# post01-caroline-wu

Caroline Wu
10/22/2017

# More Data Visualization Methods in R

## I. Introduction

As we learned the basics of the CRAN package *ggplot*, we see that the R provides a platform for data visualization and customs the graphics with different emphasis to analyze a data set in various perspectives. However, no system is perfect for all. This post will introduce another high-level graphics system called *lattice* that specializes in multivariate data sets. Since the appraoch of those two packages are rather different, I will explain some sample graphics and codes so that the readers will get a general sense of why choosing the *lattice* package and how it simplifies higher level plotting tasks.

## II. Background

This package is first written by Deepayan Sarkar to implement *Trellis* graphs, which displays a variable or the relationship between variables, conditioned on one or more other variables. When using a single-panel Trellis graph, it may seem similar to traditional R graphs but it allows more possibilities in further data analysis that only extracts some of the variables to evaluate.

## III. Available Displays

histogram() **Histogram**
densityplot() **Kernel Density Plot**
qqmath() **Theoretical Quantile Plot**
qq() **Two-sample Quantile Plot**
stripplot() **Stripchart (Comparative 1-D Scatterplots)**
bwplot() **Comparative Box-and-Whisker Plots**
dotplot() **Cleveland Dot Plot**
barchart() **Bar Plot**
xyplot() **Scatterplot**
splom() **Scatterplot Matrix**
contourplot() **Contour Plot of Surfaces**
levelplot() **False Color Level Plot of Surfaces**
wireframe() **Three-dimensional Perspective Plot of Surfaces**
cloud() **Three-dimensional Scatterplot**
parallel() **Parallel Coordinates Plot**

## IV. Some familiar plots

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: lme4
```

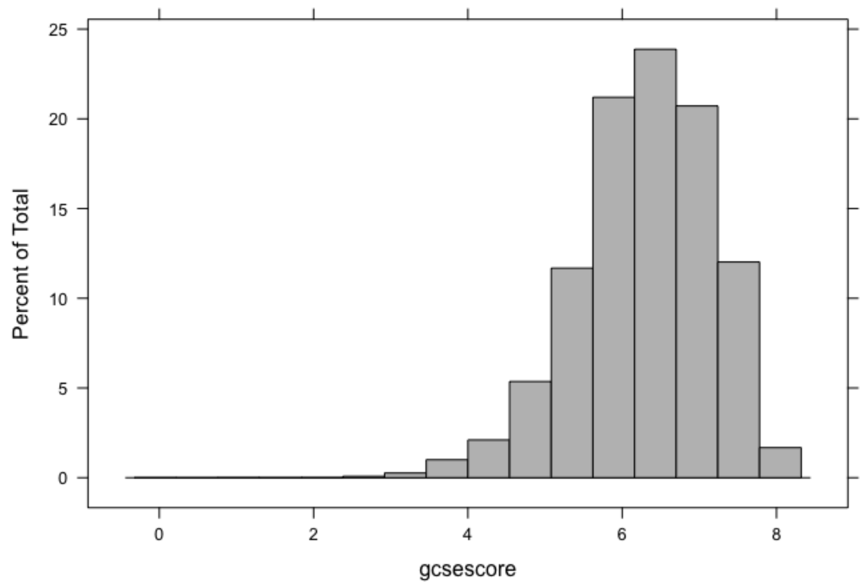
For all the examples, I will use the data set that records information on students appearing in the 1997 A-level chemistry examination in Britain

## * Histograms

Overview of all gcsescores(average score in GCSE exams, which is a continuous score that may be used as a predictor of the A-level score.)

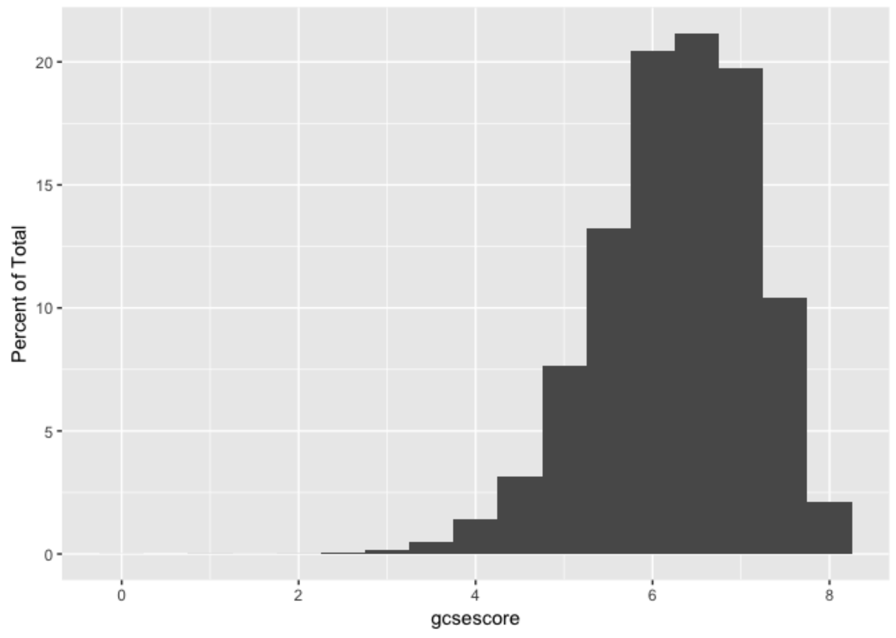
**With *lattice***

```
histogram(~ gcsescore, data = Chem97, col = "grey")
```



**With *ggplot***

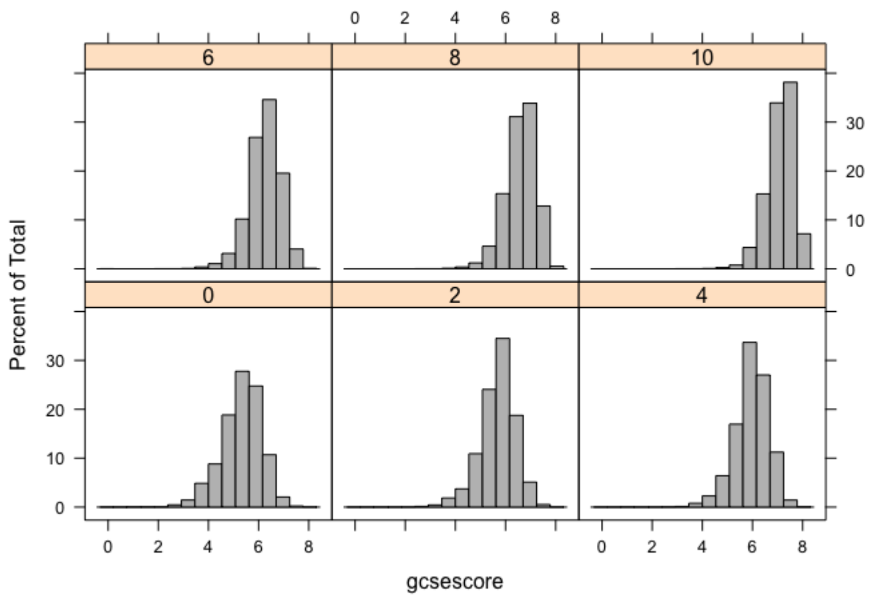
```
ggplot(Chem97, aes(gcsescore))+
  geom_histogram(binwidth = 0.5,aes(y = 100*(..count..)/sum(..count..)))+
  ylab("Percent of Total")
```



As I mentioned before, the single panel graphics between two packages do not have many variations. Now, let us add some conditioning to the graphs to explore the relationship between variables. Since gcsescore is oftern used as a predictor for A-Level scores, I want to condition the data by different scores in A-level (on scale from 0 to 10) to see the gcsescore distribution within each scores.

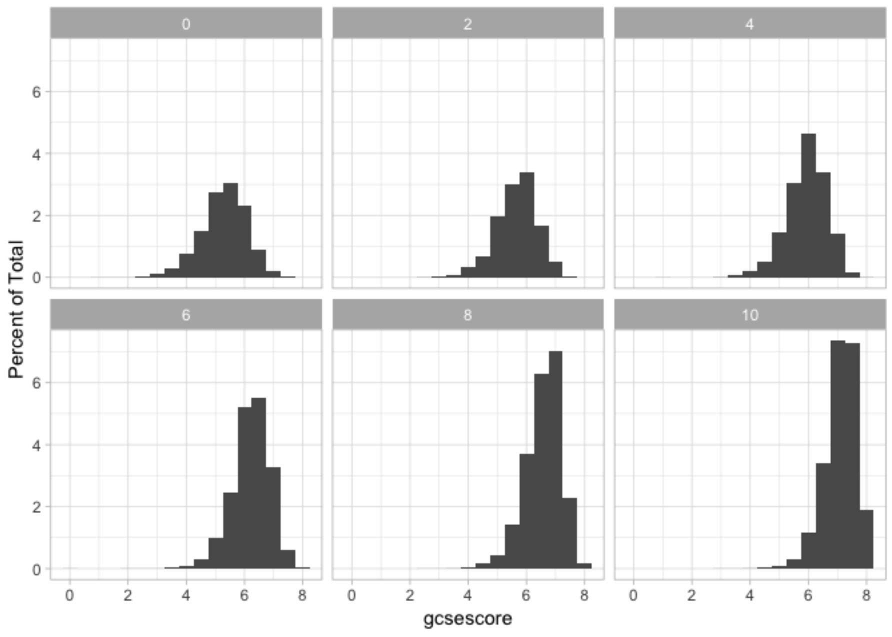
**With *lattice***

```
#syntax for conditioning: conditioning symbol | specifies the primary variable
histogram(~ gcsescore | factor(score), data = Chem97, col = "grey")
```



**With *ggplot***

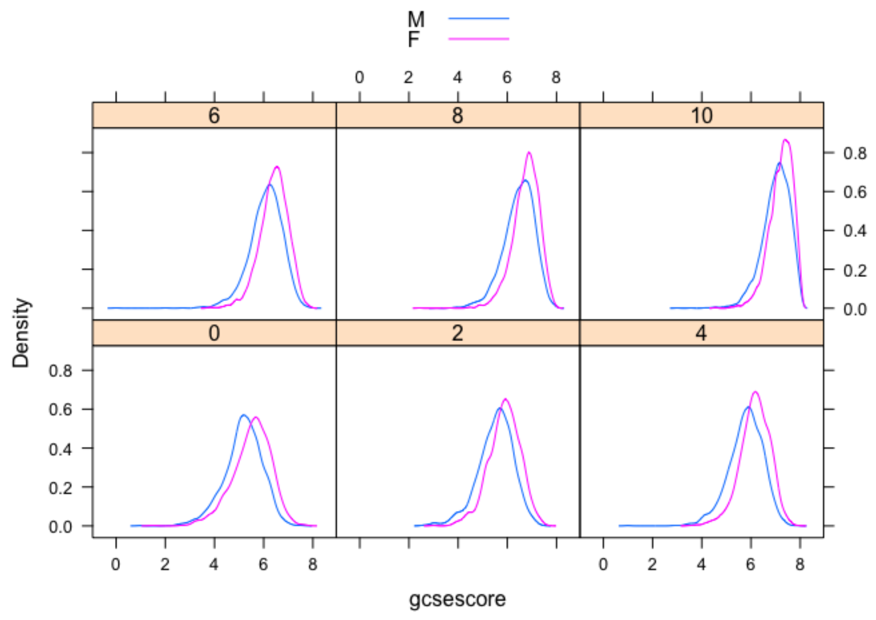
```
ggplot(Chem97, aes(gcsescore))+
  geom_histogram(binwidth = 0.5,aes(y = 100*(..count..)/sum(..count..)))+
  ylab("Percent of Total")+
  facet_wrap(~ score)+
  theme_light()
```



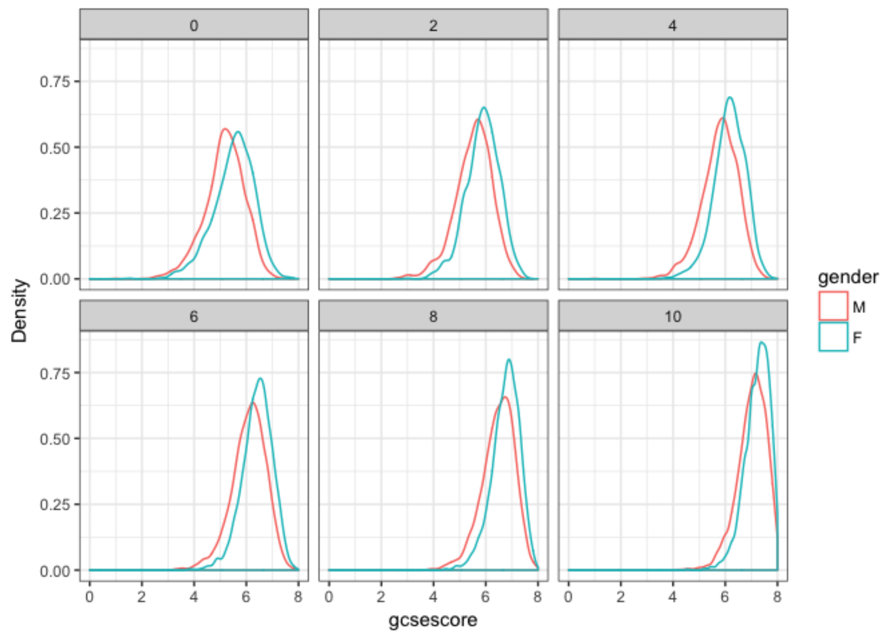
Comparing the graphs of two methods, they are quite similar. Nevertheless, the *lattice* package requires significantly less user inputs that *ggplot* to add parameters in order to ahieve the same purpose.

## * Density Plots and superposition

```
densityplot(~ gcsescore | factor(score), Chem97, groups = gender, plot.points = FALSE, auto.key = TRUE)
```



```
ggplot(Chem97, aes(gcsescore))+
  geom_density(kernel = "gaussian", aes(colour = gender))+
  ylab("Density")+
  facet_wrap(~ score)+
  theme_bw()
```



As one can see, those rather simple plotting tasks can be fulfilled by both packages with reasonable amount of codes. However, when one needs to combine statistical anlysis to data visualization, *lattice* can fulfill this requirement with more efficiency. In the next section, I will explain some ways of how *lattice* deal with more complex data visualization needs, including univariate distributions, multiway tables, scatterplots and extensions, and trivariate displays.

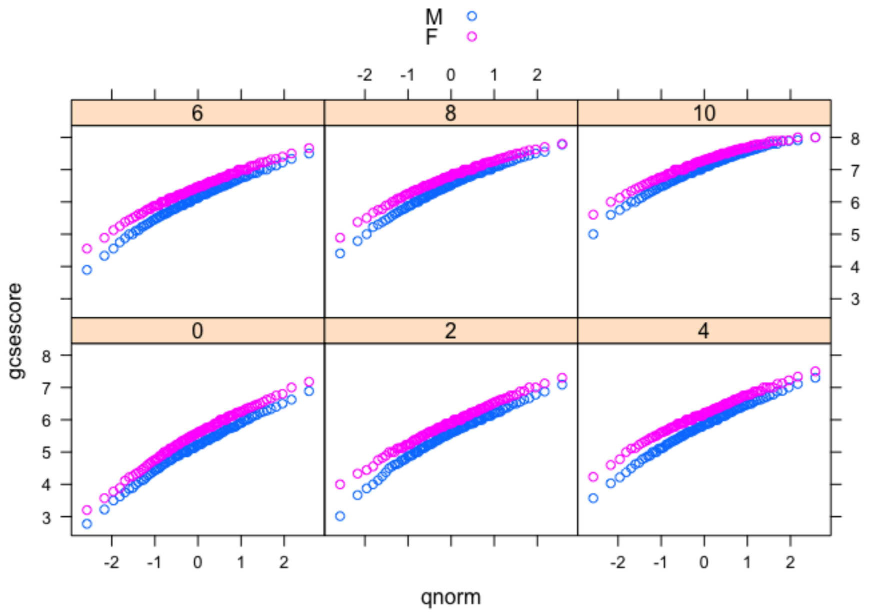
## V. Univariate distributions

### * Q-Q plots

An important statistical analysis method is to compare a data set to a known distribution to see its fit and determine whether that distribution is a good predictor for the desired data set. Q-Q plots in *lattice* are expecially useful for univariate continuous data that it compares the quantiles of the observed data against similar quantiles of a probability. This comparison is effectice since human eye can easily distinguish between a curved line (bad fit) and a straight line (good fit).

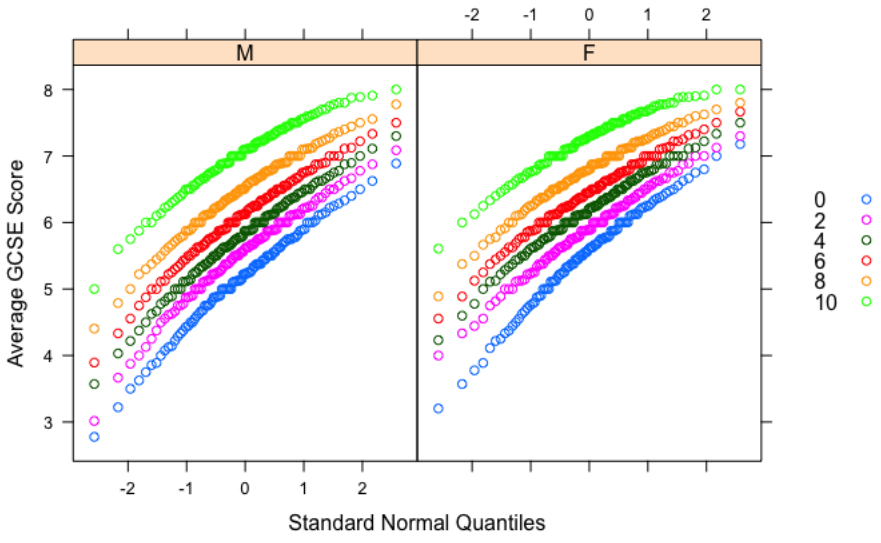
To be more specific, I will continue to use the data set that records information on students appearing in the 1997 A-level chemistry examination in Britain, conditioned on different A-Level scores.

```
qqmath(~ gcsescore | factor(score), data = Chem97, groups = gender, auto.key = TRUE,
f.value #An optional numeric vector of probabilities, quantiles corresponding to which should be plotted wi
= ppoints(100)#this is an integer indicating the number of quantiles plotted in each panel
)
```



As one can see, the graphs are slightly convex, which indicates that the data set is left-skewed. The lines for females in each scoring category are slightly above that of males. This means that within each scoring categoris, women has a slightly higher gcsescores than men. If we look at each panel more closely, we can see that as the A-level score increases, the slope of the line decreases. We can conclude from this graph that the higher A-Level score is associated with higher gcsescore. However, this decrease in slope is not very easy to observe. Instead, we can plot different scores on the same plot.

```
qqmath(~ gcsescore | gender, Chem97, groups = score, aspect = "xy",
f.value = ppoints(100), auto.key = list(space = "right"),
xlab = "Standard Normal Quantiles",
ylab = "Average GCSE Score")
```



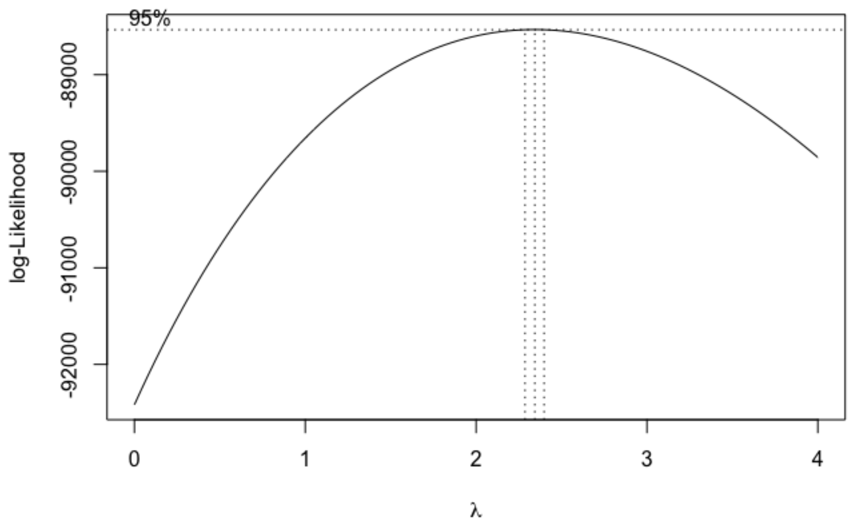
The various properties and theory of normal distributions make it a powerful model in statistical analysis. Natually, many data sets are not normal. On way to tranform those data sets so that it could be approxiametly normal is through the Box–Cox transformation, which is the scale- and location-shifted version of the power transformation.

```
#First we need to compute the optimal lambda
library("MASS")
```

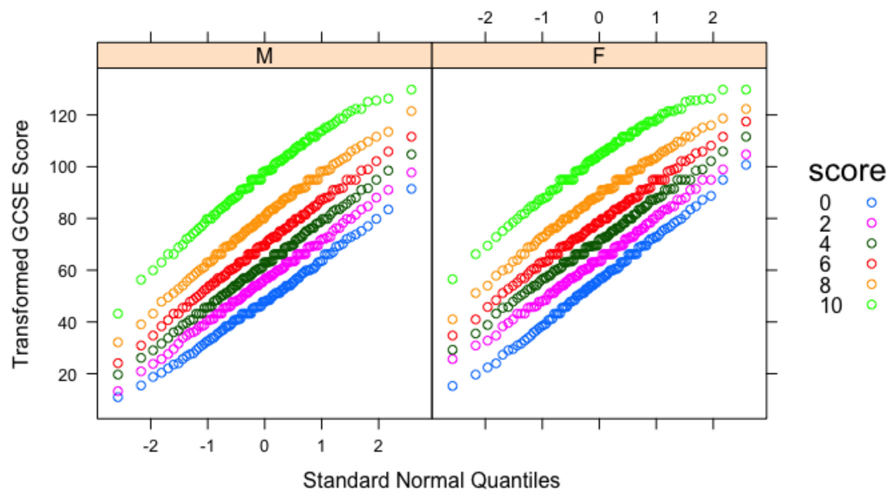
```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
Chem97.pos <- subset(Chem97, gcsescore > 0)
with(Chem97.pos,boxcox(gcsescore ~ score * gender, lambda = seq(0, 4, 1/10)))
```



```
#Plotting with lambda = 2.34
Chem97.mod <- transform(Chem97, gcsescore.trans = gcsescore^2.34)
qqmath(~ gcsescore.trans | gender, Chem97.mod, groups = score,
f.value = ppoints(100), aspect = "xy",
auto.key = list(space = "right", title = "score"),
xlab = "Standard Normal Quantiles",
ylab = "Transformed GCSE Score")
```



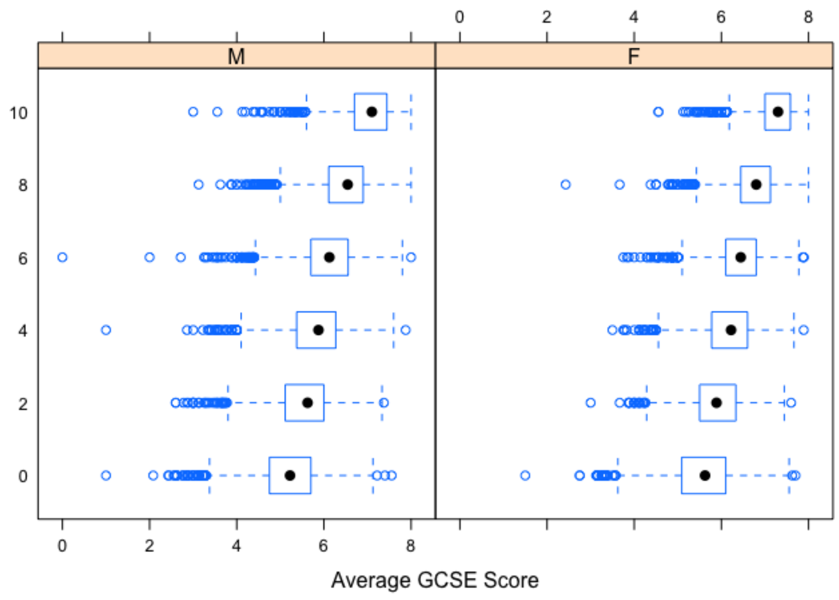
After the box-cox transformation, one can clearly see that the line for each score gets closer to a straight line, with indicates that normal approxiamation is a good fit for the transformed data set.

## * Box-and-whisker plots and violin plots

Another way to visualize a two sample plot or to compare a data set conditioned to one varialbe, we can also use the box plot and the violin plot. For both plots, users need to determine which variable to group by levels and which varialbe to condition on (Juxtapose each panel).

Box-and-whisker plots

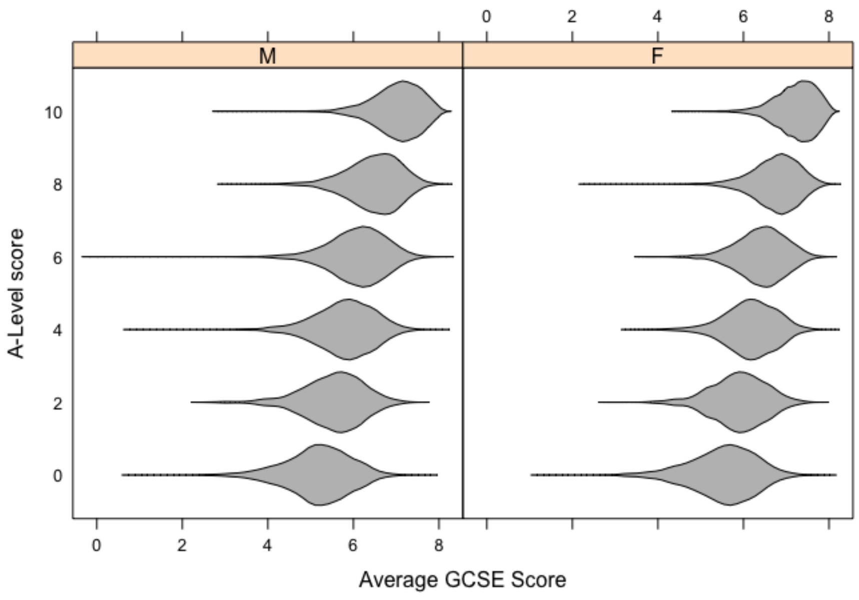
```
bwplot(factor(score) ~ gcsescore | gender, data = Chem97, xlab = "Average GCSE Score")
```



One can clearly see that the data set is left-skewed as most outliners occur on the left side. In this way, boxplot cannot efficiently display the desired distribution. It is rather difficult to compare the differences in distribution accorss each A-level score. In this case, we prefer to use the violin plot, which has similar layouts but has a flow and continuity to the plots.

Violin plots

```
bwplot(factor(score) ~ gcsescore | gender, Chem97,
panel = panel.violin, box.ratio = 5,
xlab = "Average GCSE Score",
ylab = "A-Level score", col = "grey")
```



Using the violin plots, one can observe that the distribution accross various A-level scores are all left-skewed to different extent. As the A-level score gets higher, the distribution of gcse score become more left-skewed, regardless of gender.

## * Strip plots

If I want to analyze relatively smaller data sets, or condition on some variable in a large data set, I can also use strip plots to visualize my data. This is rather similar with scatterplots but one of its variable is categorical.

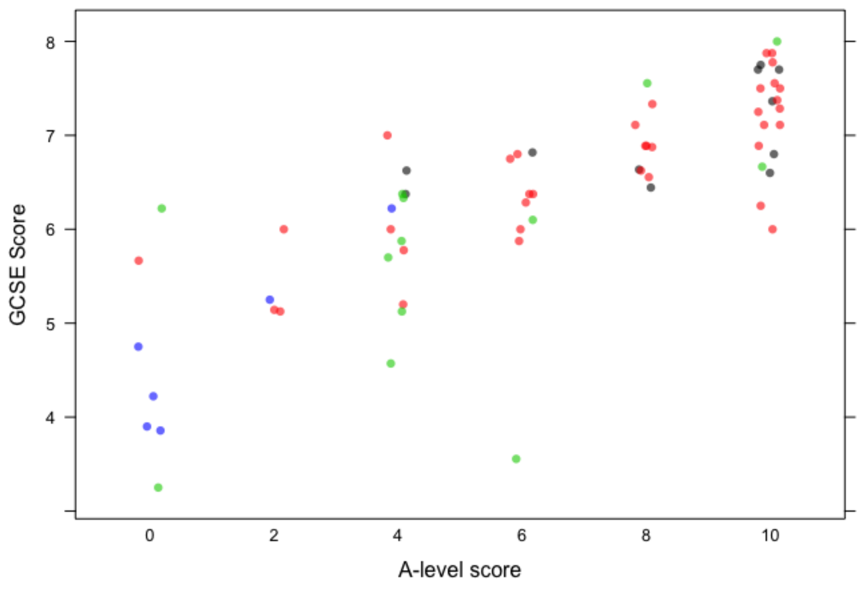
For example, I am only interested in Local Educational Authority 1 to 4 and want to look into the relationship between gcsescore and A-level score, I can produce the following univariate scatterplot and color code on each local educational authority.

```
LEA_1to4 <- filter(Chem97, lea == c("1","2","3","4"))
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length

## Warning in `==.default`(lea, c("1", "2", "3", "4")): longer object length
## is not a multiple of shorter object length
```

```
stripplot(gcsescore ~ factor(score), LEA_1to4,
jitter.data = TRUE, alpha = 0.6, pch = 16, col = LEA_1to4$lea,
xlab = "A-level score", ylab = "GCSE Score")
```



**Note**: These graphical methods above are mostly for continous distributions (with density). To visualize discrete distributions, one often uses bar charts and dot plots.

## VI. Trivariate Displays

Another perk of the *lattice* package is that it can display 3-D plots, or trivariate display. With functions cloud(), levelplot(), contourplot() and wireframe(), one can first create a 3-d scatterplot then render surfaces and 2-d projections for further data analysis. In this section, I will give examples for each function to show how it decomposes a data set to focus only on the desired variables.

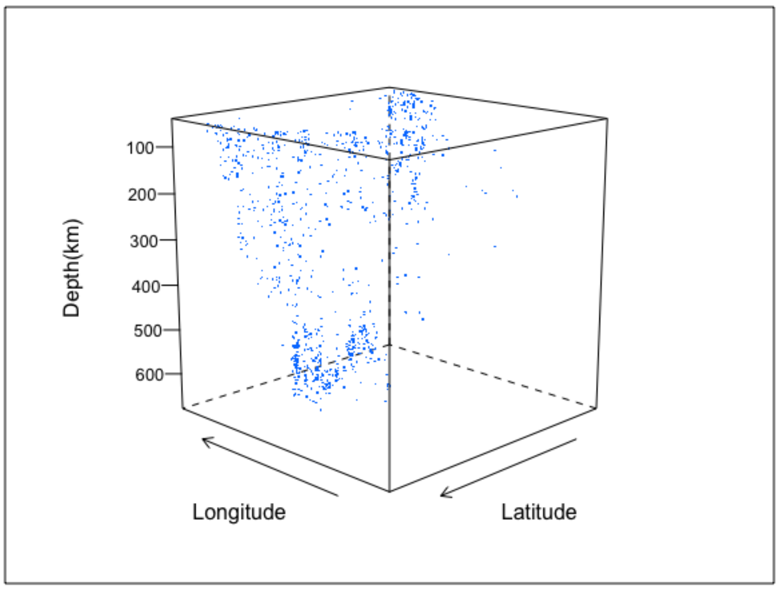
For this 3-d analysis, I will use the data set that gives the locations of 1000 seismic events (earthquakes) of MB > 4.0. The events occurred in a cube near Fiji since 1964. First, I will construct a generic 3-d plot that describes the longitute, latitute and depth of each earthquake.

## * cloud()

To use the cloud function, one needs to first define the axis with the following syntax: z~x*y
The bounds of the axis are by default the range of the axis, but one can also modify them with x/y/zlim. To determine the viewing angle, we modify the parameter screen(), with integer values for x,y,z, indicating the rotation angle of each axis. We can also the zoom() argument to shrink the plot slightly to make room for the axis labels, and the scales() argument to replace the z-axis arrow by labeled tick marks. Other graphing parameter syntax are similar to basic R graphs. One can

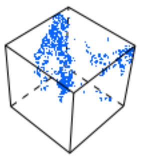
experiment with these arguments to get the ideal image. An example graph is as follows:

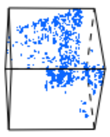
```
cloud(depth ~ lat * long, data = quakes,
zlim = rev(range(quakes$depth)),
screen = list(z = 135, x = -75), panel.aspect = 0.75,
scales = list(z = list(arrows = FALSE, distance = 1.5)),
pch = ".",
xlab = "Longitude", ylab = "Latitude",
zlab = list("Depth(km)", rot = 90))
```

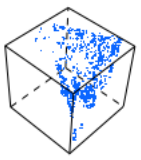


However, with 3-d graphs, one wants to see the distribution from multiple angles. One way to achieve that is through presenting the same graph with various viewing points. The first part of code is a simplified version of the previous graph and then define how each graph rotates from the previous one.

```
p <- cloud(
  depth ~ lat * long,
  quakes,
  zlim = rev(range(quakes$depth)),
  pch = ".",
  cex = 0.8  ,
  zoom = 0.8,
  xlab = NULL,
  ylab = NULL,
  zlab = NULL,
  par.settings = list(axis.line = list(col = "transparent")),
  scales = list(draw = FALSE))
npanel <- 3
rotz <- seq(-45, 45, length = npanel)
roty <- c(3, 0)
update(
  p[rep(1, 1 * npanel)],
  layout = c(1, npanel),
  panel = function(..., screen) {
  crow <- current.row()
  ccol <- current.column()
  panel.cloud(..., screen = list(z = rotz[crow],
  x = -45,
  y = roty[ccol]))})
```







## * wireframe(), levelplot(), and contourplot()

To perform such surface analysis, one needs to first pre-process the data to be represented as matrices containing evaluations of the function on a grid (continuous).

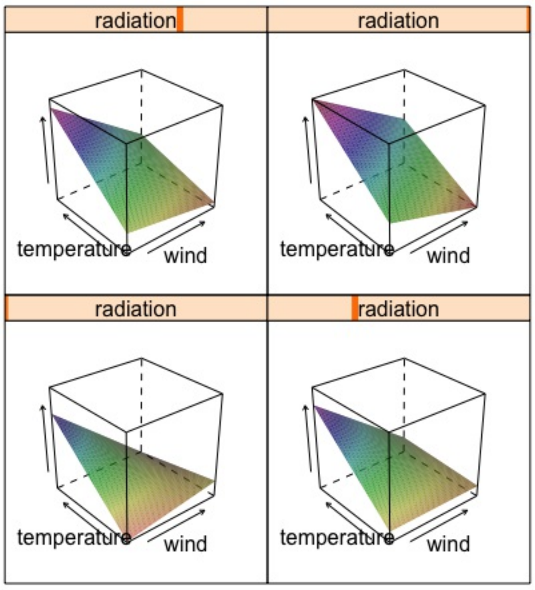
The three types of regression models that I employ here are standard linear regression, lowess regression and local regression. After defining various regression models, we calculate the margins of the grid with function with(). We then decide which two variables to define the surface and the other one as conditioning variable. The next step is to construct a full grid in the form of data frames which represents each modeled surfaces.

```
#data preperation
env <- environmental
env$ozone <- env$ozone^(1/3)
fm1.env <- lm(ozone ~ radiation * temperature * wind, env)
fm2.env <- loess(ozone ~ wind * temperature * radiation, env, span = 0.75, degree = 1)
fm3.env <- loess(ozone ~ wind * temperature * radiation, env, parametric = c("radiation", "wind"), span = 0
library("locfit")
```

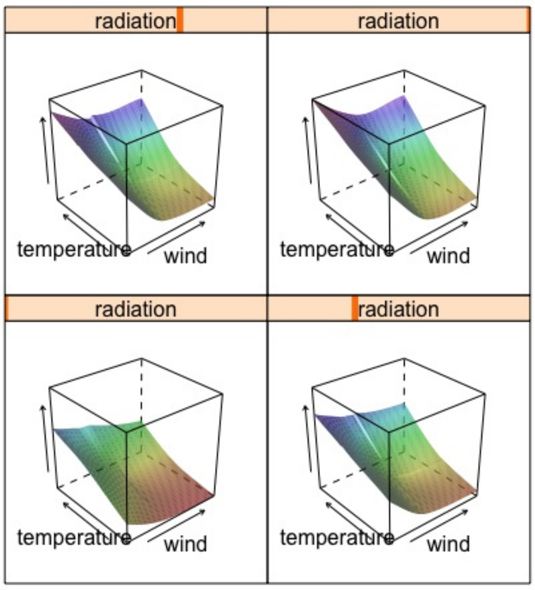
```
## locfit 1.5-9.1    2013-03-22
```

```
fm4.env <- locfit(ozone ~ wind * temperature * radiation, env)
w.mesh <- with(env, do.breaks(range(wind), 50))
t.mesh <- with(env, do.breaks(range(temperature), 50))
r.mesh <- with(env, do.breaks(range(radiation), 3))
grid <- expand.grid(wind = w.mesh, temperature = t.mesh, radiation = r.mesh)
grid[["fit.linear"]] <- predict(fm1.env, newdata = grid)
grid[["fit.loess.1"]] <- as.vector(predict(fm2.env, newdata = grid))
grid[["fit.loess.2"]] <- as.vector(predict(fm3.env, newdata = grid))
grid[["fit.locfit"]] <- predict(fm4.env, newdata = grid)
```

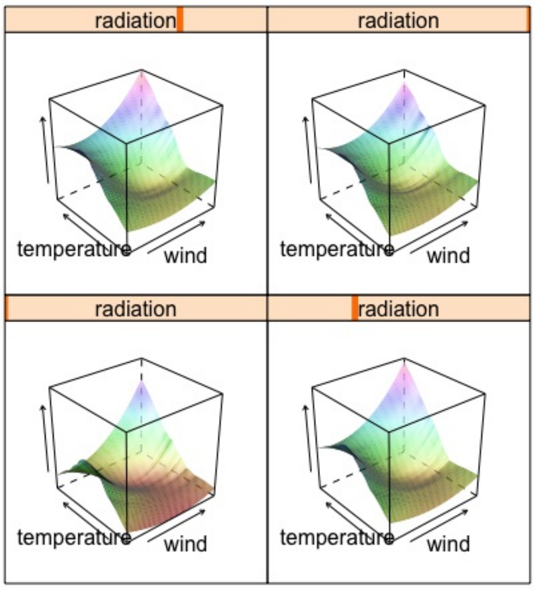
```
#Actual plotting
wireframe(fit.linear ~ wind * temperature | radiation, outer = TRUE, grid, shade = TRUE, zlab = "")
```



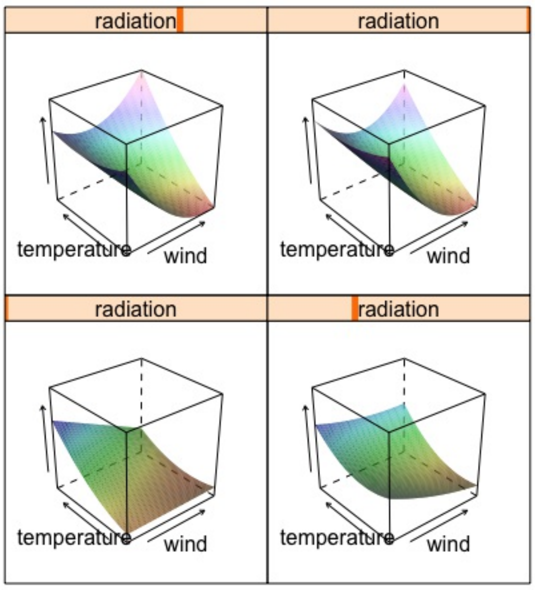
```
wireframe(fit.loess.1 ~ wind * temperature | radiation, outer = TRUE, grid, shade = TRUE, zlab = "")
```



```
wireframe(fit.loess.2 ~ wind * temperature | radiation, outer = TRUE, grid, shade = TRUE, zlab = "")
```

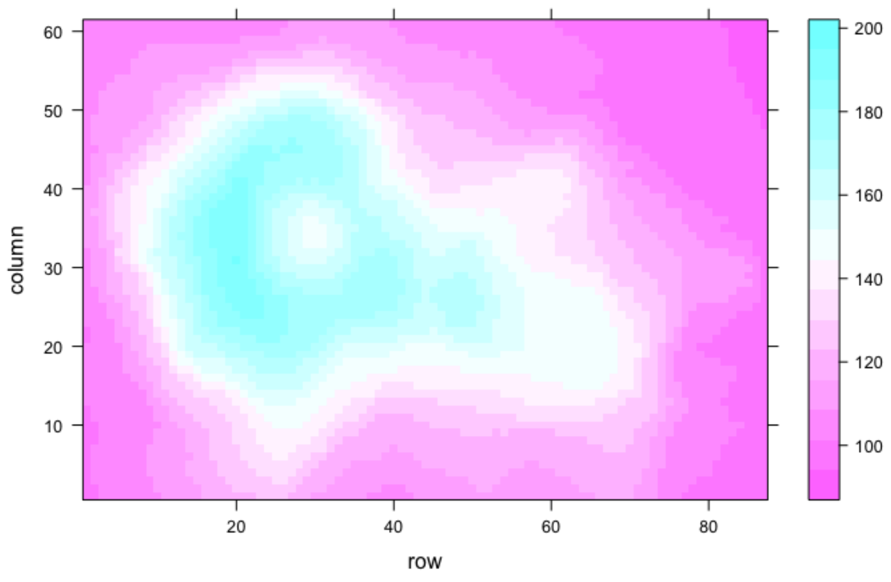


```
wireframe(fit.locfit ~ wind * temperature | radiation, outer = TRUE, grid, shade = TRUE, zlab = "")
```

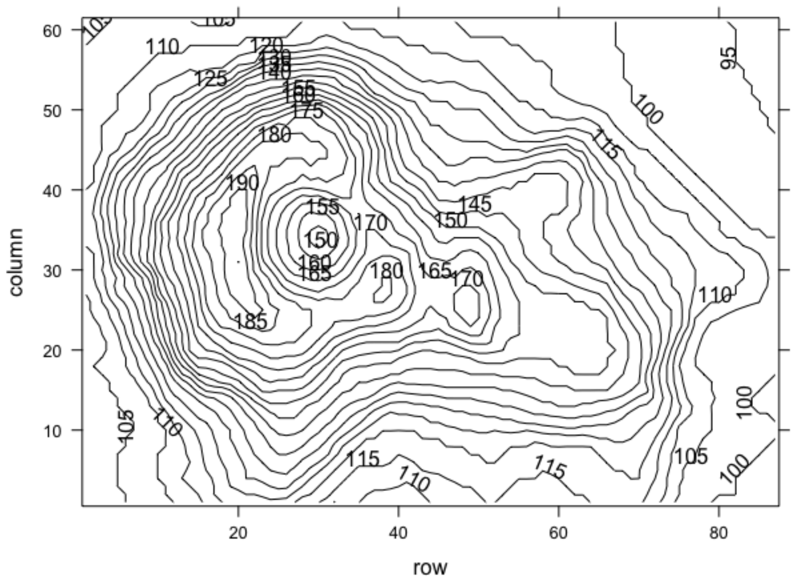


Lastly, those 3-d plotting functions works great with topography as we can see in the example below. With the topographic information of a volcano, one can use levelplot() or contourplot() to view the volcano from a bird's eyeview while the wireframe() function models a 3-d view of the volcano.

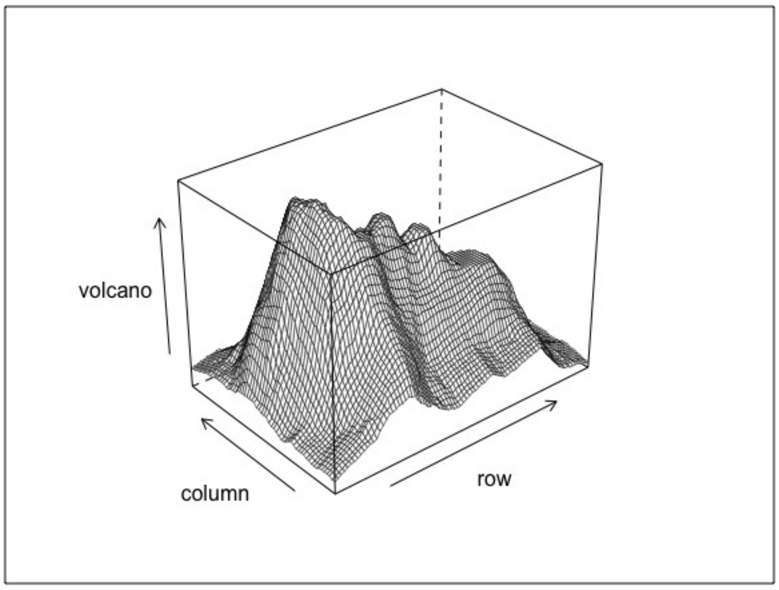
```
levelplot(volcano)
```



```r
contourplot(volcano, cuts = 30, label = TRUE)
```



```r
wireframe(volcano, panel.aspect = 0.75, zoom = 0.8, lwd = 0.5)
```



## VII. Summary

The key to this post is to present an alternate graphing package to ggplot2 as an alternate to data visualization. Those two packages have different emphasis and therefore no arbituarial conclusion for which one is better. Depending on the nature of data sets and analysis needs, one should choose the functions that can display the variables to the targeted audience in the most straight forward way.

### References

1. Lattice_Multivariate_Data_Visualization_with_R
2. Graphics and Data Visualization in R Overview
3. AN INTRODUCTION TO R
4. Lattice Graphs
5. Box–Cox transformation
6. ggplot2 Version of Figures in Lattice: Multivariate Data Visualization with R
7. local regression