**Graphics**

**Why plot data?**

1. Plotting your data should be one of the first items done after you obtain it to
   * Look for trends
   * Discover unusual observations (outliers)
   * Suggest items to examine in a more sophisticated statistical analysis
2. During a sophisticated statistical analysis, plots may be helpful to form particular conclusions (e.g., residual plots in regression analysis)
3. Once the sophisticated statistical analysis is complete, one should use plots to help explain the results to yourself and to others.

We will focus on 1) now and 3) will be useful after this sub-section

Notes for this sub-section:

* You will notice a lot of code is sometimes needed to construct plots! Please think of my code as a template. You can modify the template for your own data.
* I will primarily use the cereal data to illustrate plotting. This is because the data set is small (helpful for the first time you see a plot) and using the same data set allows you to make comparisons across the different types of plots.
* There are a few fully functional graphics packages in R. I will focus on the graphics package because it is the most widely used and within the default installation of R. Other packages will be used when graphics does not provide a particular type of plot or if the other package does something much better. The graphics package is loaded for use in R by default (it’s on the “search path”) so one does not need to use library(graphics) prior to using functions within it.

**Two-dimensional plots**

Scatter plot! See example in the Introduction to R notes.

When there are more than two variables, side-by-side scatter plots (a.k.a., scatter plot matrix) may be of interest.

Example: Cereal data (CerealGraphics.R, cereal.csv)

The pairs() function creates side-by-side scatter plots:

> pairs(formula = ~ sugar + fat + sodium, data = cereal)



> cor(cereal[,8:10])

sugar fat sodium

sugar 1.0000000 0.2397225 -0.1635699

fat 0.2397225 1.0000000 -0.0661432

sodium -0.1635699 -0.0661432 1.0000000

Examine the plot with respect to the estimated correlation matrix.

I have included another version of this plot in the program where the plotting point corresponds to shelf.

Alternatively, one can use the scatterplotMatrix() function in R’s car package. Below is the default use of it.

> library(car)

> scatterplotMatrix(formula = ~ sugar + fat + sodium, data = cereal)

**A graph of sugar and fat

Description automatically generated**

There are many items displayed in this plot! Perhaps too many. A brief explanation of these items will be given in class.

**Three-dimensional plots**

The purpose is to incorporate three separate variables into one plot.

### Scatter plot

Two variables can be plotted against each other where a characteristic of a plotting point corresponds to a qualitative variable.

Example: Cereal data (CerealGraphics.R, cereal.csv)

The plot() function can create a simple scatter plot:

> plot(x = cereal$sugar, y = cereal$fat, xlab = "Sugar",

ylab = "Fat", main = "Fat vs. Sugar \n Each variable is

adjusted for the serving size", panel.first = grid(col

= "lightgray", lty = "dotted"))



Do you see any trends or unusual observations?

Now, the shelf of the cereal is incorporated as a characteristic of a plotting point. I do this three ways (color, size, symbol), but one or two ways are typically only needed in application:

> plot(x = cereal$sugar, y = cereal$fat, xlab = "Sugar",

ylab = "Fat", main = "Fat vs. Sugar \n Each variable is

adjusted for the serving size", panel.first = grid(col

= "lightgray", lty = "dotted"), **type = "n"**)

> shelf.color <- rep(x = c("black", "red", "darkgreen",

"blue"), each = 10)

> shelf.symbol <- rep(x = 1:4, each = 10)

> #shelf.color

> #shelf.symbol

> points(x = cereal$sugar, y = cereal$fat, pch =

shelf.symbol, col = shelf.color, cex = 1 +

0.3\*shelf.symbol)

> legend(x = 0.5, y = 0.08, title = "Shelf", legend = 1:4,

bty = "n", col = c("black", "red", "darkgreen",

"blue"), pch = 1:4, pt.cex = c(1.3, 1.6, 1.9, 2.2))

> #If the legend labels were not numbers, just use

something like c("1", "2", "3", "4") where the numbers

are replaced with actual names



Of course, a legend is needed for these types of plots!

Do you see any trends or unusual observations?

The previous code took advantage of the data being sorted by shelf number. What if the data was not already sorted like this?

> # "Unsort" the data

> set.seed(9012)

> new.order <- sample(x = 1:40, size = nrow(cereal), replace = FALSE)

> cereal2 <- cereal[new.order,]

> options(width = 70) # Helps format for my notes

> head(cereal2)

ID Shelf Cereal size\_g

10 10 1 Food Club Crispy Rice 33

24 24 3 Kellogg's Corn Pops 31

29 29 3 Post Raisin Bran 59

18 18 2 Food Club Toasted Oats 33

19 19 2 Cocoa Pebbles 29

39 39 4 Post Fruit and Fibre - Dates, Raisons, Walnuts 55

sugar\_g fat\_g sodium\_mg sugar fat sodium

10 2 0.0 330 0.06060606 0.00000000 10.000000

24 14 0.0 120 0.45161290 0.00000000 3.870968

29 20 1.0 300 0.33898305 0.01694915 5.084746

18 10 1.5 150 0.30303030 0.04545455 4.545455

19 13 1.0 160 0.44827586 0.03448276 5.517241

39 17 3.0 280 0.30909091 0.05454545 5.090909

> #Sorting by shelf

> cereal3 <- cereal2[order(cereal2$Shelf),]

> head(cereal3[,c(1:3,8:10)])

ID Shelf Cereal sugar fat

10 10 1 Food Club Crispy Rice 0.06060606 0.00000000

3 3 1 Kellog's Corn Flakes 0.07142857 0.00000000

8 8 1 Capn Crunch's Peanut Butter Crunch 0.33333333 0.09259259

9 9 1 Post Honeycomb 0.37931034 0.01724138

4 4 1 Food Club Toasted Oats 0.06250000 0.06250000

6 6 1 Food Club Frosted Flakes 0.35483871 0.00000000

sodium

10 10.000000

3 10.714286

8 7.407407

9 7.586207

4 8.750000

6 5.806452

> tail(cereal3[, c(1:3,8:10)])

ID Shelf Cereal sugar fat sodium

32 32 4 Food Club Wheat Crunch 0.1000000 0.00000000 5.000000

35 35 4 Cookie Crisp 0.4000000 0.03333333 6.000000

33 33 4 Oatmeal Crisp Raisin 0.3454545 0.03636364 4.000000

34 34 4 Food Club Bran Flakes 0.1612903 0.01612903 7.096774

31 31 4 Total Raisin Bran 0.3454545 0.01818182 4.363636

37 37 4 Food Club Low Fat Granola 0.2545455 0.05454545 1.818182

### Bubble Plot

This is a scatter plot of two variables with the size of the plotting character proportional to a third variable. The third variable’s plotting character is usually a circle (bubble).

Example: Cereal data (CerealGraphics.R, cereal.csv)

Bubble plots are created with the symbols() function. Below is a bubble plot for the adjusted data.

> symbols(x = cereal$sugar, y = cereal$fat, circles =

cereal$sodium, xlab = "Sugar", ylab = "Fat",

main = "Fat vs. Sugar with symbol proportional to

sodium \n Each variable is adjusted for the serving

size", panel.first = grid(col = "lightgray", lty =

"dotted"))



The plot is difficult to interpret due to the size of some points. This is an example of where two additions can be helpful to the code to better interpret the plot:

1. Control the maximum “dimension” of the plotting symbol with the inches argument; the default is set to a value of 1 (“dimension” is not defined as radius, diameter, …)
2. Rescale the plotting symbol by using a transformation of the argument value for circles

I added inches = 0.5 to the symbols() function code:



Do you see any trends in the data?

Notice the two very small points in the lower left side of the plot. We can identify these and any other observations on the plot by using the identify() function:

identify(x = cereal$sugar, y = cereal$fat)



These points correspond to observations #26 and #30:

> cereal[c(26,30),c(1:3,8:10)]

ID Shelf Cereal sugar fat sodium

26 26 3 Post Shredded Wheat Spoon Size 0.00 0.01020408 0

30 30 3 Food Club Frosted Shreded Wheat 0.02 0.02000000 0

They have no sodium! Fortunately, R still plots a point for these observations. Overall, these points appear to be unusual in comparison to the rest due to their sodium AND also for their somewhat extreme fat and sugar values.

Question: What would happen if there were both positive and negative values for the circles argument? For example, suppose we did this plot with the standardized data values.

Alternatively, we can automatically label all of the points by also using the text() function after symbols():

> text(x = cereal$sugar, y = cereal$fat, labels = round(cereal$sodium,2))



Below is what happens to the original plot when inches = 0.5 is added and the circles argument is changed to sqrt(cereal$sodium) in the symbols() function code:



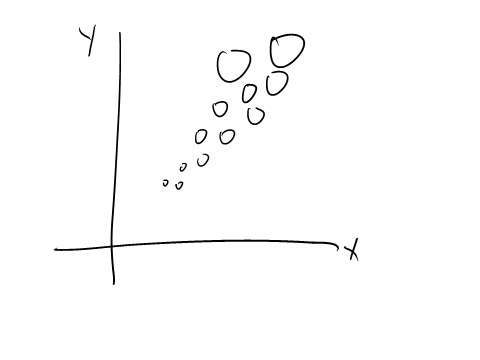
Why are the points generally more similar in size than in the last plot?

Overall, rescaling the values in the circles argument is most useful when there are only a few big circles in comparison to the rest. Try this on your own with a data set that you create! Also, the rescaling can be done with some other mathematical function too.

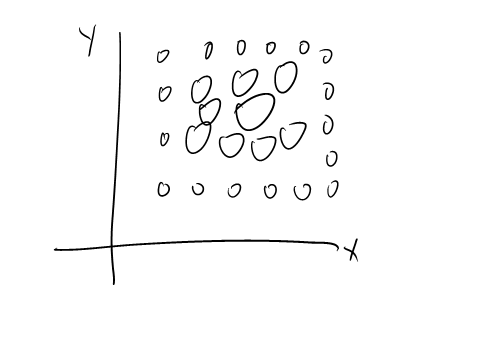
Question: How could you add shelf type to this plot?

In the last example, there were not many trends found. Below are examples of other types of trends:

Trend among all three variables:



No trend between x and y, but a trend exists with the third variable:



Bubble plots are also useful for regression model diagnostic plots.

Below is an example plot from my categorical data analysis course based on the placekicking data set described previously. The red line is an estimated logistic regression model relating the distance of a placekick to its probability of success. The plotting points are drawn at the proportion of successes for a given distance, where the plotting point is proportional to the number of trials at a particular distance.

|  |  |  |  |
| --- | --- | --- | --- |
| **Distance** | **Failures** | **Successes** | **Observed  proportion** |
| 18 | 1 | 2 | 2/3 = |
| 19 | 0 | 7 | 7/7 = 1.0 |
| 20 | 13 | 776 | 776/789 |
| 21 | 1 | 19 | 19/20 |
| 22 | 2 | 12 | 12/14 |
|  |  |  |  |
| 55 | 1 | 2 | 2/3 |
| 56 | 0 | 1 | 1/1 |
| 59 | 0 | 1 | 1/1 |
| 62 | 1 | 0 | 0/1 |
| 63 | 1 | 0 | 0/1 |
| 66 | 1 | 0 | 0/1 |



Another example of a bubble plot is available from <http://www.gapminder.org/videos/200-years-that-changed-the-world>. This plot uses time to add a fourth dimension to the plot.

### 3D Scatter Plot

The plot3d() function from the rgl package provides a very good way to do these plots in R.

Example: Cereal data (CerealGraphics.R, cereal.csv)

Similar to what we saw with the previous rgl package examples, these plots are rotatable. Below is an example plot.

> library(rgl)

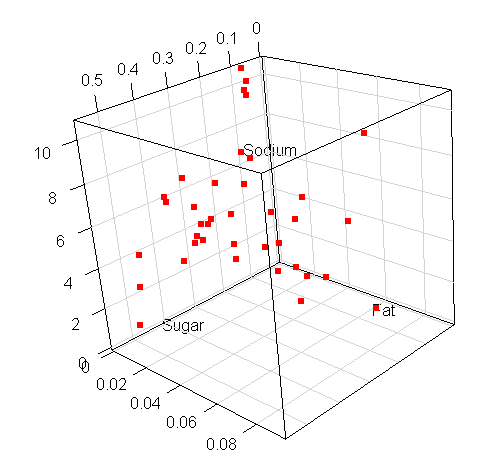
> plot3d(x = cereal$sugar, y = cereal$fat, z =

cereal$sodium, xlab = "Sugar", ylab = "Fat", zlab =

"Sodium", col = "red", size = 6) #Default size is 3

(see help file)

> grid3d(side = c("x", "y", "z"), col = "lightgray")   
 #Use "+" and "-" after "x" to change location of grid



It is best to rotate the plot to view it at a few different angles in order to decide on trends and/or unusual observations.

Below is another version of the plot where “needles” are used to help see the plotting points:

> plot3d(x = cereal$sugar, y = cereal$fat, z =

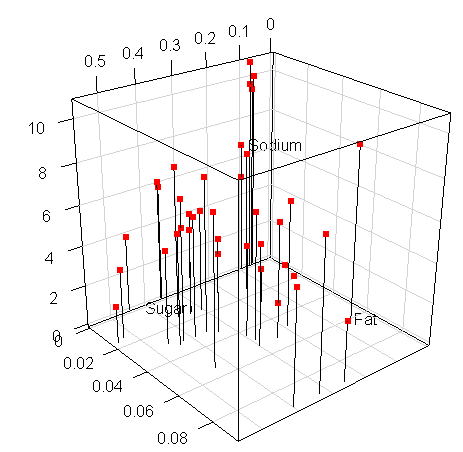
cereal$sodium, xlab = "Sugar", ylab = "Fat", zlab =

"Sodium", type = "h")

> plot3d(x = cereal$sugar, y = cereal$fat, z =

cereal$sodium, add = TRUE, col = "red", size = 6)

> grid3d(side = c("x", "y", "z"), col = "lightgray")



How could you construct these types of plots when each plotting point corresponds to shelf? This would essentially add a fourth dimension to the plot.

I show how to produce 3D scatter plots with three other packages in the corresponding program.

**Plots of Higher Dimensional Data**

Previous examples were given for how to add a few additional dimensions to plots. The plots to be discussed next can be used to plot many more dimensions.

### Chernoff Faces

This plot is briefly discussed here more for historical purposes. The star plots to be discussed shortly are better to represent multivariate data in this type of format.

Faces are used to represent multidimensional data.

* Each face is for a particular experimental unit
* Face characteristics (eyes, hair, mouth,…) are scaled to correspond to a variable.

Below is an example plot for the cereal data set:



### Star Plots

This is how these types of plots are constructed

* Each star (or sun) represents a particular experimental unit.
* The center of the star denotes the minimum value for each variable.
* For each variable, a line or “ray” extends out from the center at a length proportional to the variable value.

These plots are useful to

* Help detect outliers – stars that are very different from the others indicate possible outliers
* Validate cluster analysis results – observations grouped within the same cluster should have similar stars

Example: Cereal data (CerealGraphics.R, cereal.csv)

> stars(x = cereal[,8:10], nrow = 4, ncol = 10,

draw.segments = FALSE, key.loc = c(3,12))



> stars(x = cereal[,8:10], nrow = 4, ncol = 10,

draw.segments = TRUE, key.loc = c(3,12))

### 

Comments:

* Cereals 26 and 30 (Post Shredded Wheat Spoon Size and Food Club Frosted Shredded Wheat) appear to be low in sugar, fat, and sodium (relative to the other cereals).
* Which cereals appear to be high in fat?

8 = Capn Crunch’s Peanut Butter Crunch

14 = Capn Crunch’s Peanut Butter Crunch (same – just on different shelf!)

20 = Oreo’s O’s

25 = Post morning traditions – Raison, Date, and Pecan

* Which cereals appear to be high in sugar?

12 = Kellogg’s Smacks

16 = Marshmallow Blasted Froot Loops

* Which cereals appear to be high in sodium?

2 = Post Toasties Corn Flakes

3 = Kellogg’s Corn Flakes

10 = Food Club Crispy Rice

23 = Rice Chex

* The researcher’s hypothesis before the data collection was that shelf 1 and 2 tend to have the higher sugar content cereals. From this plot, what do you think?
* The labels argument for stars() can be used to add corresponding names to each star. I did not use it here because the cereal names would be too large. Instead, I used the default labels which is row number for the data frame.

Compare what you see in this plot with the previous plots.

The number of stars can be quite large for a large data set because each star represents an experimental unit. Thus, there are limits to the usefulness of these types of plots.

Remember that the placekicking data set has 1,425 observations! Maybe a star could be used to represent each placekicker by averaging over variable values??? This idea of averaging over experimental units (when there is something logical to average over) is a great way to still use this type of plot for large samples.

Parallel coordinates plots

Each variable is assigned to its own vertical axis. The variable values are rescaled so that minimum observation value corresponds to the bottom of the vertical axis and the maximum observation value corresponds to the top of the vertical axis (one way is (value – minimum)/(maximum – minimum)). Each experimental unit’s variable value is plotted upon the corresponding axis. Lines are drawn to connect values for the same experimental unit.

This type of plot is ideal to allow you to see trends across variables for a particular experimental unit.

There are a number of functions that can be used to construct these plots. One of the main functions is parcoord() from the MASS package. This package is already installed with R, but it needs to be called first with library(MASS). Other functions that can be used for these plots include ipcp() from the iplots package (“brushing” of points is possible) and ggparcoord() from the GGally package.

Example: Cereal data (CerealGraphics.R, cereal.csv)

> library(MASS)

> shelf.color <- rep(x = c("black", "red", "green", "blue"),

each = 10)

> parcoord(x = cereal[,c(1,8:10)], col = shelf.color, main

= "Cereal data")

> legend(locator(1), legend = c("1", "2", "3", "4"), lty =

c(1,1,1,1), col=c("black", "red", "green3", "blue"),

bty = "n", cex = 0.75)



The horizontal axis gives each variable name, and plotted above these names are the corresponding values for each experimental unit. These values are scaled so that the minimum and maximum values for each variable appear at the bottom and top, respectively, of the vertical axis. A line is drawn for each experimental unit indicating its position across the variables.

Comments:

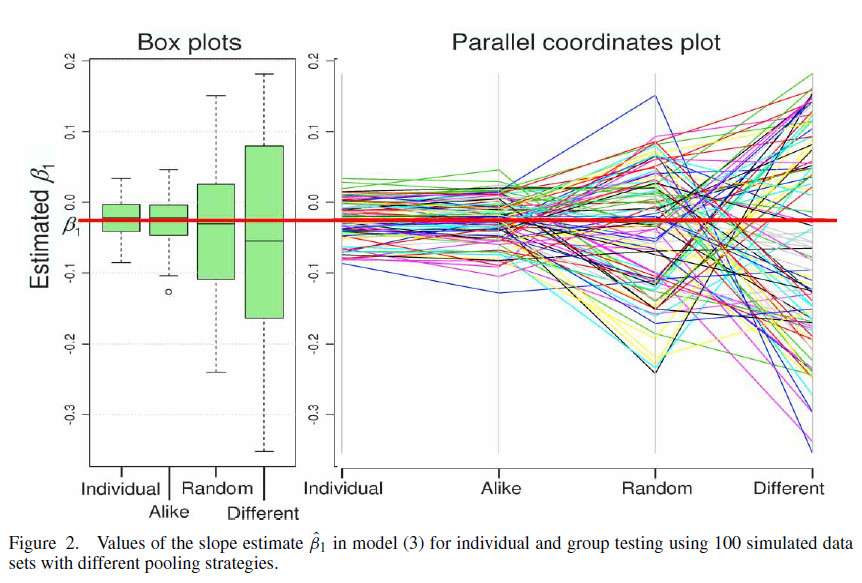
1. Parallel coordinates do not need to be drawn with the line colors corresponding to a variable. I chose to do this here because it helps particular patterns stand out better.
2. Shelf 2 cereals tend to be from the middle to the top for the sugar variable (0 is on the bottom and 1 is at the top on the vertical axis). This provides a preliminary indication that those cereals have some of the higher sugar content cereals in comparison with the other shelves.
3. There are a few unusual observations for the fat variable as indicated by their large values.
4. Examine what happens when you follow the cereal lines from one variable to another. For example, the highest in sugar content cereals do not necessarily have high fat content.

Example: Statistical research paper #1

I once used parallel coordinates plots to compare four different estimation methods. This involved simulating many different data sets and forming a data set like below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Method | | | |
| Simulated data set | Individual | Alike | Random | Different |
| 1 | -0.5 | -0.4 | 0.1 | -0.1 |
| 2 | 0.1 | 0.0 | -0.1 | 0.5 |
|  |  |  |  |  |

Each of the above numerical values represent an estimate of a regression parameter, like . I was very interested in the amount of variability for each estimation method. For this reason, I modified the parcord() function code so that the variable values were NOT rescaled. Below is the plot (color has no specific purpose) and a side-by-side box plot.



From examining this plot, what can you conclude about the estimation methods?

Here’s the actual code for parcoord():

> parcoord

function (x, col = 1, lty = 1, var.label = FALSE, ...)

{

rx <- apply(x, 2L, range, na.rm = TRUE)

x <- apply(x, 2L, function(x) (x - min(x, na.rm =

TRUE))/(max(x, na.rm = TRUE) - min(x, na.rm = TRUE)))

matplot(1L:ncol(x), t(x), type = "l", col = col, lty =

lty, xlab = "", ylab = "", axes = FALSE, ...)

axis(1, at = 1L:ncol(x), labels = colnames(x))

for (i in 1L:ncol(x)) {

lines(c(i, i), c(0, 1), col = "grey70")

if (var.label)

text(c(i, i), c(0, 1), labels = format(rx[, i],

digits = 3), xpd = NA, offset = 0.3, pos = c(1,

3), cex = 0.7)

}

invisible()

}

<bytecode: 0x0547a35c>

<environment: namespace:MASS>

I tested the code in parcoord() line-by-line in order to understand what it did, and this led to my own function based on it:

parcoord2 <- function (x, col = 1, lty = 1, ...)

{

matplot(1:ncol(x), t(x), type = "l", col = col, lty =

lty, xlab = "", ylab = "", axes = FALSE, ...)

axis(1, at = 1:ncol(x), labels = colnames(x))

axis(side = 2)

for (i in 1:ncol(x)) lines(c(i, i), c(min(x),

max(x)), col = "grey70")

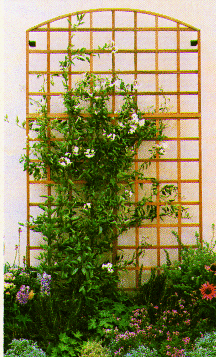
invisible()

}

Overall, parallel coordinates plots are one of my most favorite types of plots. However, one needs to watch out for the overlapping of lines. This is especially important when a variable has few values (e.g., 0 or 1) and/or there are a very large number of observations. What do you think could be done to alleviate this potential problem at least partially?

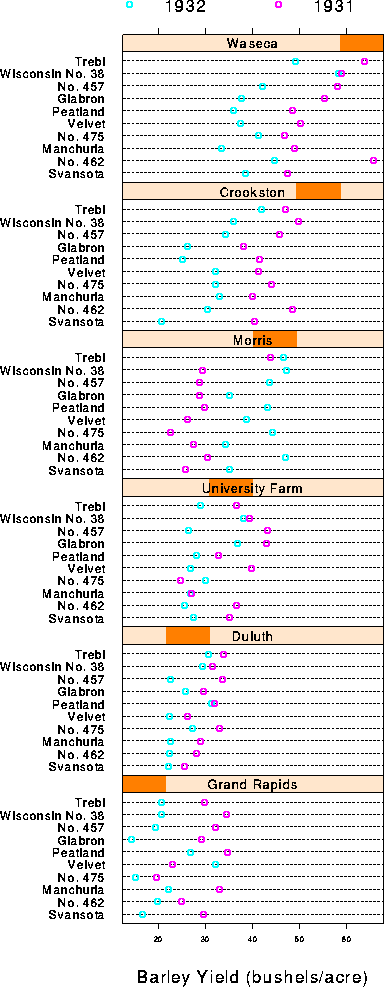
Trellis plots (co-plots)

These plots allow for the viewing of multidimensional relationships between variables through conditioning.



The picture to the right is a trellis. The important part of the picture is the squares that make up the trellis. Imagine one plot (possibly a scatter plot) within each square and that the plots are from the same data set. However, each plot represents a different subset (possibly overlapping) of the data set. The subsets are determined by conditioning on variable values.

Below is an often-shown trellis plot containing dot plots of yields for various barley varieties from an experiment. Notice the 6 different rectangles or **panels** of the trellis. They each represent a particular farming location in Minnesota. Therefore, the barley varieties vs. yield dot plots are represented conditionally on farming location.



The Morris location is different from the rest. Despite this data being used for many years to illustrate ANOVA methods, the Morris result was not noticed until Trellis plots were developed.

Was the Morris result an unusual occurrence or perhaps a data entry error? For a long time after this plot was first produced, people thought it was a data entry error. It was not until more recently (Wright, *American Statistician*, 2013) that it was explained to be an unusual occurrence.

The lattice and ggplot2 packages are the two main ways that trellis plots can be produced in R (coplot() in graphics can be used but it is not as good). I will discuss the lattice package here because its code is more like the code that we have been using. Again, there will be a lot of code for the plots. A great way to learn the code is to remove one argument at a time from my own code to see the effect on the plot.

Example: Cereal data (CerealGraphics.R, cereal.csv)

Data used in a trellis plot setting sometimes needs to be “reshaped” for plotting purposes. To illustrate the process, below is an example of a simple longitudinal data set that is reshaped from a “wide” format to a “long” format:

> #Construct a simple data set – typical format in a

repeated measures setting

> set1 <- data.frame(ID.name = c("subject1", "subject2",

"subject3"), ID.number = c(1, 2, 3), age = c(19, 16,

21), time1 = c(1, 0 ,0), time2 = c(0, 0, 1))

> set1

ID.name ID.number age time1 time2

1 subject1 1 19 1 0

2 subject2 2 16 0 0

3 subject3 3 21 0 1

> #Long format

> set2 <- reshape(data = set1, idvar = "ID.name", varying =

c("time1", "time2"), v.names = "response", direction =

"long", drop = "ID.number")

> set2

ID.name age time response

subject1.1 subject1 19 1 1

subject2.1 subject2 16 1 0

subject3.1 subject3 21 1 0

subject1.2 subject1 19 2 0

subject2.2 subject2 16 2 0

subject3.2 subject3 21 2 1

> #Remove unnecessary row names

> row.names(set2) <- NULL

> set2

ID.name age time response

1 subject1 19 1 1

2 subject2 16 1 0

3 subject3 21 1 0

4 subject1 19 2 0

5 subject2 16 2 0

6 subject3 21 2 1

> #Back to wide format

> set3 <- reshape(data = set2, timevar = "time", idvar =

"ID.name", direction = "wide")

> set3

ID.name age.1 response.1 age.2 response.2

1 subject1 19 1 19 0

2 subject2 16 0 16 0

3 subject3 21 0 21 1

Reshaping the data

Below is how I reshape the original cereal data and also its corresponding standardized format:

> cereal.long <- reshape(data = cereal, idvar = "ID", drop = c("size\_g", "sugar\_g", "fat\_g", "sodium\_mg"), varying = c("sugar", "fat", "sodium"), timevar = "content",

v.names = "amount", times = c("sugar", "fat",

"sodium"), direction = "long")

> row.names(cereal.long) <- NULL

> head(cereal.long)

ID Shelf Cereal content

1 1 1 Kellog's Razzle Dazzle Rice Crispies sugar

2 2 1 Post Toasties Corn Flakes sugar

3 3 1 Kellog's Corn Flakes sugar

4 4 1 Food Club Toasted Oats sugar

5 5 1 Frosted Cheerios sugar

6 6 1 Food Club Frosted Flakes sugar

amount

1 0.35714286

2 0.07142857

3 0.07142857

4 0.06250000

5 0.43333333

6 0.35483871

> #Construct long format of the data with standardized

values

> Z.cereal <- data.frame(cereal[,1:3], scale(cereal[,8:10]))

> head(Z.cereal)

ID Shelf Cereal content

1 1 1 Kellog's Razzle Dazzle Rice Crispies sugar

2 2 1 Post Toasties Corn Flakes sugar

3 3 1 Kellog's Corn Flakes sugar

4 4 1 Food Club Toasted Oats sugar

5 5 1 Frosted Cheerios sugar

6 6 1 Food Club Frosted Flakes sugar

amount

1 0.35714286

2 0.07142857

3 0.07142857

4 0.06250000

5 0.43333333

6 0.35483871

> Z.cereal.long <- reshape(data = Z.cereal, idvar = "ID",

varying = c("sugar", "fat", "sodium"), timevar =

"content", v.names = "amount", times = c("sugar",

"fat", "sodium"), direction = "long")

> row.names(Z.cereal.long) <- NULL

> head(Z.cereal.long)

ID Shelf Cereal content

1 1 1 Kellog's Razzle Dazzle Rice Crispies sugar

2 2 1 Post Toasties Corn Flakes sugar

3 3 1 Kellog's Corn Flakes sugar

4 4 1 Food Club Toasted Oats sugar

5 5 1 Frosted Cheerios sugar

6 6 1 Food Club Frosted Flakes sugar

amount

1 0.4528441

2 -1.4575233

3 -1.4575233

4 -1.5172223

5 0.9622754

6 0.4374379

Histograms:

> library(lattice)

> histogram(x = ~ amount | content + Shelf, data =

Z.cereal.long, type = "percent", layout = c(3,4),

xlab = "Standardized amount", main = "Histograms of

cereal content")

A graph of a bar chart

Description automatically generated with medium confidence

Do you see any trends in the data?

Notice the shelf number is incorporated into the plot by vertical dark lines. If this is not your preference, there are ways to include the actual shelf number by having R treat the shelf as a factor (i.e., character) rather than a numerical value.

> Z.cereal.long2 <- data.frame(Z.cereal.long, Shelf.char =

factor(Z.cereal.long$Shelf))

> histogram(x = ~ amount | content + Shelf.char, data =

Z.cereal.long2, type = "percent", layout = c(3,4),

xlab = "Standardized amount", main = "Histograms of

cereal content")

A graph of different sizes and colors

Description automatically generated with medium confidence

We can even include “Shelf = “ on the plot:

> Z.cereal.long2 <- data.frame(Z.cereal.long, Shelf.char =

factor(paste("Shelf =", Z.cereal.long$Shelf)))

> histogram(x = ~ amount | content + Shelf.char, data =

Z.cereal.long2, type = "percent", layout = c(3,4),

xlab = "Standardized amount", main = "Histograms of

cereal content")

A graph of a graph of sugar and sodium

Description automatically generated with medium confidence

Finer control of the panels in a trellis plot can be controlled by the panel argument:

> # Add kernel density estimate to plot

> histogram(x = ~ amount | content + Shelf, data =

Z.cereal.long, type = "density", layout = c(3,4),

xlab = "Standardized amount", main = "Histograms of

cereal content",

panel = function(x, ...) {

panel.histogram(x, ...)

panel.densityplot(x, col = "black", ...)

}

)

A graph of food content

Description automatically generated

The panel argument allows for a function to be called (in this case, it is written within the original function call) and applied to each panel. The x argument of the new function takes the contents from the original x in histogram() for a particular content and Shelf.

Scatter plots

> xyplot(x = sugar ~ fat | factor(paste("Shelf =",

cereal.long$Shelf)), data = cereal, main = "Scatter

plots by shelf")

A diagram of a scatter plot

Description automatically generated

Notice how I simply formed the “Shelf = ” panel label within the function call rather than create a new data frame as I did before.

> xyplot(x = sugar ~ fat | factor(paste("Shelf =",

cereal.long$Shelf)), data = cereal,

main = "Scatter plots with linear regression model",

panel = function(x, y, ...) {

panel.xyplot(x, y, col = "black")

panel.grid(lty = "dotted", col = "lightgray")

panel.lmline(x, y, col = "red", lty = "solid")

}

)

A diagram of a scatter plot

Description automatically generated

> cloud(x = sugar ~ fat + sodium | factor(paste("Shelf =",

cereal.long$Shelf)), data = cereal, main = "Scatter

plots by shelf")

A diagram of a cube

Description automatically generated

Forming a shingle

There may be times when you want to condition on a continuous variable. This can be done by forming a “shingle” for each panel.

> sodium.group <- equal.count(x = cereal$sodium, number = 3, overlap = 0.1)

> sodium.group

Data:

[1] 6.071429 9.642857 10.714286 8.750000 7.000000

[6] 5.806452 7.407407 7.407407 7.586207 10.000000

[11] 6.333333 1.851852 4.687500 7.407407 7.666667

[16] 3.500000 5.666667 4.545455 5.517241 5.555556

[21] 5.185185 6.333333 9.354839 3.870968 2.962963

[26] 0.000000 5.818182 6.000000 5.084746 0.000000

[31] 4.363636 5.000000 4.000000 7.096774 6.000000

[36] 2.096774 1.818182 4.727273 5.090909 6.666667

Intervals:

min max count

1 -0.003081664 5.003082 14

2 4.996918336 6.336415 14

3 6.330251669 10.717367 15

Overlap between adjacent intervals:

[1] 1 2

> levels(sodium.group)

[,1] [,2]

[1,] -0.003081664 5.003082

[2,] 4.996918336 6.336415

[3,] 6.330251669 10.717367

> xyplot(x = sugar ~ fat | sodium.group, data = cereal,

groups = factor(Shelf), auto.key = TRUE)

A diagram of different types of sugar

Description automatically generated

The equal.count() function found 3 separate groups for the sodium content where a 10% overlap in number of observations was allowed between the groups. The darker color in the panel title regions indicate the approximate range of sodium values for a particular group.

Example: Statistical research paper #2

I have constructed trellis plots for a few of my papers to summarize Monte Carlo simulation results. In one of my papers, I used a trellis plot to summarize the Monte Carlo simulation results that evaluated 8 different hypothesis testing methods. Here’s the set-up:

* There were 500 data sets simulated using settings that specified the null hypothesis of interest to be true.
* For each simulated data set, the hypothesis testing methods were applied using α = 0.05.
* If the hypothesis testing method held the correct size, one would expect the approximate proportion of null hypothesis rejections to be equal to 5%.
* Due to randomness of Monte Carlo simulation, I would expect all rejection rates to be within



This same process was applied to 62 different data simulation settings. The multivariate data set produced had the following form:

|  |  |  |  |  | Methods | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Simulation run | Marginal table | Marginals | n | OR | RS |  | Bon |
| 1 | 2×2 | 0.6, 0.8 | 20 | All 2 | 0.052 |  | 0.038 |
| 2 | 2×2 | 0.6, 0.8 | 20 | All 25 | 0.056 |  | 0.052 |
|  |  |  |  |  |  |  |  |
| 62 | 510 | 0.05, …, 0.95 | 200 | Different | 0.070 |  | 0.046 |

where the numerical values are given just for illustrative purposes and are not actual values found in the simulation study.



Do you see any trends?

Note that it would be very difficult to do this exact plot completely in R. Instead, one can construct as much of the plot as possible in R, copy it into PowerPoint (or some other software package), and add annotations to the plot.