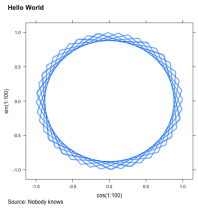
Lattice graphics

Ron Wehrens

December 14, 2016
Biometris
Wageningen University & Research



Displaying data (and models)

Good data graphics:
display data accurately and
clearly

def·i·ni·tion n. 1.
The teacher gave de
of the new words.
of an image (pict

Seminal reference:
"Visualizing data"
William S. Cleveland, (1993)

Displaying data (and models)

Good data graphics:
display data accurately and clearly

which a televised image
broadcast signal is rec

def·i·ni·tion n. 1.
The teacher gave de
of the new words.
of an image (pict

Seminal reference:
"Visualizing data"
William S. Cleveland, (1993)

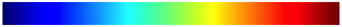
"How to display data badly",

Howard Wainer, The American Statistician (1984)

1. show as few data as possible
2. hide what data you do show
3. ignore the visual metaphor
4. only order matters
5. graph data out of context
6. change scales in mid-axis
7. emphasize the trivial
8. jiggle the baseline
9. Austria first!
10. Label illegibly, incompletely, incorrectly and ambiguously
11. more is murkier
12. if it has been done well in the past, do it differently

Example: choice of colors

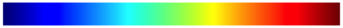
Matlab's colorjet:



- introduces artefacts
- highlights specific regions
- no BW print
- bad for colour-impaired people

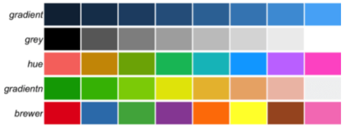
Example: choice of colors

Matlab's colorjet:



- introduces artefacts
- highlights specific regions
- no BW print
- bad for colour-impaired people

Alternatives:



From the `RColorBrewer` package

Plotting systems in R

1. `graphics` (R Core team, ...)
   base graphics, based on Cleveland principles

Plotting systems in R

1. `graphics` (R Core team, ...)
   base graphics, based on Cleveland principles
2. `lattice` (Deepayan Sarkar, ...)
   "Trellis" graphics, especially good for multi-panel plots

Plotting systems in R

1. `graphics` (R Core team, ...)
   base graphics, based on Cleveland principles
2. `lattice` (Deepayan Sarkar, ...)
   "Trellis" graphics, especially good for multi-panel plots
3. `ggplot2` (Hadley Wickham, ...)
   "tries to take the good parts of base and lattice graphics and none of the bad parts" ...

Plotting systems in R

1. `graphics` (R Core team, ...)
   base graphics, based on Cleveland principles
2. `lattice` (Deepayan Sarkar, ...)
   "Trellis" graphics, especially good for multi-panel plots
3. `ggplot2` (Hadley Wickham, ...)
   "tries to take the good parts of base and lattice graphics and none of the bad parts" ...

Plotting systems in R

1. `graphics` (R Core team, ...)
   base graphics, based on Cleveland principles
2. `lattice` (Deepayan Sarkar, ...)
   "Trellis" graphics, especially good for multi-panel plots
3. `ggplot2` (Hadley Wickham, ...)
   "tries to take the good parts of base and lattice graphics and none of the bad parts" ... but very un-R-like syntax

Plotting systems in R

1. `graphics` (R Core team, ...)
   base graphics, based on Cleveland principles
2. `lattice` (Deepayan Sarkar, ...)
   "Trellis" graphics, especially good for multi-panel plots
3. `ggplot2` (Hadley Wickham, ...)
   "tries to take the good parts of base and lattice graphics and none of the bad parts" ... but very un-R-like syntax

Both `lattice` and `ggplot2` make extensive use of the `grid` package (Paul Murrel, R Core team)

Example: the `mtcars` data

```
data(mtcars)
class(mtcars)
```

```
[1] "data.frame"
```

```
dim(mtcars)
```

```
[1] 32 11
```

Example: the mtcars data

```
data(mtcars)
class(mtcars)

[1] "data.frame"

dim(mtcars)

[1] 32 11
```

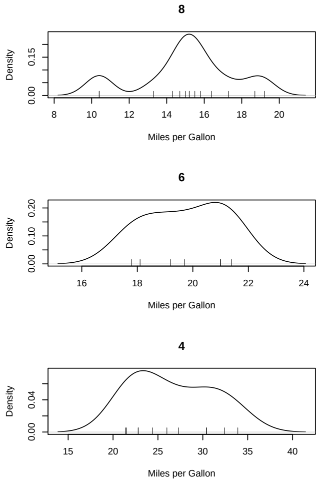
```
head(mtcars[,1:4])

                   mpg cyl disp  hp
Mazda RX4         21.0   6  160 110
Mazda RX4 Wag     21.0   6  160 110
Datsun 710        22.8   4  108  93
Hornet 4 Drive    21.4   6  258 110
Hornet Sportabout 18.7   8  360 175
Valiant           18.1   6  225 105
```

Miles per gallon...

```r
ncyl <- c(8, 6, 4)
par(mfrow = c(3,1))

for (nc in ncyl) {
  idx <- which(mtcars$cyl == nc)

  plot(density(mtcars$mpg[idx]),
       main = nc,
       xlab = "Miles per Gallon",
       ylab = "Density")
  rug(mtcars$mpg[idx], ticksize = .1)
}
```

Miles per gallon...

```r
ncyl <- c(8, 6, 4)
par(mfrow = c(3,1))

for (nc in ncyl) {
  idx <- which(mtcars$cyl == nc)

  plot(density(mtcars$mpg[idx]),
       main = nc,
       xlab = "Miles per Gallon",
       ylab = "Density")
  rug(mtcars$mpg[idx], ticksize = .1)
}
```

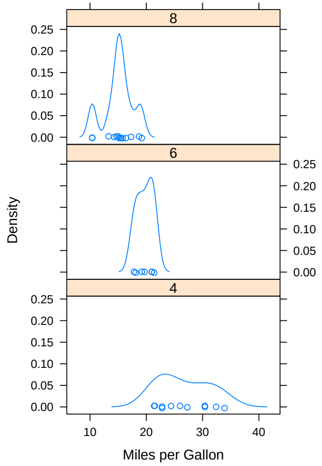


The same in lattice

```
mtcars$cyl.f <- factor(mtcars$cyl)
densityplot(~ mpg | cyl.f,
  data = mtcars,
  xlab = "Miles per Gallon",
  layout = c(1,3))
```

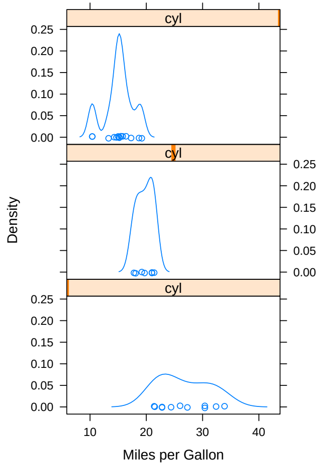
The same in lattice

```r
mtcars$cyl.f <- factor(mtcars$cyl)
densityplot(~ mpg | cyl.f,
  data = mtcars,
  xlab = "Miles per Gallon",
  layout = c(1,3))
```



By the way...

```
densityplot(~ mpg | cyl,
  data = mtcars,
  xlab = "Miles per Gallon",
  layout = c(1,3))
```

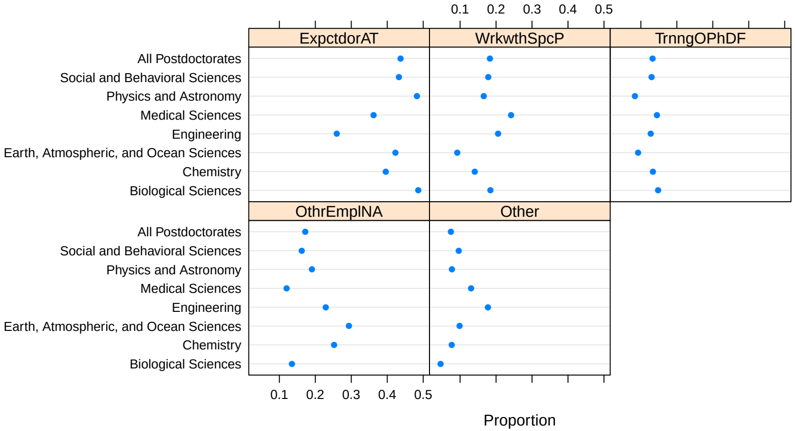


And a more complicated example...

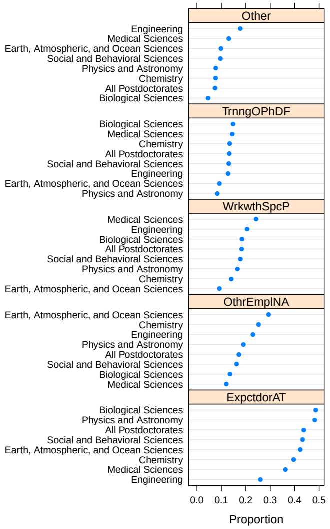
How about two factors?

```
data(postdoc, package = "latticeExtra")
rownames(postdoc)

[1] "Biological Sciences"
[2] "Chemistry"
[3] "Earth, Atmospheric, and Ocean Sciences"
[4] "Engineering"
[5] "Medical Sciences"
[6] "Physics and Astronomy"
[7] "Social and Behavioral Sciences"
[8] "All Postdoctorates"

colnames(postdoc)

[1] "Expected or Additional Training" "Work with Specific Person"
[3] "Training Outside PhD Field"      "Other Employment Not Available"
[5] "Other"
```

The postdoc plot (first version)



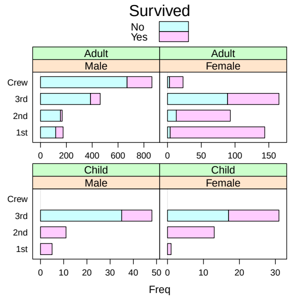
Another version...



- Each panel sorted: easy interpretation
- Common x scale: easy comparison
- Include zero on x scale

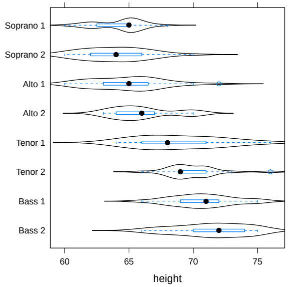
Basic lattice plots: univariate

- `barchart`
- `bwplot`
- `densityplot`
- `stripplot`
- `dotplot`
- `histogram`
- `qqmath`



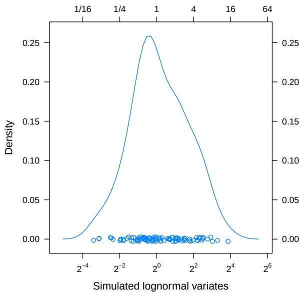
Basic lattice plots: univariate

- `barchart`
- `bwplot`
- `densityplot`
- `stripplot`
- `dotplot`
- `histogram`
- `qqmath`



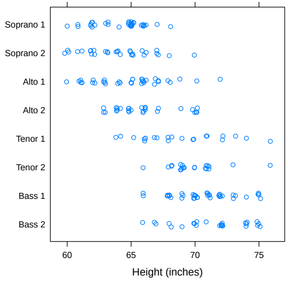
Basic lattice plots: univariate

- barchart
- bwplot
- densityplot
- stripplot
- dotplot
- histogram
- qqmath



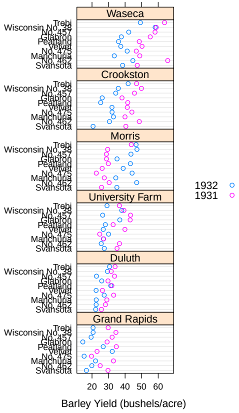
Basic lattice plots: univariate

- barchart
- bwplot
- densityplot
- stripplot
- dotplot
- histogram
- qqmath



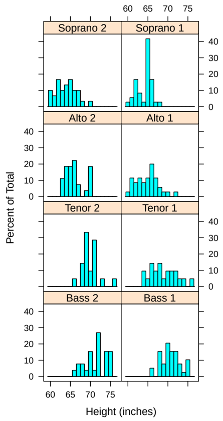
Basic lattice plots: univariate

- `barchart`
- `bwplot`
- `densityplot`
- `stripplot`
- `dotplot`
- `histogram`
- `qqmath`



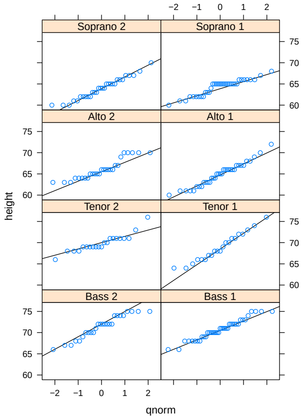
Basic lattice plots: univariate

- `barchart`
- `bwplot`
- `densityplot`
- `stripplot`
- `dotplot`
- `histogram`
- `qqmath`



Basic lattice plots: univariate

- `barchart`
- `bwplot`
- `densityplot`
- `stripplot`
- `dotplot`
- `histogram`
- `qqmath`



Further lattice plots

Bivariate, trivariate

- `qq`
- `xyplot`
- `levelplot`
- `contourplot`
- `cloud`
- `wireframe`

Hypervariate, miscellaneous

- `splom`
- `parallel`
- `rfs`
- `tmd`

Key features

- formula interface

Key features

- formula interface
- especially useful for `data.frame` objects

Key features

- formula interface
- especially useful for `data.frame` objects
- also methods for other variable types

Key features

- formula interface
- especially useful for `data.frame` objects
- also methods for other variable types
- automatic splits into panels

Key features

- formula interface
- especially useful for `data.frame` objects
- also methods for other variable types
- automatic splits into panels
- defined by factor levels or variable ranges

Key features

- formula interface
- especially useful for `data.frame` objects
- also methods for other variable types
- automatic splits into panels
- defined by factor levels or variable ranges
- indications of groups

Key features

- formula interface
- especially useful for `data.frame` objects
- also methods for other variable types
- automatic splits into panels
- defined by factor levels or variable ranges
- indications of groups
- automatic keys/legends

Key features

- formula interface
- especially useful for `data.frame` objects
- also methods for other variable types
- automatic splits into panels
- defined by factor levels or variable ranges
- indications of groups
- automatic keys/legends
- highly customizable

Standard plot customizations

Data: locations and characteristics of earthquakes near Fiji

Simplest possible case: show locations

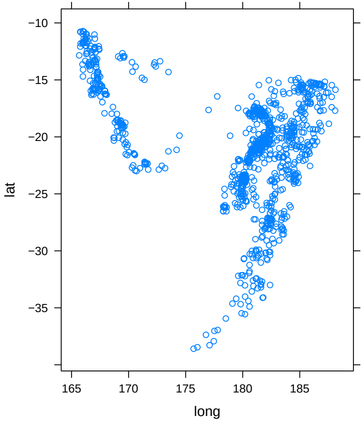
```
xyplot(lat ~ long,
   data = quakes,
   aspect = "iso")
```

Standard plot customizations

Data: locations and characteristics of earthquakes near Fiji

Simplest possible case: show locations

```
xyplot(lat ~ long,
   data = quakes,
   aspect = "iso")
```



Standard plot customizations

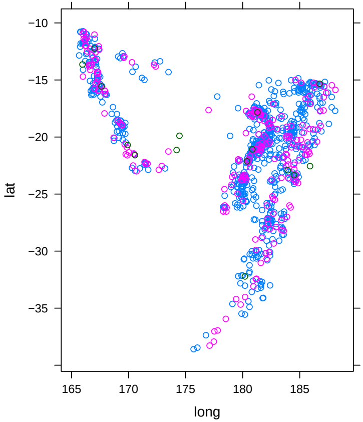
Now: location and magnitude

```
xyplot(lat ~ long,
   data = quakes,
   groups =  cut(mag, 3),
   aspect = "iso")
```

Standard plot customizations

Now: location and magnitude

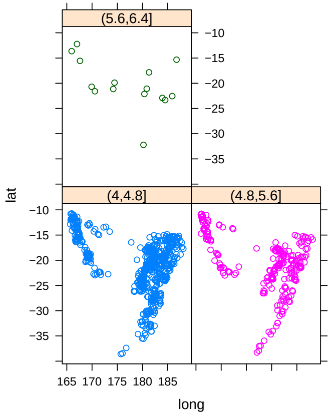
```
xyplot(lat ~ long,
   data = quakes,
   groups =  cut(mag, 3),
   aspect = "iso")
```



Standard plot customizations

Now: location and magnitude

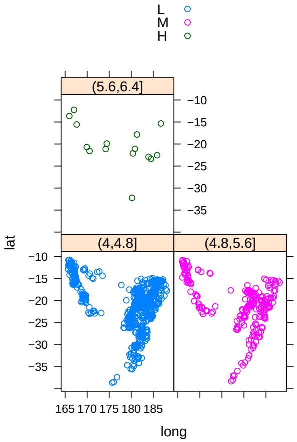
```
xyplot(lat ~ long | cut(mag, 3),
   data = quakes,
   groups =  cut(mag, 3),
   aspect = "iso")
```



Standard plot customizations

Now: location and magnitude

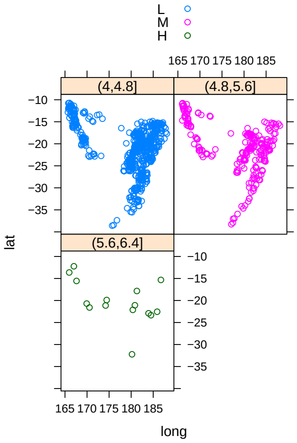
```
xyplot(lat ~ long | cut(mag, 3),
   data = quakes,
   groups =  cut(mag, 3),
   aspect = "iso",
   auto.key =
     list(text = c("L", "M", "H"),
          space = "top"))
```



Standard plot customizations

Now: location and magnitude

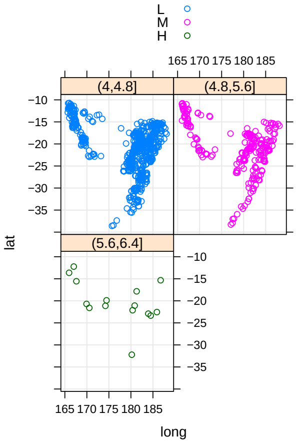
```
xyplot(lat ~ long | cut(mag, 3),
   data = quakes,
   groups =  cut(mag, 3),
   aspect = "iso",
   as.table = TRUE,
   auto.key =
     list(text = c("L", "M", "H"),
          space = "top"))
```



Standard plot customizations

Now: location and magnitude

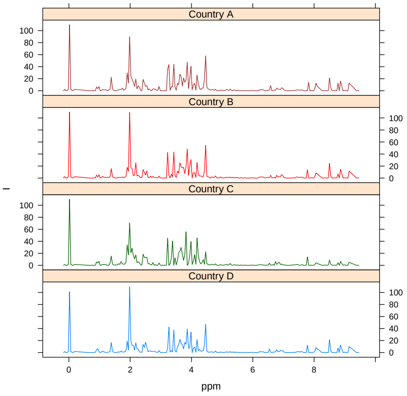
```
xyplot(lat ~ long | cut(mag, 3),
   data = quakes,
   groups =  cut(mag, 3),
   aspect = "iso",
   as.table = TRUE,
   auto.key =
     list(text = c("L", "M", "H"),
          space = "top"),
   type = c("p", "g"))
```



An elegant example: NMR spectra

"Naive" plot leads to an inverted x axis:

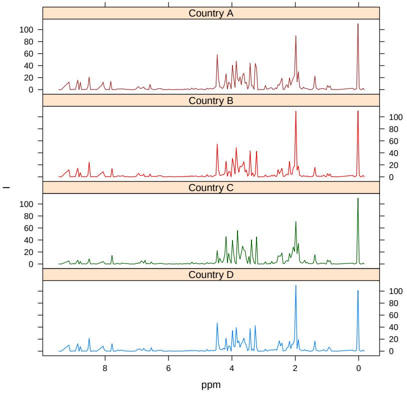
```
xyplot(I ~ ppm | Origin,
  groups = Sample,
  data = NMR.df,
  type = "l",
  layout = c(1,4),
  as.table = TRUE)
```



An elegant example: NMR spectra

... but this is easily remedied:

```
xyplot(I ~ ppm | Origin,
  groups = Sample,
  data = NMR.df,
  type = "l",
  layout = c(1,4),
  as.table = TRUE,
  xlim =
    rev(extendrange(NMR.df$ppm)))
```

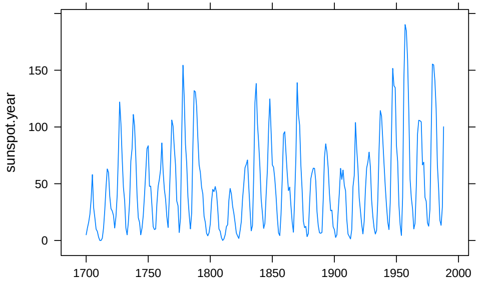


Banking: the sunspots data

```
myplot <-
  xyplot(sunspot.year ~
           1700:1988,
         xlab = "",
         type = "l")
```

Default:

`aspect = "fill”`



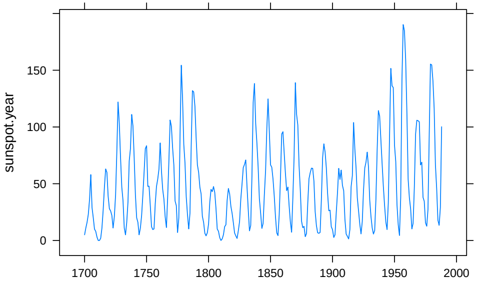
Banking: the sunspots data

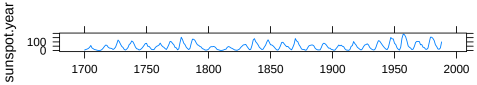
```
myplot <-
  xyplot(sunspot.year ~
            1700:1988,
          xlab = "",
          type = "l")
```

Default:

`aspect = "fill”`

```
myplot2 <-
  update(myplot,
          aspect = "xy")
```





The `prepanel` and `panel` functions

"For the little things...
that take soooo much time"

The `prepanel` and `panel` functions

"For the little things...
that take soooo much time"

- `packet`: group of data points to be shown in one "shingle"

The `prepanel` and `panel` functions

"For the little things...
that take soooo much time"

- `packet`: group of data points to be shown in one "shingle"
- `prepanel` function: determines rectangle to display the packet

The `prepanel` and `panel` functions

"For the little things...
that take soooo much time"

- `packet`: group of data points to be shown in one "shingle"
- `prepanel` function: determines rectangle to display the packet
- `panel` function: determines how to display the packet

The `prepanel` and `panel` functions

"For the little things...
that take soooo much time"

- `packet`: group of data points to be shown in one "shingle"
- `prepanel` function: determines rectangle to display the packet
- `panel` function: determines how to display the packet

The `prepanel` and `panel` functions

"For the little things...
that take soooo much time"

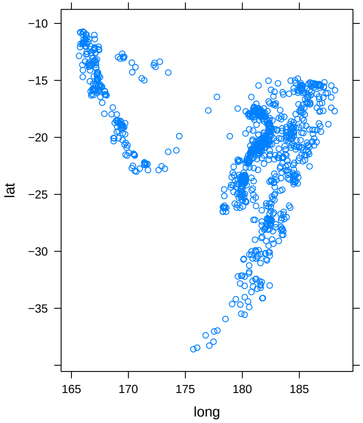
- `packet`: group of data points to be shown in one "shingle"
- `prepanel` function: determines rectangle to display the packet
- `panel` function: determines how to display the packet

For many plot types there are basic "panel" functions:
`panel.xyplot`,
`panel.stripplot`,
`panel.splom`, ...

Many supporting plotting functions also have "panel" variants:
`panel.points`, `panel.lines`,
`panel.text`

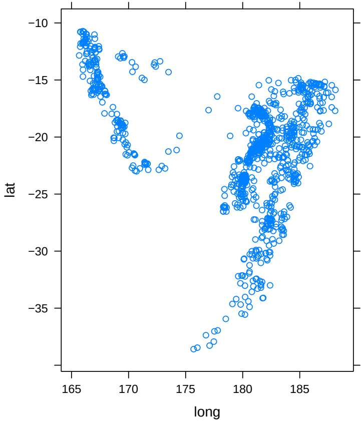
Building it up

```
xyplot(lat ~ long,
   data = quakes,
   aspect = "iso")
```



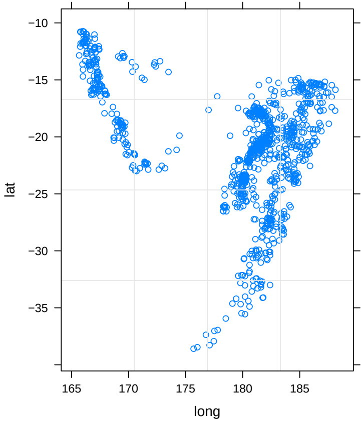
Building it up

```
xyplot(lat ~ long,
  data = quakes,
  aspect = "iso",
  panel = function(...) {
    panel.xyplot(...)
    })
```



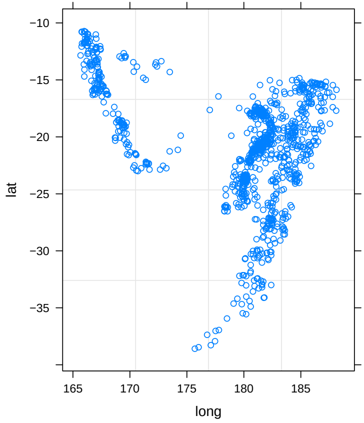
Building it up

```
xyplot(lat ~ long,
  data = quakes,
  aspect = "iso",
  panel = function(...) {
    panel.xyplot(...)
    panel.grid()
  })
```



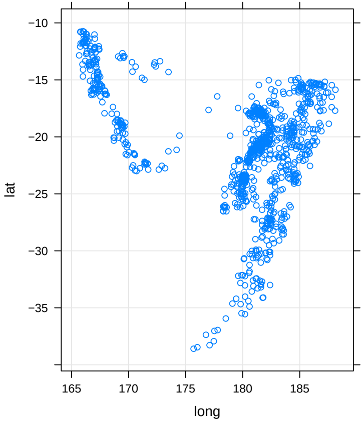
Building it up

```
xyplot(lat ~ long,
  data = quakes,
  aspect = "iso",
  panel = function(...) {
    panel.grid()

    panel.xyplot(...)
  })
```



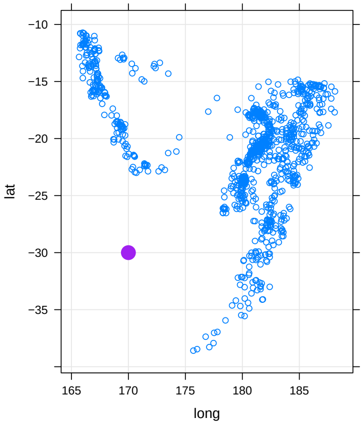
Building it up

```
xyplot(lat ~ long,
  data = quakes,
  aspect = "iso",
  panel = function(...) {
    panel.grid(h=-1, v=-1)

    panel.xyplot(...)
  })
```



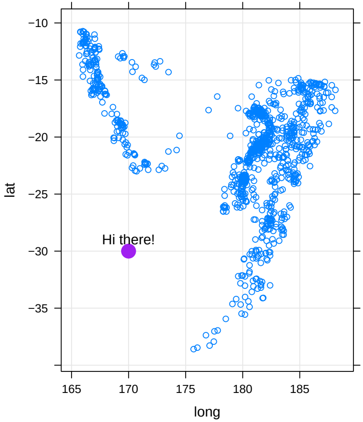
Building it up

```
xyplot(lat ~ long,
  data = quakes,
  aspect = "iso",
  panel = function(...) {
    panel.grid(h=-1, v=-1)
    panel.points(170, -30,
        cex = 2, col = "purple",
        pch = 19)

    panel.xyplot(...)
  })
```



Building it up

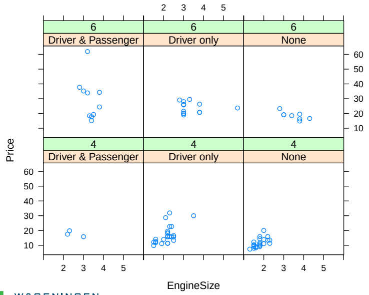
```
xyplot(lat ~ long,
  data = quakes,
  aspect = "iso",
  panel = function(...) {
    panel.grid(h=-1, v=-1)
    panel.points(170, -30,
      cex = 2, col = "purple",
      pch = 19)
    panel.text(170, -30,
      "Hi there!",
      pos = 3)

    panel.xyplot(...)
  })
```



The latticeExtra package

Normal plot:

```
(pcars <-
   xyplot(Price ~ EngineSize |
     AirBags + Cylinders,
     data = Cars93,
     subset = Cylinders %in%
       c(4,6)))
```



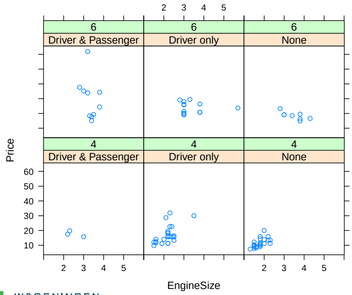
The `latticeExtra` package

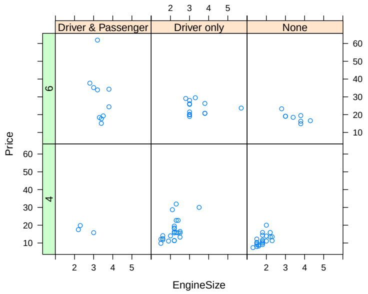
Normal plot:

```
(pcars <-
   xyplot(Price ~ EngineSize |
     AirBags + Cylinders,
     data = Cars93,
     subset = Cylinders %in%
       c(4,6)))
```

Now move one set of labels to the side...

```
(pcars2 <-
  useOuterStrips(pcars))
```

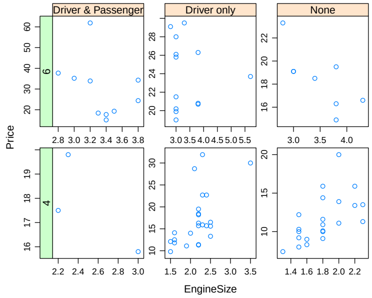




Combine axis ranges...

Every packet its own axes...

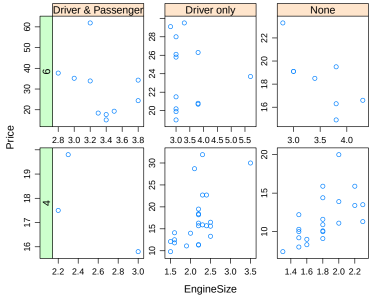
```
(pcars3 <-
   update(pcars2,
          scale = "free"))
```



Combine axis ranges...

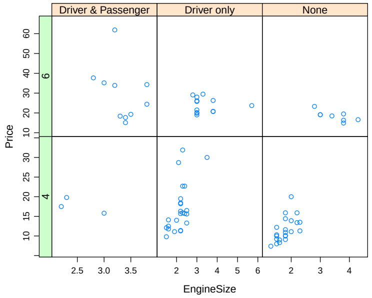
Every packet its own axes...

```
(pcars3 <-
   update(pcars2,
          scale = "free"))
```



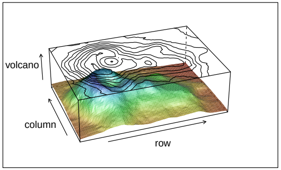
Combined in a meaningful way

```
combineLimits(pcars3)
```



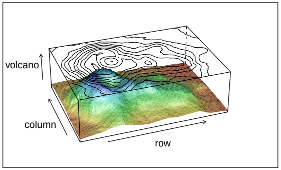
Conclusions

- You can do quite complicated things...



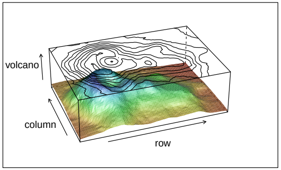
Conclusions

- You can do quite complicated things...
- Resist!



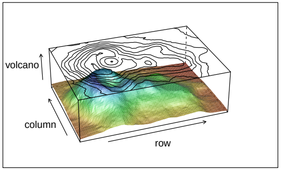
Conclusions

- You can do quite complicated things...
- Resist!
- Only if you must



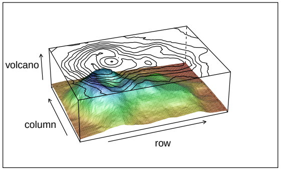
Conclusions

- You can do quite complicated things...
- Resist!
- Only if you must
- Which means: if it makes the plot better

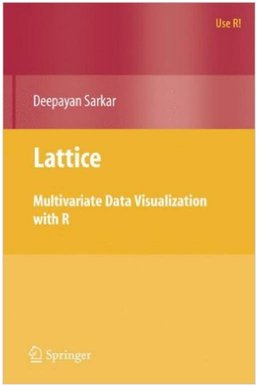


Conclusions

- You can do quite complicated things...
- Resist!
- Only if you must
- Which means: if it makes the plot better
- Not treated:
  - themes
  - grouping (e.g., `panel.superpose` instead of `panel.xyplot`)
  - labels and legends
  - axis coordinates and annotation
  - ...



Acknowledgements


Use R!

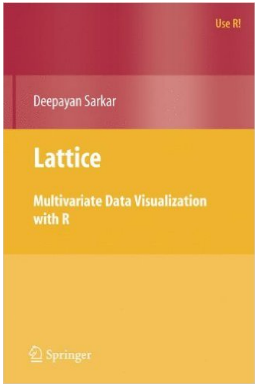
Deepayan Sarkar

Lattice

Multivariate Data Visualization with R

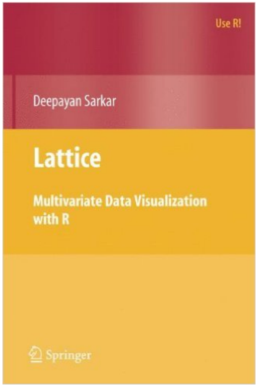
Springer

Acknowledgements



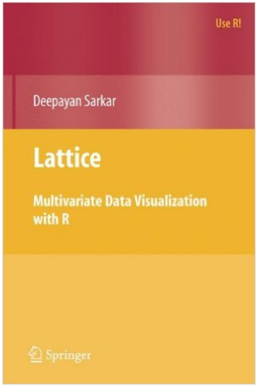
- `lmdvr.r-forge.r-project.org` contains all code to reproduce the book figures

Acknowledgements



- `lmdvr.r-forge.r-project.org` contains all code to reproduce the book figures
- `lattice.r-forge.r-project.org`: vignettes, code, ...

Acknowledgements


Use R!

Deepayan Sarkar

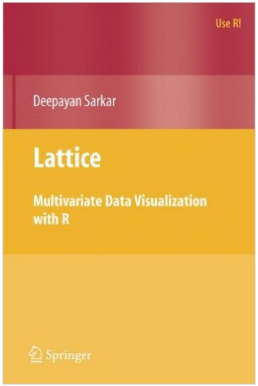
Lattice

Multivariate Data Visualization with R

Springer

- `lmdvr.r-forge.r-project.org` contains all code to reproduce the book figures
- `lattice.r-forge.r-project.org`: vignettes, code, ...

Acknowledgements



- `lmdvr.r-forge.r-project.org` contains all code to reproduce the book figures
- `lattice.r-forge.r-project.org`: vignettes, code, ...

Google