Data Visualization in R
3. Grid & lattice graphics

Michael Friendly
SCS Short Course
Sep/Oct, 2018

http://www.datavis.ca/courses/RGraphics/
Overview

- Overview of grid-based graphics
  - grid: low-level graphics functions
  - lattice, vcd, ggplot2: high-level functions

The grid graphics system for R provides an alternative and more powerful way to develop data graphics in R.

The lattice package provides functions for drawing all standard plots, plus:
- more pleasing defaults
- create and modify graphic “themes”
- collections ("small multiples") of simpler graphs from subsets of the data.
Lattice, son of Trellis graphics

- Complex multivariate data can often be better visualized by conditioning & grouping
  - show how some relationship changes over other variables
  - Tufte: “small multiples”: separate panels, arranged for visual comparison
  - Cleveland et al.: Trellis graphs for S+, ~ 1980
  - Deepayan Sarkar: lattice package, ~ 2000
Lattice ideas in a nutshell

• All plots can be described by plot formulas
  - \(~ y\) Some univariate plot (boxplot, histogram, boxplot, ...)
  - \(~ y \mid A\) Univariate, separate panels for levels of factor A
  - \(~ y \mid z\) Univariate, cutting z into discrete ranges
  - \(y \sim x\) Bivariate
  - \(y \sim x \mid A\) Bivariate, separate panels for levels of A
  - \(y \sim x \mid A + B\) multiple conditioning variables
  - \(y_1 + y_2 \sim x_1 + x_2\) multiple Y and X variables

• Conditioning variables define “panels” in a plot
  - These can be laid out on a “page” in various ways
  - panel functions get the data for a subset and “render” (plot) it
  - High-level functions handle panel layout, and call panel functions

• Customize
  - graphic “themes” generalize par() settings
  - Combine multiple panel functions, write new ones.
These are the high-level plot functions in lattice

These schematic examples have all been “rendered” using the default lattice theme settings

From: Murrell, *R Graphics*, Fig. 4.3
Lattice plots have analogs in traditional graphics. All use formula-style arguments for what to plot: y ~ x, or conditioning: y ~ x|A.

<table>
<thead>
<tr>
<th>lattice function</th>
<th>description</th>
<th>formula examples</th>
<th>base analog</th>
</tr>
</thead>
<tbody>
<tr>
<td>barchart()</td>
<td>bar chart</td>
<td>x<del>A or A</del>x</td>
<td>barplot()</td>
</tr>
<tr>
<td>bwplot()</td>
<td>boxplot</td>
<td>x<del>A or A</del>x</td>
<td>boxplot()</td>
</tr>
<tr>
<td>densityplot()</td>
<td>kernal density plot</td>
<td>~x</td>
<td>A*B</td>
</tr>
<tr>
<td>dotplot()</td>
<td>dotplot</td>
<td>~x</td>
<td>A</td>
</tr>
<tr>
<td>histogram()</td>
<td>histogram</td>
<td>~x</td>
<td>hist()</td>
</tr>
<tr>
<td>stripplot()</td>
<td>strip plots</td>
<td>A<del>x or x</del>A</td>
<td>stripchart()</td>
</tr>
<tr>
<td>xyplot()</td>
<td>scatterplot</td>
<td>y~x</td>
<td>A</td>
</tr>
<tr>
<td>contourplot()</td>
<td>3D contour plot</td>
<td>z~x*y</td>
<td>contour()</td>
</tr>
<tr>
<td>cloud()</td>
<td>3D scatterplot</td>
<td>z~x*y</td>
<td>A</td>
</tr>
<tr>
<td>levelplot()</td>
<td>3D level plot</td>
<td>z~y</td>
<td>x</td>
</tr>
<tr>
<td>parallel()</td>
<td>parallel coordinates plot</td>
<td>data frame</td>
<td>NA</td>
</tr>
<tr>
<td>splom()</td>
<td>scatterplot matrix</td>
<td>data frame</td>
<td>pairs()</td>
</tr>
<tr>
<td>wireframe()</td>
<td>3D surface graph</td>
<td>z~y</td>
<td>x</td>
</tr>
</tbody>
</table>
Lattice plots: formulas, conditioning & grouping

For 1D plots, the formula argument, $\sim y$, specifies the variable to be plotted

- Conditioning: $\sim y | \text{group}$ gives multipanel plots for the levels of the group factor
- Grouping: $\sim y, \text{group} = \text{superposes}$ plots for the levels of group

```r
densityplot(~mpg, 
data=mtcars, ...)
densityplot(~mpg | cyl, 
data=mtcars, ...)
densityplot(~mpg, groups=cyl, 
data=mtcars, ...)
```
Ethanol data: Ethanol fuel was burned in a single-cylinder engine.
How do emissions of nitrous oxide ($\text{NO}_x$) depend on
• engine compression ratio (C) and
• equivalence ratio (EE), a measure of richness of the air and ethanol fuel mixture

\texttt{xyplot()} for lattice scatterplots:

\texttt{xyplot(NOx \sim C \mid EE, data = ethanol, ...)}

Same plot, with aspect="xy": sets aspect ratio to “bank to 45°”
As in base graphics, some computation is often required to make a simpler or better version of some plot.

- 2D plots of the ethanol data suggest something that might better be seen in 3D
- This requires calculating a fitted response surface, and drawing it
- It doesn’t show the data, and uses a non-parametric smoother, not a lm() model

```r
require(stats)
with(ethanol, {
  eth.lo <- loess(NOx ~ C * E, span = 1/3, parametric = "C",
                  drop.square = "C", family="symmetric")
  eth.marginal <- list(C = seq(min(C), max(C), length.out = 25),
                        E = seq(min(E), max(E), length.out = 25))
  eth.grid <- expand.grid(eth.marginal)
  eth.fit <- predict(eth.lo, eth.grid)
  wireframe(eth.fit ~ eth.grid$C * eth.grid$E,
            shade=TRUE,
            screen = list(z = 40, x = -60, y=0),
            distance = .1,
            xlab = "C", ylab = "E", zlab = "NOx")
})
```

This example is complex. It uses:

- loess() to calculate smoothed values of NOx
- predict() to evaluate these over ranges of C & E
- wireframe() to plot these with nice shading

If this plot is believed, it gives a much simpler description of dependence, NOx ~ C * E
Detour: Modeling what we see

- Graphs of the ethanol data suggest a systematic, but complex relationship between NOx ~ C + E
  - Traditional parametric linear models handle this very semi-well
  - E.g., try a model with terms in C, E, \( E^2 \) and interactions

```
> eth.mod2 <- lm(NOx ~ (C + poly(E,2))^2, data=ethanol)
> Anova(eth.mod2)
Anova Table (Type II tests)

Response: NOx

<table>
<thead>
<tr>
<th>Sum Sq</th>
<th>Df</th>
<th>F value</th>
<th>Pr(&gt;F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.032</td>
<td>1</td>
<td>25.2282</td>
<td>2.925e-06 ***</td>
</tr>
<tr>
<td>91.838</td>
<td>2</td>
<td>230.2103</td>
<td>&lt; 2.2e-16 ***</td>
</tr>
<tr>
<td>3.322</td>
<td>2</td>
<td>8.3271</td>
<td>0.0005101 ***</td>
</tr>
<tr>
<td>16.356</td>
<td>82</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```
Detour: Modeling what we see

- The R model formula, $\text{NOx} \sim (C + \text{poly}(E, 2))^2$ is a short-hand notation.
- The expanded version is nearly
  $$\text{NOx} \sim C + E + E^2 + C:E + C:E^2$$
- Interpretation:
  - $C + E$: overall linear effects (slopes) of $C$ & $E$ on NOx
  - $E^2$: quadratic effect (curvature) of equivalence ratio on NOx
  - $C:E$: does the slope for $E$ change linearly with $C$?
  - $C:E^2$: does the curvature for $E$ change linearly with $C$?
Detour: Modeling what we see

summary() for a given model gives significance tests of model terms

> summary(eth.mod2)

Call:
  lm(formula = NOx ~ (C + poly(E, 2))^2, data = ethanol)

Residuals:
   Min     1Q   Median     3Q    Max
-0.84489 -0.37039 -0.00367  0.39327  0.76796

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.16206    0.16009   7.259 2.01e-10 ***
   C          0.06572    0.01265   5.193 1.48e-06 ***
poly(E, 2)1  4.81844    1.56979   3.069  0.002907 **
poly(E, 2)2 -12.15328    1.61916  -7.506  6.60e-11 ***
C:poly(E, 2)1 -0.46307    0.11615  -3.987  0.000145 ***
C:poly(E, 2)2  0.15492    0.11720   1.322  0.189909

---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4466 on 82 degrees of freedom
Multiple R-squared: 0.8535,  Adjusted R-squared: 0.8445
F-statistic: 95.52 on 5 and 82 DF,  p-value: < 2.2e-16
Effect plots: Seeing what we model

In many cases, effect plots help to visualize a fitted model. These all use lattice graphics to render the plot.

```r
plot(Effect("C", eth.mod2))
plot(Effect("E", eth.mod2))
```
Effect plots: Seeing what we model

The strength of lattice graphics for conditioning is used in multipanel effect plots. Details of the layout and conditioning levels can all be controlled by options.

```r
plot(allEffects(eth.mod2), layout=c(6,1), xlab="Equivalence ratio")
```
Detour: gam

- Generalized additive models (gam) are like generalized linear models (glm), but allow non-parametric “smoothed” s() terms
  - degree of smoothing ~ # degrees of freedom
  - models can have linear & smoothed s() terms
  - approx. significance tests are available for smooth terms

```r
> library(mgcv)
> eth.gam1 <- gam(NOx ~ C + s(E), data=ethanol)
> summary(eth.gam1)

Parametric coefficients:
  Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.291342   0.088898  14.526  < 2e-16 ***
C           0.055345   0.007062   7.837 1.88e-11 ***
---
Approximate significance of smooth terms:

  edf Ref.df     F p-value
s(E) 7.553  8.469 208.8  <2e-16 ***
---
R-sq.(adj) =  0.953   Deviance explained = 95.8%
GCV = 0.067206   Scale est. = 0.05991   n = 88
```

This is sometimes called “semi-parametric regression”

The edf for the smoothed term is found using cross-validation

There are other kinds of smoothing models
Plotting “gam” objects

The mgcv package contains a number of plot methods for “gam” objects

```r
plot(eth.gam1, shade=TRUE, shade.col="pink", all.terms=TRUE, residuals=TRUE, pages=1)
```
Plotting “gam” objects

vis.gam() is like a 3D version of an effect plot. It shows the fitted values for two predictors, holding constant all others.

vis.gam(eth.gam1, color="topo", theta=70, phi=30)
vis.gam(eth.gam1, color="topo", plot.type="contour")

NB: The result is similar to what we got using loess(). However, this is a full-fledged statistical model, so we can find confidence intervals, prediction intervals, etc.
Lattice panel functions

- Lattice plots use **panel functions** to add info to a plot
  - `panel.grid()` – grid lines
  - `panel.xyplot(x, y, type=, ...)` – various types of (x, y) plots
  - `panel.lmline()` – add regression line
  - `panel.loess()` – smoothed loess curve
  - many others ...

```r
EE <- equal.count(ethanol$E, number=9, overlap=1/4)
xyplot(NOx ~ C | EE, data = ethanol, 
  prepanel = function(x, y) prepanel.loess(x, y, span = 1), 
  xlab = "Compression ratio", ylab = "NOx (micrograms/J)",
  panel = function(x, y) {
    panel.grid(h=-1, v= 2)
    panel.xyplot(x, y)
    panel.loess(x,y, span=1)
  },
  aspect = "xy")
```
splom() draws a scatterplot matrix. As with other lattice functions, a type= argument can be used to invoke several panel functions.

```r
splom(ethanol,
     type=c("p", "r", "smooth"),
     col.line = "red",
     pch=16, lwd=3,
     main="Ethanol data")
```
The Trellis approach allows creating effective graphs with a consistent look and feel. It uses “themes” to define colour, size and other features of components of a graph.

A theme consists of settings for the attributes of various graphical elements.

The current settings are displayed with `show.settings()`.

This differs from base graphics, where `par()` settings are used inconsistently across different graph types.
Lattice themes and settings

- Get theme settings with `trellis.par.get()`
- Set new ones with `trellis.par.set()`

```r
> my.theme <- trellis.par.get()
> names(my.theme)
[1] "grid.pars"         "fontsize"          "background"        "panel.background"
[5] "clip"              "add.line"          "add.text"          "plot.polygons"
[9] "box.dot"           "box.rectangle"    "box.umbrella"      "dot.line"
[13] "dot.symbol"        "plot.line"         "plot.symbol"       "reference.line"
[17] "strip.background"  "strip.shingle"     "strip.border"      "superpose.line"
[21] "superpose.symbol"  "superpose.polygon" "regions"           "shade.colors"
[25] "axis.line"         "axis.text"         "axis.components"   "layout.heights"
[29] "layout.widths"     "box.3d"            "par.xlab.text"     "par.ylab.text"
[33] "par.zlab.text"     "par.main.text"    "par.sub.text"
```

There are 35 different attributes, each of which is a list of more basic settings

```r
> names(my.theme$plot.symbol)
[1] "alpha" "cex"  "col"  "font"  "pch"  "fill"
> names(my.theme$plot.line)
[1] "alpha" "col"  "lty"  "lwd"
```
I like to use filled point symbols (pch=16) and make lines thicker

```r
my.theme$plot.line$lwd <- 2
my.theme$plot.symbol$pch <- 16
my.theme$superpose.symbol$pch <- rep(16, 7)

#establish my.theme
trellis.par.set(my.theme)
show.settings()
```

points are now filled circles & lines are thicker

NB: This is tedious, but useful if you are writing a paper or a book. Do it ONCE, for all figures!
Lattice themes: color to BW

Lattice plots are “trellis” objects. They can be printed with different themes w/o changing your code

```r
plt <- barchart(Class ~ Freq | Sex + Age, 
                 data = as.data.frame(Titanic), 
                 groups = Survived, stack = TRUE, 
                 layout = c(4, 1), 
                 auto.key = list(title = "Survived", columns = 2), 
                 scales = list(x = "free"))
print(plt)

trellis.device(color = FALSE)
print(plt)
```

As this example demonstrates, lattice themes are generally well-designed to handle color vs. B/W
Boxplots show some aspects of the shape of distributions: median, IQR, outliers, ...

**Violin plots** use a mirrored kernel density plot instead

```
bwplot(Ann ~ cut(Lat, pretty(Lat, 20)),
       data=nasa, subset=(abs(Lat)<60),
       xlab='Latitude', ylab='Solar radiation G(0) (kWh/m²)')
```

For lattice, this is just a boxplot using a different panel function: `panel.violin()`

```
bwplot(Ann ~ cut(Lat, pretty(Lat, 20)),
       data=nasa, subset=(abs(Lat)<60),
       xlab='Latitude', ylab='Solar radiation G(0) (kWh/m²)',
       panel = panel.violin)
```

You can combine these using a custom panel function that calls both

```r
my.panel <- function(..., box.ratio) {
  panel.violin(..., col = "lightblue",
    varwidth = FALSE, box.ratio = box.ratio)
  panel.bwplot(..., col='black',
    cex=0.9, pch='|', fill='red', box.ratio = .25)
}
```

Use it:

```r
bwplot(Ann ~ cut(Lat, pretty(Lat, 40)),
  data=nasa, subset=(abs(Lat)<60),
  xlab='Latitude', ylab='Solar radiation G(0) (kWh/m²)',
  horizontal=FALSE,
  panel = my.panel,
  par.settings = list(box.rectangle=list(col='black'),
    plot.symbol = list(pch='.', cex = 0.1)),
  scales=list(x=list(rot=45, cex=0.5))
}
```

Notes:
cut(): breaks a quantitative variable to a factor
subset: use only -60 < Lat < 60
par.settings: set some plot attributes
scales: tweak labeling of x axis, rotating labels
How does solar radiation vary with latitude, over months of the year?
• The result of this plot suggests some sort of scientific explanation
• Models to confirm/reject any of these would have to take the distributions into account

How was this graph produced?
• What was the plot formula?
• What was the panel function?
• What plot attributes were modified?
Data munging for plots & models

- Very often, the difficult problems in data analysis and graphics concern:
  - How to get my data into a format required for analysis?
  - How to get my data into a format for plotting?
  - How to get my model results into a table or plot?
- The first step is to understand the structure of your data

```r
> str(nasa)
'data.frame': 64800 obs. of 15 variables:
$ Lat: int -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 ...
$ Lon: int -180 -179 -178 -177 -176 -175 -174 -173 -172 -171 ...
$ Feb: num 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 ...
$ Mar: num 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 ...
$ Apr: num 0 0 0 0 0 0 0 0 0 0 ...
$ May: num 0 0 0 0 0 0 0 0 0 0 ...
$ Jun: num 0 0 0 0 0 0 0 0 0 0 ...
$ Jul: num 0 0 0 0 0 0 0 0 0 0 ...
$ Aug: num 0 0 0 0 0 0 0 0 0 0 ...
$ Sep: num 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
$ Nov: num 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 ...
$ Dec: num 11 11 11 11 11 11 11 11 11 11 ...
```

Previous plots used the annual average (Ann) against Latitude (Lat), with a plot formula:

```
Ann ~ cut(Lat, pretty(Lat, 40))
```

But now, we want to plot monthly values, Jan:Dec
The solution used here works, but it is opaque, in that it tries to coerce the data into what is required for plot formulas for lattice

```r
> (x <- paste(names(nasa)[3:14], collapse='+'))
[1] "Jan+Feb+Mar+Apr+May+Jun+Jul+Aug+Sep+Oct+Nov+Dec"
> (formula <- as.formula(paste(x, '~cut(Lat, pretty(Lat, 20))', sep='')))  
Jan + Feb + Mar + Apr + May + Jun + Jul + Aug + Sep + Oct + Nov + Dec ~ cut(Lat, pretty(Lat, 20))
```

With this, the monthly plot can be produced by:

```r
bwplot(formula, data=nasa, subset=(abs(Lat)<60),
xlab='Latitude', ylab='G(0) (kWh/m²)',
outer=TRUE, as.table=TRUE, horizontal=FALSE,
col='lightblue',
panel=panel.violin,
scales=list(x=list(rot=70, cex=0.5))
```