The lattice package

- Trellis graphics for R (originally developed in S)
- Powerful high-level data visualization system
- Provides common statistical graphics with conditioning
  - emphasis on multivariate data
  - sufficient for typical graphics needs
  - flexible enough to handle most nonstandard requirements
- Traditional user interface:
  - collection of high level functions: `xyplot()`, `dotplot()`, etc.
  - interface based on formula and data source
Outline

- Introduction, simple examples
- Overview of features
- A few case studies
Further reading

- The Lattice book (Springer’s UseR! series, March 2008)
- Handouts (including Chapter 2 of the Lattice book)
- Online documentation in the lattice package
## High-level functions in lattice

<table>
<thead>
<tr>
<th>Function</th>
<th>Default Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram()</td>
<td>Histogram</td>
</tr>
<tr>
<td>densityplot()</td>
<td>Kernel Density Plot</td>
</tr>
<tr>
<td>qqmath()</td>
<td>Theoretical Quantile Plot</td>
</tr>
<tr>
<td>qq()</td>
<td>Two-sample Quantile Plot</td>
</tr>
<tr>
<td>stripplot()</td>
<td>Stripchart (Comparative 1-D Scatter Plots)</td>
</tr>
<tr>
<td>bwplot()</td>
<td>Comparative Box-and-Whisker Plots</td>
</tr>
<tr>
<td>barchart()</td>
<td>Bar Plot</td>
</tr>
<tr>
<td>dotplot()</td>
<td>Cleveland Dot Plot</td>
</tr>
<tr>
<td>xyplot()</td>
<td>Scatter Plot</td>
</tr>
<tr>
<td>splom()</td>
<td>Scatter-Plot Matrix</td>
</tr>
<tr>
<td>contourplot()</td>
<td>Contour Plot of Surfaces</td>
</tr>
<tr>
<td>levelplot()</td>
<td>False Color Level Plot of Surfaces</td>
</tr>
<tr>
<td>wireframe()</td>
<td>Three-dimensional Perspective Plot of Surfaces</td>
</tr>
<tr>
<td>cloud()</td>
<td>Three-dimensional Scatter Plot</td>
</tr>
<tr>
<td>parallel()</td>
<td>Parallel Coordinates Plot</td>
</tr>
</tbody>
</table>
The Chem97 dataset

- 1997 A-level Chemistry examination in Britain

```r
> data(Chem97, package = "mlmRev")
> head(Chem97[,c("score", "gender", "gcsescore")])

   score gender gcsescore
1     4    F    6.625
2    10    F    7.625
3    10    F    7.250
4    10    F    7.500
5     8    F    6.444
6    10    F    7.750
```
> `histogram(~ gcsescore, data = Chem97)`

![Histogram of gcsescore](image-url)
> `histogram(~ gcsescore | factor(score), data = Chem97)`

```
> gcsescore
Percent of Total
0
10
20
30
0 2 4 6 8
0 2
0 2 4 6 8
4
6
0 2 4 6 8
8
0
10
20
30
10
```
> densityplot(~ gcsescore | factor(score), Chem97,
plot.points = FALSE,
groups = gender, auto.key = TRUE)
Trellis Philosophy: Part I

- Display specified in terms of
  - Type of display (histogram, densityplot, etc.)
  - Variables with specific roles

- Typical roles for variables
  - Primary variables: used for the main graphical display
  - Conditioning variables: used to divide into subgroups and juxta-poss (multipanel conditioning)
  - Grouping variable: divide into subgroups and superpose

- Primary interface: high-level functions
  - Each function corresponds to a display type
  - Specification of roles depends on display type
    - Usually specified through the formula and the `groups` argument
> qqmath(~ gcsescore | factor(score), Chem97, groups = gender,
  f.value = ppoints(100), auto.key = TRUE,
  type = c("p", "g"), aspect = "xy")
> qq(gender ~ gcse score | factor(score), Chem97,
  f.value = ppoints(100), type = c("p", "g"), aspect = 1)
> bwplot(factor(score) ~ gcsescore | gender, Chem97)
> bwplot(gcsescore ~ gender | factor(score), Chem97, layout = c(6, 1))
> stripplot(depth ~ factor(mag), data = quakes,
  jitter.data = TRUE, alpha = 0.6)
The VADeaths dataset

- Death rates in Virginia, 1941, among different population subgroups

```r
> VADeaths

<table>
<thead>
<tr>
<th>Rural</th>
<th>Male</th>
<th>Rural</th>
<th>Female</th>
<th>Urban</th>
<th>Male</th>
<th>Urban</th>
<th>Female</th>
</tr>
</thead>
<tbody>
<tr>
<td>50-54</td>
<td>11.7</td>
<td>8.7</td>
<td>15.4</td>
<td>8.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55-59</td>
<td>18.1</td>
<td>11.7</td>
<td>24.3</td>
<td>13.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60-64</td>
<td>26.9</td>
<td>20.3</td>
<td>37.0</td>
<td>19.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65-69</td>
<td>41.0</td>
<td>30.9</td>
<td>54.6</td>
<td>35.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70-74</td>
<td>66.0</td>
<td>54.3</td>
<td>71.1</td>
<td>50.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
> barchart(VADeaths, groups = FALSE, layout = c(4, 1))
> dotplot(VADeaths, groups = FALSE, layout = c(4, 1))
> dotplot(VADeaths, type = "o",
          auto.key = list(points = TRUE, lines = TRUE,
                          space = "right"))
> data(Earthquake, package = "nlme")
> xyplot(accel ~ distance, data = Earthquake)
```r
> xyplot(accel ~ distance, data = Earthquake,
scales = list(log = TRUE),
type = c("p", "g", "smooth"))
```
```r
> Depth <- equal.count(quakes$depth, number=8, overlap=.1)
> summary(Depth)

Intervals:

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>39.5</td>
<td>63.5</td>
<td>138</td>
</tr>
<tr>
<td>2</td>
<td>60.5</td>
<td>102.5</td>
<td>138</td>
</tr>
<tr>
<td>3</td>
<td>97.5</td>
<td>175.5</td>
<td>138</td>
</tr>
<tr>
<td>4</td>
<td>161.5</td>
<td>249.5</td>
<td>142</td>
</tr>
<tr>
<td>5</td>
<td>242.5</td>
<td>460.5</td>
<td>138</td>
</tr>
<tr>
<td>6</td>
<td>421.5</td>
<td>543.5</td>
<td>137</td>
</tr>
<tr>
<td>7</td>
<td>537.5</td>
<td>590.5</td>
<td>140</td>
</tr>
<tr>
<td>8</td>
<td>586.5</td>
<td>680.5</td>
<td>137</td>
</tr>
</tbody>
</table>

Overlap between adjacent intervals:

[1] 16 14 19 15 14 15 15
```
> xyplot(lat ~ long | Depth, data = quakes)
> cloud(depth ~ lat * long, data = quakes, 
zlim = rev(range(quakes$depth)),
screen = list(z = 105, x = -70), panel.aspect = 0.75)
> cloud(depth ~ lat * long, data = quakes,
  zlim = rev(range(quakes$depth)),
  screen = list(z = 80, x = -70), panel.aspect = 0.75)
More high-level functions

- More high-level functions in `lattice`
  - Won’t discuss, but examples in manual page
- Other Trellis high-level functions can be defined in other packages, e.g.,
  - `ecdfplot()`, `mapplot()` in the `latticeExtra` package
  - `hexbinplot()` in the `hexbin` package
The "trellis" object model

- One important feature of lattice:
  - High-level functions do not actually plot anything
  - They return an object of class "trellis"
  - Display created when such objects are print()-ed or plot()-ed
- Usually not noticed because of automatic printing rule
- Can be used to arrange multiple plots
- Other uses as well
> dp.uspe <-
    dotplot(t(USPersonalExpenditure),
    groups = FALSE, layout = c(1, 5),
    xlab = "Expenditure (billion dollars)")
> dp.uspe.log <-
    dotplot(t(USPersonalExpenditure),
    groups = FALSE, layout = c(1, 5),
    scales = list(x = list(log = 2)),
    xlab = "Expenditure (billion dollars)")
> plot(dp.uspe, split = c(1, 1, 2, 1))
> plot(dp.uspe.log, split = c(2, 1, 2, 1), newpage = FALSE)
Trellis Philosophy: Part I

- Display specified in terms of
  - Type of display (histogram, densityplot, etc.)
  - Variables with specific roles

- Typical roles for variables
  - Primary variables: used for the main graphical display
  - Conditioning variables: used to divide into subgroups and juxtapose (multipanel conditioning)
  - Grouping variable: divide into subgroups and superpose

- Primary interface: high-level functions
  - Each function corresponds to a display type
  - Specification of roles depends on display type
    - Usually specified through the formula and the `groups` argument
Trellis Philosophy: Part II

- Design goals:
  - Enable effective graphics by encouraging good graphical practice (e.g., Cleveland, 1985)
  - Remove the burden from the user as much as possible by building in good defaults into software

- Some obvious examples:
  - Use as much of the available space as possible
  - Encourage direct comparison by superposition (grouping)
  - Enable comparison when juxtaposing (conditioning):
    - use common axes
    - add common reference objects (such as grids)

- Inevitable departure from traditional R graphics paradigms
Trellis Philosophy: Part III

- Any serious graphics system must also be flexible
- **lattice** tries to balance flexibility and ease of use using the following model:
  - A display is made up of various elements
  - Coordinated defaults provide meaningful results, but
  - Each element can be controlled independently
  - The main elements are:
    - the primary (panel) display
    - axis annotation
    - strip annotation (describing the conditioning process)
    - legends (typically describing the grouping process)
The full system would take too long to describe
Handouts cover the important aspects
Online documentation has details; start with Lattice
We discuss a few advanced ideas using some case studies
Case studies

- Adding regression lines to scatter plots
- Reordering levels of a factor
Example 1: Growth curves

- Heights of boys from Oxford over time
- 26 boys, height measured on 9 occasions

```r
> data(Oxboys, package = "nlme")
> head(Oxboys)

<table>
<thead>
<tr>
<th>Subject</th>
<th>age</th>
<th>height</th>
<th>Occasion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.0000</td>
<td>140.5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>-0.7479</td>
<td>143.4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>-0.4630</td>
<td>144.8</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>-0.1643</td>
<td>147.1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>-0.0027</td>
<td>147.7</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0.2466</td>
<td>150.2</td>
<td>6</td>
</tr>
</tbody>
</table>
```
> xyplot(height ~ age | Subject, data = Oxboys,
  strip = FALSE, aspect = "xy", pch = 16,
  xlab = "Standardized age", ylab = "Height (cm)")
Example 2: Exam scores

- GCSE exam scores on a science subject. Two components:
  - course work
  - written paper

- 1905 students

```r
> data(Gcsemv, package = "mlmRev")
> head(Gcsemv)
```

<table>
<thead>
<tr>
<th>school</th>
<th>student</th>
<th>gender</th>
<th>written</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20920</td>
<td>16</td>
<td>M</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>20920</td>
<td>25</td>
<td>F</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>20920</td>
<td>27</td>
<td>F</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>20920</td>
<td>31</td>
<td>F</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>20920</td>
<td>42</td>
<td>M</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>20920</td>
<td>62</td>
<td>F</td>
<td>36</td>
</tr>
</tbody>
</table>
> xyplot(written ~ course | gender, data = Gcsemv,
  xlab = "Coursework score",
  ylab = "Written exam score")
Adding to a Lattice display

- Traditional R graphics encourages incremental additions
- The Lattice analogue is to write panel functions
A simple panel function

- Things to know:
  - Panel functions are functions (!)
  - They are responsible for graphical content inside panels
  - They get executed once for every panel
  - Every high level function has a default panel function
    e.g., `xyplot()` has default panel function `panel.xyplot()`
A simple panel function

- So, equivalent call:

```r
> xyplot(written ~ course | gender, data = Gcsemv,
  xlab = "Coursework score",
  ylab = "Written exam score",
  panel = panel.xyplot)
```
A simple panel function

- So, equivalent call:

```r
> xyplot(written ~ course | gender, data = Gcsemv,
    xlab = "Coursework score",
    ylab = "Written exam score",
    panel = function(...) {
        panel.xyplot(...)
    })
```
A simple panel function

- So, equivalent call:

```r
> xyplot(written ~ course | gender, data = Gcsemv,
       xlab = "Coursework score",
       ylab = "Written exam score",
       panel = function(x, y, ...) {
           panel.xyplot(x, y, ...)
       })
```
Now, we can add a couple of elements:

```r
> xyplot(written ~ course | gender, data = Gcsemv,
      xlab = "Coursework score",
      ylab = "Written exam score",
      panel = function(x, y, ...) {
          panel.grid(h = -1, v = -1)
          panel.xyplot(x, y, ...)
          panel.loess(x, y, ..., col = "black")
          panel.rug(x = x[is.na(y)],
                     y = y[is.na(x)])
      })
```
Panel functions

Another useful feature: argument passing

```r
> xyplot(written ~ course | gender, data = Gcsemv,
    panel = function(x, y, ...) {
        panel.xyplot(x, y, ...
        type = c("g", "p", "smooth"),
        col.line = "black")
    })
```

is equivalent to

```r
> xyplot(written ~ course | gender, data = Gcsemv,
    type = c("g", "p", "smooth"), col.line = "black")
```
Passing arguments to panel functions

- Requires knowledge of arguments supported by panel function
- Each high-level function has a corresponding default panel function, named as “panel.” followed by the function name. For example,
  - `histogram()` has panel function `panel.histogram`
  - `dotplot()` has panel function `panel.dotplot`
- Most have useful arguments that support common variants
Back to regression lines

- **Oxboys**: model height on age

\[
y_{ij} = \mu + b_i + x_{ij} + x_{ij}^2 + \epsilon_{ij}
\]

- Mixed effect model that can be fit with **lme4**

```r
> library(lme4)
> fm.poly <- lmer(height ~ poly(age, 2) + (1 | Subject),
                   data = Oxboys)
```

- Goal: plot of data with fitted curve superposed
> xyplot(height ~ age | Subject,
  data = Oxboys, strip = FALSE, aspect = "xy",
  type = "p", pch = 16,
  xlab = "Standardized age", ylab = "Height (cm)"
> xyplot(fitted(fm.poly) ~ age | Subject,
   data = Oxboys, strip = FALSE, aspect = "xy",
   type = "l", lwd = 2,
   xlab = "Standardized age", ylab = "Height (cm)"
> xyplot(height + fitted(fm.poly) ~ age | Subject, 
    data = Oxboys, strip = FALSE, aspect = "xy", pch = 16, 
    lwd = 2, type = c("p", "l"), distribute.type = TRUE, 
    xlab = "Standardized age", ylab = "Height (cm)"
)
GCSE exam scores

- \textbf{Gcsemv}: model written score by coursework and gender
- A similar approach does not work as well
  - \( x \) values are not ordered
  - missing values are omitted from fitted model
> fm <- lm(written ~ course + I(course^2) + gender, Gcsemv)
> xyplot(written + fitted(fm) ~ course | gender,
    data = subset(Gcsemv, !(is.na(written) | is.na(course))),
    type = c("p", "l"), distribute.type = TRUE)
• Built-in solution: Simple Linear Regression in each panel

```r
> xyplot(written ~ course | gender, Gcsemv,
  type = c("p", "r"), col.line = "black")
```
GCSE exam scores

- More complex models need a little more work
- Consider three models:
  
  ```r
  > fm0 <- lm(written ~ course, Gcsemv)
  > fm1 <- lm(written ~ course + gender, Gcsemv)
  > fm2 <- lm(written ~ course * gender, Gcsemv)
  ```
- Goal: compare fm2 and fm1 with fm0
**Solution:** evaluate fits separately and combine

```r
> course.rng <- range(Gcsemv$course, finite = TRUE)
> grid <-
    expand.grid(course = do.breaks(course.rng, 30),
                 gender = unique(Gcsemv$gender))
> fm0.pred <-
    cbind(grid,
          written = predict(fm0, newdata = grid))
> fm1.pred <-
    cbind(grid,
          written = predict(fm1, newdata = grid))
> fm2.pred <-
    cbind(grid,
          written = predict(fm2, newdata = grid))
> orig <- Gcsemv[c("course", "gender", "written")]
```
> str(orig)

'data.frame': 1905 obs. of 3 variables:
  $ course : num NA 71.2 76.8 87.9 44.4 NA 89.8 17.5 32.4 84.2 ...
  $ gender : Factor w/ 2 levels "F","M": 2 1 1 1 2 1 1 2 2 1 ...
  $ written: num 23 NA 39 36 16 36 49 25 NA 48 ...

> str(fm0.pred)

'data.frame': 62 obs. of 3 variables:
  $ course : num 9.25 12.28 15.30 18.32 21.35 ...
  $ gender : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
  $ written: num 21.6 22.7 23.9 25.1 26.3 ...
```r
> combined <-
  make.groups(original = orig,
              fm0  = fm0.pred,
              fm2  = fm2.pred)

> str(combined)

'data.frame': 2029 obs. of  4 variables:
$ course : num  NA  71.2  76.8  87.9  44.4 NA  89.8 17.5 32.4 84.2 ...
$ gender : Factor w/ 2 levels "F","M": 2 1 1 1 2 1 1 2 2 1 ...
$ written: num  23 NA  39  36  16  36  49 25 NA 48 ...
$ which : Factor w/ 3 levels "original","fm0",...
```
> xyplot(written ~ course | gender,
  data = combined, groups = which,
  type = c("p", "l", "l"), distribute.type = TRUE)
Reordering factor levels

- Levels of categorical variables often have no intrinsic order
- The default in `factor()` is to use `sort(unique(x))`
  - Implies alphabetical order for factors converted from character
- Usually irrelevant in analyses
- Can strongly affect impact in a graphical display
Example

- Population density in US states in 1975

```r
> state <- data.frame(name = state.name,
                          region = state.region,
                          state.x77)
> state$Density <- with(state, Population / Area)
> dotplot(name ~ Density, state)
> dotplot(name ~ Density, state,
            scales = list(x = list(log = TRUE)))
```
The reorder() function

\[
\text{dotplot}\left(\text{reorder}(\text{name}, \text{Density}) \sim \text{Density}, \text{state}\right)
\]

\[
\text{dotplot}\left(\text{reorder}(\text{name}, \text{Density}) \sim \text{Density}, \text{state}, \text{scales} = \text{list}(x = \text{list}(\text{log} = \text{TRUE}))\right)
\]

- Reorders levels of a factor by another variable
- Optional summary function, default `mean()`
Reordering by multiple variables

- Not directly supported, but...
- Order is preserved within ties

```r
> state$region <- with(state, reorder(region, Frost, median))
> state$name <- with(state, reorder(reorder(name, Frost), as.numeric(region)))

> p <-
    dotplot(name ~ Frost | region, state,
            strip = FALSE, strip.left = TRUE, layout = c(1, 4),
            scales = list(y = list(relation = "free", rot = 0)))

> plot(p,
      panel.height = list(x = table(state$region),
                           units = "null"))
```
Ordering panels using index.cond

- Order panels by some summary of panel data
- Example: death rates due to cancer in US counties, 2001-2003

```
data(USCancerRates, package = "latticeExtra")
xyplot(rate.male ~ rate.female | state, USCancerRates,
       index.cond = function(x, y, ...) {
         median(y - x, na.rm = TRUE)
       },
       aspect = "iso",
       panel = function(...) {
         panel.grid(h = -1, y = -1)
         panel.abline(0, 1)
         panel.xyplot(...)
       },
       pch = ".")
```
Take home message

- Panel functions provide finest level of control
- Built-in panel functions are also powerful
  - Easily taken advantage of using argument passing
  - Requires knowledge of arguments (read documentation!)
  - Special function `panel.superpose()` useful for grouping
- Several useful functions make life a little simpler
  - `reorder()`, `make.groups()`, etc.