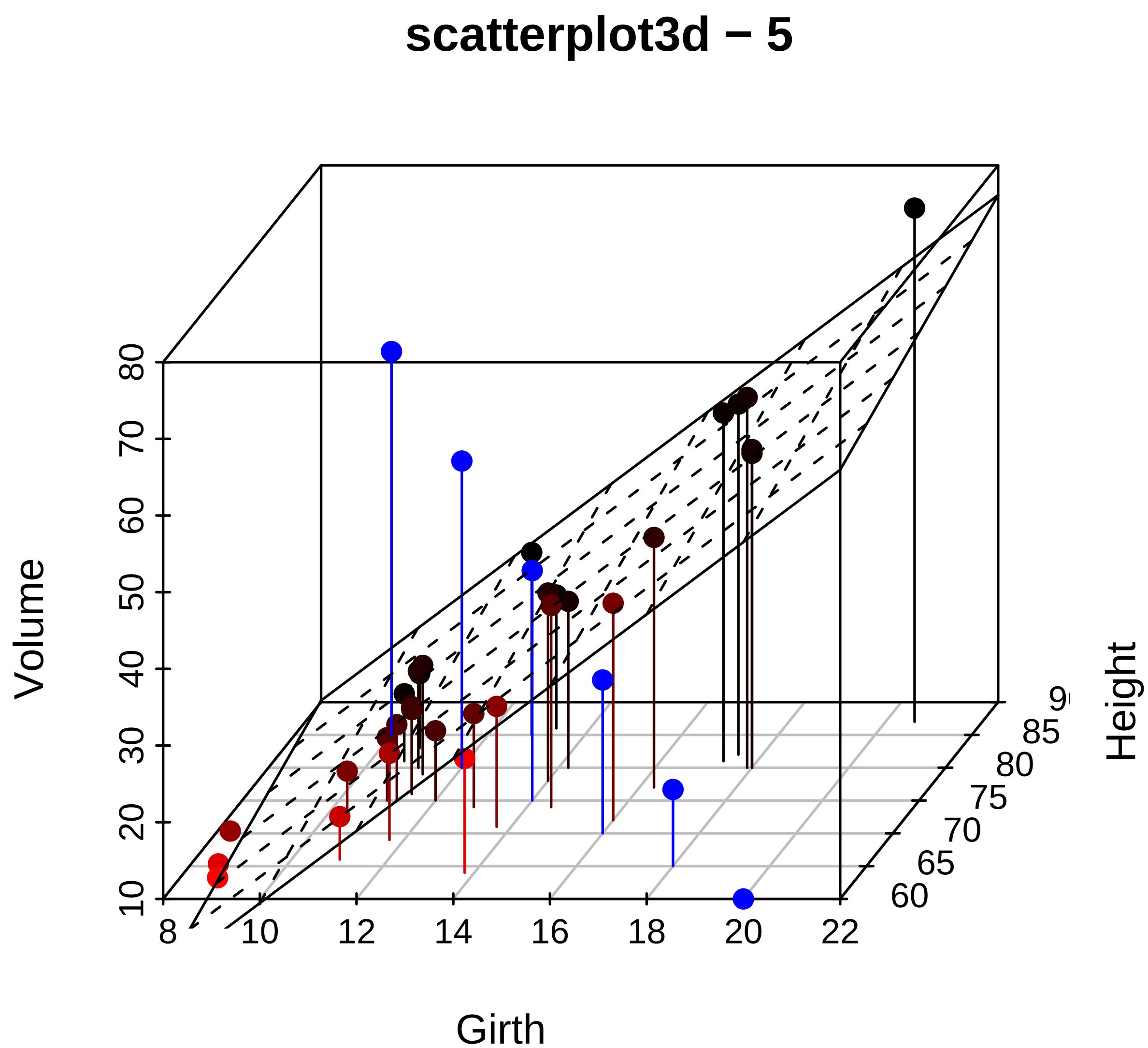
3D surface: `wireframe()`



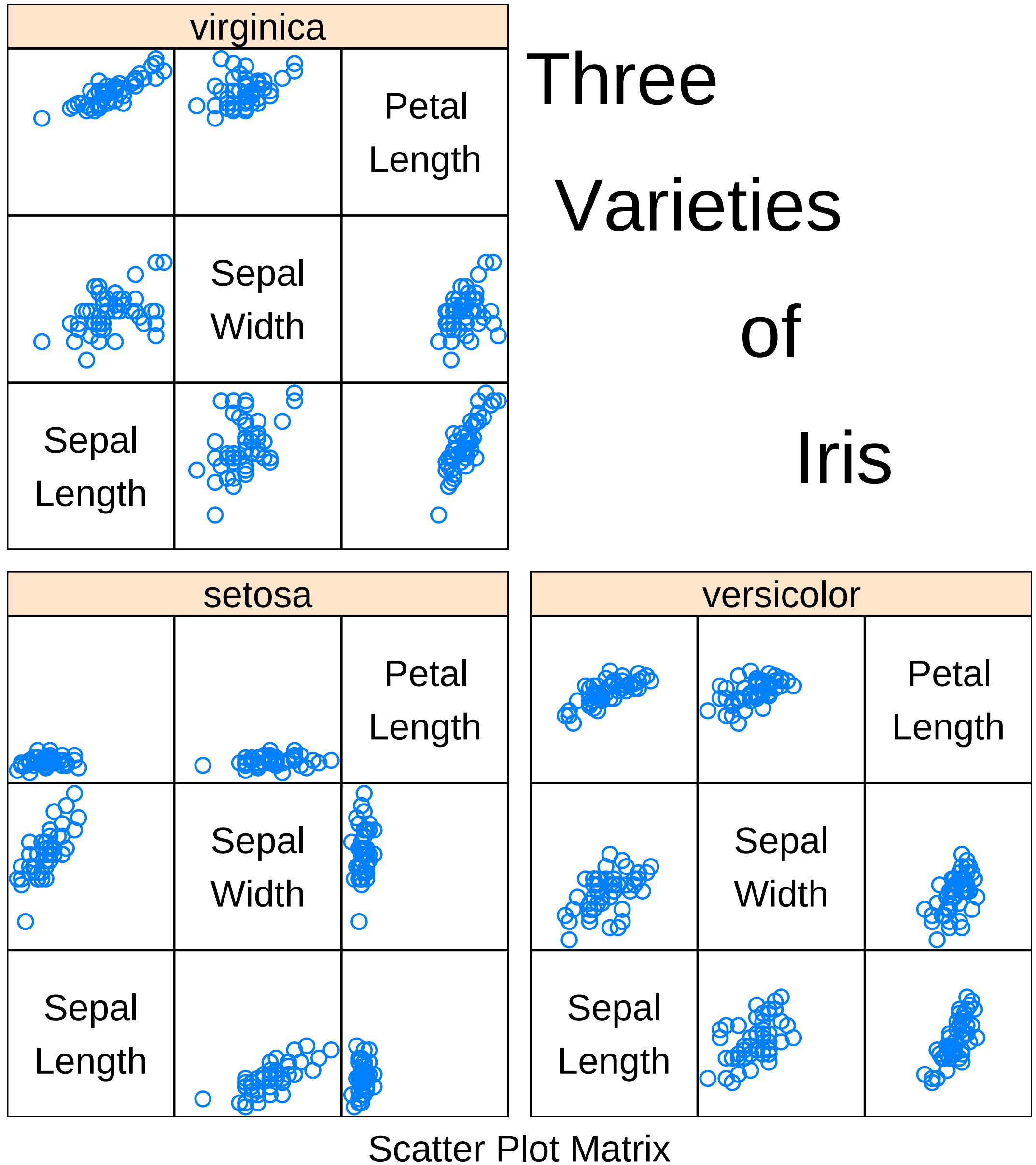
Conditional plots: `coplot()`



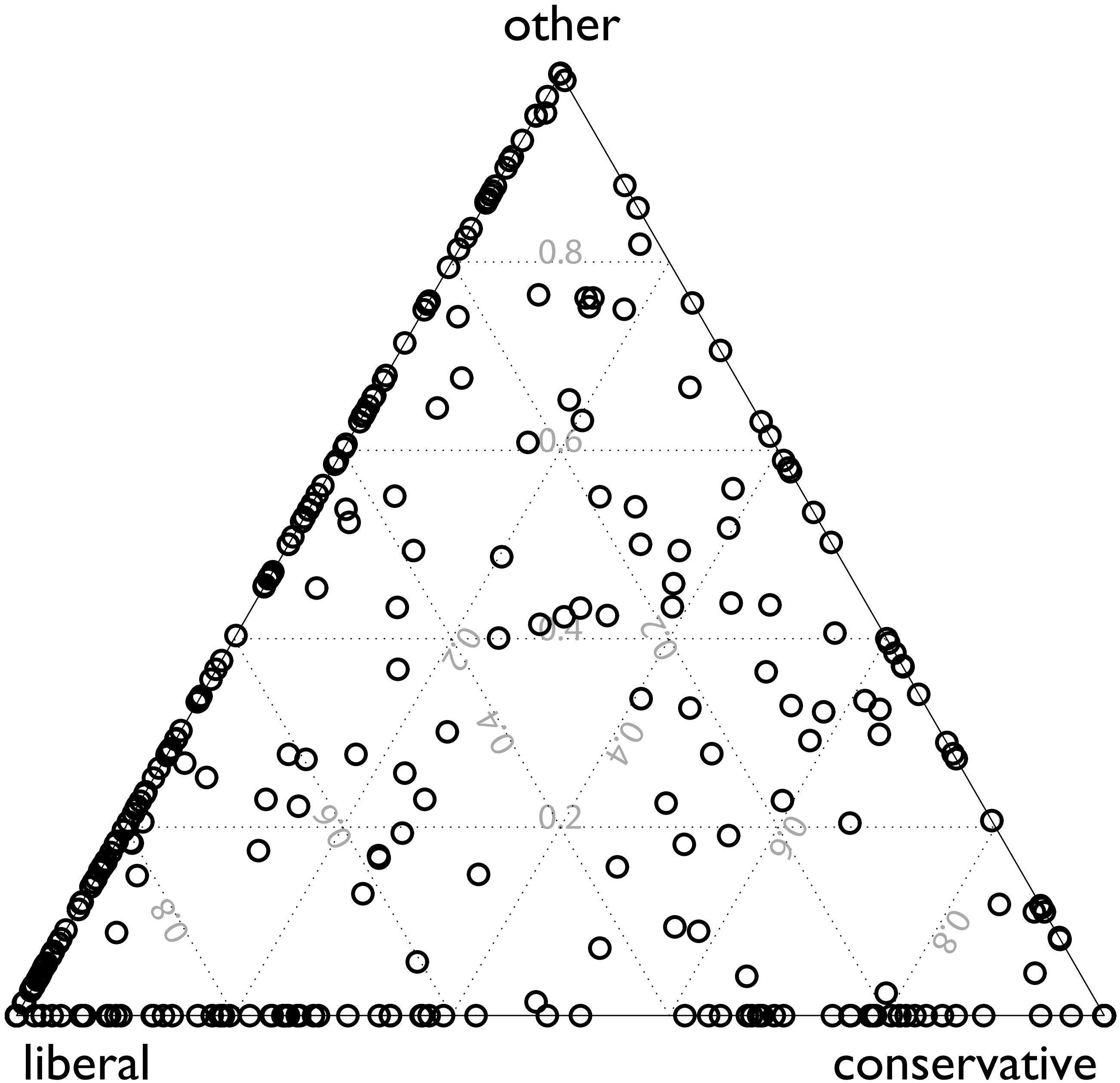
3D scatter: `scatterplot3d()` in own library



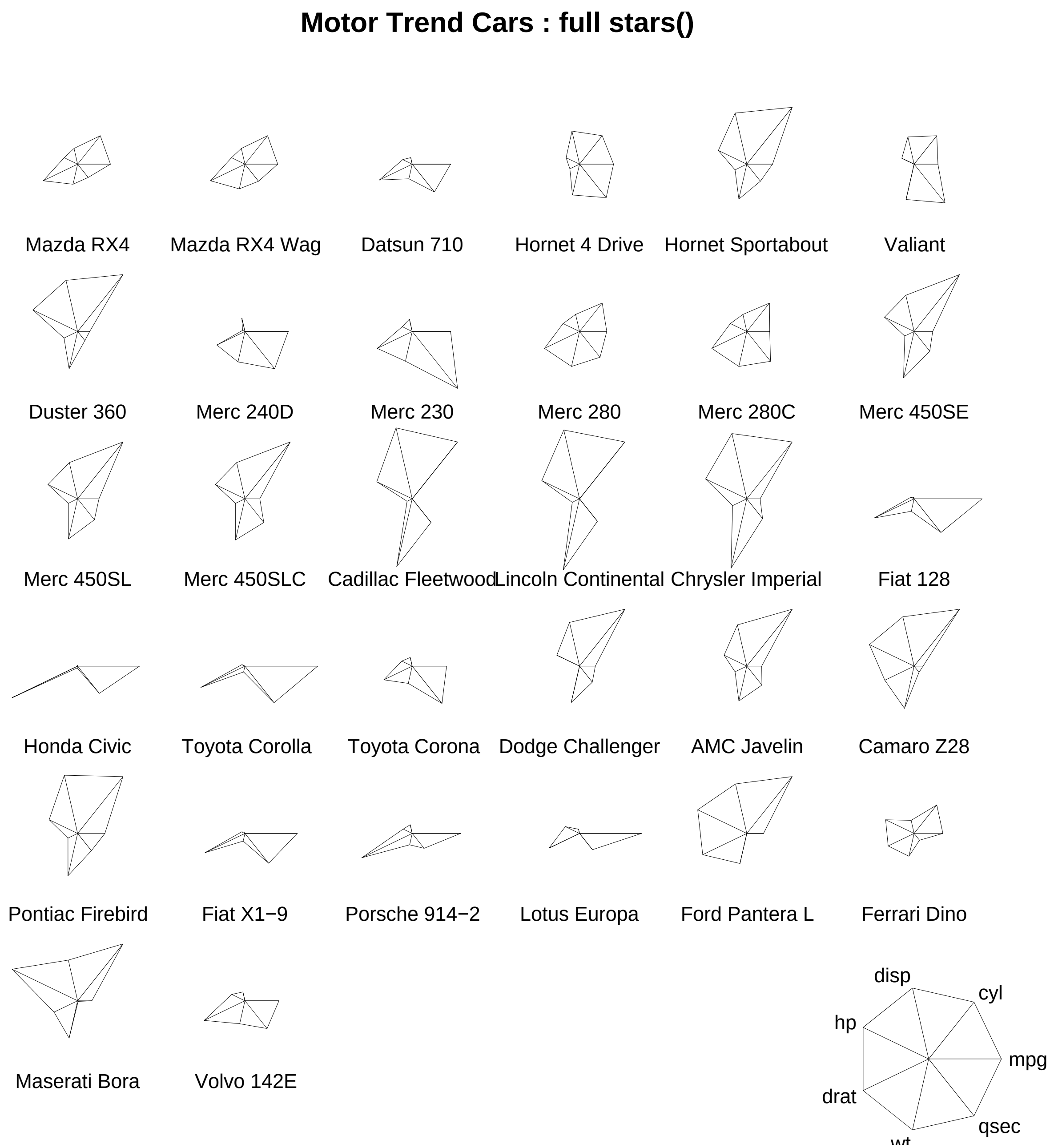
Scatterplot matrix: `splom()`



Ternary plot: `ternaryplot()` in `vcd`



Star plot: `stars()`



Stem-and-leaf plot

```
stem> stem(log10(islands))

  The decimal point is at the |

  1 | 1111112222233444
  1 | 5555556666667899999
  2 | 3344
  2 | 59
  3 |
  3 | 5678
  4 | 012
```

Basic customization

For any given high-level plotting command, there are many options listed in help

```
barplot(height, width = 1, space = NULL,
        names.arg = NULL, legend.text = NULL, beside = FALSE,
        horiz = FALSE, density = NULL, angle = 45,
        col = NULL, border = par("fg"),
        main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
        xlim = NULL, ylim = NULL, xpd = TRUE,
        axes = TRUE, axisnames = TRUE,
        cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
        inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0, ...)
```

Just the tip of the iceberg: notice the `...`

This means you can pass other, unspecified commands throough `barplot`

Basic customization

The most important (semi-) documented parameters to send through `...` are settings to `par()`

Most base (traditional) graphics options are set through `par()`

`par()` has no effect on lattice or grid graphics

Consult `help(par)` for the full list of options

Some key examples, grouped functionally

`par()` settings

Customizing text size:

cex	Text size (a multiplier)
cex.axis	Text size of tick numbers
cex.lab	Text size of axes labels
cex.main	Text size of plot title
cex.sub	Text size of plot subtitle

note the latter will multiply off the basic `cex`

`par()` settings

More text specific formatting

font	Font face (bold, italic)
font.axis	etc
srt	Rotation of text in plot (degrees)
las	Rotation of text in margin (degrees)

Note the distinction between text in the plot and outside.

Text in the plot is plotted with `text()`

Text outside the plot is plotted with `mtext()`, which was designed to put on titles, etc.

`par()` settings

Formatting for most any object

bg	background color
col	Color of lines, symbols in plot
col.axis	Color of tick numbers, etc

The above expect colors (see `colors()` for a list of names

`par()` settings

Formatting for lines and symbols

lty	Line type (solid, dashed, etc)
lwd	Line width (default too large; try really small, e.g., 0)
pch	Data symbol type; see `example(points)`

You will very often need to set the above

More `par()` settings

Formatting for axes

lab	Number of ticks
xaxp	Number of ticks for xaxis
tck,tcl	Length of ticks relative to plot/text
mgp	Axis spacing: axis title, tick labels, axis line

These may seem trivial, but affect the aesthetics of the plot & effective use of space

`R` defaults to excessive mgp, which looks ugly & wastes space

`par()` settings

More formating for axes

The following commands are special:
they are primitives in `par()` that can't be set inside the `...` of high-level commands

You must set them with `par()` first

usr	Ranges of axes, (xmin, xmax, ymin, ymax)
xlog	Log scale for x axis?
ylog	Log scale for y axis?

You can also make a logged axis by hand, as we will do now

Making a Scatterplot from Scratch

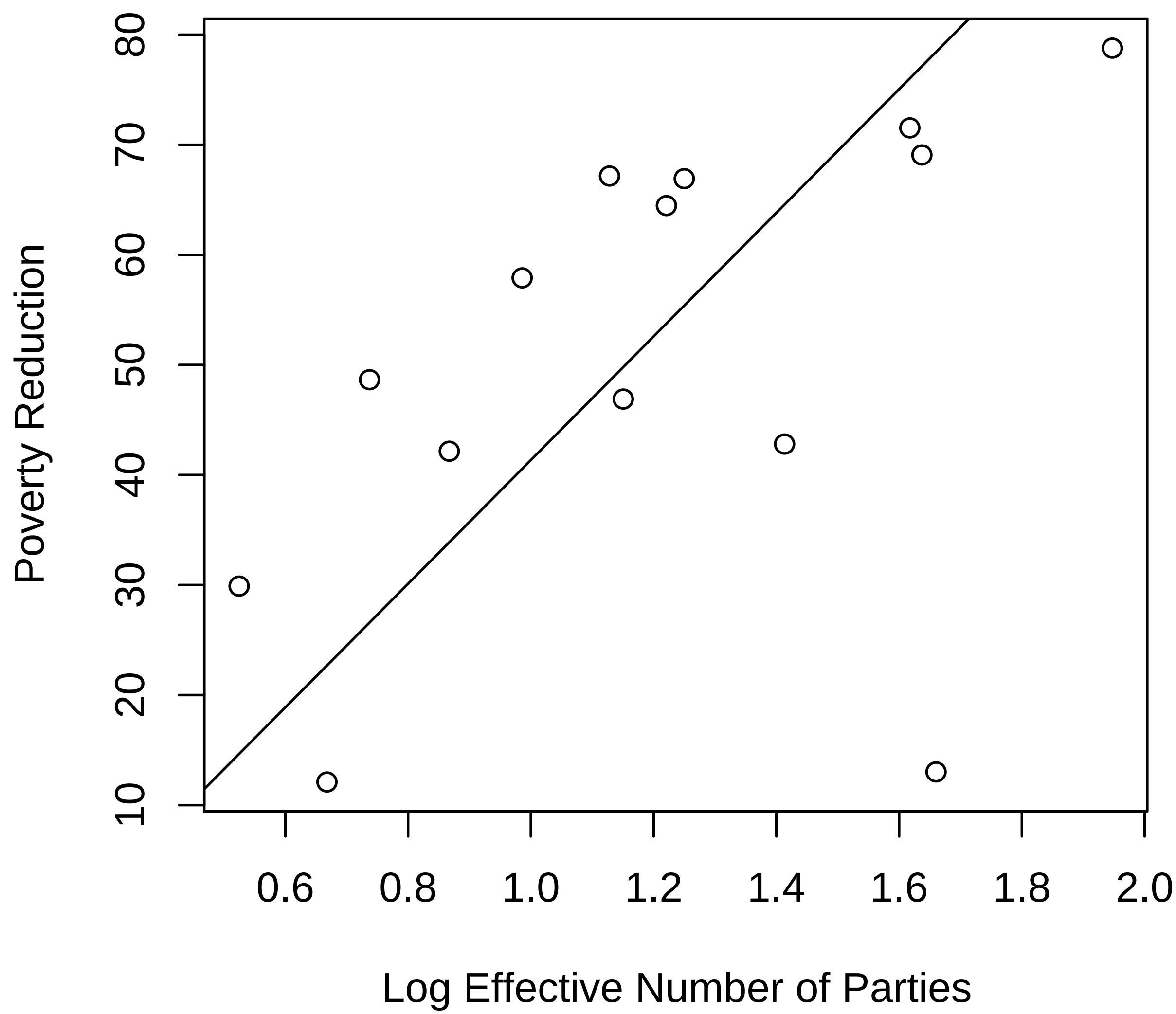
Using the Redistribution data, make a quick scatterplot for screen display:

```
# Make a plot of the data (automatic axes, etc)
plot(x=lnenp,
     y=povred,
     xlab="Log Effective Number of Parties",
     ylab="Poverty Reduction")

# One way to add a regression line to the plot
abline(lm.result$coefficients[1],    # Intercept
       lm.result$coefficients[2],    # Slope
       col="black")

# The above is easy for bivariate models
# For multivariate models, you need to calculate
# an appropriate intercept to take account
# of all the other covariates
```

A simple plot



What do we learn about the data from this plot?

What is problematic about this plot?

A better scatterplot from scratch

Let's make a better scatterplot, and save it to the disk as a PDF

First, let's find the confidence intervals for the fitted model:

```
# Generate expected values & CIs for povred at each lnenp

# Make a list of hypothetical effective number of parties values
lnenp.hyp <- seq(0.5,2,0.1)

# Use this list as "newdata" for the predict command
xnew <- list(lnenp=lnenp.hyp
             )

# Pass the fitted model and newdata to predict, and
# ask for 95 % CIs around the Y-hat
povred.pred <- predict(lm.result,
                       newdata=xnew,
                       interval="confidence",
                       level=0.95
                       )
```

Plotting preliminaries

```
# Open a pdf file for plotting
pdf("redist.pdf",
    height=5,
    width=5)

# Create a new plot
plot.new()
```

Plotting preliminaries

```
# Set the plotting region limits
par(usr=c(0.5,2,0,100))

# Create the x-axis
x.ticks <- c(2,3,4,5,6,7)
axis(1,                   # Which axis to make (1 indicates x)
     at=log(x.ticks),     # Where to put the ticks
     labels=x.ticks       # How to label the ticks
     )

# Create the y-axis
axis(2,at=seq(0,100,10))

# Add plot titles
title(xlab="Effective Number of Parties",
      ylab="Poverty Reduction"
      )
```

Plot the CI as a shaded polygon

```
# Plot ci for the regression line
# Make the x-coord of a confidence envelope polygon
xpoly <- c(lnenp.hyp,
           rev(lnenp.hyp),
           lnenp.hyp[1])

# Make the y-coord of a confidence envelope polygon
ypoly <- c(povred.pred[,2],
           rev(povred.pred[,3]),
           povred.pred[1,2])

# Choose the color of the polygon
col <- "gray70"

# Plot the polygon first, before the points & lines
polygon(x=xpoly,
        y=ypoly,
        col=col,
        border=FALSE
        )
```

Add the regression line and the data

```
# Plot the expected values for the regression model
lines(x=lnenp.hyp,
      y=povred.pred[,1],
      col="black")

# Plot the data for the regression model
#points(x=lnenp,
#       y=povred,
#       col="black",   # see colors() for color names
#       pch=1)         # see example(points) for symbols
```

Use colors and shapes to show categorical covariates

```
points(x=lnenp[maj==1],
       y=povred[maj==1],
       col="blue",      # see colors() for color names
       pch=17)          # see example(points) for symbols

points(x=lnenp[pr==1],
       y=povred[pr==1],
       col="green",     # see colors() for color names
       pch=15)          # see example(points) for symbols

points(x=lnenp[unam==1],
       y=povred[unam==1],
       col="red",       # see colors() for color names
       pch=16)          # see example(points) for symbols
```

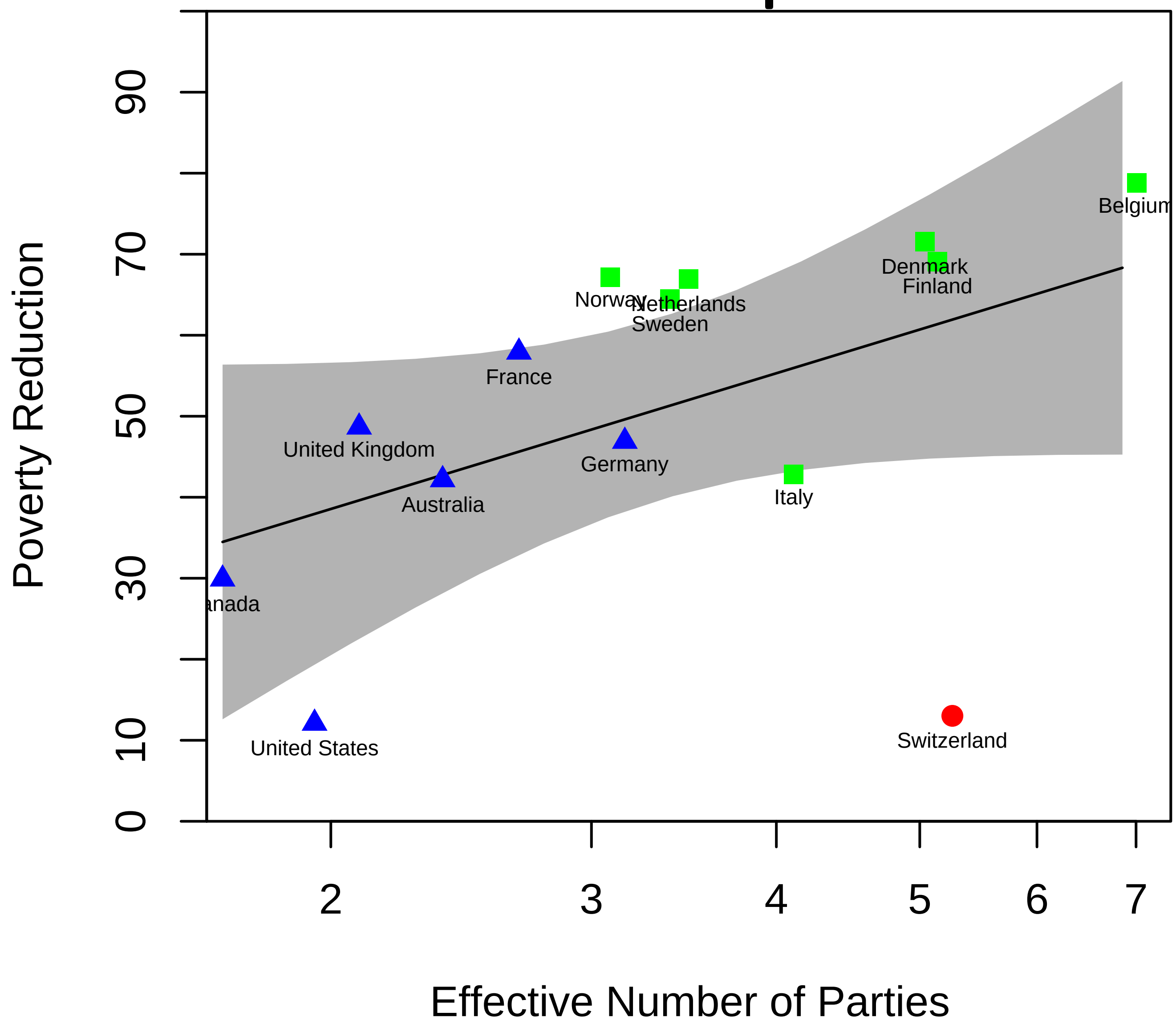
Label the points and close the plot

```
text(x=lnenp,
     y=povred-3,
     labels=cty,
     col="black",
     cex=0.5
     )

# Finish drawing the box around the plot area
box()

# Close the device (ie, save the graph)
dev.off()
```

The finished plot



What does this tell us about the data?

What could we improve, in the plot or the model?