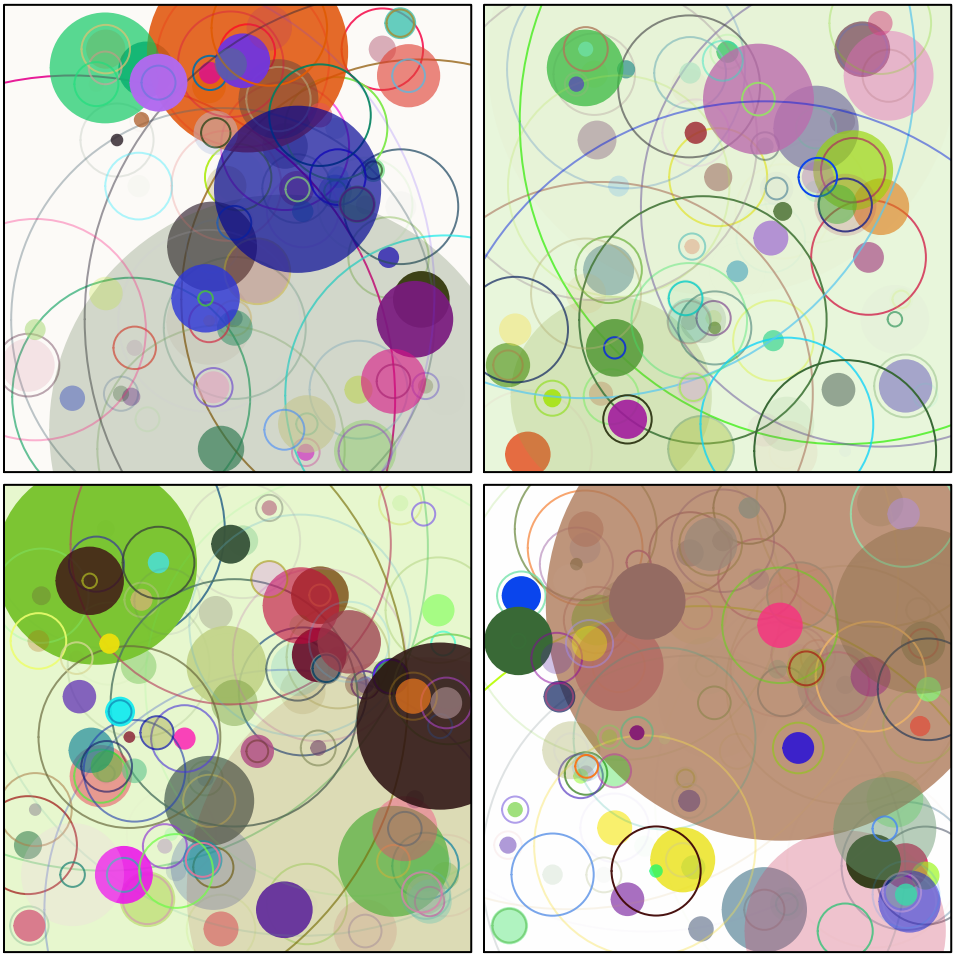
Basic Graphics

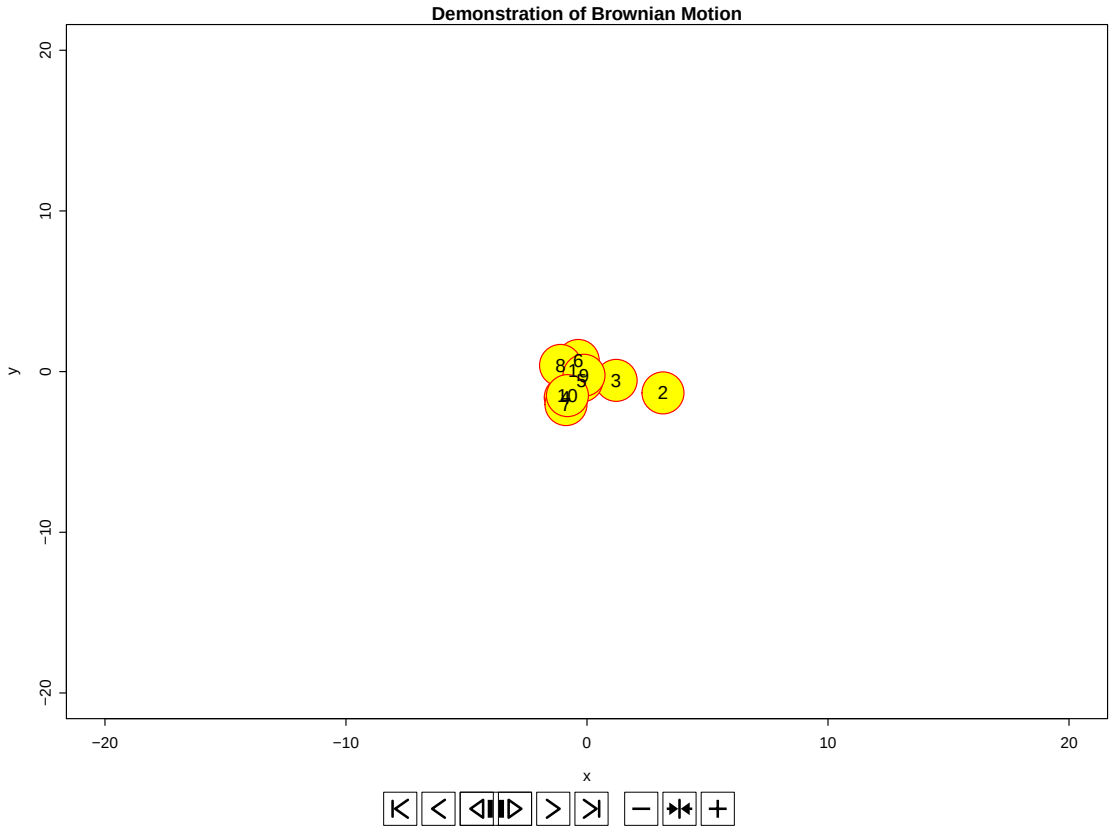
April 19, 2017

Contents

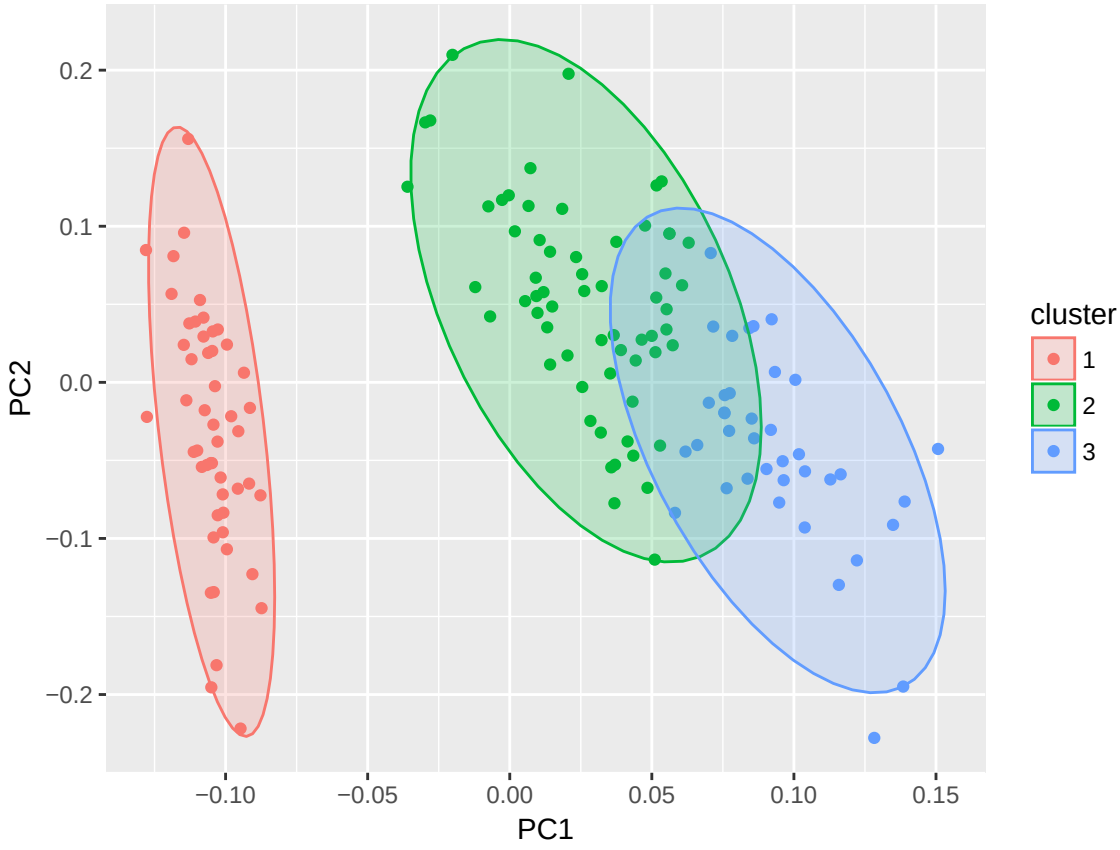
```
par(mar = c(0.2, 0.2, 0.2, 0.2), mfrow = c(2, 2))
for (n in c(63, 60, 76, 74)) {
        set.seed(711)
        plot.new()
        size = c(replicate(n, 1/rbeta(2, 1.5, 4)))
        center = t(replicate(n, runif(2)))
        center = center[rep(1:n, each = 2), ]
        color = apply(replicate(2 * n, sample(c(0:9,
                LETTERS[1:6]), 8, TRUE)), 2, function(x) sprintf("#%s",
        points(center, cex = size, pch = rep(20:21, n),
               col = color)
        box()
}
```



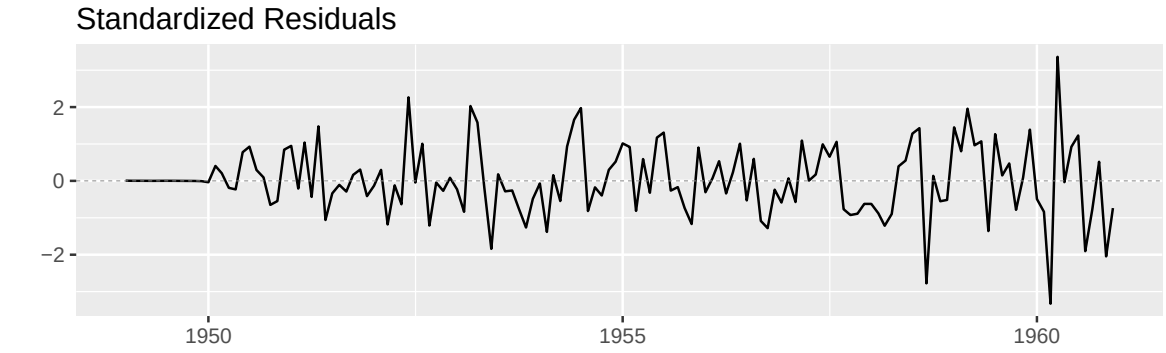
```
library(animation)
saveLatex({
        par(mar = c(3, 3, 1, 0.5), mgp = c(2, 0.5, 0), tcl = -0.3, cex.axis = 0.8,
            cex.lab = 0.8, cex.main = 1)
        brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow",
                        main = "Demonstration of Brownian Motion")
}, img.name = "BM", ani.opts = "controls,loop,width=0.95\\textwidth",
latex.filename = ifelse(interactive(), "brownian_motion.tex", ""),
interval = 0.15, nmax = 30, ani.dev = "pdf", ani.type = "pdf", ani.width = 7,
ani.height = 7, documentclass = paste("\\documentclass{article}",
                                      "\\usepackage[papersize={7in,7in},margin=0.3in]{geometry}",
                                       sep = "\n"))
```

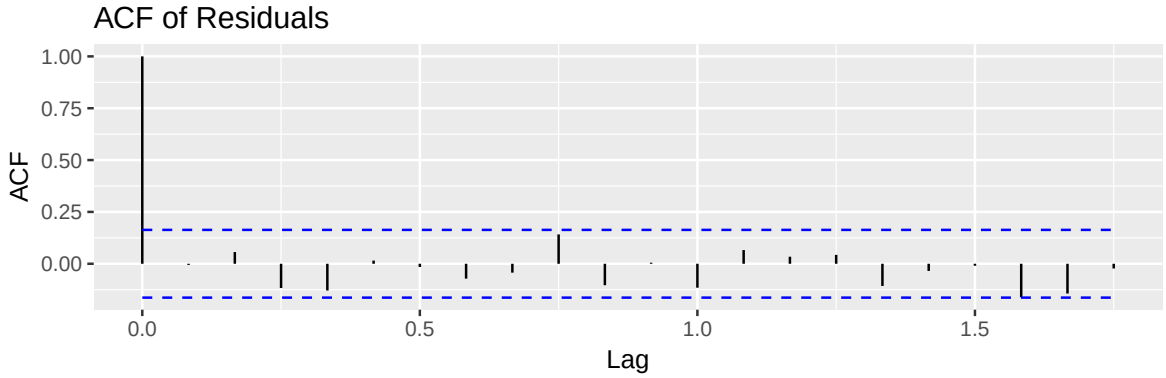


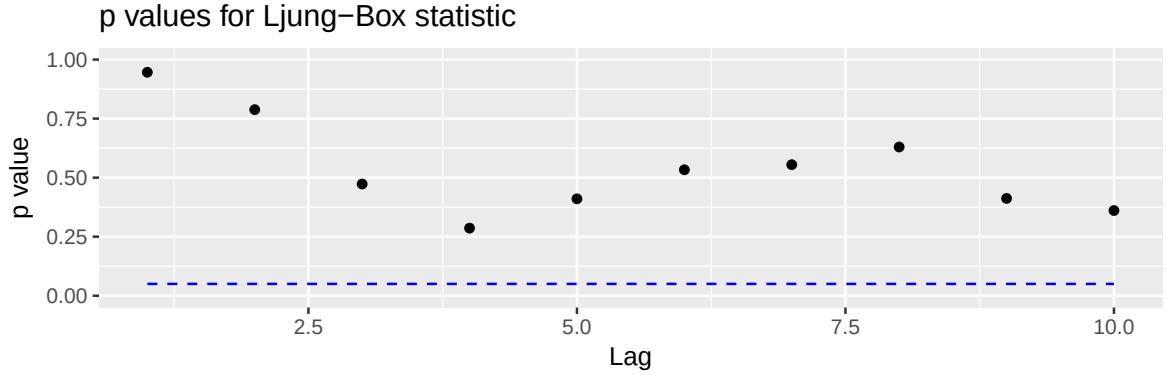
```
## Warning:  package 'ggfortify' was built under R version 3.3.3
## Loading required package:  methods
## Loading required package:  ggplot2
## Warning:  package 'ggplot2' was built under R version 3.3.3
```



```
## Loading required package:  zoo
##
## Attaching package:  'zoo'
## The following objects are masked from 'package:base':
##
##    as.Date, as.Date.numeric
## Loading required package:  timeDate
## This is forecast 7.3
##
## Attaching package:  'forecast'
## The following object is masked from 'package:ggfortify':
##
##    gglagplot
```







1 Basic Graphics

Why do we use graphs in data analysis?

- To understand data properties
- To find patterns in data
- To suggest modeling strategies
- To "debug" analyses
- To communicate results

Some resources

- R Graph Gallery
- R Bloggers

1.1 Plotting Systems in R

In this course, most graphs were produced using R's base graphics system. In addition to base graphics, we have graphics systems provided by the lattice, and ggplot2 packages.

1.1.1 The Base Plotting System ("artist's palette" model)

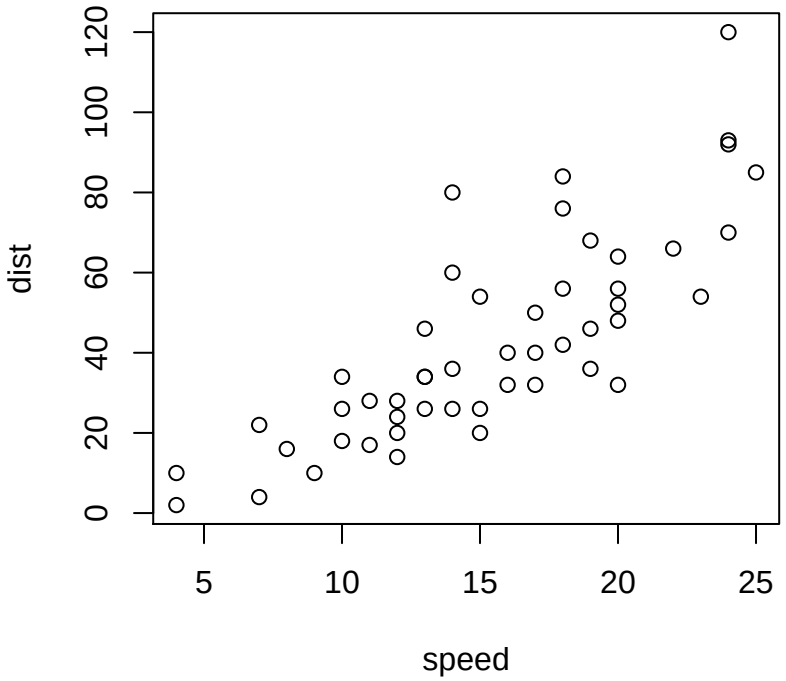
feature

- "Artist's palette" model
- Start with blank canvas and build up from there
- Start with plot function (or similar)
- Use annotation functions to add/modify (text, lines, points, axis)

Advantage and disadvantage

- Convenient, mirrors how we think of building plots and analyzing data
- Can't go back once plot has started (i.e. to adjust margins); need to plan in advance
- Difficult to "translate" to others once a new plot has been created (no graphical "language")
- Plot is just a series of R commands

```
library(datasets)
data(cars)
with(cars,plot(speed,dist))
```



1.1.2 The Lattice System (Entire plot specified by one function; conditioning)

The lattice package , written by Deepayan Sarkar (2008), implements trellis graphics as outlined by Cleveland (1985, 1993) and described on the Trellis website (http:// netlib.bell-labs.com/cm/ms/departments/sia/project/trellis/).

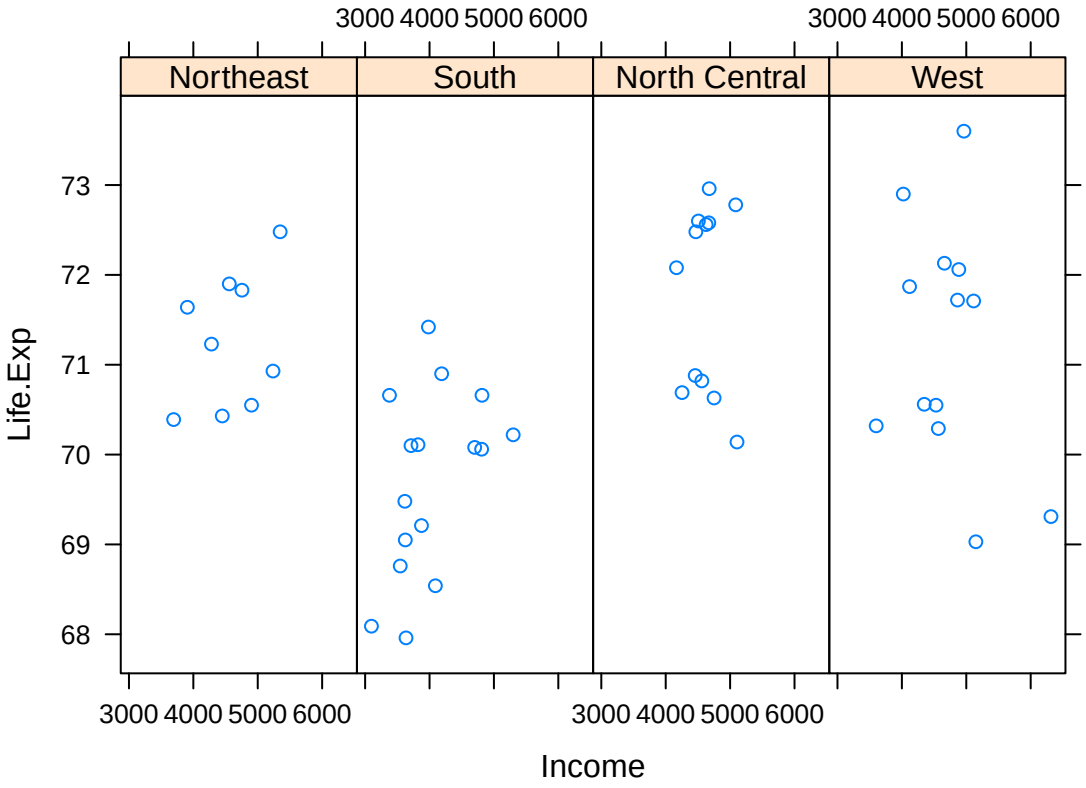
feature

- Plots are created with a single function call (xyplot, bwplot, etc.)
- Most useful for conditioning types of plots: Looking at how y changes with x across levels of z
- Things like margins/spacing set automatically because entire plot is specified at once
- Good for puttng many many plots on a screen

Advantage and disadvantage

- Sometimes awkward to specify an entire plot in a single function call
- Annotation in plot is not especially intuitive
- Use of panel functions and subscripts difficult to wield and requires intense preparation
- Cannot "add" to the plot once it is created

```
library(lattice)
state <- data.frame(state.x77, region = state.region)
xyplot(Life.Exp ~ Income | region, data = state,layout =c(4,1))
```



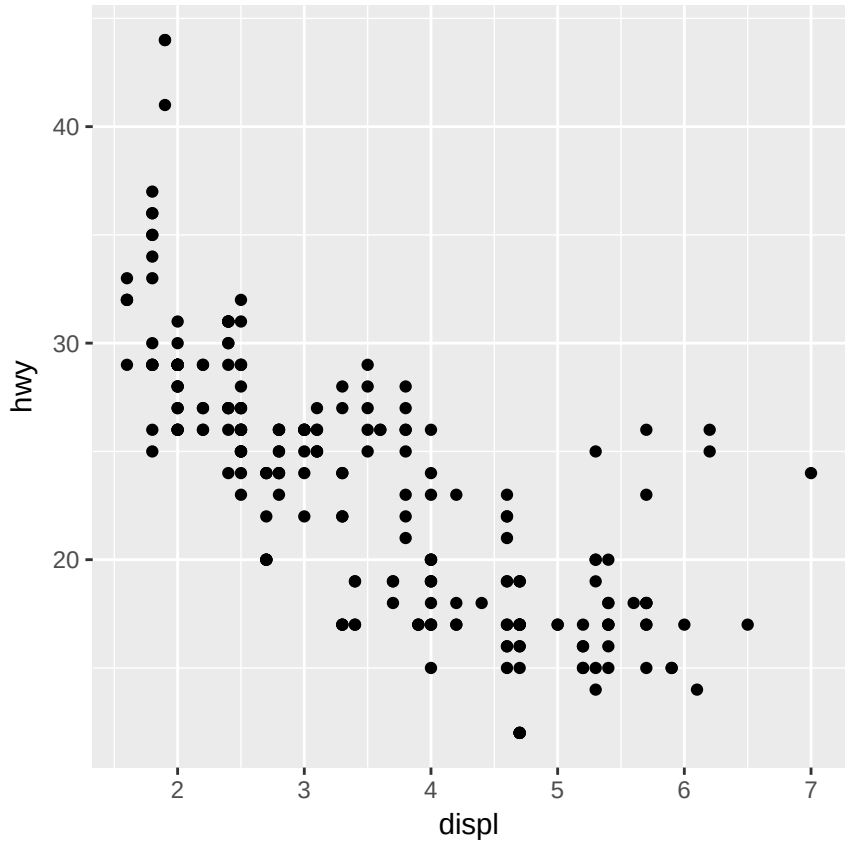
1.1.3 The ggplot2 System (Mixes elements of Base and Lattice)

The ggplot2 package , written by Hadley Wickham (2009a), provides a system for creating graphs based on the grammar of graphics described by Wilkinson (2005) and expanded by Wickham (2009b). The intention of the ggplot2 package is to provide a comprehensive, grammar-based system for generating graphs in a unified and coherent manner, allowing users to create new and innovative data visualizations.

feature

- Splits the difference between base and lattice in a number of ways
- Automatically deals with spacings, text, titles but also allows you to annotate by "adding" to a plot
- Superficial similarity to lattice but generally easier/more intuitive to use
- Default mode makes many choices for you (but you can still customize to your heart's desire)

```
library(ggplot2)
data(mpg)
qplot(displ, hwy, data = mpg)
```




```
rm(list=ls())
```

1.2 What is a Graphics Device?

- A graphics device is something where you can make a plot appear
  - A window on your computer (screen device)
  - A PDF file (file device)
  - A PNG or JPEG file (file device)
  - A scalable vector graphics (SVG) file (file device)
- When you make a plot in R, it has to be "sent" to a specific graphics device
- The most common place for a plot to be "sent" is the screen device
  - On a Mac the screen device is launched with the quartz()
  - On Windows the screen device is launched with windows()
  - On Unix/Linux the screen device is launched with x11()
- When making a plot, you need to consider how the plot will be used to determine what device the plot should be sent to. The list of devices is found in ?Devices.
- For quick visualizations and exploratory analysis, usually you want to use the screen device.

  - Functions like plot in base, xyplot in lattice, or qplot in ggplot2 will default to sending a plot to the screen device.
  - On a given platform (Mac, Windows, Unix/Linux) there is only one screen device.

- For plots that may be printed out or be incorporated into a document (e.g. papers/reports, slide presentations), usually a file device is more appropriate
  - There are many different file devices to choose from

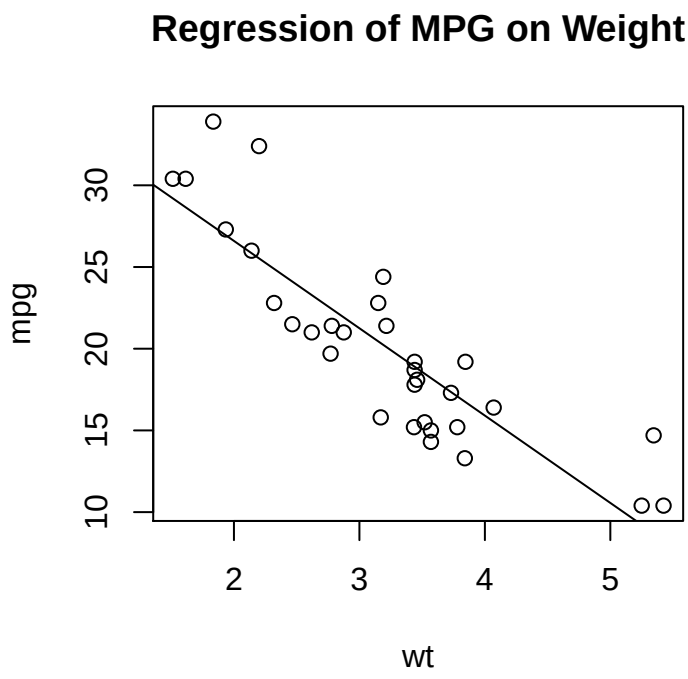
- NOTE: Not all graphics devices are available on all platforms (i.e. you cannot launch the windows() on a Mac)

1.3 How Does a Plot Get Created?

There are two basic approaches to plotting. The first is most common:

1. Call a plotting function like plot, xyplot, or qplot
2. The plot appears on the screen device
3. Annotate plot if necessary
4. Enjoy

```
attach(mtcars)

## The following object is masked from package:ggplot2:
##
##     mpg

   plot(wt, mpg)   ## Make plot appear on screen device
   abline(lm(mpg ~ wt))
   title("Regression of MPG on Weight") ## Annotate with a title
```




```
detach(mtcars)
```

The second approach to plotting is most commonly used for file devices:

1. Explicitly launch a graphics device
2. Call a plotting function to make a plot (Note: if you are using a file device, no plot will appear on the screen)
3. Annotate plot if necessary
4. Explicitly close graphics device with dev.off() (this is very important!)

```
pdf("mygraph.pdf")
   attach(mtcars)
   plot(wt, mpg)
   abline(lm(mpg ~ wt))
   title("Regression of MPG on Weight")
   detach(mtcars)
dev.off()
```

The first statement attaches the data frame mtcars. The second statement opens a graphics window and generates a scatter plot between automobile weight on the horizontal axis and miles per gallon on the vertical axis. The third statement adds a line of best fit. The fourth statement adds a title. In addition to pdf(), you can use the functions win.metafile(), png(), jpeg(), bmp(), tiff(), xfig(), and postscript() to save graphs in other formats.

Saving graphs via the GUI will be platform specific. On a Windows platform, select File > Save As from the graphics window, and choose the format and location desired in the resulting dialog. Creating a new graph

by issuing a high-level plotting command such as plot(), hist() (for histograms), or boxplot() will typically overwrite a previous graph.

Copying a plot to another device can be useful because some plots require a lot of code and it can be a pain to type all that in again for a different device.

- dev.copy: copy a plot from one device to another
- dev.copy2pdf: specifically copy a plot to a PDF file

```
attach(mtcars)
   plot(wt, mpg)   ## Make plot appear on screen device
   abline(lm(mpg ~ wt))
   title("Regression of MPG on Weight") ## Annotate with a title
   dev.copy(png, file = "testpng.png") ## Copy my plot to a PNG file
   dev.off() ## Don't forget to close the PNG device!
detach(mtcars)
```

1.4 Plot() function

The following data describes patient response to two drugs at five dosage levels.

```
dose <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)
plot(dose, drugA, type = "b")
```

plot() is a generic function that plots objects in R (its output will vary according to the type of object being plotted). In this case, plot(x, y, type="b") places x on the horizontal axis and y on the vertical axis, plots the (x, y) data points, and connects them with line segments. The option type="b" indicates that both points and lines should be plotted. Use help(plot) to view other options.

1.5 Graphical parameters

You can customize many features of a graph (fonts, colors, axes, titles) through options called graphical parameters.

One way is to specify these options through the par() function. Values set in this manner will be in effect for the rest of the session or until they're changed. The format is par(optionname=value, optionname=value, ...). Specifying par() without parameters produces a list of the current graphical settings. Adding the no.readonly=TRUE option produces a list of current graphical settings that can be modified.

```
par()
opar <- par(no.readonly=TRUE)
par(lty=2, pch=17)
plot(dose, drugA, type="b")
par(opar)
plot(dose, drugA, type="b")
example(points)
```

The first statement makes a copy of the current settings. The second statement changes the default line type to dashed (lty=2) and the default symbol for plotting points to a solid triangle (pch=17). You then generate the plot and restore the original settings.

A second way to specify graphical parameters is by providing the optionname=value pairs directly to a high-level plotting function. In this case, the options are only in effect for that specific graph.

```
plot(dose, drugA, type="b", lty=2, pch=17)
```

Not all high-level plotting functions allow you to specify all possible graphical parameters. See the help for a specific plotting function (such as ?plot, ?hist, or ?boxplot) to determine which graphical parameters can be set in this way.

Symbols and lines

```
pch #Specifies the symbol to use when plotting points.
cex #Specifies the symbol size. 1=default, 1.5 is 50% larger, 0.5 is 50% smaller, and so forth.
lty #Specifies the line type.
lwd #Specifies the line width. lwd is expressed relative to the default (default=1).
```

Taking these options together, the code

```
plot(dose, drugA, type="b", lty=3, lwd=3, pch=15, cex=2)
```

would produce a plot with a dotted line that was three times wider than the default width, connecting points displayed as filled squares that are twice as large as the default symbol size. Try ?pch.

Color

There are several color-related parameters in R. You can specify colors in R by index, name, hexadecimal, RGB, or HSV. For example, col=1, col="white", col="#FFFFFF", col=rgb(1,1,1), and col=hsv(0,0,1) are equivalent ways of specifying the color white. The function rgb()creates colors based on red-green-blue values, whereas hsv() creates colors based on hue-saturation values. See the help feature on these functions for more details.

The function colors() returns all available color names. Earl F. Glynn has created an excellent online chart of R colors, available at http://research.stowers-institute. org/efg/R/Color/Chart. R also has a number of functions that can be used to create vectors of contiguous colors. These include rainbow(), heat.colors(), terrain. colors(), topo.colors(), and cm.colors(). For example, rainbow(10) produces 10 contiguous “rainbow" colors. Gray levels are generated with the gray() function. In this case, you specify gray levels as a vector of numbers between 0 and 1. gray(0:10/10) would produce 10 gray levels.

```
n <- 10
mycolors <- rainbow(n)
pie(rep(1, n), labels=mycolors, col=mycolors)
mygrays <- gray(0:n/n)
pie(rep(1, n), labels=mygrays, col=mygrays)
```

Parameters for specifying color are shown as following.

```
col #Default plotting color.
col.axis #Color for axis text.
col.lab #Color for axis labels.
col.main #Color for titles.
col.sub #Color for subtitles.
fg #The plot's foreground color.
bg #The plot's background color.
```

Text characteristics

Graphic parameters are also used to specify text size, font, and style. Font family and style can be controlled with font options.

```
cex #Number indicating the amount by which plotted text should be scaled relative to the default.
cex.axis #Magnification of axis text relative to cex.
cex.lab #Magnification of axis labels relative to cex.
cex.main #Magnification of titles relative to cex.
cex.sub #Magnification of subtitles relative to cex.
font #Integer specifying font to use for plotted text..
1=plain, 2=bold, 3=italic, 4=bold italic, 5=symbol.
font.axis #Font for axis text.
font.lab #Font for axis labels.
font.main #Font for titles.
font.sub #Font for subtitles.
ps #Font point size (roughly 1/72 inch). The text size = ps*cex.
```

For example, all graphs created after the statement

```
par(font.lab=3, cex.lab=1.5, font.main=4, cex.main=2)
```

will have italic axis labels that are 1.5 times the default text size, and bold italic titles that are twice the default text size.

If graphs will be output in PDF or PostScript format, changing the font family is relatively straightforward. For PDFs, use names(pdfFonts()) to find out which fonts are available on your system and pdf(file="myplot.pdf", family="fontname") to generate the plots.

```
names(pdfFonts())
pdf(file="myplot.pdf", family="fontname")
pie(rep(1, n), labels=mycolors, col=mycolors)
dev.off()
```

Graph and margin dimensions

Finally, you also can control the plot dimensions and margin sizes using the following parameters.

```
pin #Plot dimensions (width, height) in inches.
mai #Numerical vector indicating margin size where c(bottom,
     left, top, right) is expressed in inches.
mar #Numerical vector indicating margin size
     where c(bottom, left, top, right) is expressed in lines.
     The default is c(5, 4, 4, 2) + 0.1.
```

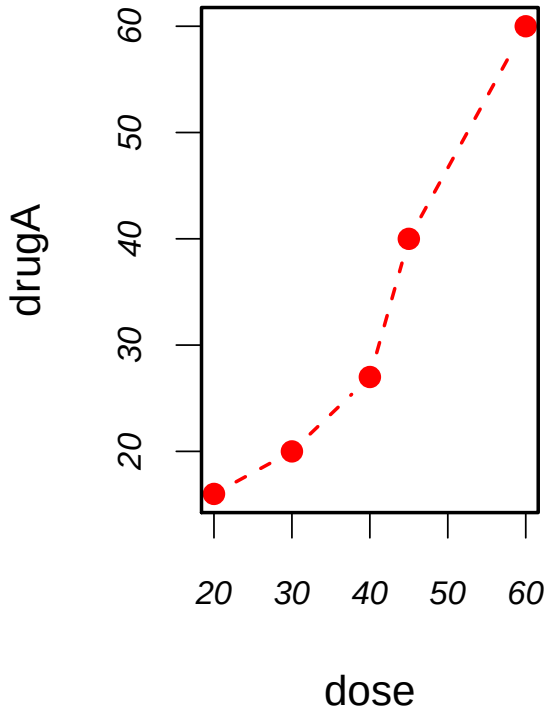
The code

```
par(pin=c(4,3), mai=c(1,.5, 1, .2))
```

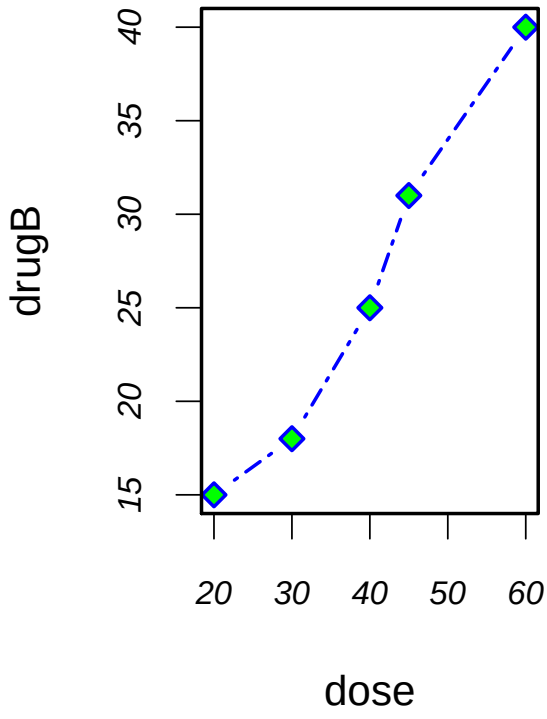
produces graphs that are 4 inches wide by 3 inches tall, with a 1-inch margin on the bottom and top, a 0.5-inch margin on the left, and a 0.2-inch margin on the right. For a complete tutorial on margins, see Earl F. Glynn's comprehensive online tutorial.
(http://research.stowers-institute.org/efg/R/Graphics/Basics/mar-oma/)

An Example

```
dose <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)
opar <- par(no.readonly=TRUE)
par(pin=c(2, 3))
par(lwd=2, cex=1.5)
par(cex.axis=.75, font.axis=3)
plot(dose, drugA, type="b", pch=19, lty=2, col="red")
```




```
plot(dose, drugB, type="b", pch=23, lty=6, col="blue", bg="green")
```



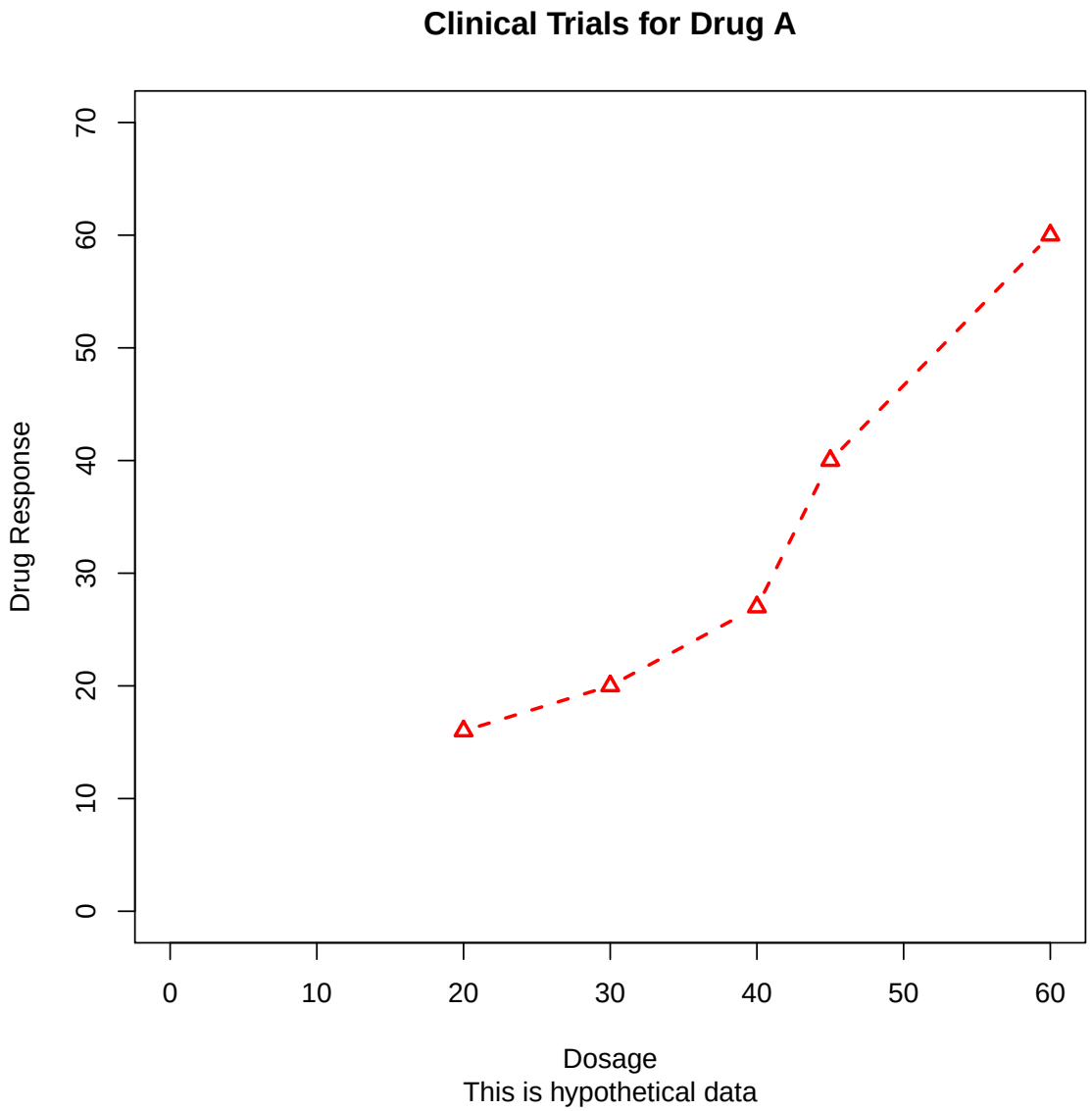

```
par(opar)
```

First you enter your data as vectors, then save the current graphical parameter settings (so that you can restore them later). You modify the default graphical parameters so that graphs will be 2 inches wide by 3 inches tall. Additionally, lines will be twice the default width and symbols will be 1.5 times the default size. Axis text will be set to italic and scaled to 75 percent of the default. The first plot is then created using filled red circles and dashed lines. The second plot is created using filled green filled diamonds and a blue border and blue dashed lines. Finally, you restore the original graphical parameter settings. Note that parameters set with the par() function apply to both graphs, whereas parameters specified in the plot functions only apply to that specific graph.

1.6 Add text, customized axes, and legends

Many high-level plotting functions (for example, plot, hist, boxplot) allow you to include axis and text options, as well as graphical parameters. For example, the following adds a title (main), subtitle (sub), axis labels (xlab, ylab), and axis ranges (xlim, ylim).

```
plot(dose, drugA, type="b",
     col="red", lty=2, pch=2, lwd=2,
     main="Clinical Trials for Drug A",
     sub="This is hypothetical data",
     xlab="Dosage", ylab="Drug Response",
     xlim=c(0, 60), ylim=c(0, 70))
```



Some high-level plotting functions include default titles and labels. You can remove them by adding ann=FALSE in the plot() statement or in a separate par() statement.

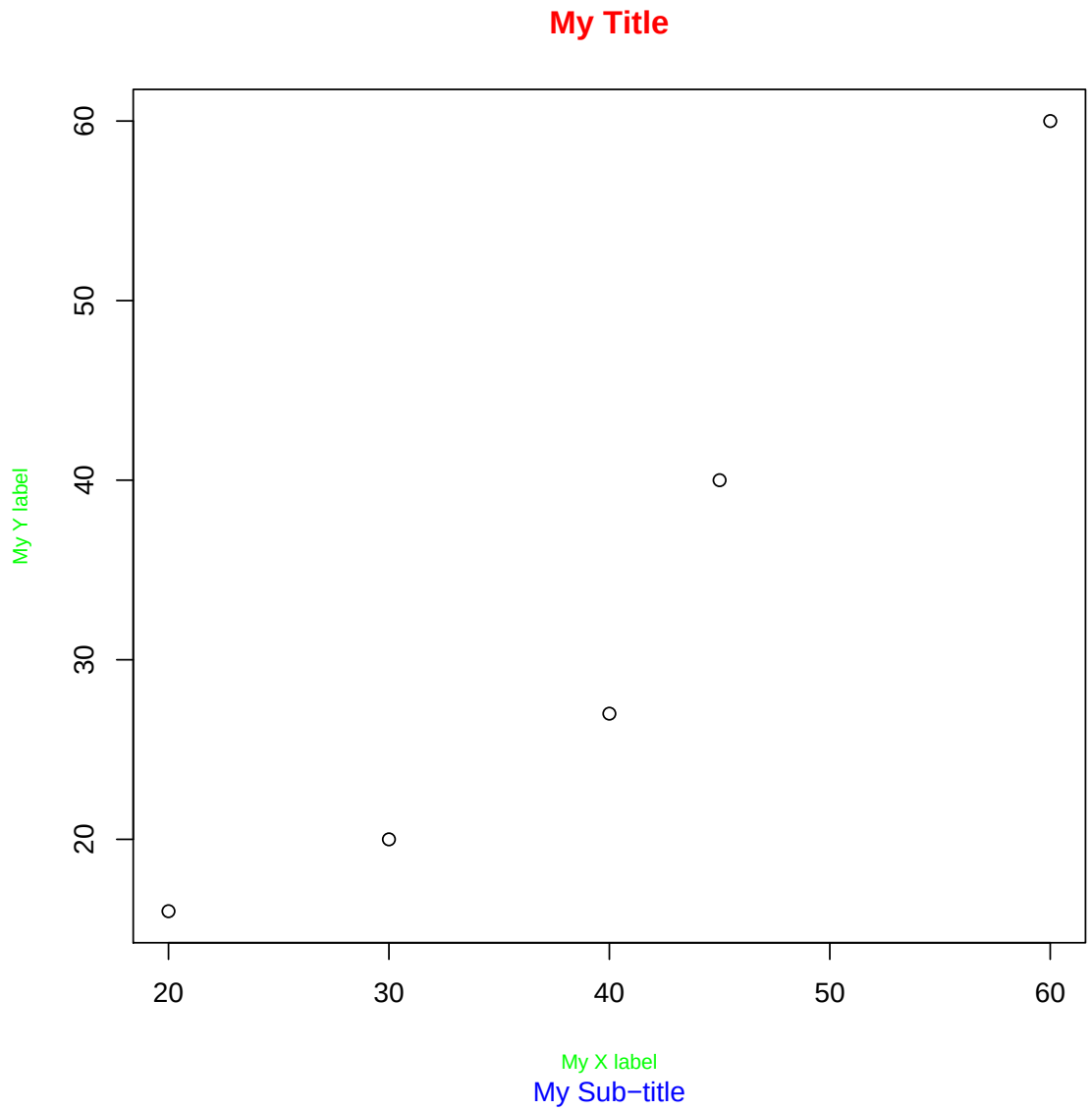
Titles

Use the title() function to add title and axis labels to a plot.

```
title(main="main title", sub="sub-title",
      xlab="x-axis label", ylab="y-axis label")
```

Graphical parameters (such as text size, font, rotation, and color) can also be specified in the title() function.

```
plot(dose, drugA,
     title(main="My Title", col.main="red",
     sub="My Sub-title", col.sub="blue",
     xlab="My X label", ylab="My Y label",
     col.lab="green", cex.lab=0.75), ann=FALSE)
```



The above statement produces a red title and a blue subtitle, and creates green x and y labels that are 25 percent smaller than the default text size.

Axes

Rather than using R's default axes, you can create custom axes with the axis() function.

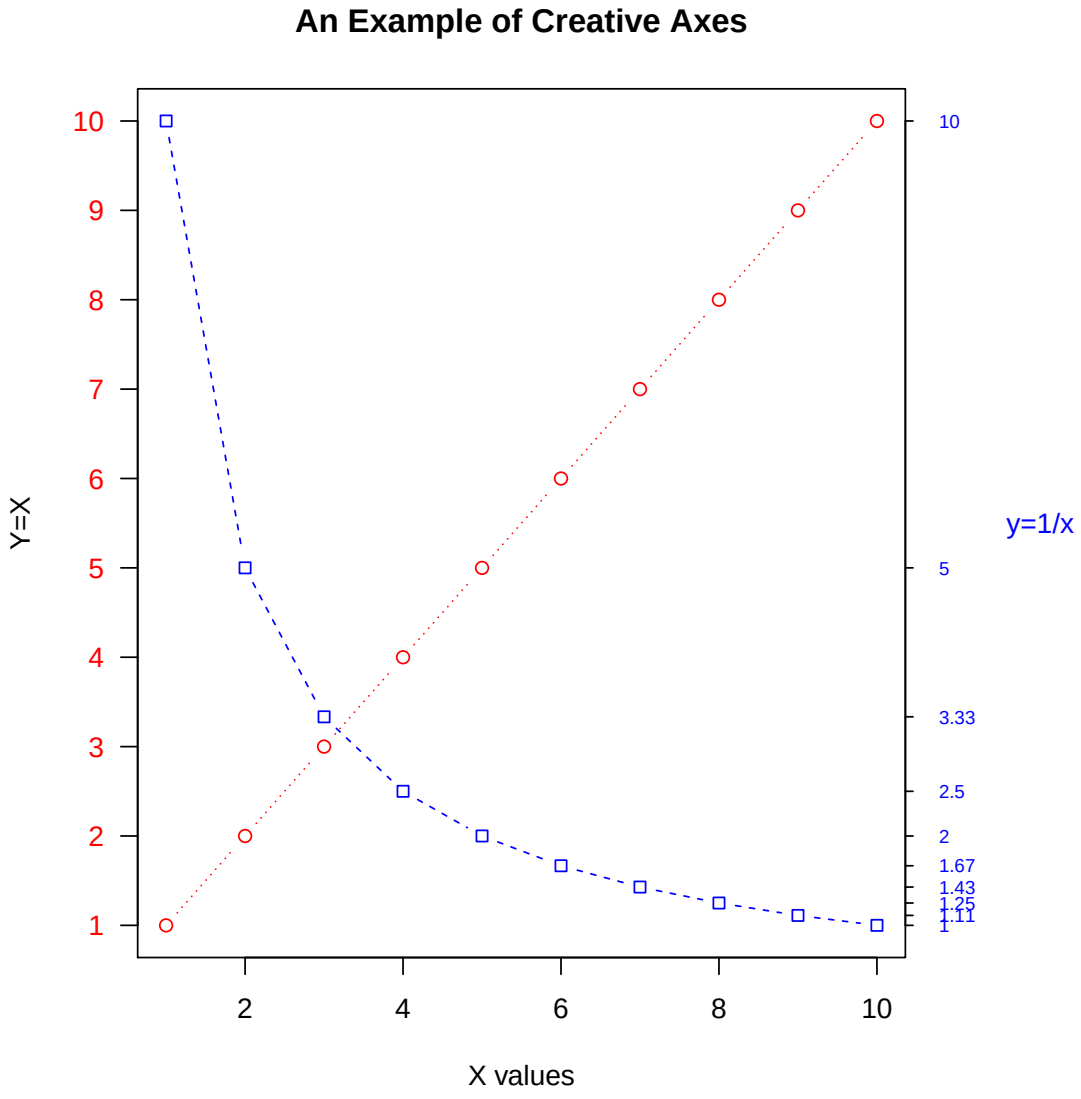
```
axis(side, at=, labels=, pos=, lty=, col=, las=, tck=)
```

When creating a custom axis, you should suppress the axis automatically generated by the high-level plotting function. The option axes=FALSE suppresses all axes (including all axis frame lines, unless you add the option frame.plot=TRUE). The options xaxt="n" and yaxt="n" suppress the x- and y-axis, respectively (leaving the frame lines, without ticks).

```
side #An integer indicating the side of the graph to draw the axis
     (1=bottom, 2=left, 3=top, 4=right).
at #A numeric vector indicating where tick marks should be drawn.
   labels #A character vector of labels to be placed at the tick marks
   (if NULL, the at values will be used).
pos #The coordinate at which the axis line is to be drawn
    (that is, the value on the other axis where it crosses).
lty #Line type.
col #The line and tick mark color.
las #Labels are parallel (=0) or perpendicular (=2) to the axis.
tck #Length of tick mark as a fraction of the plotting region
    (a negative number is outside the graph, a positive number is inside,
    0 suppresses ticks, 1 creates gridlines); the default is -0.01.
```

An Example

```
x <- c(1:10)
y <- x
z <- 10/x
opar <- par(no.readonly = TRUE)
par(mar = c(5, 4, 4, 8) + 0.1)
plot(x, y, type = "b", pch = 21, col = "red", yaxt = "n",
     lty = 3, ann = FALSE)
lines(x, z, type = "b", pch = 22, col = "blue", lty = 2)
axis(2, at = x, labels = x, col.axis = "red", las = 2)
axis(4, at = z, labels = round(z, digits = 2), col.axis = "blue",
     las = 2, cex.axis = 0.7, tck = -0.01)
mtext("y=1/x", side = 4, line = 3, cex.lab = 1, las = 2,        col = "blue")
title("An Example of Creative Axes", xlab = "X values",        ylab = "Y=X")
par(opar)
```



Minor tick marks

Notice that each of the graphs you've created so far have major tick marks but not minor tick marks. To create minor tick marks, you'll need the minor.tick() function in the Hmisc package. If you don't already have Hmisc installed, be sure to install it first.

```
install.package(Hmisc)
minor.tick(nx=2, ny=3, tick.ratio=0.5)
```

The length of the tick marks will be 50 percent as long as the major tick marks.

Reference lines

The abline() function is used to add reference lines to our graph.

```
abline(v=seq(1, 10, 2), lty=2, col="blue")
```

adds dashed blue vertical lines at x = 1, 3, 5, 7, and 9.

Legend

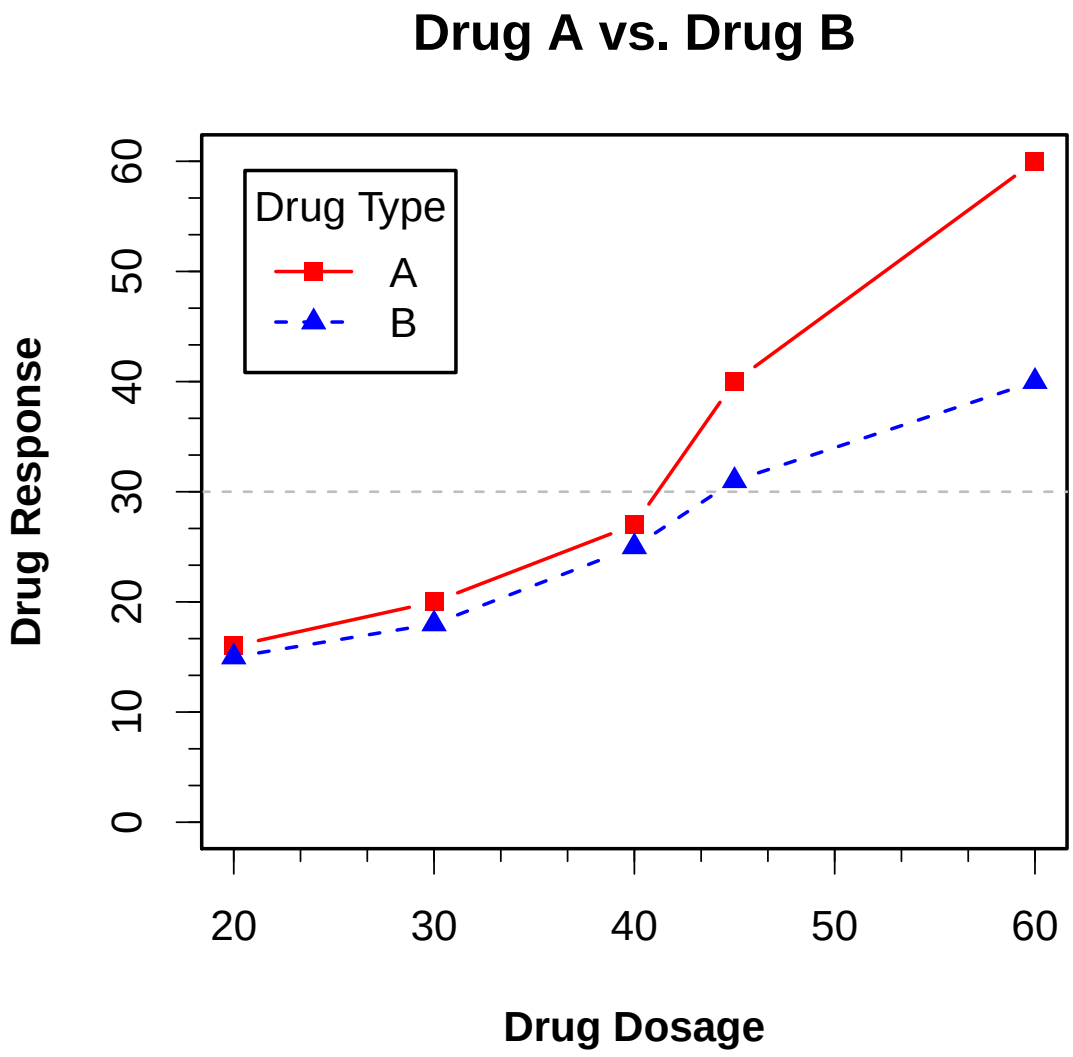
When more than one set of data or group is incorporated into a graph, a legend can help you to identify what's being represented by each bar, pie slice, or line. A legend can be added (not surprisingly) with the legend() function.

```
legend(location, title, legend)
```

For more on legends, see help(legend).

An Example

```
dose <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)
opar <- par(no.readonly = TRUE)
par(lwd = 2, cex = 1.5, font.lab = 2)
plot(dose, drugA, type = "b", pch = 15, lty = 1, col = "red",
    ylim = c(0, 60), main = "Drug A vs. Drug B",
    xlab = "Drug Dosage",      ylab = "Drug Response")
lines(dose, drugB, type = "b", pch = 17, lty = 2,      col = "blue")
abline(h = c(30), lwd = 1.5, lty = 2, col = "grey")
library(Hmisc)
minor.tick(nx = 3, ny = 3, tick.ratio = 0.5)
legend("topleft", inset = 0.05, title = "Drug Type", c("A", "B"),
    lty = c(1, 2), pch = c(15, 17), col = c("red", "blue"))
par(opar)
```



Text annotations

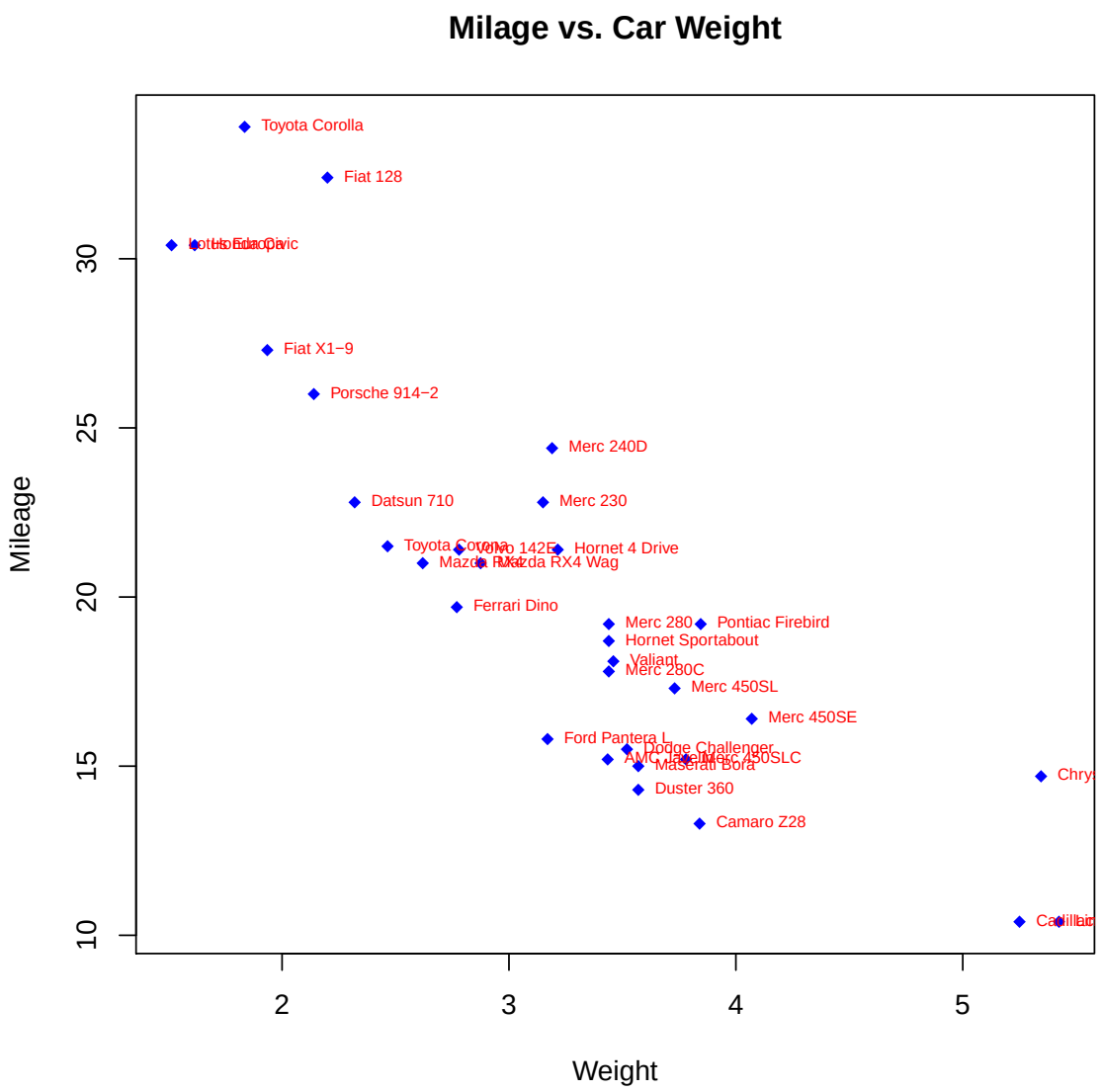
Text can be added to graphs using the text() and mtext() functions. text() places text within the graph whereas mtext() places text in one of the four margins.

```
text(location, "text to place", pos)
mtext("text to place", side, line=n)
```

See ?text() and ?mtext() for the details of the common options.

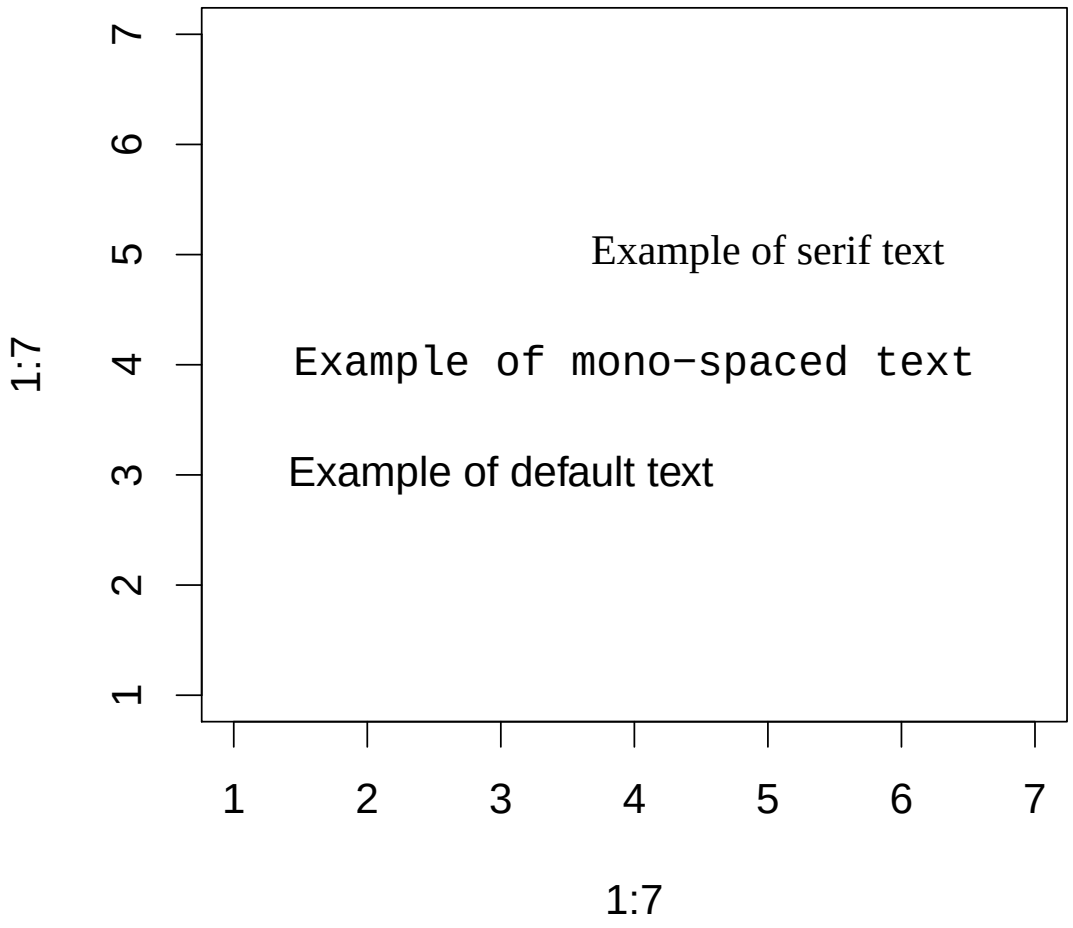
```
attach(mtcars)
plot(wt, mpg, main = "Milage vs. Car Weight", xlab = "Weight", ylab = "Mileage",
    pch = 18, col = "blue")
text(wt, mpg, row.names(mtcars), cex = 0.6, pos = 4, col = "red")
detach(mtcars)
```

```
## The following object is masked from package:ggplot2:
##
##     mpg
```



the following code can be used to display font families

```
opar <- par(no.readonly=TRUE)
par(cex=1.5)
plot(1:7,1:7,type="n")
text(3,3,"Example of default text")
text(4,4,family="mono","Example of mono-spaced text")
text(5,5,family="serif","Example of serif text")
```

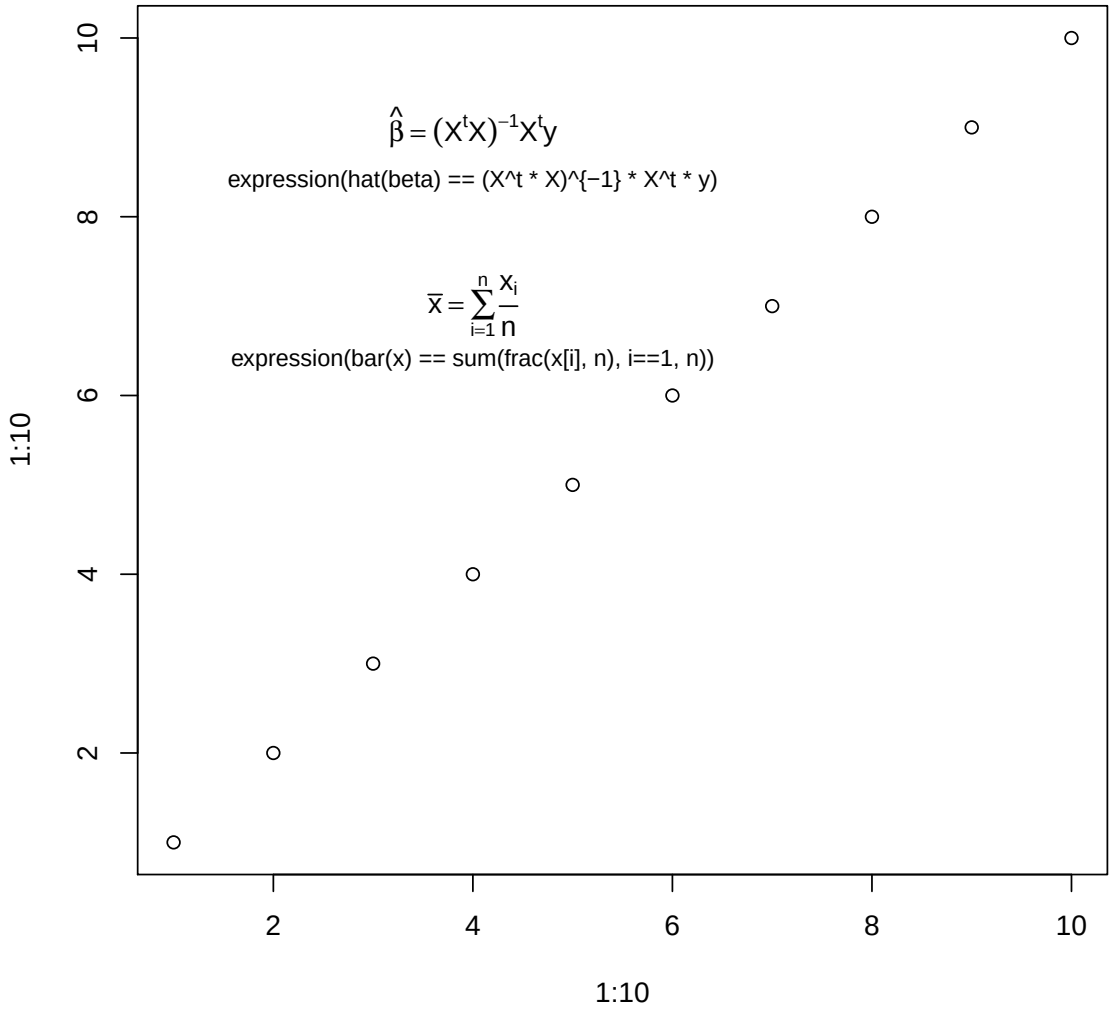



```
par(opar)
```

Math annotations

Finally, you can add mathematical symbols and formulas to a graph using TEX-like rules. See help(plotmath) for details and examples. You can also try demo(plotmath) to see this in action. The plotmath() function can be used to add mathematical symbols to titles, axis labels, or text annotation in the body or margins of the graph.

```
plot(1:10, 1:10)
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(4, 8.4, "expression(hat(beta) == (X^t * X)^{-1} * X^t * y)",      cex = .8)
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))
text(4, 6.4, "expression(bar(x) == sum(frac(x[i], n), i==1, n))",      cex = .8)
```



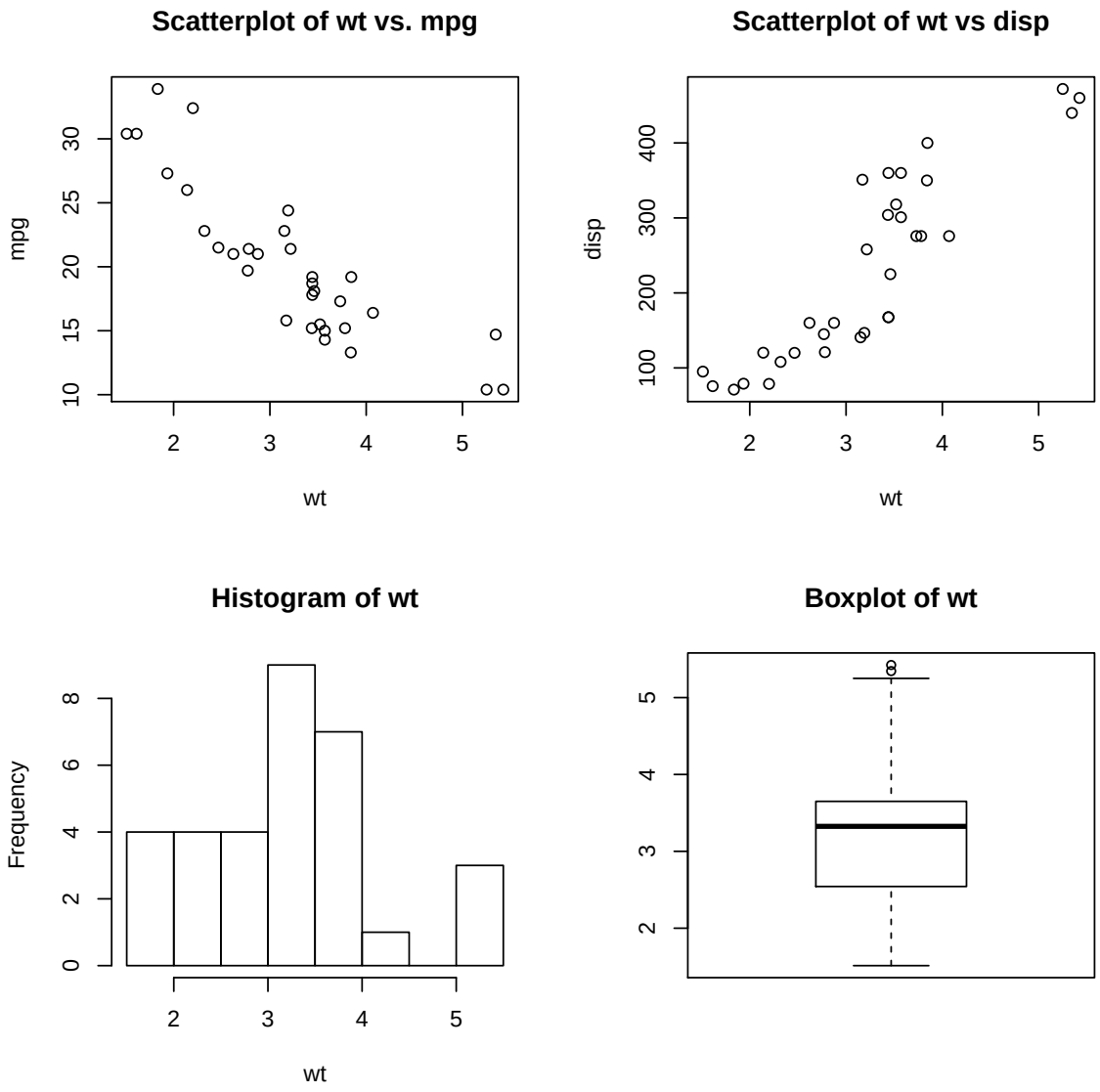
Combining graphs

R makes it easy to combine several graphs into one overall graph, using either the par() or layout() function. At this point, don't worry about the specific types of graphs being combined; our focus here is on the general methods used to combine them.

With the par() function, you can include the graphical parameter mfrow=c(nrows, ncols) to create a matrix of nrows x ncols plots that are filled in by row. Alternatively, you can use mfcol=c(nrows, ncols) to fill the matrix by columns.

```
attach(mtcars)

## The following object is masked from package:ggplot2:
##
##     mpg
```

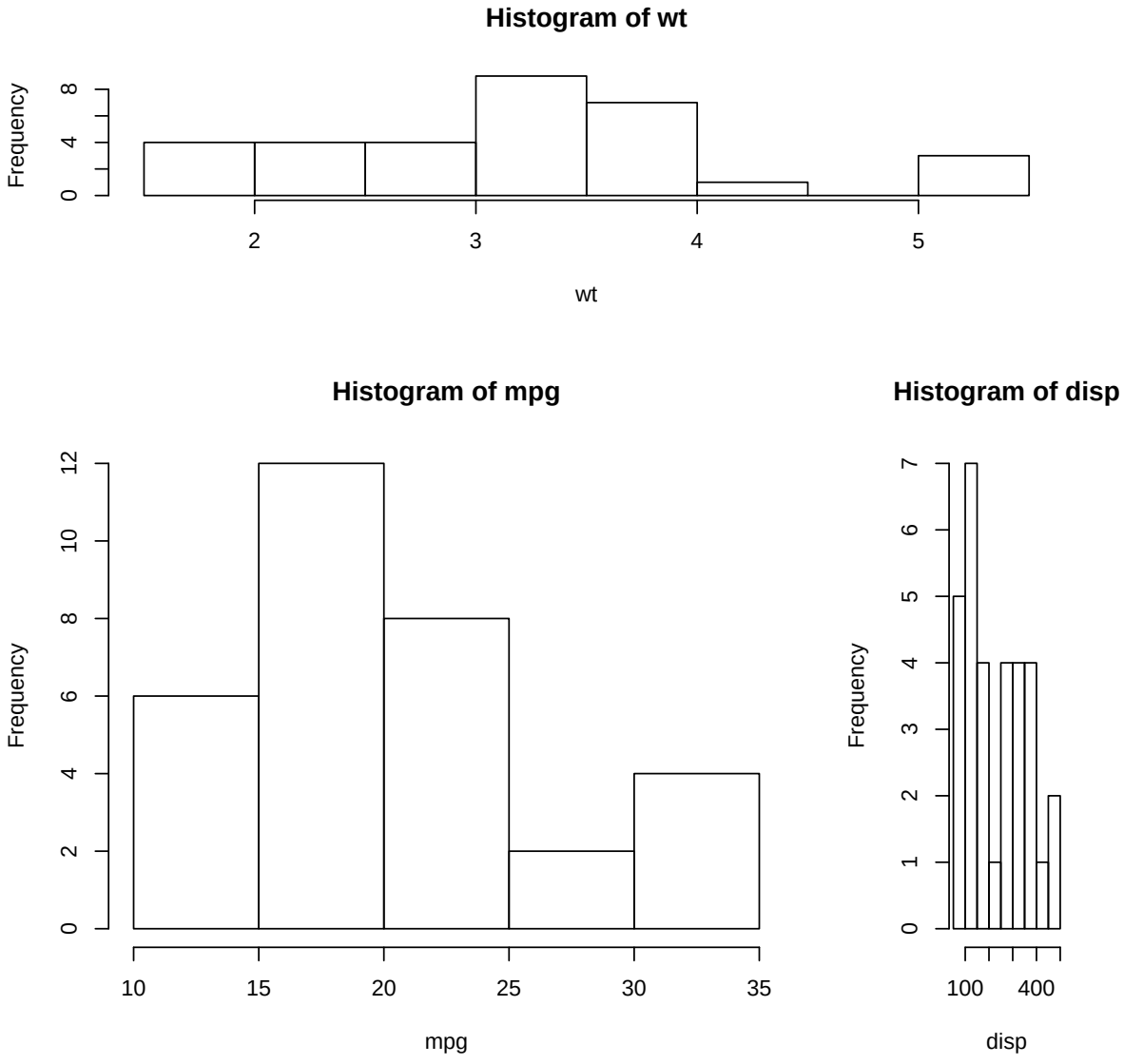
```
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(wt,mpg, main="Scatterplot of wt vs. mpg")
plot(wt,disp, main="Scatterplot of wt vs disp")
hist(wt, main="Histogram of wt")
boxplot(wt, main="Boxplot of wt")
```




```
par(opar)
detach(mtcars)
```

The layout() function has the form layout(mat) where mat is a matrix object specifying the location of the multiple plots to combine. Optionally, you can include widths= and heights= options in the layout() function to control the size of each figure more precisely. These options have the form widths = a vector of values for the widths of columns heights = a vector of values for the heights of rows.

```
attach(mtcars)

## The following object is masked from package:ggplot2:
##
##     mpg

layout(matrix(c(1, 1, 2, 3), 2, 2, byrow = TRUE), widths=c(3, 1), heights=c(1, 2))
hist(wt)
hist(mpg)
hist(disp)
```



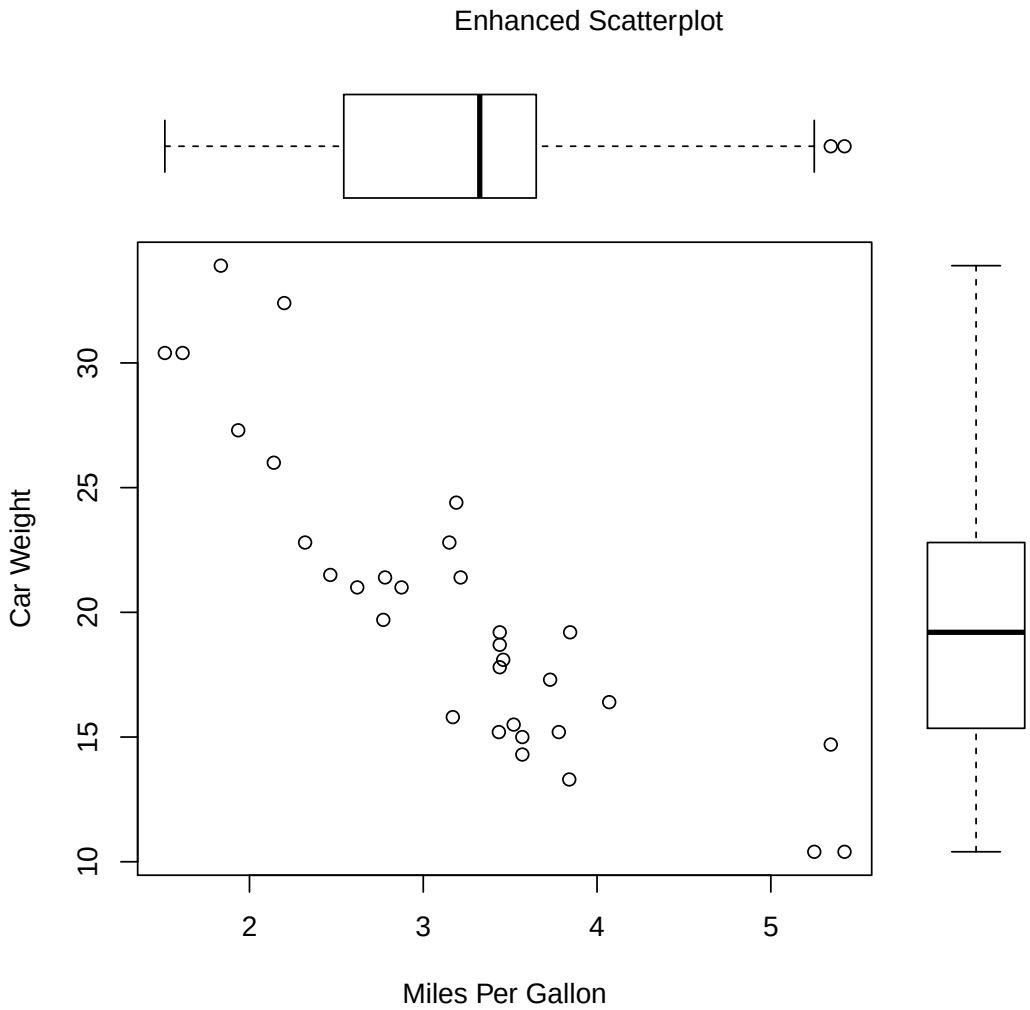

```
detach(mtcars)
```

As you can see, the layout() function gives you easy control over both the number and placement of graphs in a final image and the relative sizes of these graphs. See help(layout) for more details.

Creating a figure arrangement with fine control

There are times when you want to arrange or superimpose several figures to create a single meaningful plot. Doing so requires fine control over the placement of the figures. You can accomplish this with the fig= graphical parameter.

```
opar <- par(no.readonly = TRUE)
par(fig = c(0, 0.8, 0, 0.8))
plot(mtcars$wt, mtcars$mpg, xlab = "Miles Per Gallon", ylab = "Car Weight")
par(fig = c(0, 0.8, 0.5, 1), new = TRUE)
boxplot(mtcars$wt, horizontal = TRUE, axes = FALSE)
par(fig = c(0.6, 1, 0, 0.8), new = TRUE)
boxplot(mtcars$mpg, axes = FALSE)
mtext("Enhanced Scatterplot", side = 3, outer = TRUE, line = -5)
```



```
par(opar)
```

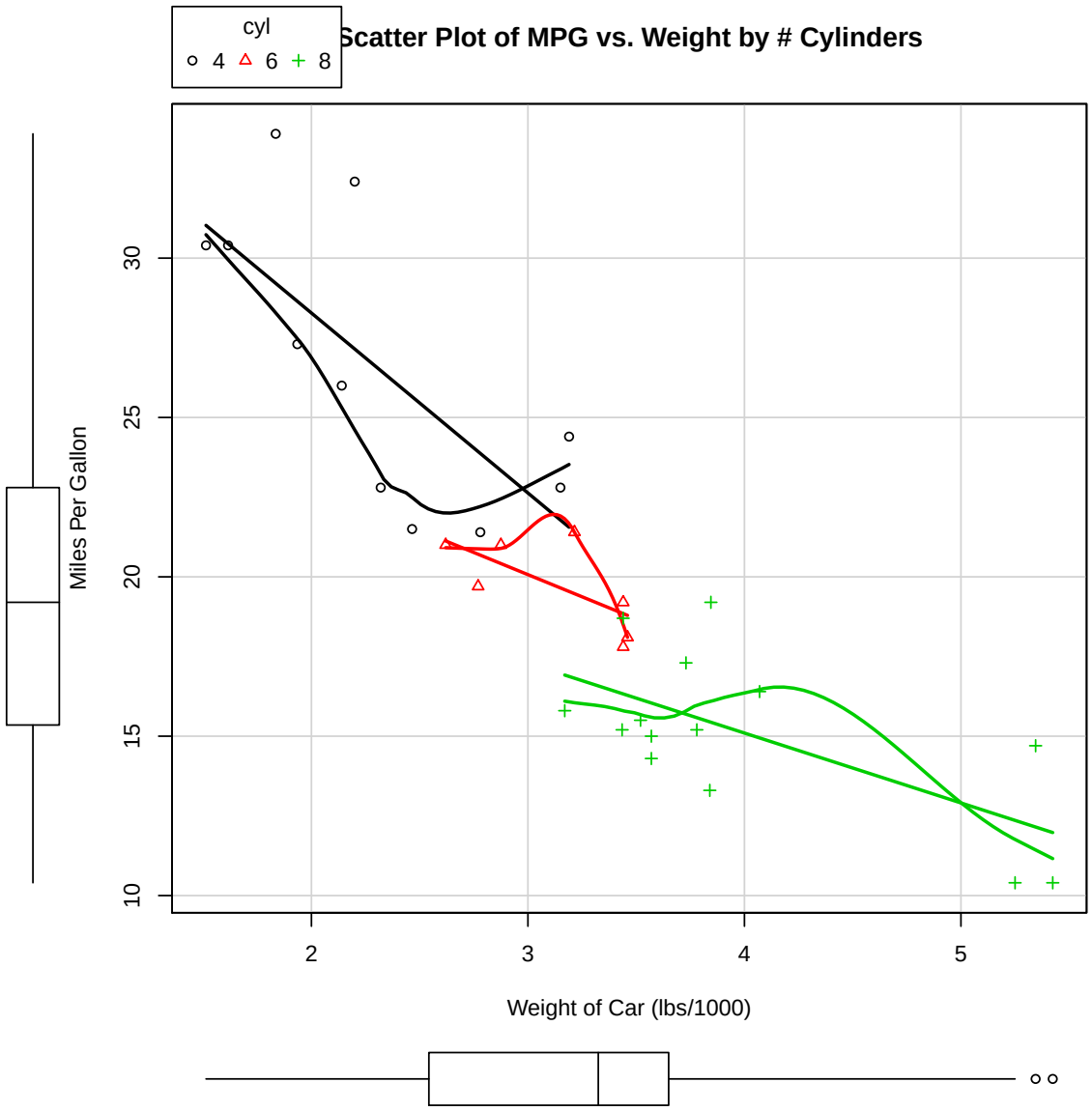
To understand how this graph was created, think of the full graph area as going from (0,0) in the lower-left corner to (1,1) in the upper-right corner. The format of the fig= parameter is a numerical vector of the form c(x1, x2, y1, y2).

The first fig= sets up the scatter plot going from 0 to 0.8 on the x-axis and 0 to 0.8 on the y-axis. The top box plot goes from 0 to 0.8 on the x-axis and 0.5 to 1 on the y-axis. The right-hand box plot goes from 0.6 to 1 on the x-axis and 0 to 0.8 on the y-axis. fig= starts a new plot, so when adding a figure to an existing graph, include the new=TRUE option.

1.7 Scatter plots

The s catterplot() function in the c ar package offers many enhanced features and convenience functions for producing scatter plots, including fit lines, marginal box plots, confidence ellipses, plotting by subgroups, and interactive point identification.

```
library(car)
scatterplot(mpg ~ wt | cyl, data=mtcars, lwd=2,
main="Scatter Plot of MPG vs. Weight by # Cylinders",
xlab="Weight of Car (lbs/1000)",
ylab="Miles Per Gallon",
legend.plot=TRUE,
id.method="identify",
labels=row.names(mtcars),
boxplots="xy"
)
```



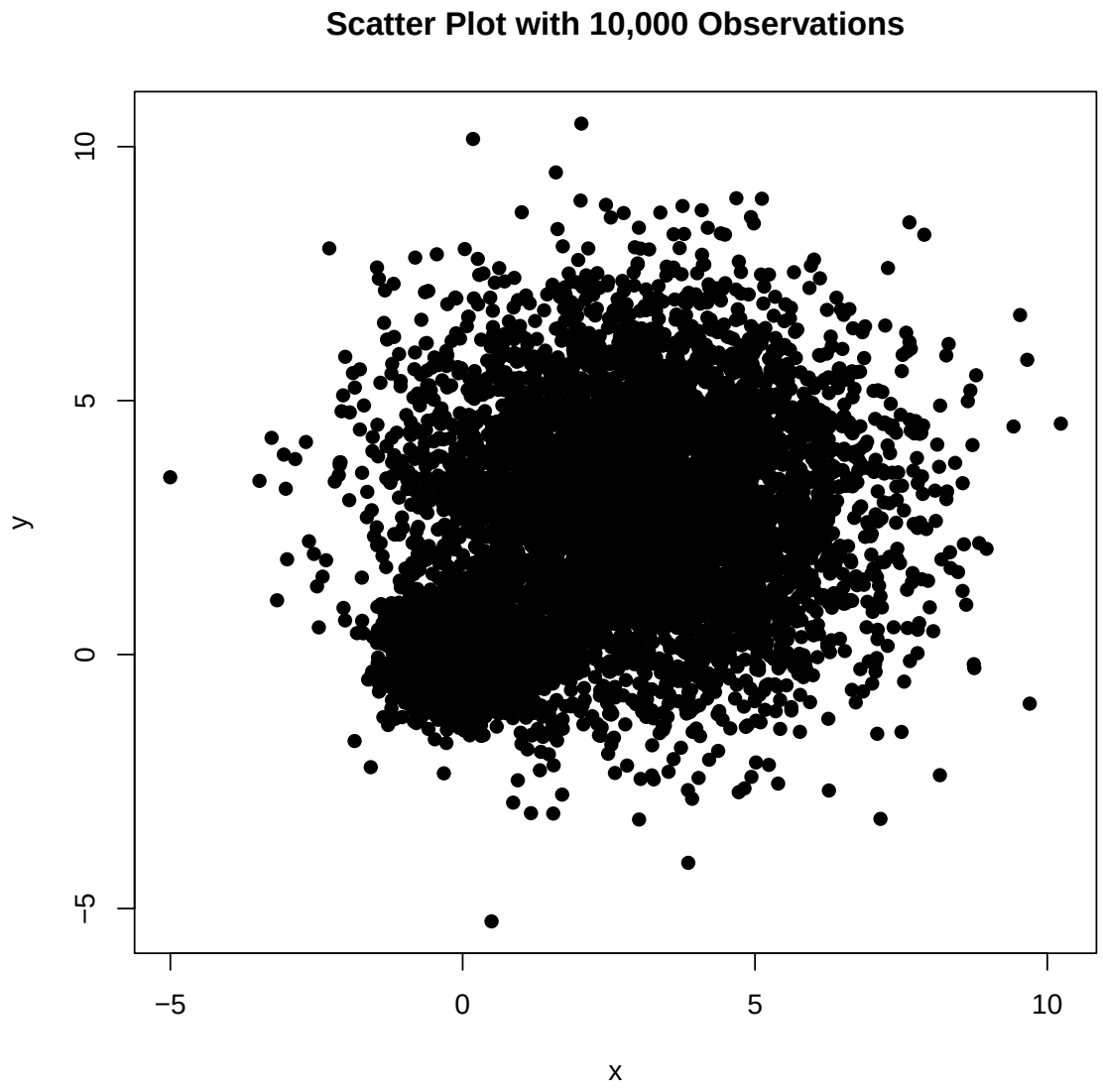
Here, the scatterplot() function is used to plot miles per gallon versus weight for automobiles that have four, six, or eight cylinders. The formula mpg ~ wt | cyl indicates conditioning (that is, separate plots between mpg and wt for each level of cyl).

1.8 High-density scatter plots

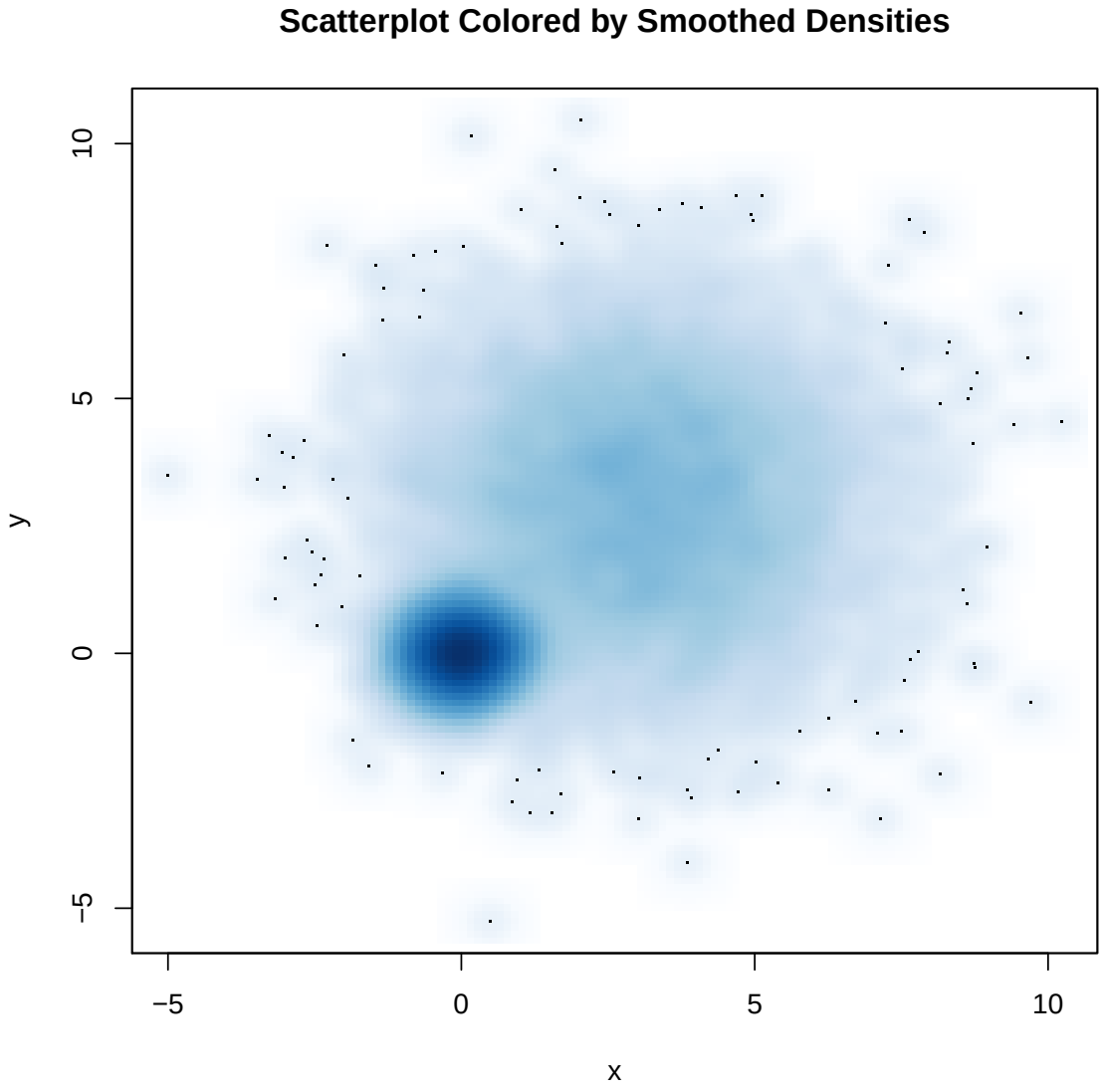
When there's a significant overlap among data points, scatter plots become less useful for observing relationships. Consider the following contrived example with 10,000 observations falling into two overlapping clusters of data:

```
set.seed(1234)
n <- 10000
c1 <- matrix(rnorm(n, mean=0, sd=.5), ncol=2)
c2 <- matrix(rnorm(n, mean=3, sd=2), ncol=2)
mydata <- rbind(c1, c2)
```

```r
mydata <- as.data.frame(mydata)
names(mydata) <- c("x", "y")
with(mydata, plot(x, y, pch=19, main="Scatter Plot with 10,000 Observations"))
```



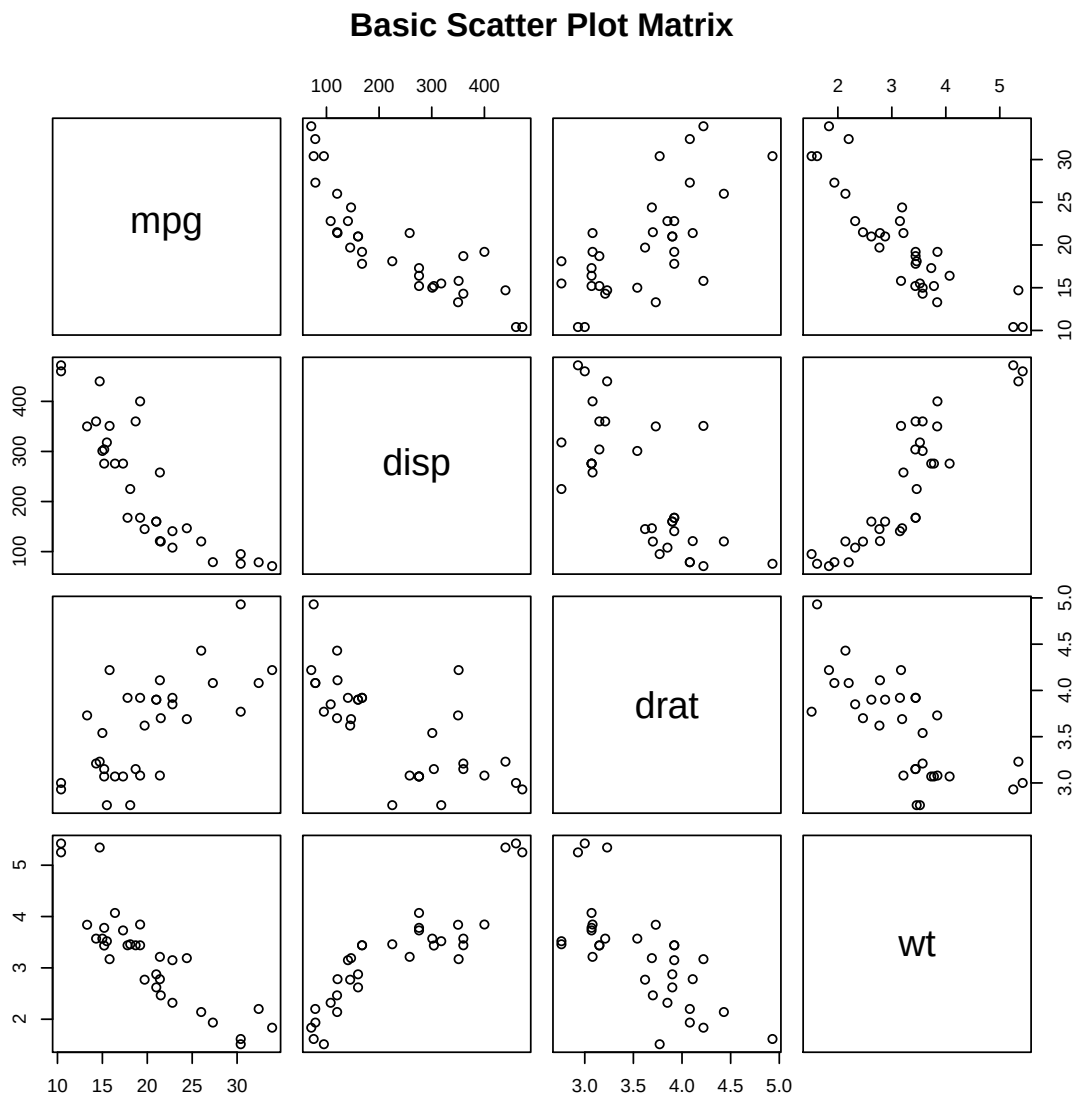

```r
with(mydata, smoothScatter(x, y, main="Scatterplot Colored by Smoothed Densities"))
```



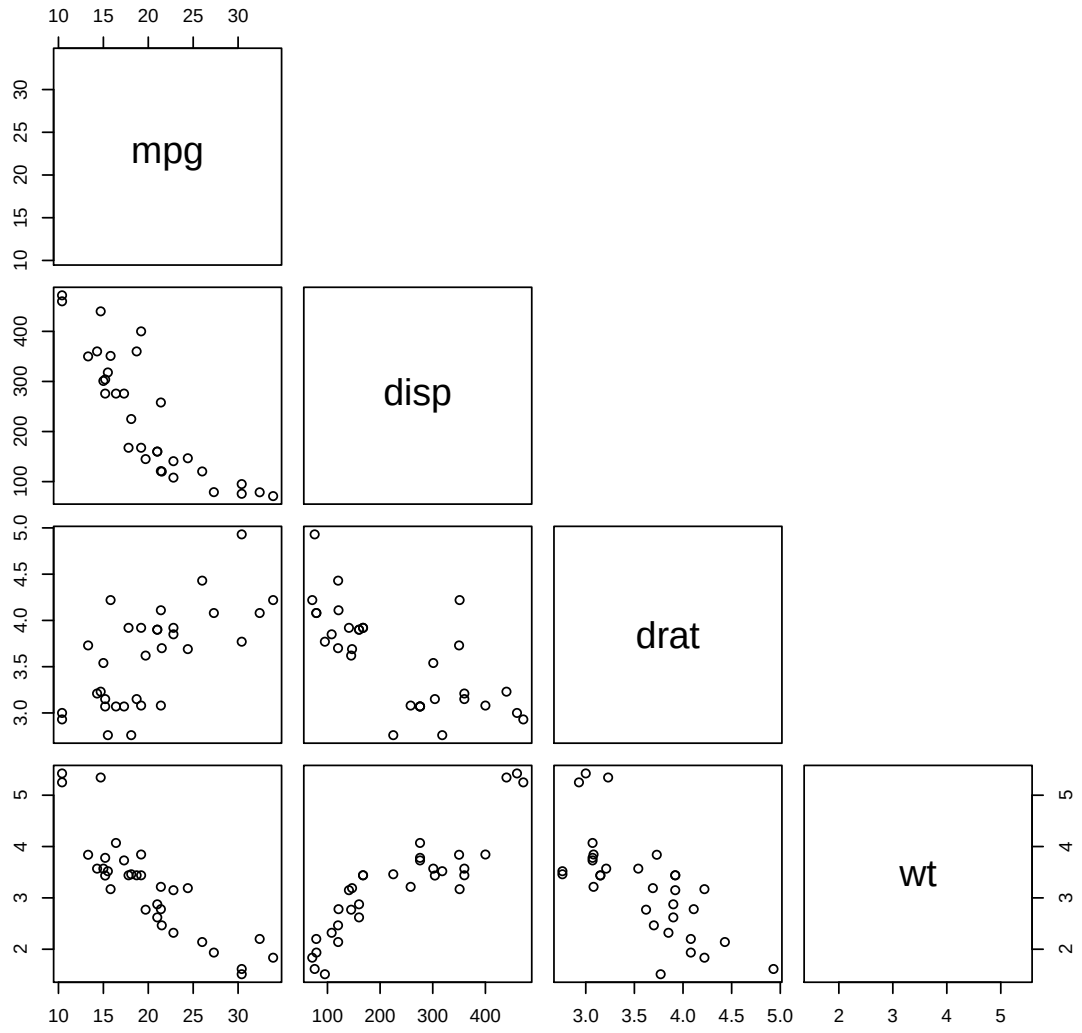
1.9 Scatter plot matrices

The following code produces a scatter plot matrix for the variables mpg, disp, drat, and wt:

```
pairs(~mpg+disp+drat+wt, data=mtcars, main="Basic Scatter Plot Matrix")
```




```
pairs(~mpg+disp+drat+wt, data=mtcars, main="Basic Scatter Plot Matrix",upper.panel=NULL)
```



The s catterplotMatrix() function in the c ar package can also produce scatter plot matrices and can optionally do the following:

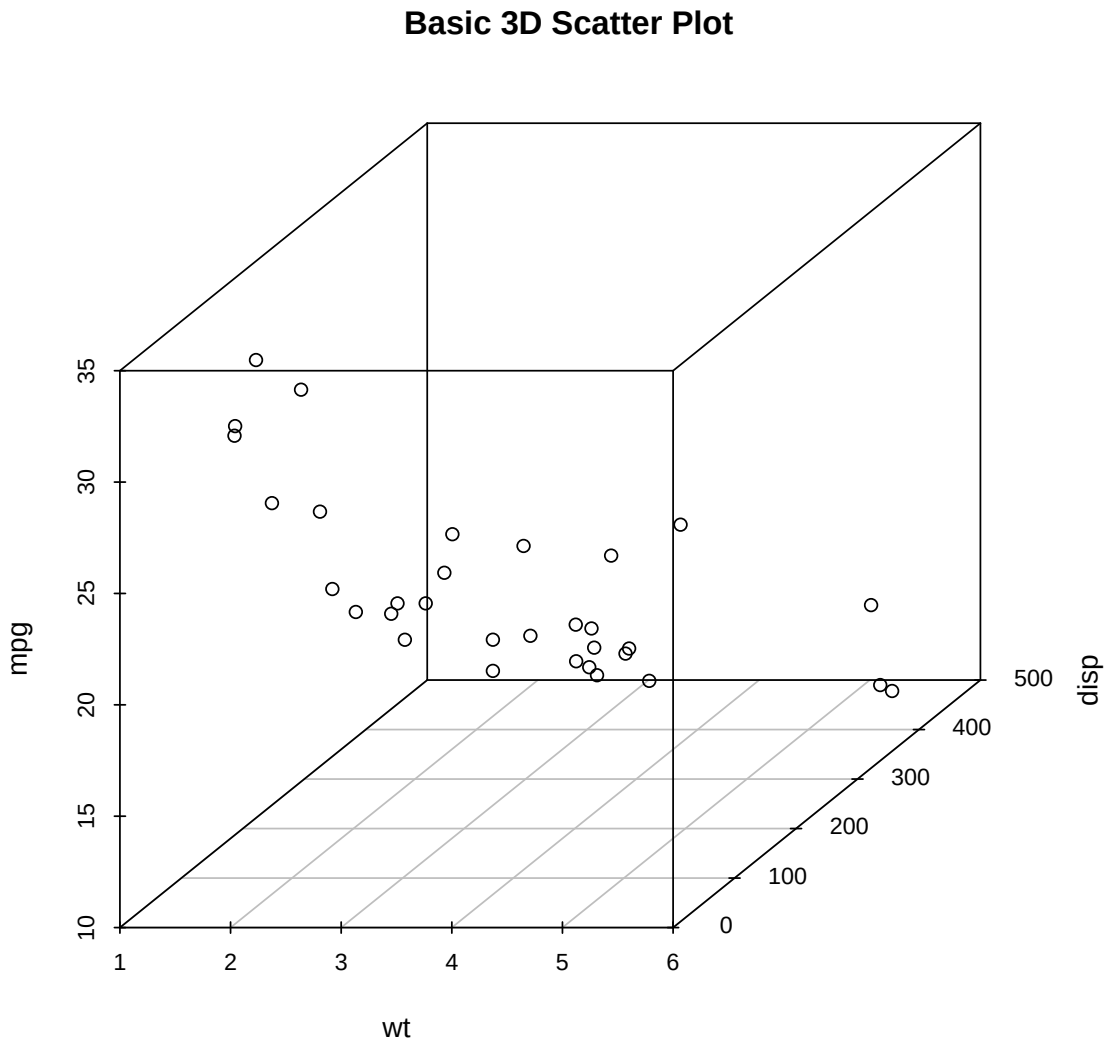
- Condition the scatter plot matrix on a factor
- Include linear and loess fit lines
- Place box plots, densities, or histograms in the principal diagonal
- Add rug plots in the margins of the cells

```
library(car)
scatterplotMatrix(~ mpg + disp + drat + wt, data=mtcars,
     spread=FALSE, lty.smooth=2, main="Scatter Plot Matrix via car Package")
scatterplotMatrix(~ mpg + disp + drat + wt | cyl,
     data=mtcars, spread=FALSE, diagonal="histogram", main="Scatter Plot Matrix via car Package")
```

1.10 3D scatter plots

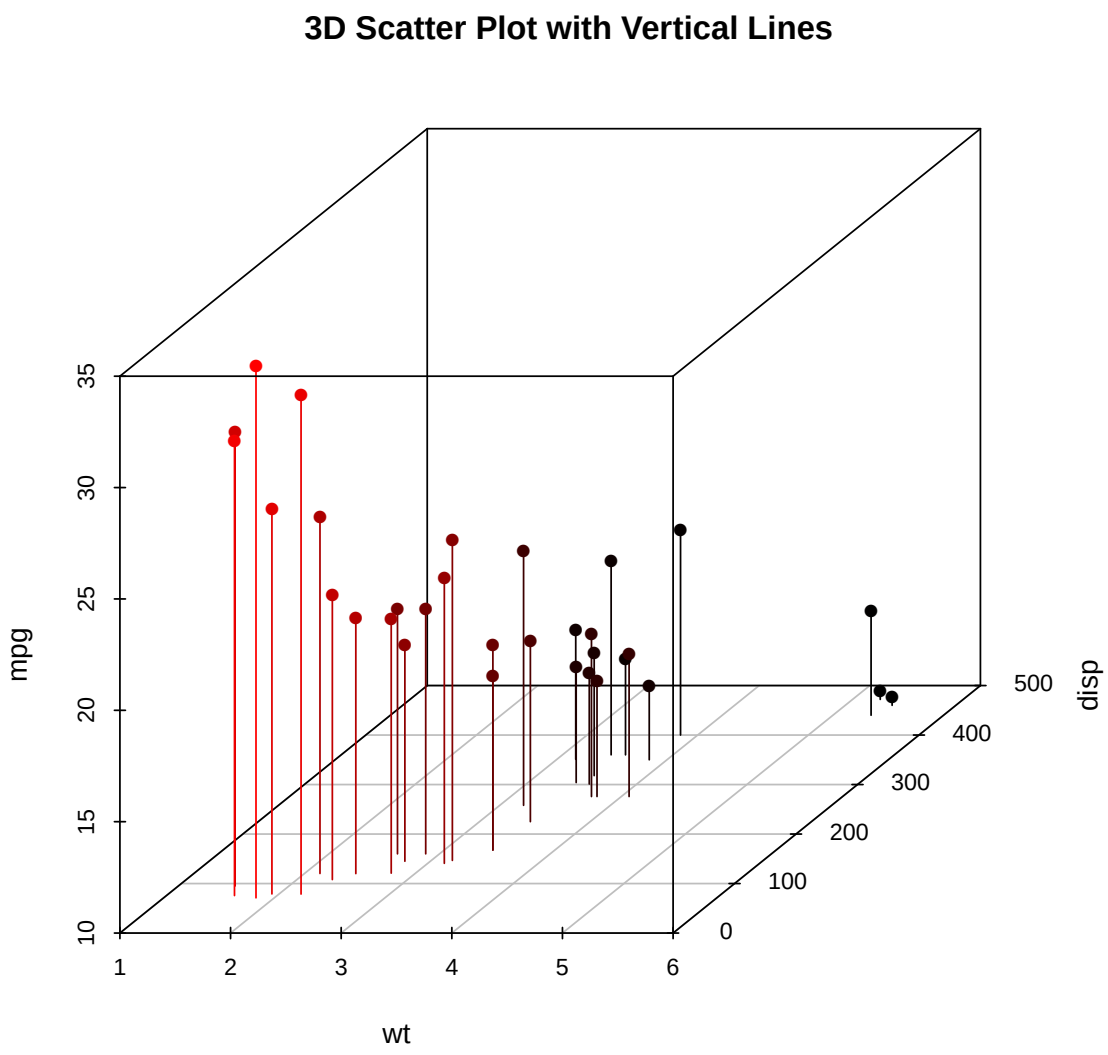
Scatter plots and scatter plot matrices display bivariate relationships. What if you want to visualize the interaction of three quantitative variables at once? In this case, you can use a 3D scatter plot.

```
library(scatterplot3d)
attach(mtcars)

## The following object is masked from package:ggplot2:
##
##     mpg

scatterplot3d(wt, disp, mpg, main="Basic 3D Scatter Plot")
```



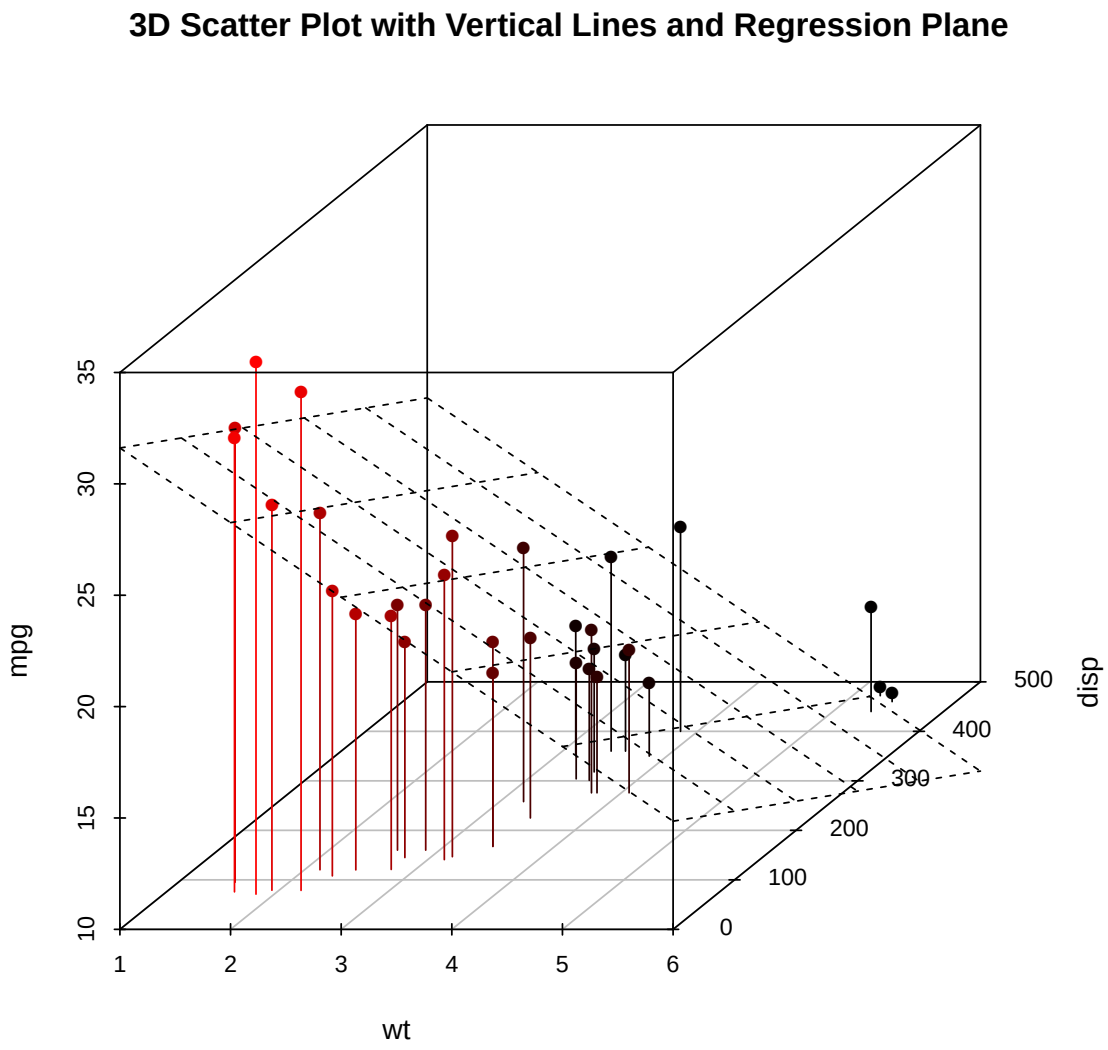
The scatterplot3d() function offers many options, including the ability to specify symbols, axes, colors, lines, grids, highlighting, and angles.

```
scatterplot3d(wt, disp, mpg, pch=16, highlight.3d=TRUE,
              type="h", main="3D Scatter Plot with Vertical Lines")
```



Let's take the previous graph and add a regression plane. The necessary code is:

```
s3d <-scatterplot3d(wt, disp, mpg, pch=16, highlight.3d=TRUE,
   type="h", main="3D Scatter Plot with Vertical Lines and Regression Plane")
fit <- lm(mpg ~ wt+disp)
s3d$plane3d(fit)
```




```
detach(mtcars)
```

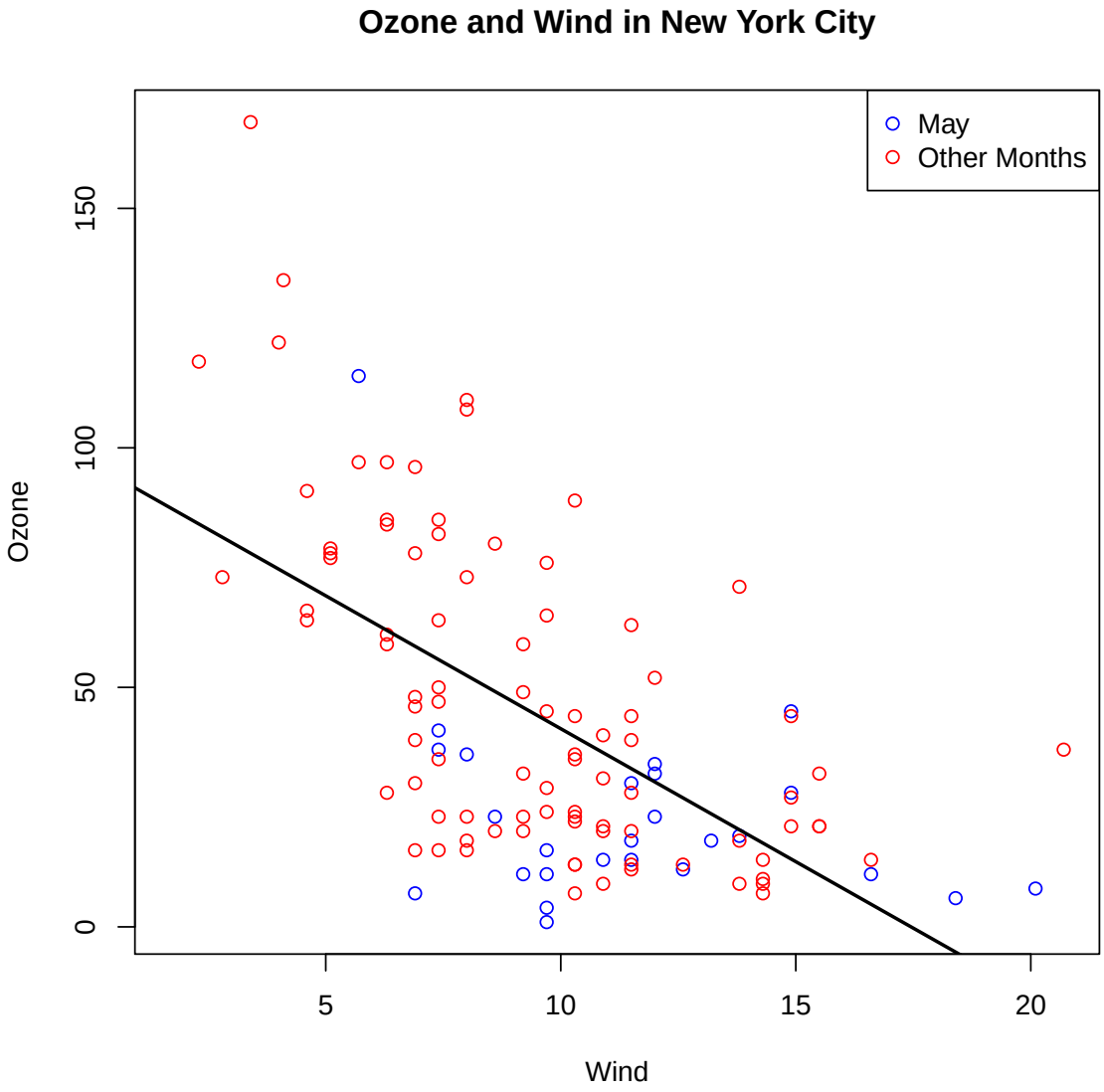
SPINNING 3D SCATTER PLOTS

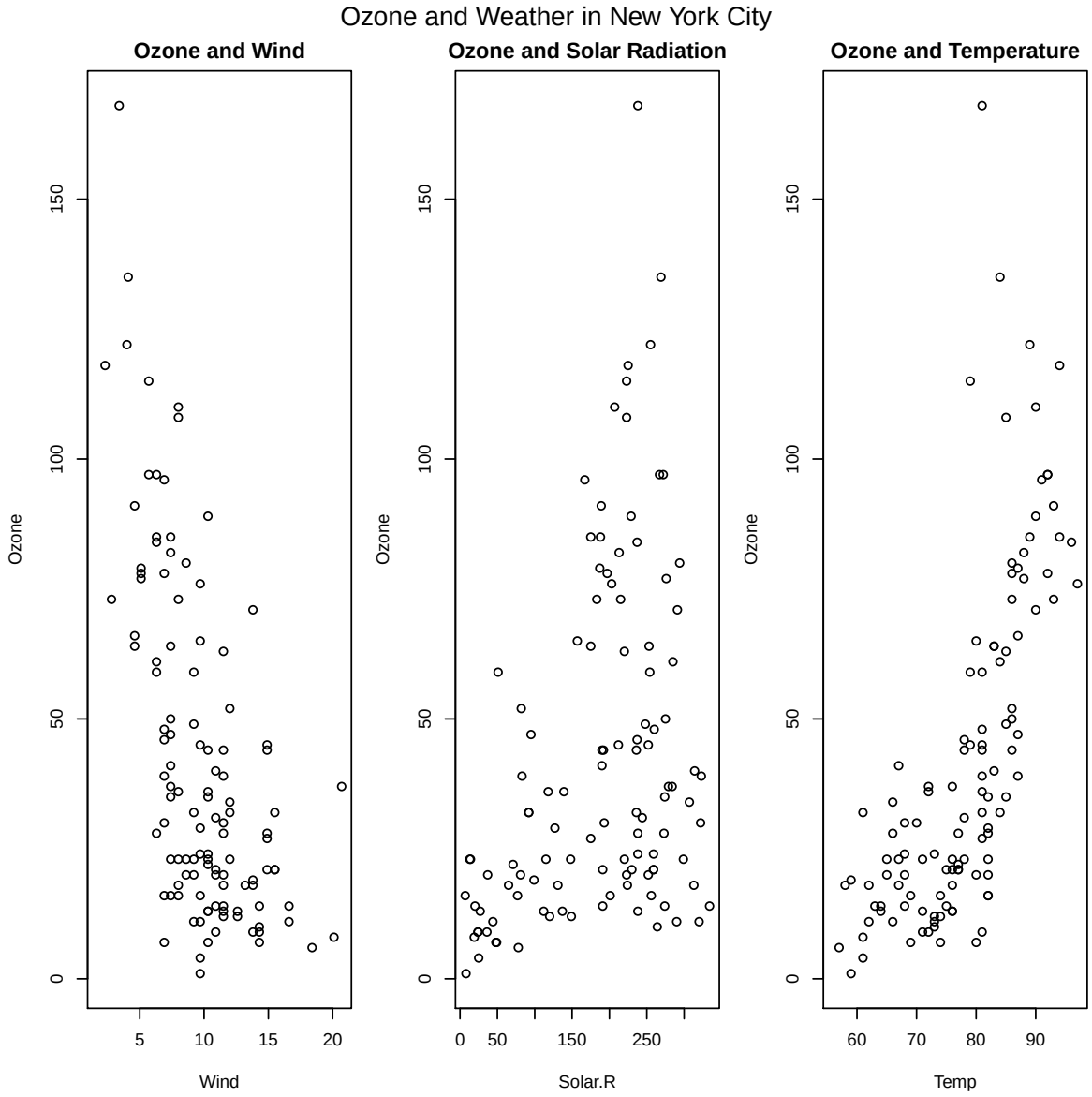
Three-dimensional scatter plots are much easier to interpret if you can interact with them. R provides several mechanisms for rotating graphs so that you can see the plotted points from more than one angle.

```
library(rgl)
attach(mtcars)
plot3d(wt, disp, mpg, col="red", size=5)
library(Rcmdr)
scatter3d(wt, disp, mpg)
detach(mtcars)
```

1.11 Exercise

Try to plot the following graphs. data=library(datasets)





Appendix

Some Important Base Graphics Parameters

Many base plotting functions share a set of parameters. Here are a few key ones:

- pch: the plotting symbol (default is open circle)
- lty: the line type (default is solid line), can be dashed, dotted, etc.
- lwd: the line width, specified as an integer multiple
- col: the plotting color, specified as a number, string, or hex code;the colors() function gives you a vector of colors by name

- xlab: character string for the x-axis label
- ylab: character string for the y-axis label

The par() function is used to specify global graphics parameters that affect all plots in an R session. These parameters can be overridden when specified as arguments to specific plotting functions.

- las: the orientation of the axis labels on the plot
- bg: the background color
- mar: the margin size
- oma: the outer margin size (default is 0 for all sides)
- mfrow: number of plots per row, column (plots are filled row-wise)
- mfcol: number of plots per row, column (plots are filled column-wise)

Base Plotting Functions

- plot: make a scatterplot, or other type of plot depending on the class of the object being plotted
- lines: add lines to a plot, given a vector x values and a corresponding vector of y values (or a 2- column matrix); this function just connects the dots
- points: add points to a plot
- text: add text labels to a plot using specified x, y coordinates
- title: add annotations to x, y axis labels, title, subtitle, outer margin
- mtext: add arbitrary text to the margins (inner or outer) of the plot
- axis: adding axis ticks/labels