

# Introduction to R Graphics

\* R has a powerful environment for visualization of scientific data

- It provides publication quality graphics, which are fully programmable
- Easily reproducible
- Full LaTeX and Sweave support
- Lots of packages and functions with built-in graphics support
- On-screen graphics
- Postscript, PDF, jpeg, png, SVG

[http://faculty.ucr.edu/~tgirke/HTML\\_Presentations/Manuals/Rgraphics/Rgraphics.pdf](http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rgraphics/Rgraphics.pdf)

# Graphics: History

## \* R Graph Gallery

<http://gallery.r-enthusiasts.com/>

## • R Graphic Manual and Gallery

<http://rgm2.lab.nig.ac.jp/RGM2/images.php?show=all&pageID=2087>

## • Grid Graphics – Paul Murrell

<http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html>

## \* Lattice Graphics

<http://lmdvr.r-forge.r-project.org/figures/figures.html>

[http://faculty.ucr.edu/~tgirke/HTML\\_Presentations/Manuals/Rgraphics/Rgraphics.pdf](http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rgraphics/Rgraphics.pdf)

# Graphics: History

R graphics can be confusing because there are no less than 4 different systems. Let's list them out here and talk about which one(s) to use.

## **Low-Level Capability**

Base Graphics (Has Low and High Level functions)  
Grid Graphics

## **High-Level Capability**

Lattice Graphics  
ggplot2

# Graphics: History

## **Base Graphics**

- \* Oldest and most commonly used
- Uses a "pen-on-paper" model. You can only draw on top of the object. Cannot erase, modify, or delete what has already been drawn.
- Has both high and low level plotting routines (unique to Base)
- Base graphics are fast.
- Lots of documentation and "google" support

# Graphics: History

## Grid Graphics

- \* Developed in 2000 by Paul Murrell
- Provides a rich set of graphics primitives
- Uses a system of objects and view ports to make complex objects easier.
- You will almost never use this directly unless you want to do in-depth programming

# Graphics: History

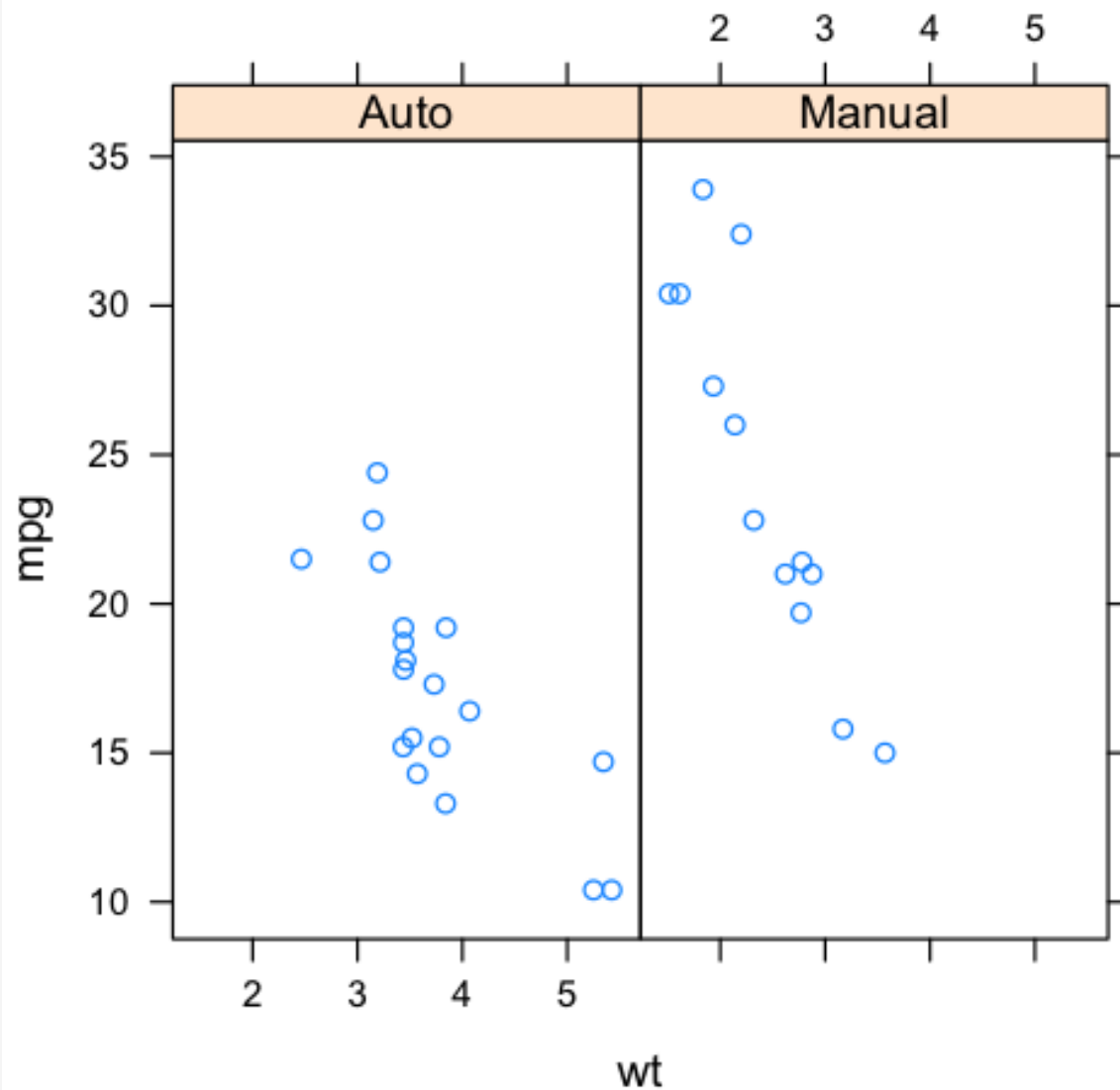
## Lattice package

\* Developed by Deepayan Sarkar to implement the trellis graphics system described in "Visualizing Data" by Cleveland.

- Easy to create conditioned plots with automatic creation of axes, legends, and other annotations
- Usually considered to be an improvement over Base graphics.

```
library(lattice)
xyplot(mpg~wt | factor(am,labels=c("Auto","Manual")), data=mtcars)
```

# Graphics: History



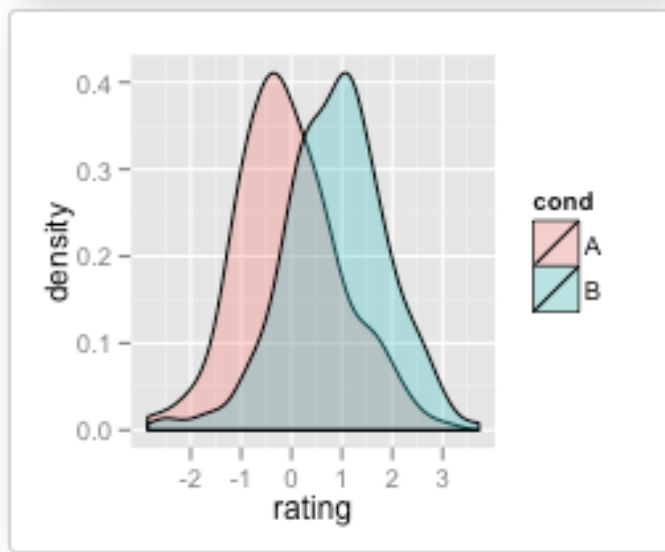
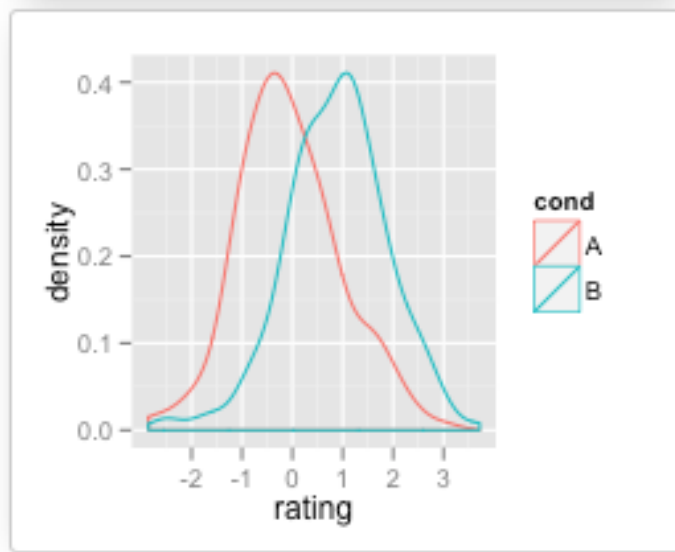
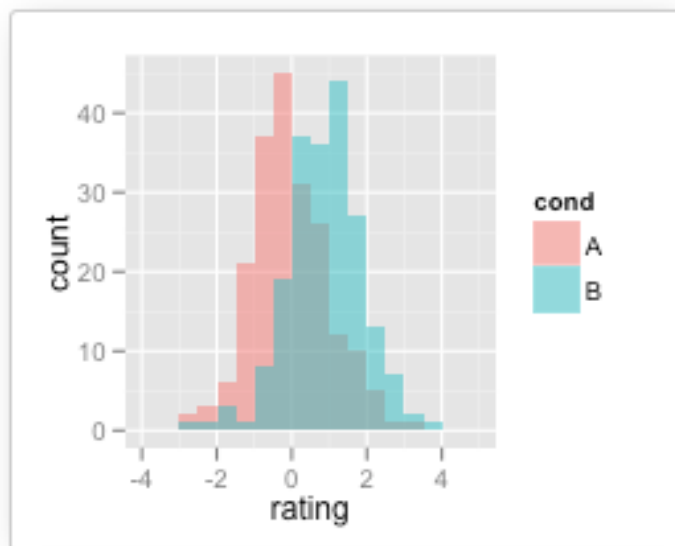
# Graphics: History

## ggplot2

- \* Developed starting in 2005 by Hadley Wickham
- ggplot2 is an implementation of [Leland Wilkinson](#)'s *Grammar of Graphics*--a general scheme for data visualization which breaks up graph into semantic components such as scales and layers.
- ggplot2 can serve as a replacement for the base graphics in R and contains a number of defaults for web and print display of common scales.
- Is said to be much slower than Base graphics but this isn't a major thing (in my opinion)



# Graphics: History



# Graphics: Chart Types

**plot(x,y) where x and y are continuous:**

X/Y, scatterplot, pairs, sunflower plots

**plot(x,[y]) where x and y are categorical. Note that y can be optional:**

dotplot, barplot, stacked bar plot, pie chart

**plot(x) where x is a single continuous variable:**

dotplot, barplot, stripchart, boxplot, density, histogram, QQ Plot

**plot(x,y) where one of x and y is continuous and the other is discrete**

Side-by-Side dotplot and boxplot, notched boxplot

# Graphics

## BASE Graphics

# Graphics: Base

**Base Graphics** - Some low level plotting functions (a select list):

FUNCTION NAME	PURPOSE
<code>points(x,y)</code>	Adds points to an existing plot
<code>lines(x,y)</code>	Adds lines to an existing plot
<code>arrows(x,y)</code>	Draws arrows on an existing plot
<code>text(x,y,labels,...)</code>	Adds text to an existing plot
<code>abline(a,b)</code>	Adds a line of slope b and intercept a
<code>polygon(x,y,...)</code>	Draws a polygon
<code>legend(x,y,legend)</code>	Adds a legend to the plot
<code>title("title")</code>	Adds a title to the plot
<code>axis</code>	Adds an axis to the current plot
<code>mtext</code>	Write text in one of the four margins
<code>segments</code>	Draws line segments on an existing plot

# Graphics: Base

**Base Graphics** - Some high level plotting functions (a select list):

FUNCTION NAME	PURPOSE
<code>plot(x,y)</code>	Generic x-y plots
<code>barplot(x)</code>	Creates a barplot of a table object
<code>boxplot(x)</code>	Creates a boxplot of numeric vector
<code>hist(x)</code>	Histogram of numeric data
<code>pie(x)</code>	Pie chart of a table object
<code>dotchart(x)</code>	Dot Plot of a vector or matrix
<code>qqnorm(x)</code>	Normal qqplot of numeric vector
<code>qqline</code>	Draws the qqline
<code>pairs(x)</code>	Scatterplot of matrix or data frame
<code>stripchart</code>	1D Scatterplot
<code>coplot(x ~ y   f)</code>	Conditioned plot by factor

# Graphics: Base

**Base Graphics** – Some arguments to high level functions:

FUNCTION NAME	PURPOSE
add=TRUE	Adds a new plot on top of another (kind of)
axes=FALSE	Suppresses axis creation – you then make your own
xlab="STRING"	Makes the X label
ylab="STRING"	Makes the y-label
main="STRING"	Gives the plot a main title
sub="STRING"	Gives a subtitle
type="p"	Plot individual points
type="l"	Plot lines
type="b"	Plot points connected by lines
type="o"	Plot points overlaid by lines
type="n"	Suppresses plotting but sets up device. Good for

# Graphics: Base

**Base Graphics** – Some arguments to high level functions:

FUNCTION NAME	PURPOSE
mar	Specifies margins around plot area
col	Specify color of plot symbols
pch	Specify type of symbol example(pch)
lwd	Specify size of plot symbols
cex	Control font sizes (see also cex.main, cex.axis, cex.lab)
las	Direction of axis labels in relation to axis
lty	If lines are used this specifies line type (dashed, etc)
type="l"	Plot lines
type="b"	Plot points connected by lines
type="o"	Plot points overlaid by lines
type="n"	Suppresses plotting but sets up device. Good for when you

# Graphics: Base

You can save your on-screen graphics to a popular file type for use within a program. You can always do screen grabs too.

FUNCTION	RESULT OUTPUT
<code>pdf("file.pdf")</code>	Creates a PDF file called "file.pdf"
<code>png("file.png")</code>	Creates a PNG file
<code>jpeg("file.jpg")</code>	Creates a JPG file
<code>bmp("file.bmp")</code>	Creates a BMP file
<code>postscript("file.ps")</code>	Creates a Postscript file
<code>win.meta("file.wmf")</code>	Creates a Windows meta file

```
> png("mytest.png")  
  
> plot(mtcats$mpg) # Simple, but you get the point  
  
> dev.off()
```



# Graphics: Base: Foundations

```
plot(0:10, 0:10, type="n", xlab="X", ylab="Y", axes=FALSE)

abline(h=seq(0,10,2),lty=3,col="gray90")

abline(v=seq(0,10,2),lty=3,col="gray90")

text(5,5, "Plot Stuff Here", col="red", cex=1.5)

box("plot", col="red", lty = "dotted")

box("inner", col="blue", lty = "dashed")

mtext("South Margin",1,cex=1.2,col="blue")

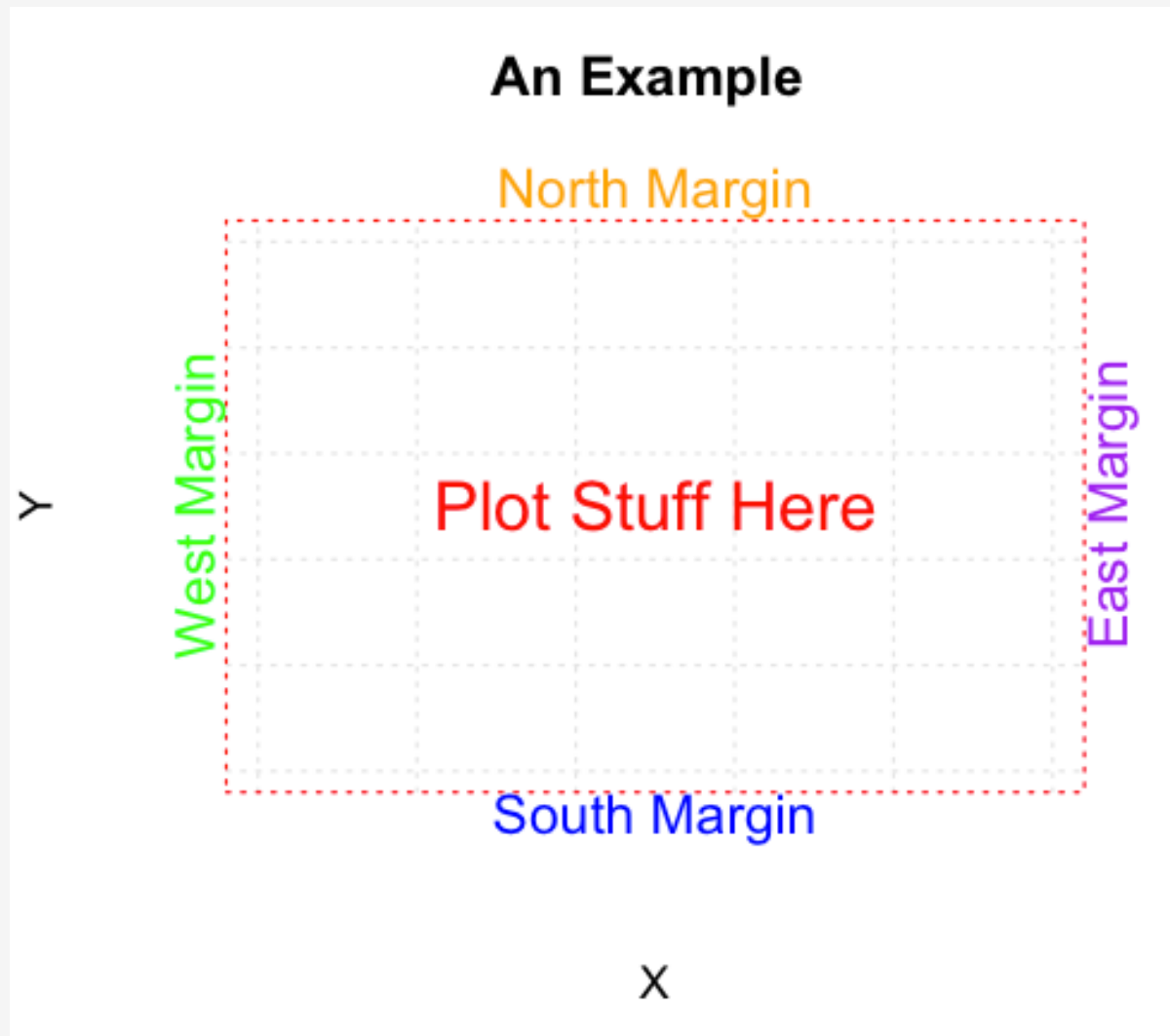
mtext("West Margin",2,cex=1.2,col="green")

mtext("North Margin",3,cex=1.2,col="orange")

mtext("East Margin",4,cex=1.2,col="purple")

title("An Example Plot")
```

# Graphics: Base: Foundations



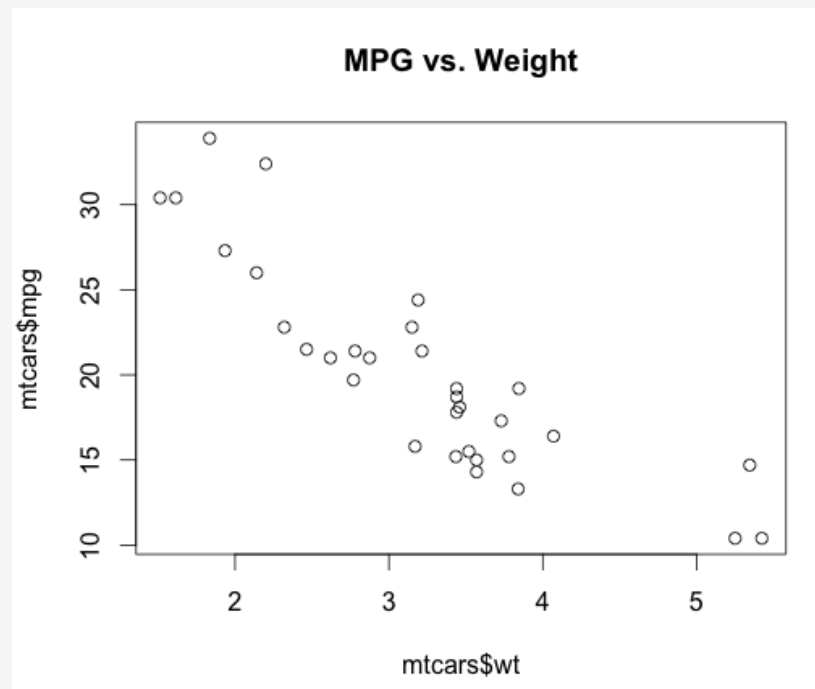
# Graphics: Base: Foundations

Let's do some basic plotting. These two commands do the same thing. Given two vectors, x and y (of same length), do a scatterplot:

```
plot(x,y)    # Traditional way
```

Using mtcars:

```
plot(mtcars$wt, mtcars$mpg, main="MPG vs. Weight")
```

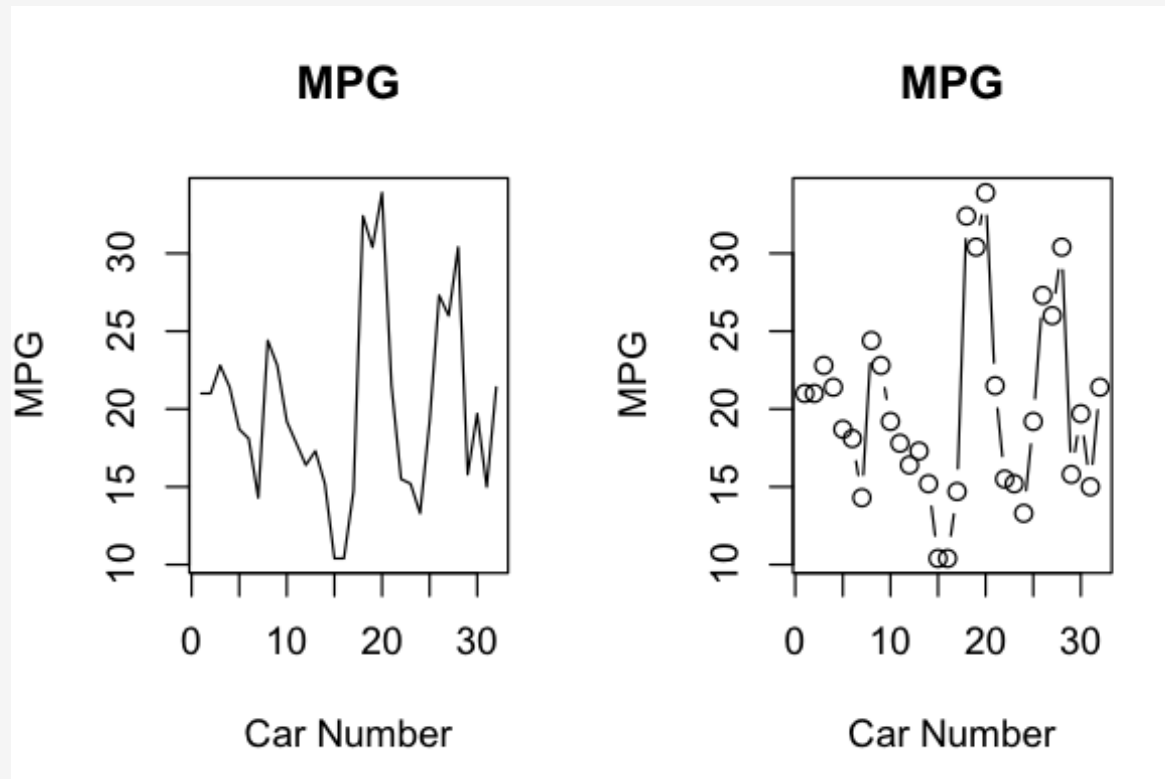


# Graphics: Base: Foundations

We can also plot a single variable:

```
plot(mtcars$mpg, main="MPG", type="l", xlab="Car Number", ylab="MPG")
```

```
plot(mtcars$mpg, main="MPG", type="b", xlab="Car Number", ylab="MPG")
```

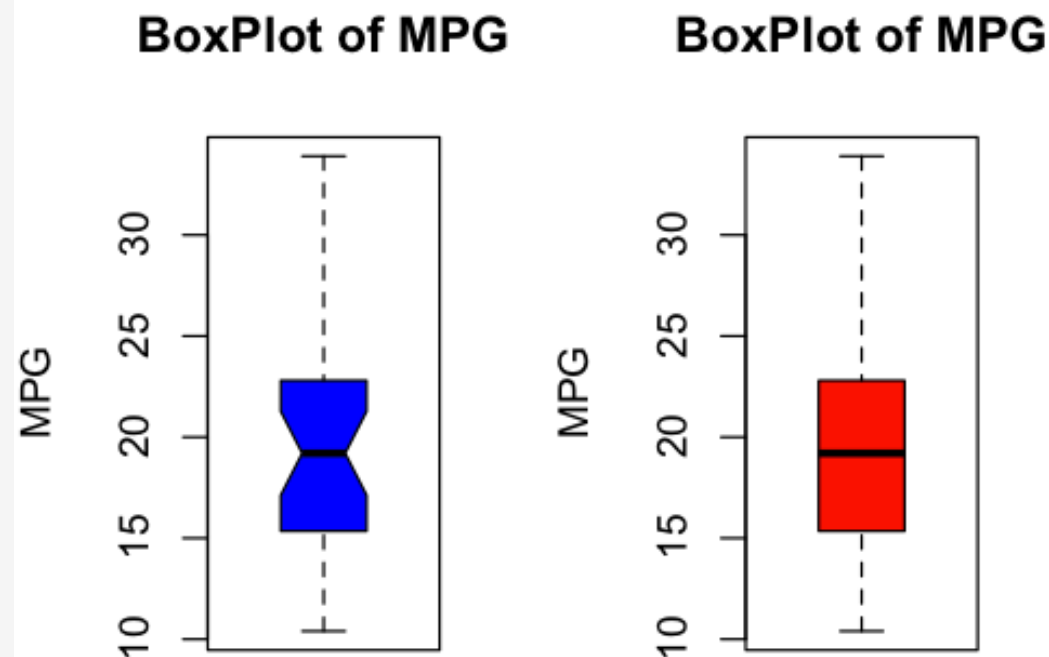


# Graphics: Base: Foundations

We can also plot a single variable:

```
boxplot(mtcars$mpg, main="BoxPlot of MPG",  
        ylab="MPG",col="blue",notch=TRUE)
```

```
boxplot(mtcars$mpg,main="BoxPlot of MPG",ylab="MPG",col="red")
```



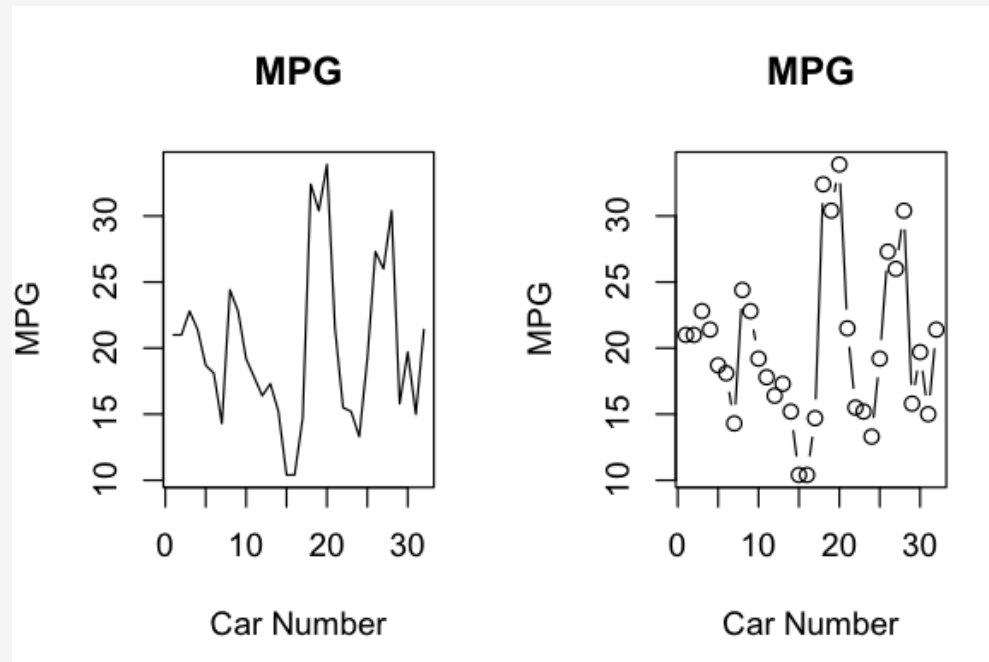
# Graphics: Base: Panels

How can I get two plots to be on the same page ?

```
par(mfrow=c(1,2)) # One row and two columns
```

```
plot(mtcars$mpg, main="MPG", type="l",xlab="Car Number",ylab="MPG")
```

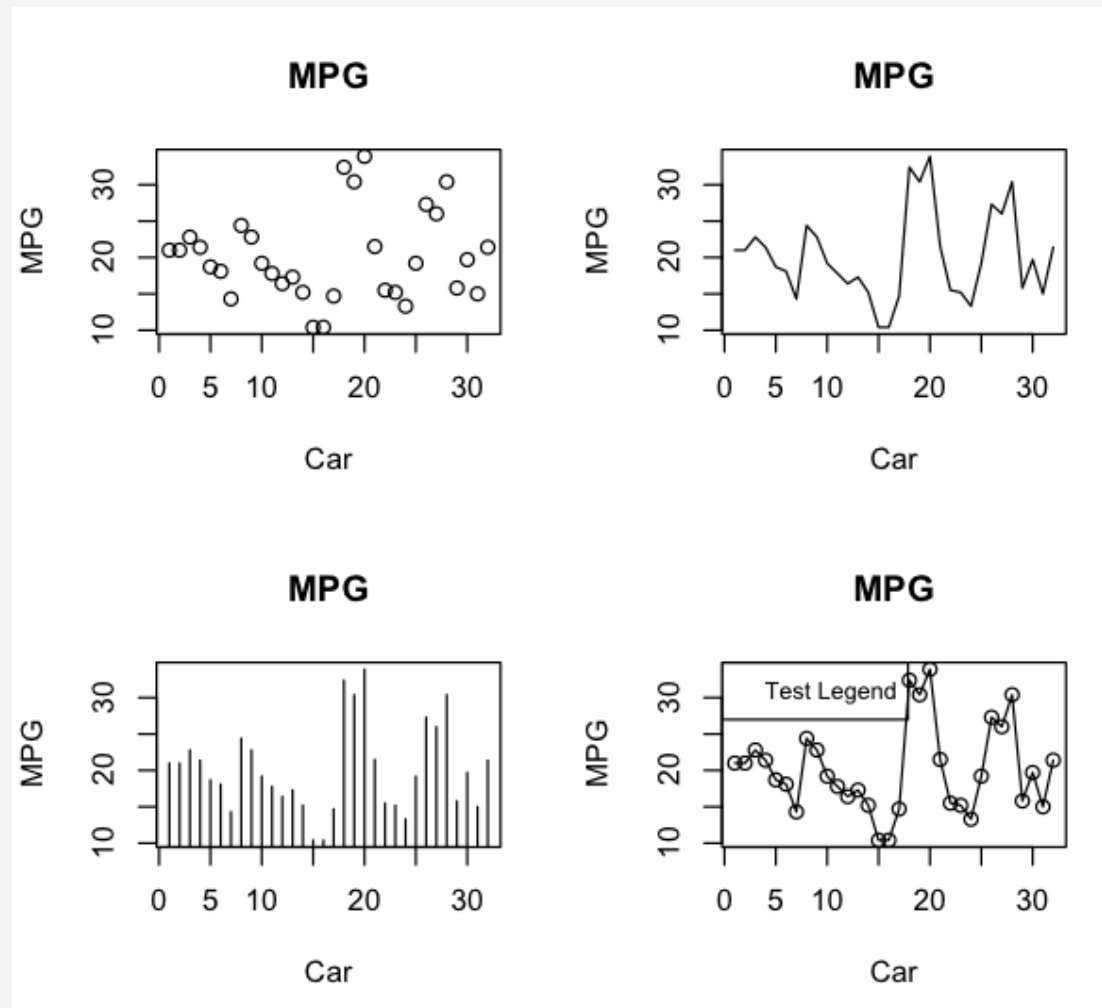
```
plot(mtcars$mpg, main="MPG", type="b",xlab="Car Number",ylab="MPG")
```



# Graphics: Base: Panels

```
par(mfrow=c(2,2))  
  
plot(mtcars$mpg,main="MPG",xlab="Car",ylab="MPG",type="p")  
plot(mtcars$mpg,main="MPG",xlab="Car",ylab="MPG",type="l")  
plot(mtcars$mpg,main="MPG",xlab="Car",ylab="MPG",type="h")  
plot(mtcars$mpg,main="MPG",xlab="Car",ylab="MPG",type="o")  
  
legend("topleft",legend=c("Test Legend"),cex=0.8)
```

# Graphics: Base: Panels





# Graphics: Base: MultiPanel

We usually take this approach when we want to plot data across different categories. Like the mpg vs weight across cylinder types. We have three unique cylinder values:

```
unique(mtcars$cyl)  # We have three categories so let's create 3 plots  
[1] 6 4 8
```

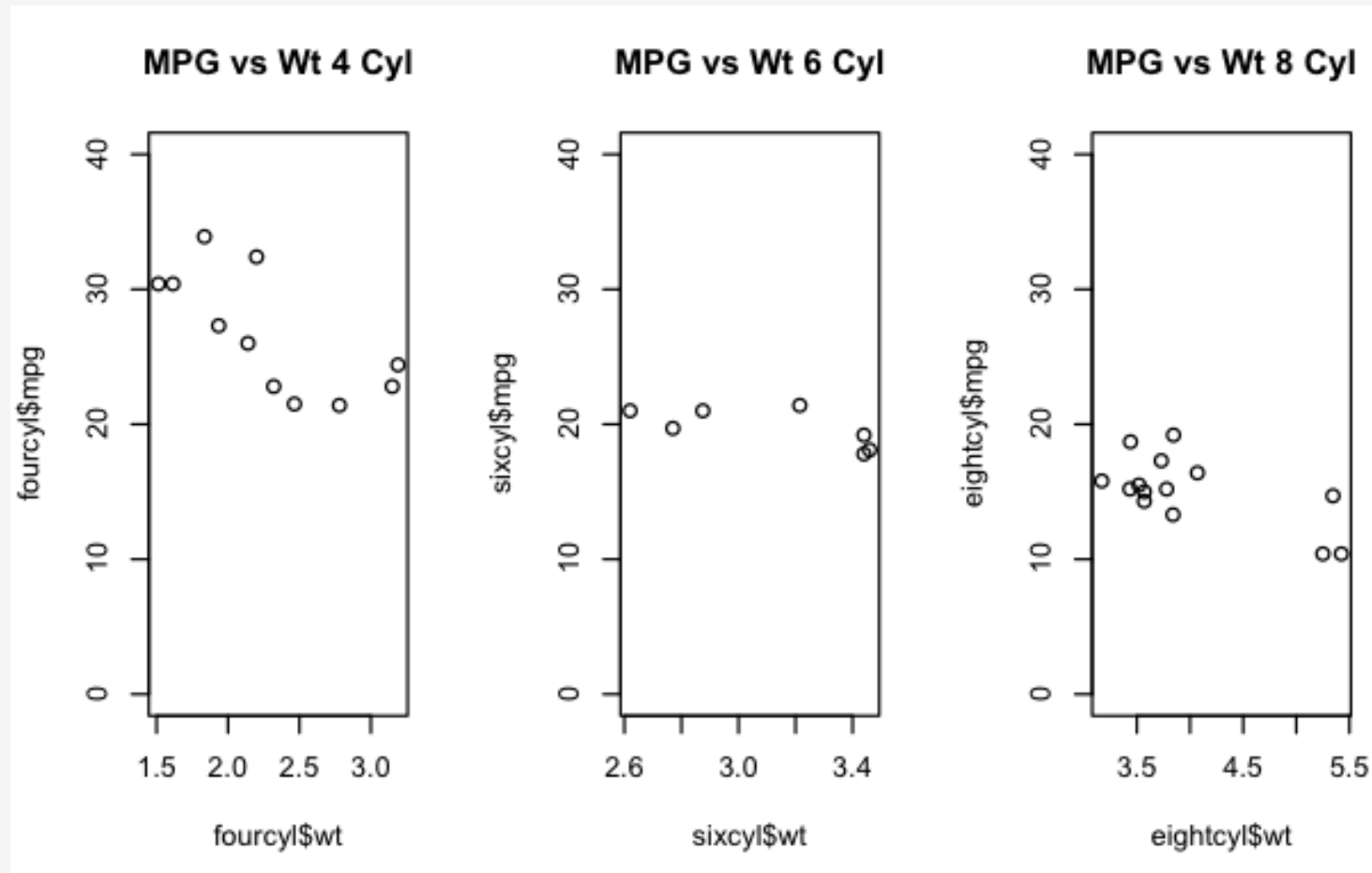
```
par(mfrow=c(1,3))  # One row and three columns
```

```
fourcyl  <- mtcars[mtcars$cyl == 4,]  
sixcyl   <- mtcars[mtcars$cyl == 6,]  
eightcyl <- mtcars[mtcars$cyl == 8,]
```

```
plot(fourcyl$wt, fourcyl$mpg, main = "MPG vs Wt 4 Cyl", ylim=c(0,40))  
plot(sixcyl$wt, sixcyl$mpg, main = "MPG vs Wt 6 Cyl", ylim=c(0,40))  
plot(eightcyl$wt, eightcyl$mpg, main = "MPG vs Wt 8 Cyl", ylim=c(0,40))
```

```
par(mfrow=c(1,1)) # Reset the plot window
```

# Graphics: Base: MultiPanel



# Graphics: Base: MultiPanel

We could automate this using the "split" approach:

```
par(mfrow=c(1,3))  # One row and three columns
mysplits <- split(mtcars, mtcars$cyl)

for (ii in 1:length(mysplits)) {
  plot(mysplits[[ii]]$wt, mysplits[[ii]]$mpg,
       ylim <- c(0,40),
       main=paste("MPG vs weight for",names(mysplits[ii])))
}

# Better yet we could make this into a function

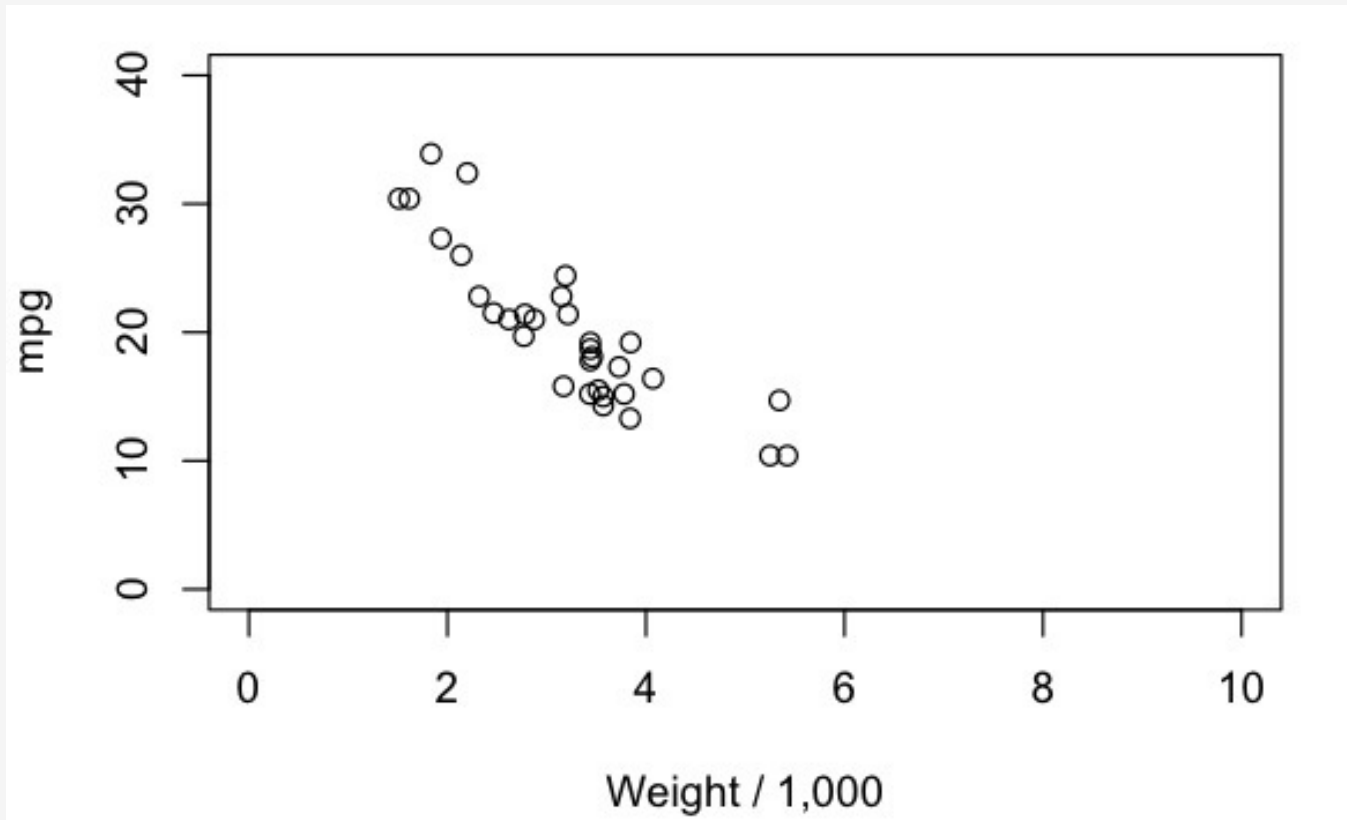
cyl.plot <- function(df, fac, numrows=1, numcols=3) {
  par(mfrow=c(numrows,numcols))
  mysplits <- split(df,fac)
  for (ii in 1:length(mysplits)) {
    plot(mysplits[[ii]]$wt, mysplits[[ii]]$mpg,
         ylim = c(0,40),
         main=paste("MPG vs weight for",names(mysplits[ii])))
  }
}

cyl.plot(mtcars,mtcars$cyl)
```

# Graphics: Base: Arguments

We can set plot limits and add annotations

```
plot(mtcars$wt, mtcars$mpg, xlab = "Weight / 1,000", ylab = "MPG",  
     xlim = c(0,10), ylim = c(0,40))
```



# Graphics: Base: Arguments

We can add a legend:

```
plot(mtcars$wt, mtcars$mpg, xlab = "Weight / 1,000", ylab = "MPG",  
     xlim = c(0,10), ylim = c(0,40))
```

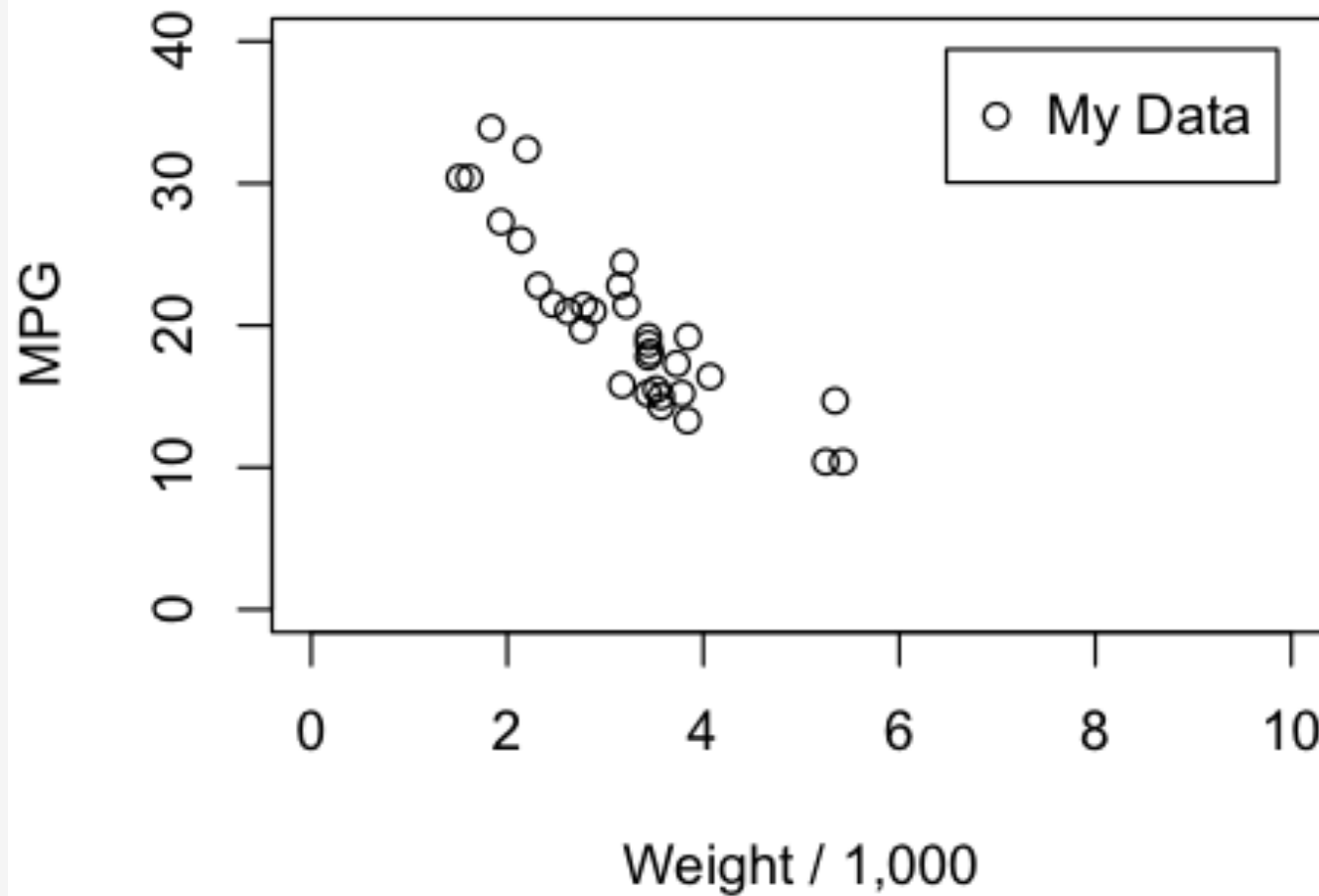
```
legend("topright", inset=0.05, "My Data", pch=1, col="black")
```

# Could use specific coordinates also

```
legend(6.5,35, inset=0.05, "My Data", pch=1, col="black")
```

We specify location of the legend in terms of the data coordinates

# Graphics: Base: Plotting



# Graphics: Base: Annotation

We could also put up our own grid using some "primitive" graphics functions:

```
plot(mtcars$wt, mtcars$mpg,
      xlab = "Weight / 1,000",
      main = "MPG vs. WT")

abline(v=c(2,3,4,5),lty=2,col="gray90")

# Draws vertical dashed lines at 2,3,4,5

abline(h=c(10,15,20,25,30), lty=2, col="gray90")

# Horizontal lines at 10,15,20,25,30

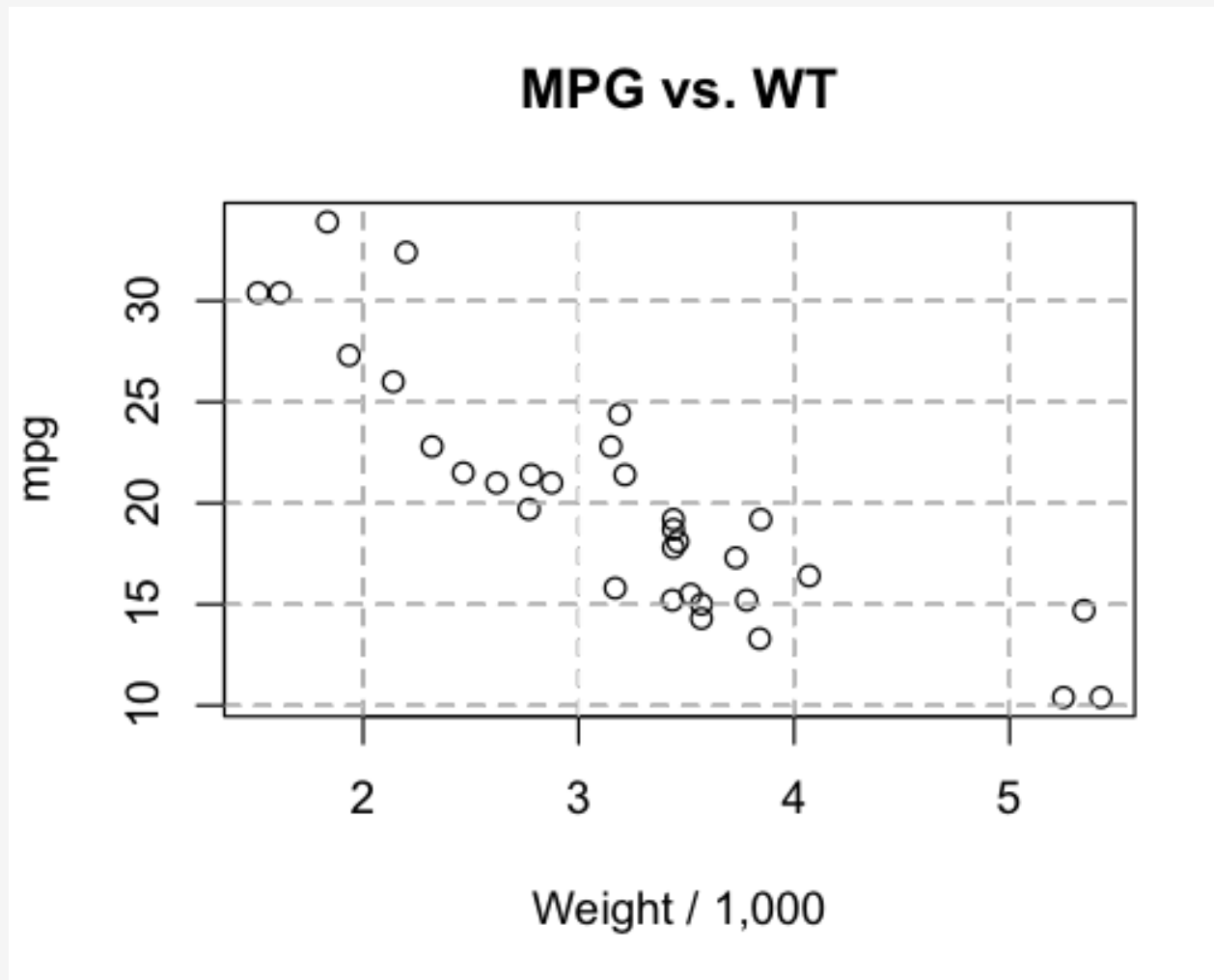
# Could do:

abline(v=2:5,lty=2,col="gray90")

abline(h=seq(10,30,5),lty=2,col="gray90")
```

# Graphics: Base: Annotation

We could also put up our own grid using some "primitive" graphics functions:



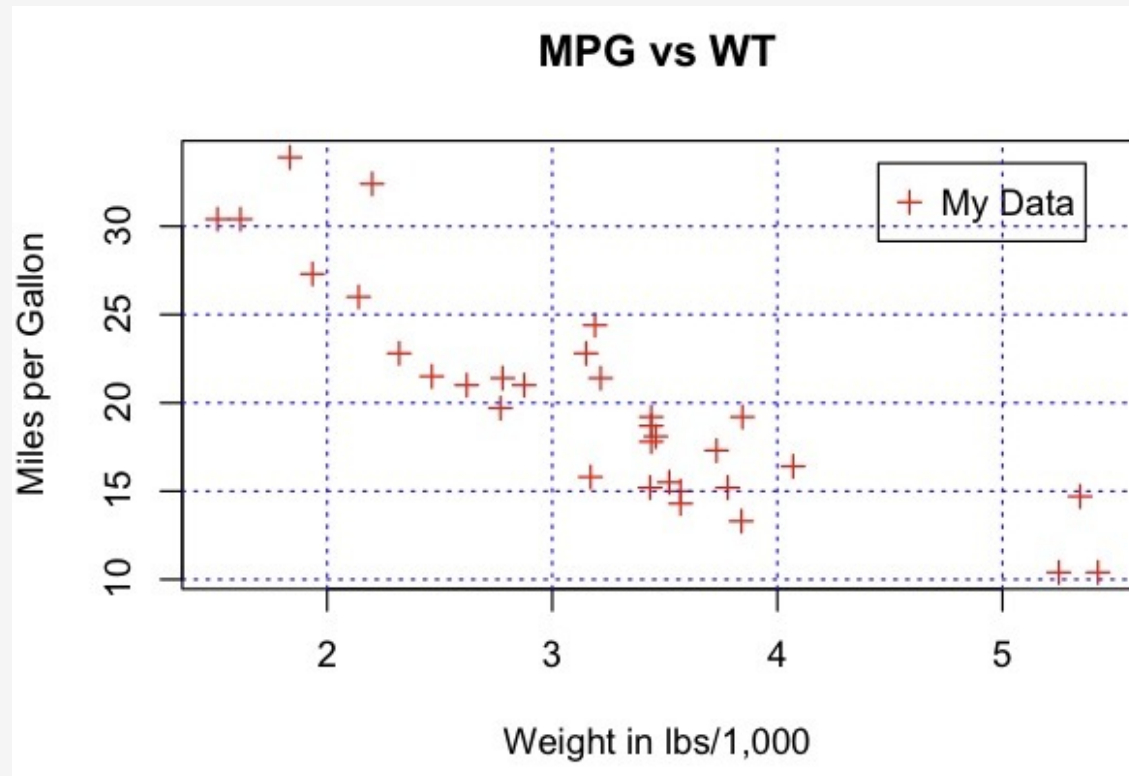


# Graphics: Base: Plot Character

```
plot(mtcars$wt, mtcars$mpg, main="MPG vs WT", col="red",  
      xlab="Weight in lbs/1,000",  
      ylab="Miles per Gallon",  
      pch = 3)
```

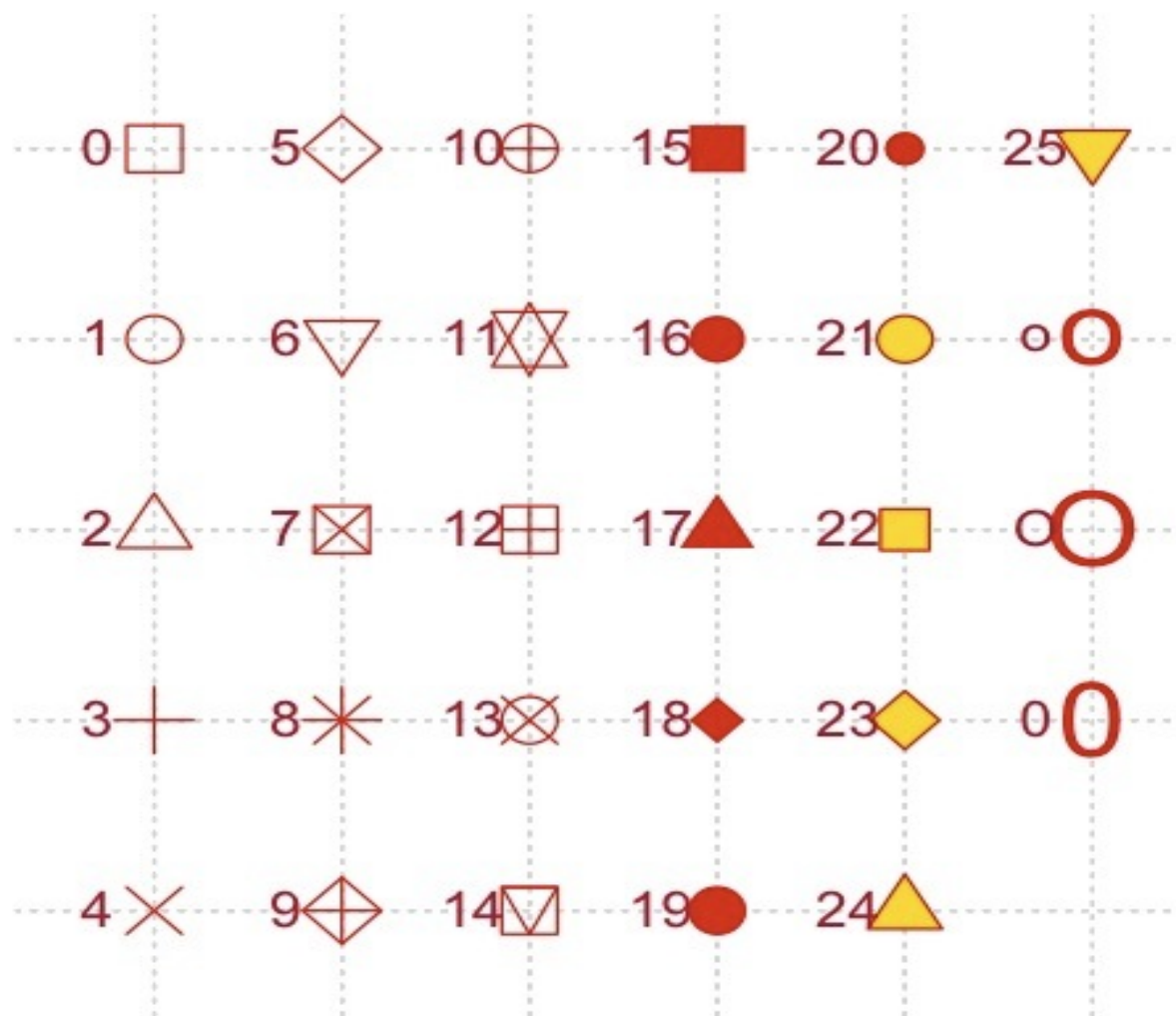
```
legend("topright", inset=0.05, "My Data", pch = 3, col="red")
```

```
grid(col="blue")
```



# Graphics: Base: Plot Characters

```
example(pch)
```



# Graphics: Base: Layered Plot

We could also use information from a data frame to help us print different characters based on value. Like in mtcars. Let's plot MPG vs Weight but pick a different plot character based on Transmission Type. Here is one way to do it:

- 1) Create a blank plot that sets the limits and title
- 2) Extract records for automatic transmission into a data frame
- 3) Extract records for manual transmission into a data frame
- 4) Use the points command to plot these two different groups using a different pch value

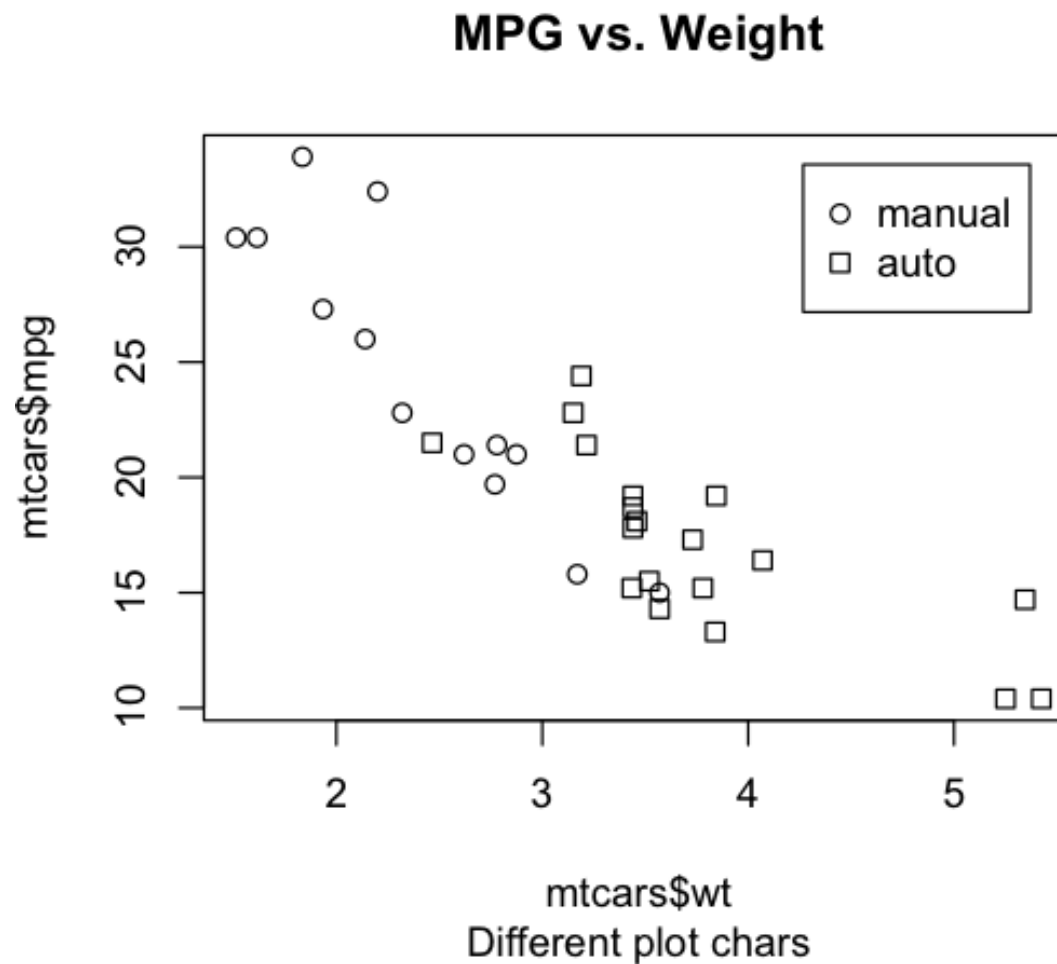
```
plot(mtcars$wt, mtcars$mpg, type="n", main="MPG vs. Weight") # A null plot
```

```
auto <- mtcars[mtcars$am == 0,]  
manu <- mtcars[mtcars$am == 1,]
```

```
points(auto$wt, auto$mpg, pch = 0)  
points(manu$wt, manu$mpg, pch = 1)
```

```
legend("topright", inset=0.05, c("manual", "auto"),  
      pch = c(1, 0))
```

# Graphics: Base: Layered Plot



# Graphics: Base: Layered Plot

But this would be working too hard. No programming is required. Just recognize that the plot characters are selected by a number from 0 to 25. We can exploit this:

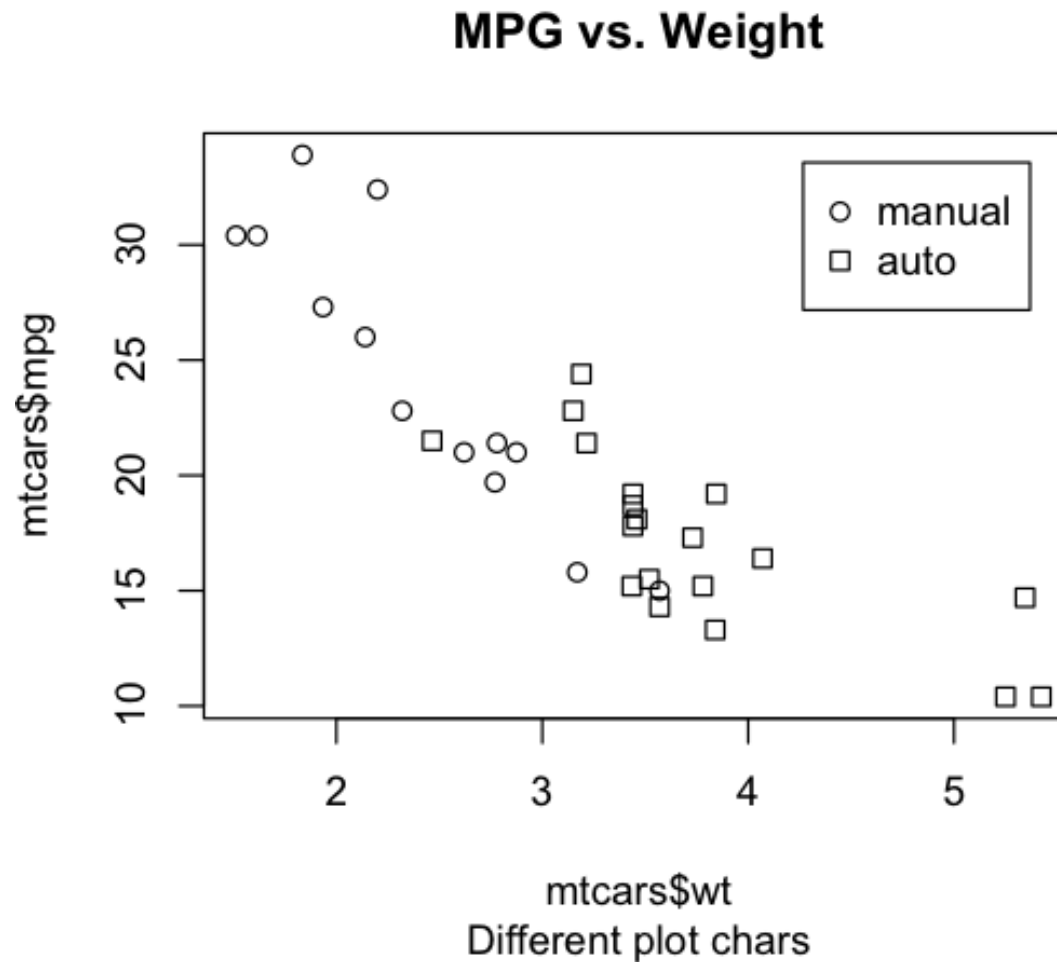
```
mtcars$am  
[1] 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1
```

We see that am is 0 or 1 which just so happen to also represent valid print characters

```
plot(mtcars$wt, mtcars$mpg, pch=mtcars$am,  
     main="MPG vs. Weight", sub="Different plot chars")
```

```
legend("topright", inset=0.05, c("manual", "auto"),  
      pch = unique(mtcars$am))
```

# Graphics: Base: Layered Plot



# Graphics: Base: Layered

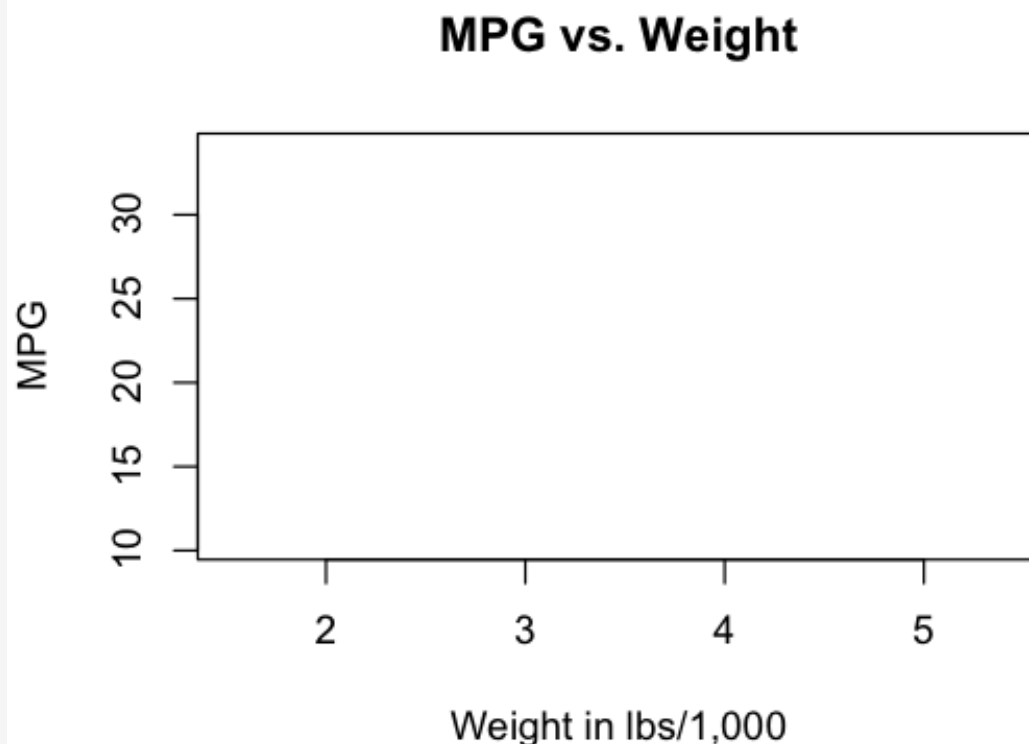
It is also possible to build a plot in layers. We initialize a "blank" plot using the plot command but we specify a type of "n".

Let's plot wt vs MPG and do it such that the records with a weight below the mean weight are in red and those above the mean weight are in blue

# Graphics: Base: Layered

It is also possible to build a plot in layers. We initialize a "blank" plot using the plot command but we specify a type of "n".

```
plot(mtcars$wt,mtcars$mpg,type="n",xlab="Weight in lbs/1,000",  
     ylab="MPG", main="MPG vs. Weight")
```





# Graphics: Base: Layered

How is this useful ? Well we can add points or lines in stages. This allows us to plot things on an existing plot using specific colors or print characters.

```
plot(mtcars$wt,mtcars$mpg,type="n",xlab="Weight in lbs/1,000",  
     ylab="MPG", main="MPG vs. Weight")
```

# Let's get records for each category

```
above.mean <- mtcars[mtcars$wt >= mean(mtcars$wt),]
```

```
below.mean <- mtcars[mtcars$wt < mean(mtcars$wt),]
```

# Use the points command to plot each group

```
points(below.mean$wt,below.mean$mpg,col="red")
```

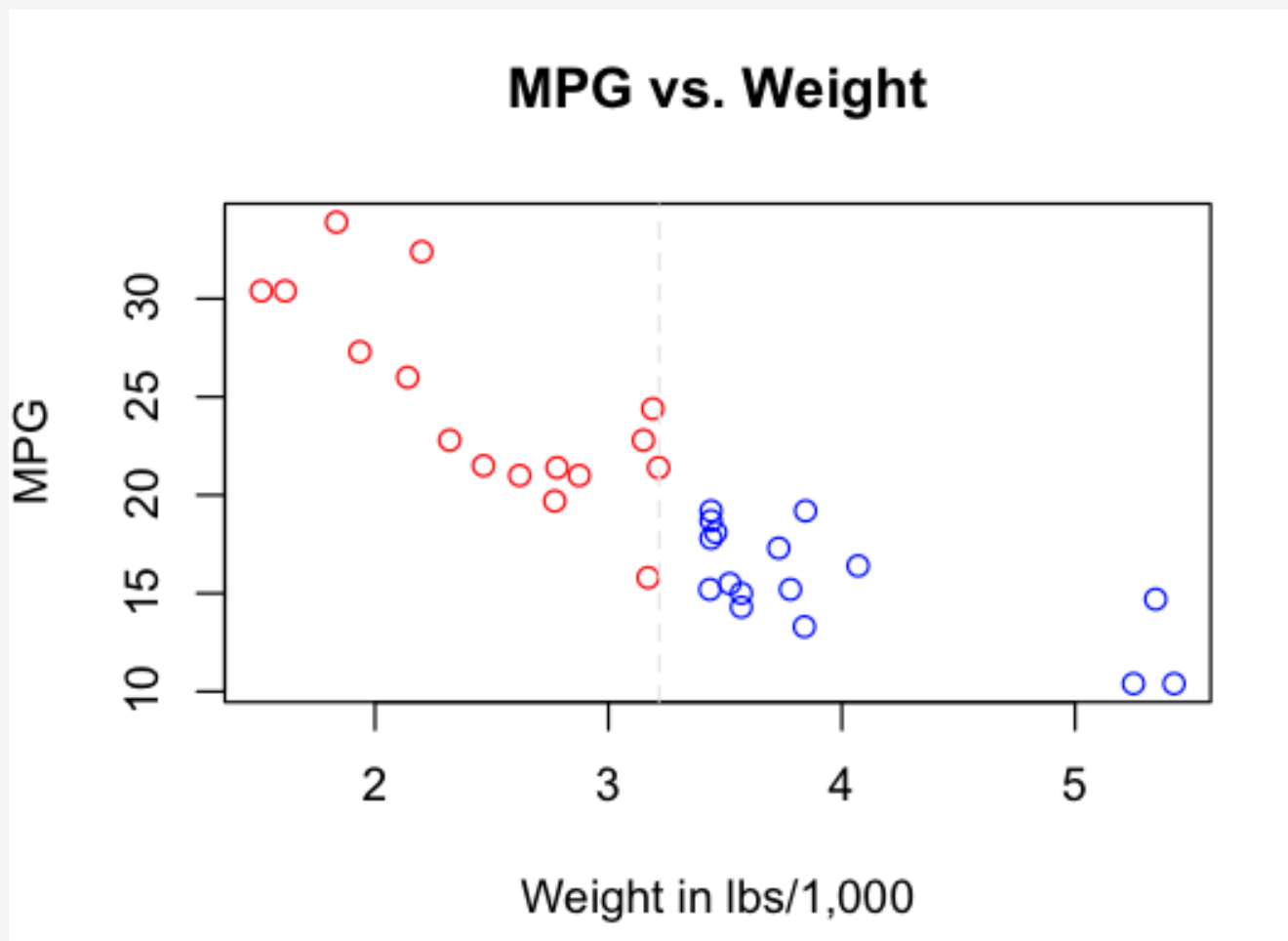
```
points(above.mean$wt,above.mean$mpg,col="blue")
```

# Draw a vertical line where the mean(wt) is

```
abline(v=mean(mtcars$wt),lty=2,col="gray90")
```

# Graphics: Base: Layered

How is this useful ? Well we can add points or lines in stages. This allows us to plot things on an existing plot using specific colors or print characters.



# Graphics: Base: Layered

Unfortunately there is nothing in the existing data set that tells us if a given row's weight value is greater than or below the mean weight. We could handle this a couple of ways - one of which is to use our knowledge of for loops.

```
colvec <- ifelse(mtcars$wt >= mean(mtcars$wt),"blue","red")
```

```
colvec
```

```
[1] "red"  "red"  "red"  "red"  "blue" "blue" "blue" "red"  "red"  "blue"
[11] "blue" "blue" "blue" "blue" "blue" "blue" "blue" "red"  "red"  "red"
[21] "red"  "blue" "blue" "blue" "blue" "red"  "red"  "red"  "red"  "red"
[31] "blue" "red"
```

```
plot(mtcars$wt,mtcars$mpg,col=colvec)
```

# Graphics: Base: text

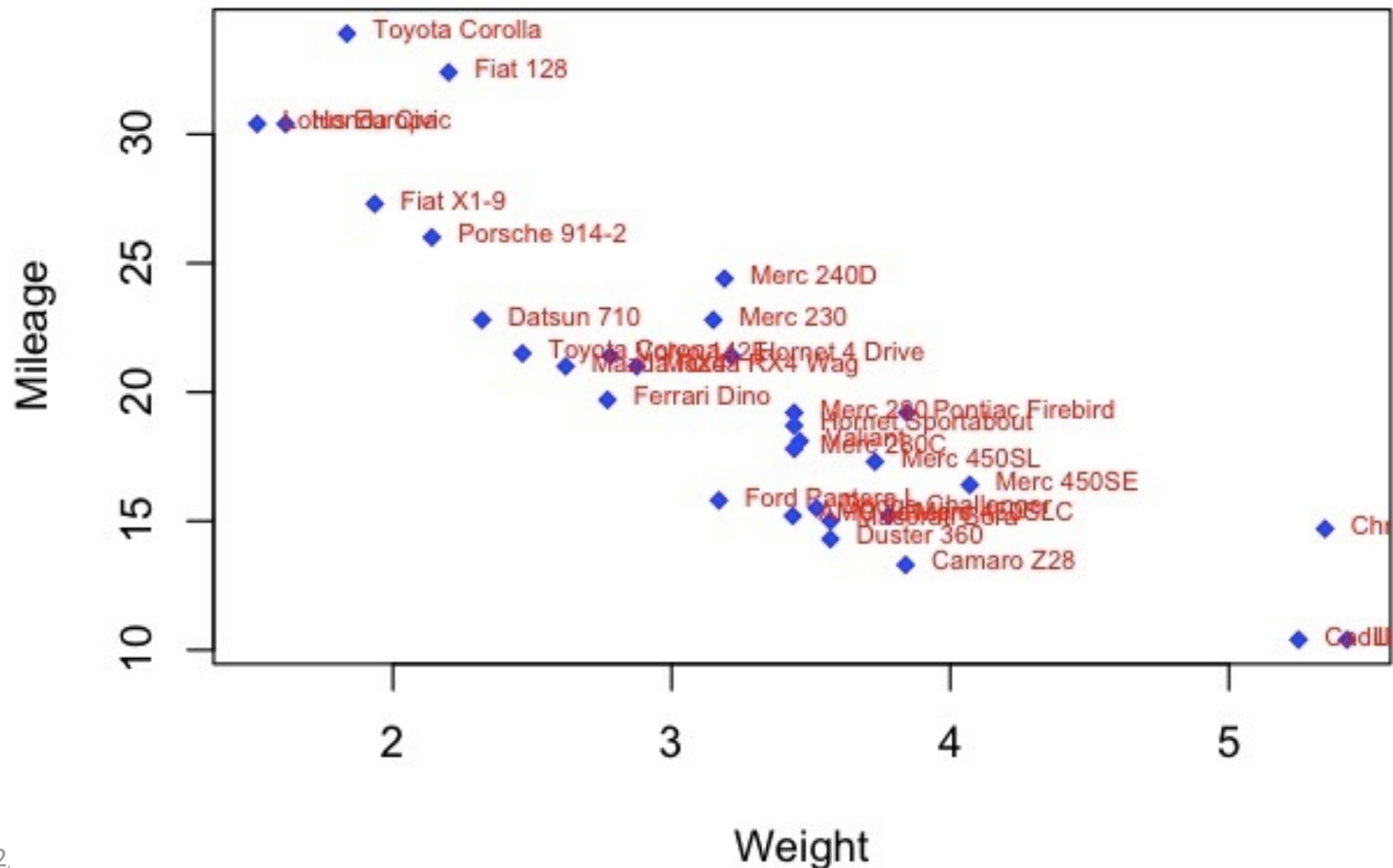
We can add text to our plot with no problem.

```
plot(mtcars$wt, mtcars$mpg, main="Mileage vs. Car Weight",  
      xlab="Weight",  
      ylab="Mileage",  
      pch=18, col="blue")
```

```
text(mtcars$wt, mtcars$mpg, # Note we cannot use the formula in text  
      row.names(mtcars),    # Get the row names  
      cex=0.6,               # Scaling of the font size  
      pos=4,                 # 1=below, 2=left, 3=above, 4=right  
      col="red")
```

# Graphics: Base: text

## Milage vs. Car Weight

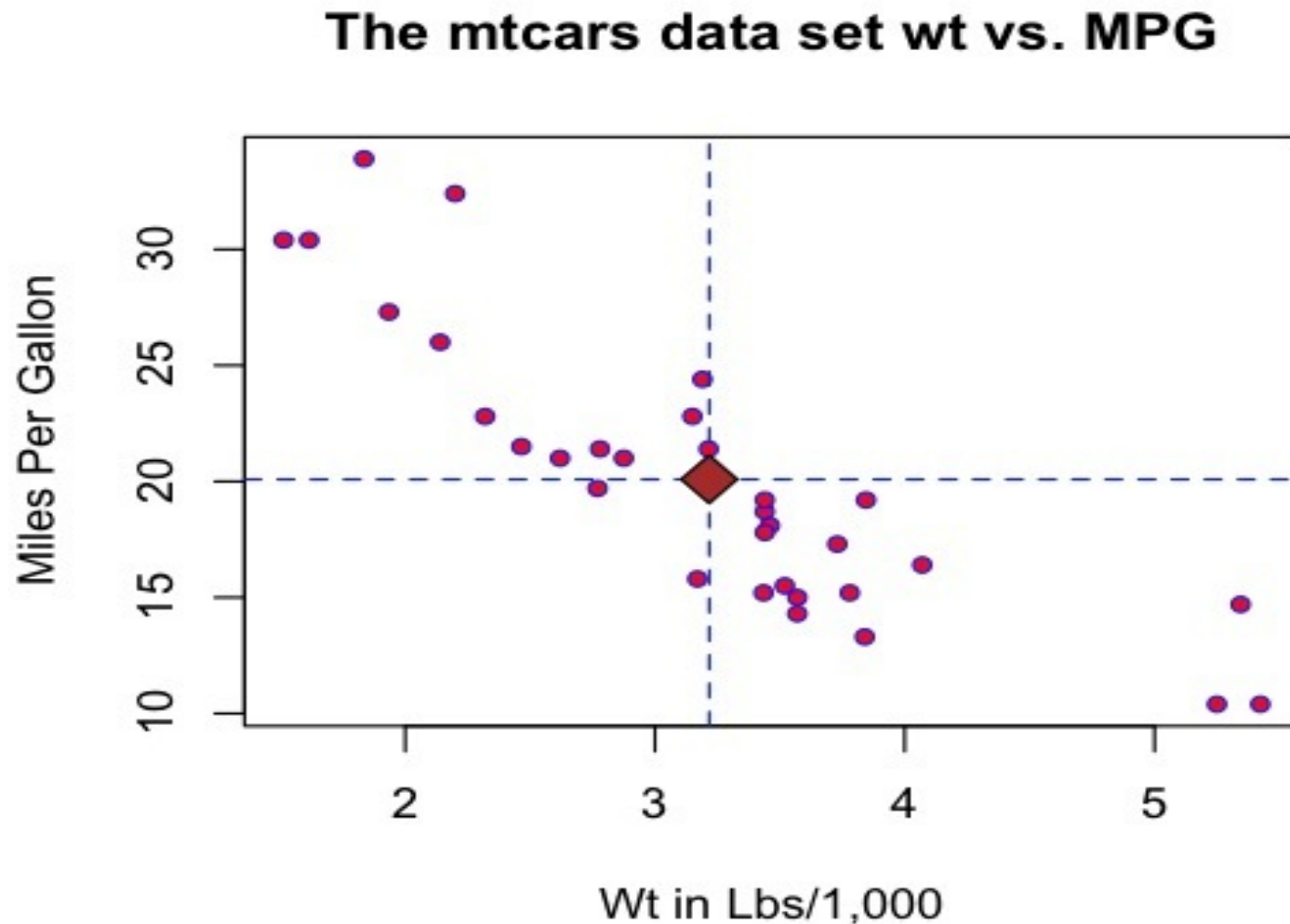


# Supplemental: Advanced Annotation

Let's look at a more involved annotation example. We'll use the same data:

```
plot(mtcars$mpg ~ mtcars$wt, cex=0.8,  
     pch=21, col="blue", bg="red",  
     xlab="Wt in Lbs/1,000",  
     ylab="Miles Per Gallon")  
  
title(main="The mtcars data set wt vs. MPG")  
  
# Next draw a vertical line at the mean of the weight  
  
abline(v=mean(mtcars$wt), lty=2, col="blue")  
  
# Next draw a horizontal line at the man of the MPG  
  
abline(h=mean(mtcars$mpg), lty=2, col="blue")  
  
points(mean(mtcars$wt),          # Draws a diamond at the common mean  
       mean(mtcars$mpg),  
       pch=23, col="black",  
       bg="brown",  
       cex=2)
```

# Supplemental: Advanced Annotation



# Supplemental: Advanced Annotation

**Let's put some custom text on the graph to indicate the mean.**

```
text(mean(mtcars$wt),min(mtcars$mpg),  
      paste("Mean:",round(mean(mtcars$wt),2)),pos=4)  
  
text(min(mtcars$wt),mean(mtcars$mpg),  
      paste("Mean:",round(mean(mtcars$mpg),2)),adj=c(0,1))
```

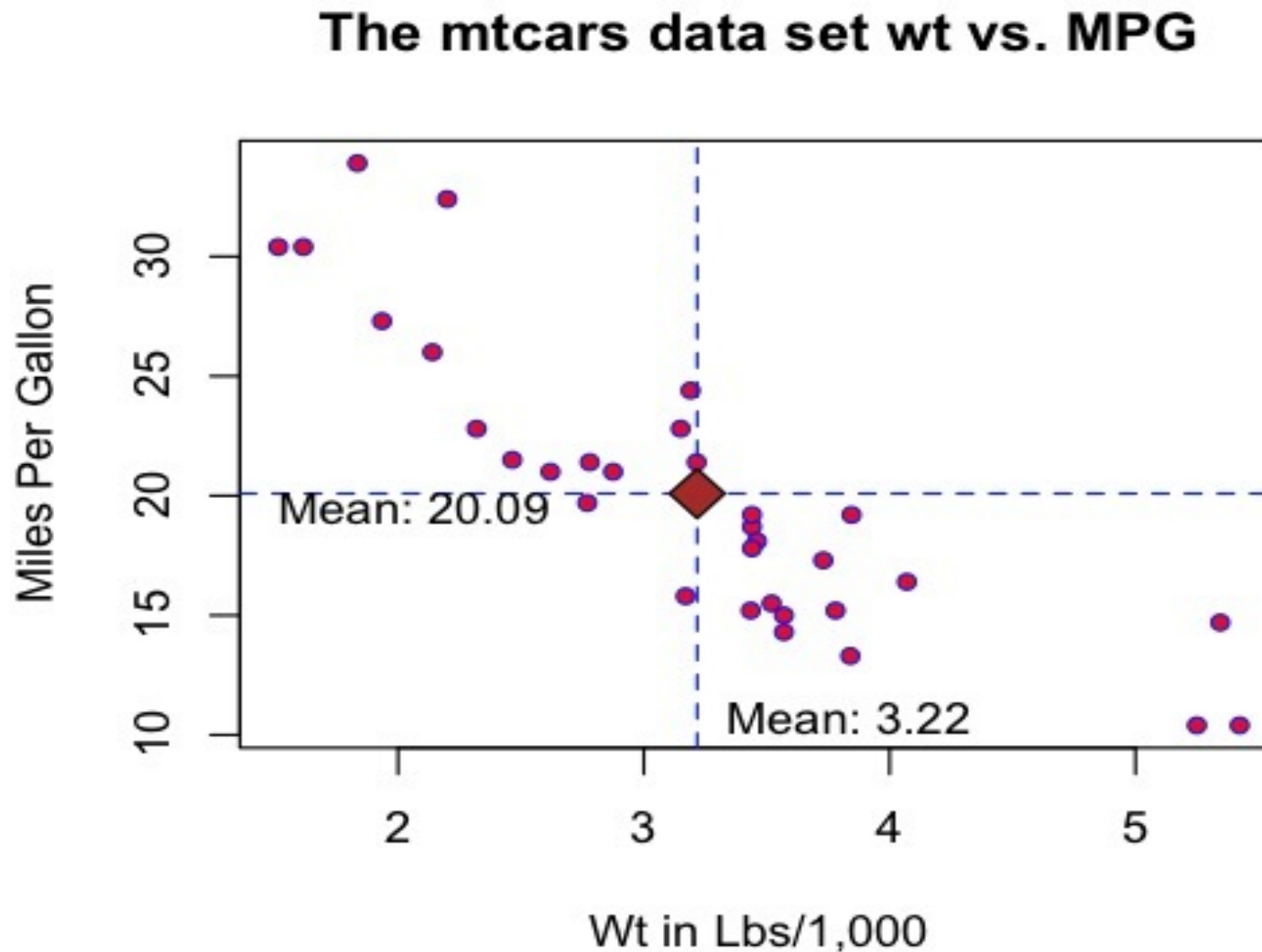
**Note that this is basically equivalent to:**

```
text(3.2,10.4,paste("Mean:",round(mean(mtcars$wt),2)),pos=4)  
  
text(2,20.09,paste("Mean:",round(mean(mtcars$mpg),2)))
```



# Supplemental: Advanced Annotation

Let's look at a more involved annotation example. We'll use the same data:



# Custom Axis

Sometimes we want to draw an axis ourselves because R's defaults aren't what we want. Imagine a set of observations over time such as stock market activity.

Here is a data frame you can read in that tracks actual stock market performance for Microsoft, (MSFT), for each trading day of the year 2014.

```
url <- "http://stevie42.bitbucket.org/bios545r/SUPP.DIR/stock.data.14.csv"
```

```
msft <- read.csv(url)
```

```
head(msft)
```

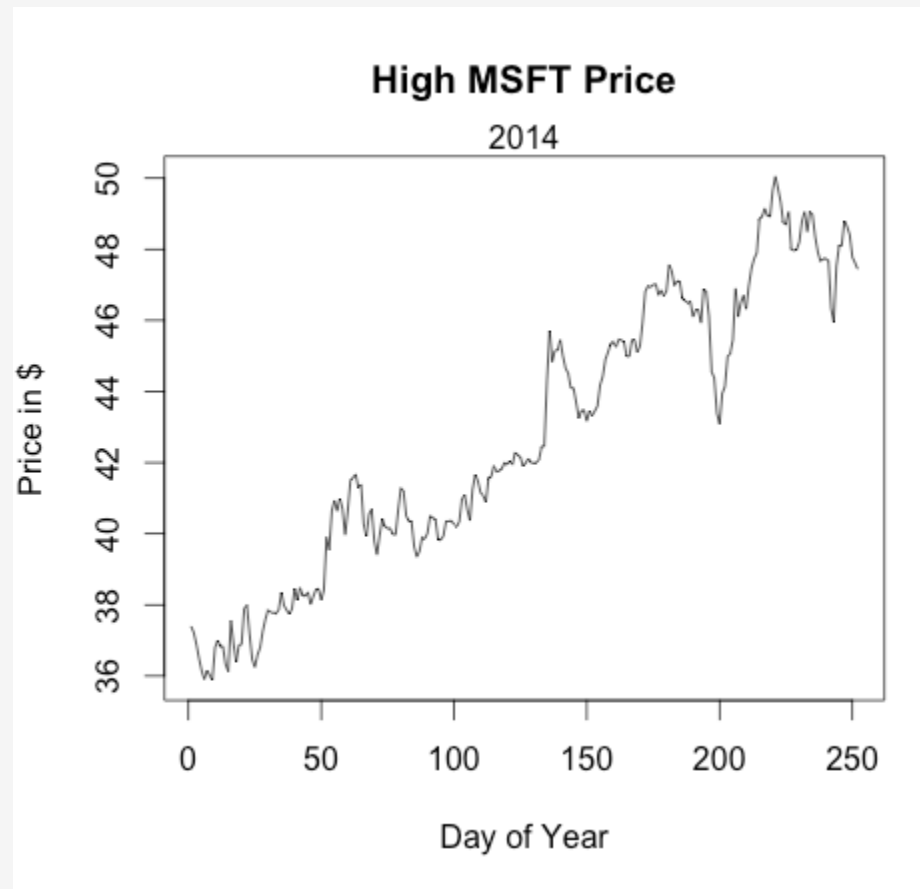
	Date	Open	High	Low	Close	Volume	Adj.Close
1	2014-01-02	37.35	37.40	37.10	37.16	30632200	36.17
2	2014-01-03	37.20	37.22	36.60	36.91	31134800	35.93
3	2014-01-06	36.85	36.89	36.11	36.13	43603700	35.17
4	2014-01-07	36.33	36.49	36.21	36.41	35802800	35.44
5	2014-01-08	36.00	36.14	35.58	35.76	59971700	34.81
6	2014-01-09	35.88	35.91	35.40	35.53	36516300	34.58

# Custom Axis

```
url <- "http://steviep42.bitbucket.org/bios545r/SUPP.DIR/stock.data.14.csv"
msft <- read.csv(url)

plot(msft$High,type="l",main="High MSFT Price",
      xlab="Day of Year",ylab="Price in $")

mtext("2014",3)
```

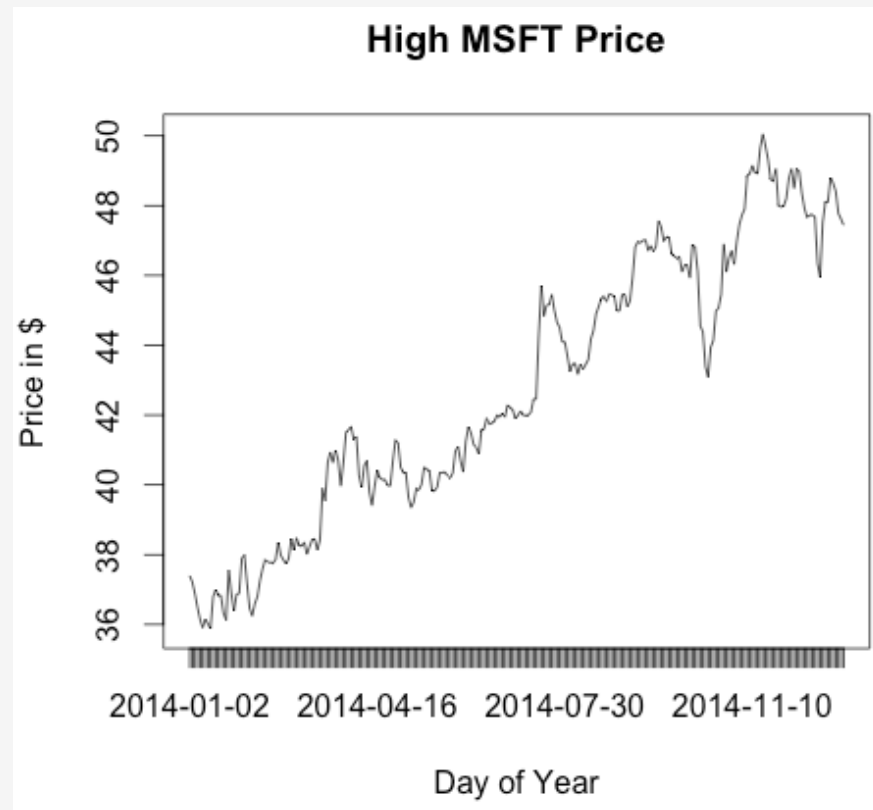


# Custom Axis

The day number is okay but we could also use the actual dates as labels. But that could be a problem. First, we use the `xaxt` argument to suppress the printing of the x-axis

```
plot(msft$High,type="l",main="High MSFT Price",  
     xlab="Day of Year",ylab="Price in $", xaxt="n")
```

```
axis(1,at=1:nrow(msft),labels=msft$Date)
```



# Custom Axis

That wasn't so good because the X-axis got really crowded. We can print labels for the x-axis every 30 days or so using this approach.

We could alter this to accommodate an arbitrary number of days and labels.

Notice how we generate sequence that we then use to index into the Dates.

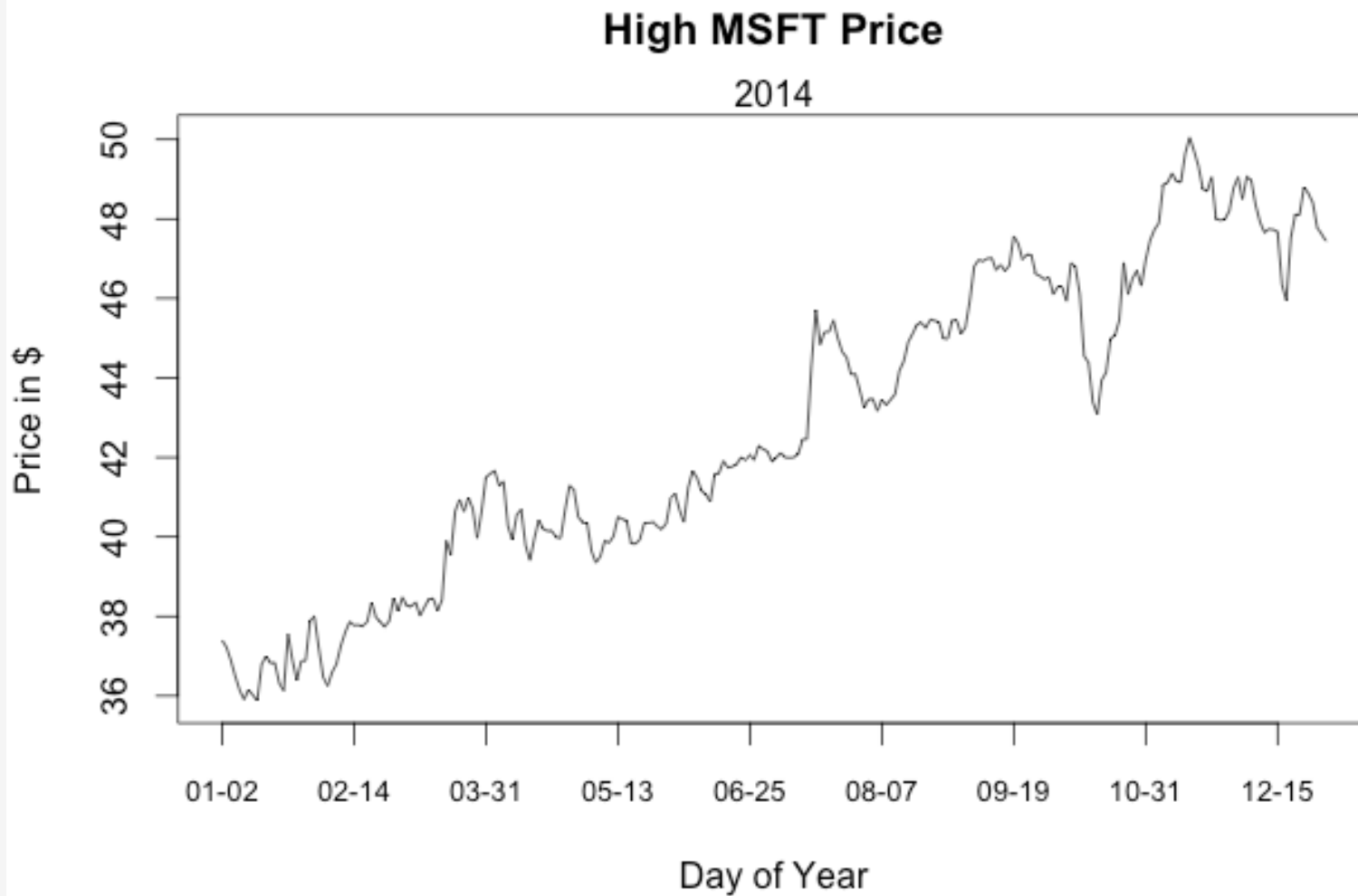
```
plot(msft$High,type="l",main="High MSFT Price",  
      xlab="Day of Year",ylab="Price in $", xaxt="n")  
mtext("2014",3)
```

```
dseq <- seq(1,nrow(msft),30)
```

```
axislabs <- substr(msft$Date[dseq],6,10)
```

```
axis(1, at=dseq, labels=axislabs, cex.axis=0.8)
```

# Custom Axis



# Graphics: Colors

```
length(colors()) # The colors function returns a vector of colors  
[1] 657
```

```
colors()[1:5]  
[1] "white"          "aliceblue"      "antiquewhite"  "antiquewhite1"  
"antiquewhite2"
```

23	bisque4	#8B7D6B	139	125	107
24	black	#000000	0	0	0
25	blanchedalmond	#FFEBCD	255	235	205
26	blue	#0000FF	0	0	255
27	blue1	#0000FF	0	0	255
28	blue2	#0000EE	0	0	238
29	blue3	#0000CD	0	0	205
30	blue4	#00008B	0	0	139
31	blueviolet	#8A2BE2	138	43	226
32	brown	#A52A2A	165	42	42
33	brown1	#FF4040	255	64	64
34	brown2	#EE3B3B	238	59	59
35	brown3	#CD3333	205	51	51

# Graphics: Colors

```
grep("yellow",colors(),value=TRUE)
```

```
"greenyellow"      "lightgoldenrodyellow" "lightyellow"  
"lightyellow1"     "lightyellow2"  
"lightyellow3"     "lightyellow4"         "yellow"  
"yellow1"          "yellow2"              "yellow3"  
"yellow4"          "yellowgreen"
```

```
grep("purple",colors(),value=TRUE)
```

```
"mediumpurple"     "mediumpurple1" "mediumpurple2" "mediumpurple3"  
"mediumpurple4"   "purple"        "purple1"  
"purple2"         "purple3"       "purple4"
```

Get a copy of the PDF Color Chart from:

<http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf>



# Graphics: Colors

R also has some built in palettes that give you a color scheme from which to choose:

Palettes

package:grDevices

R Documentation

Color Palettes

Description:

Create a vector of 'n' contiguous colors.

Usage:

```
rainbow(n, s = 1, v = 1, start = 0, end = max(1, n - 1)/n, alpha = 1)
heat.colors(n, alpha = 1)
terrain.colors(n, alpha = 1)
topo.colors(n, alpha = 1)
cm.colors(n, alpha = 1)
```

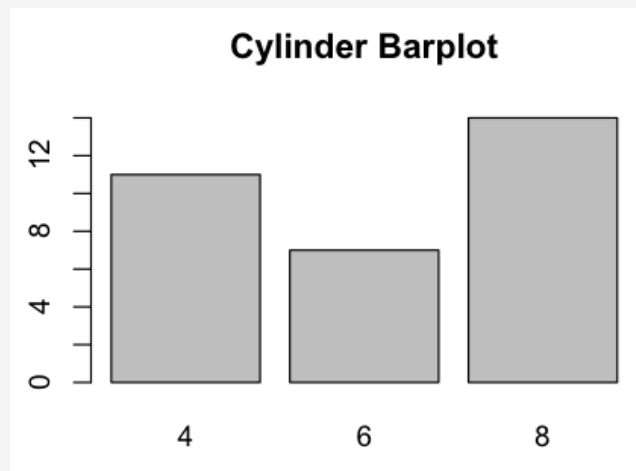
# Graphics: Base: Barplot

If we have some categories we want to look at we can easily visualize it. Barplots are for plotting tables. Let's count up all the cars by cylinder type from mtcars:

```
table(mtcars$cyl)
```

```
 4  6  8  
11  7 14
```

```
barplot(table(mtcars$cyl), axes=T, main = "Cylinder Barplot")
```

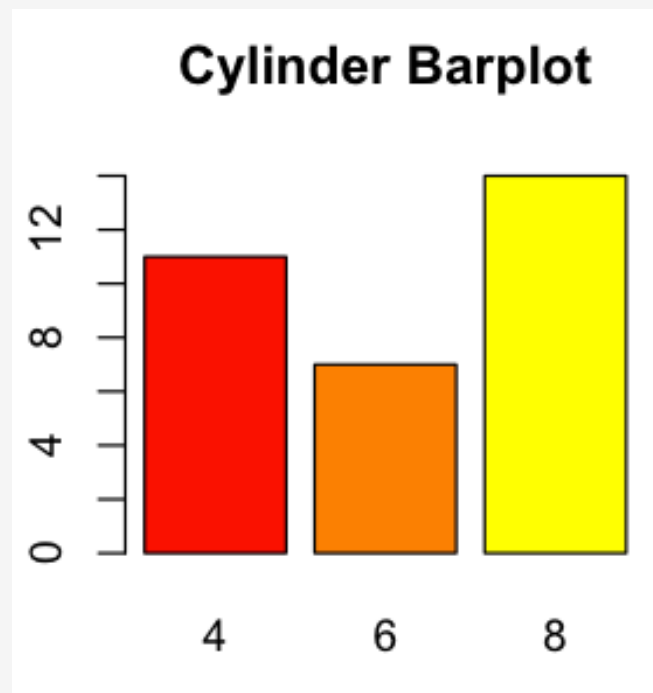


# Graphics: Base: Barplot

```
table(mtcars$cyl)
```

```
 4  6  8  
11  7 14
```

```
barplot(table(mtcars$cyl), axes=T,  
        main = "Cylinder Barplot", col=heat.colors(3))
```

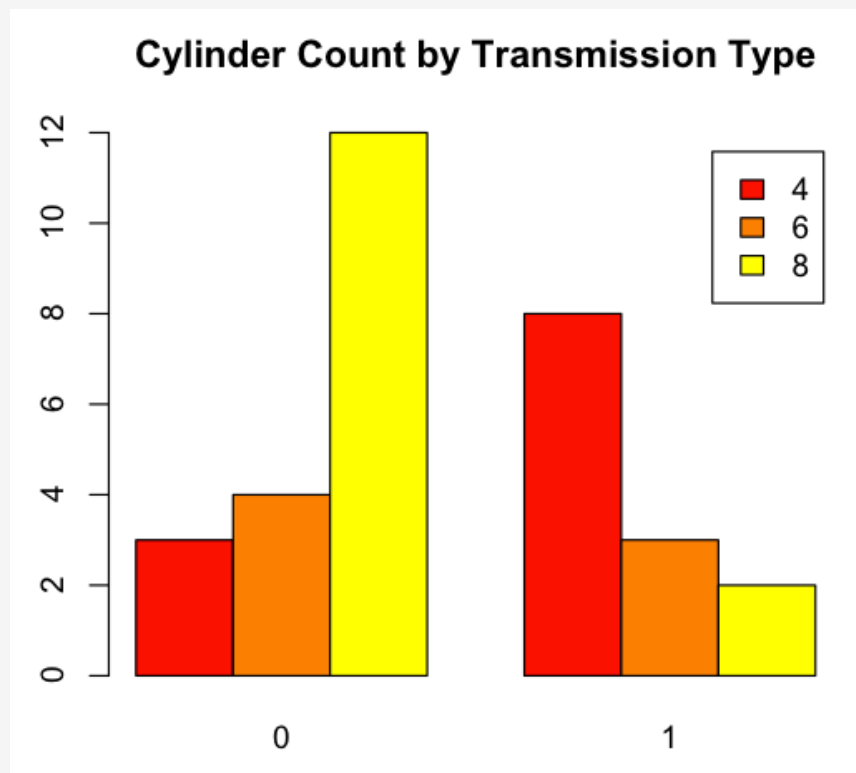


# Graphics: Base: Barplot

```
table(mtcars$cyl,mtcars$am)    # A bigger table
```

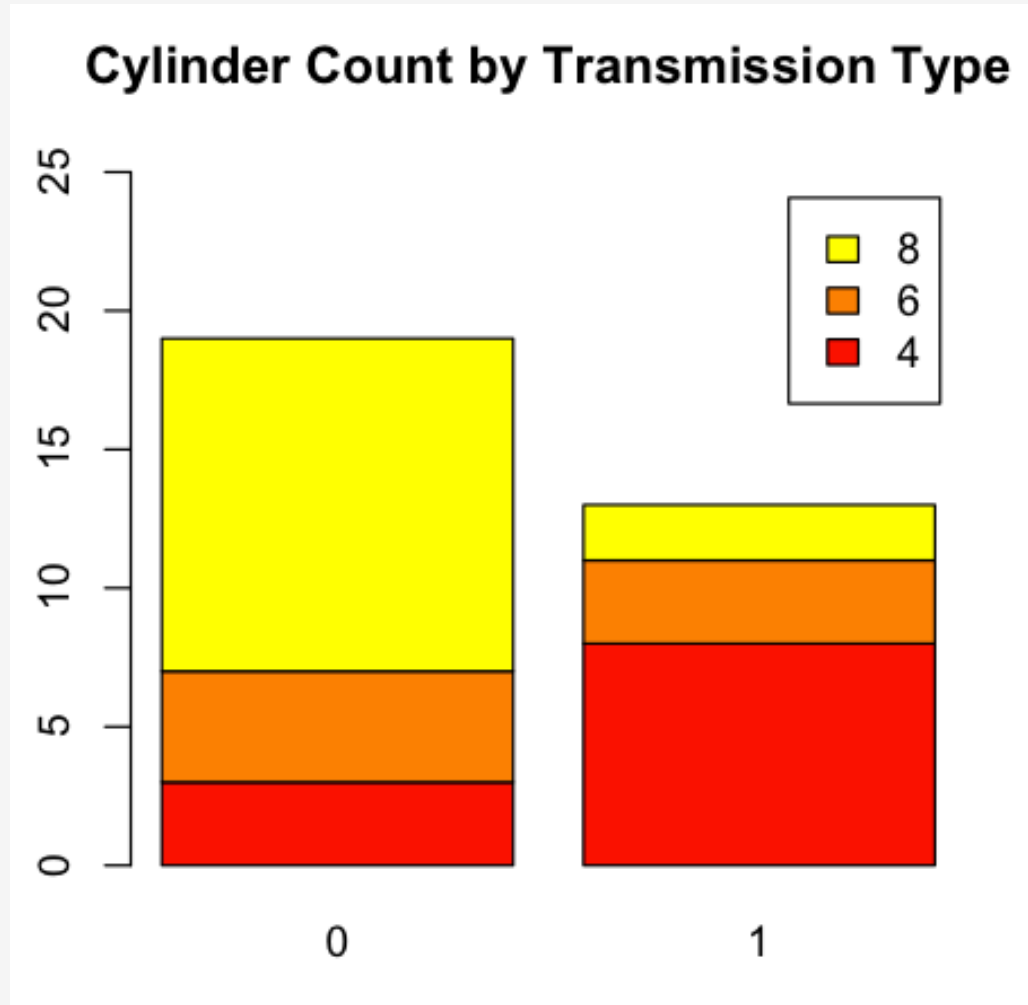
	0	1
4	3	8
6	4	3
8	12	2

```
barplot(table(mtcars$cyl,mtcars$am), legend = T, beside = T,  
col=heat.colors(3), main='Cylinder Count by Transmission Type')
```



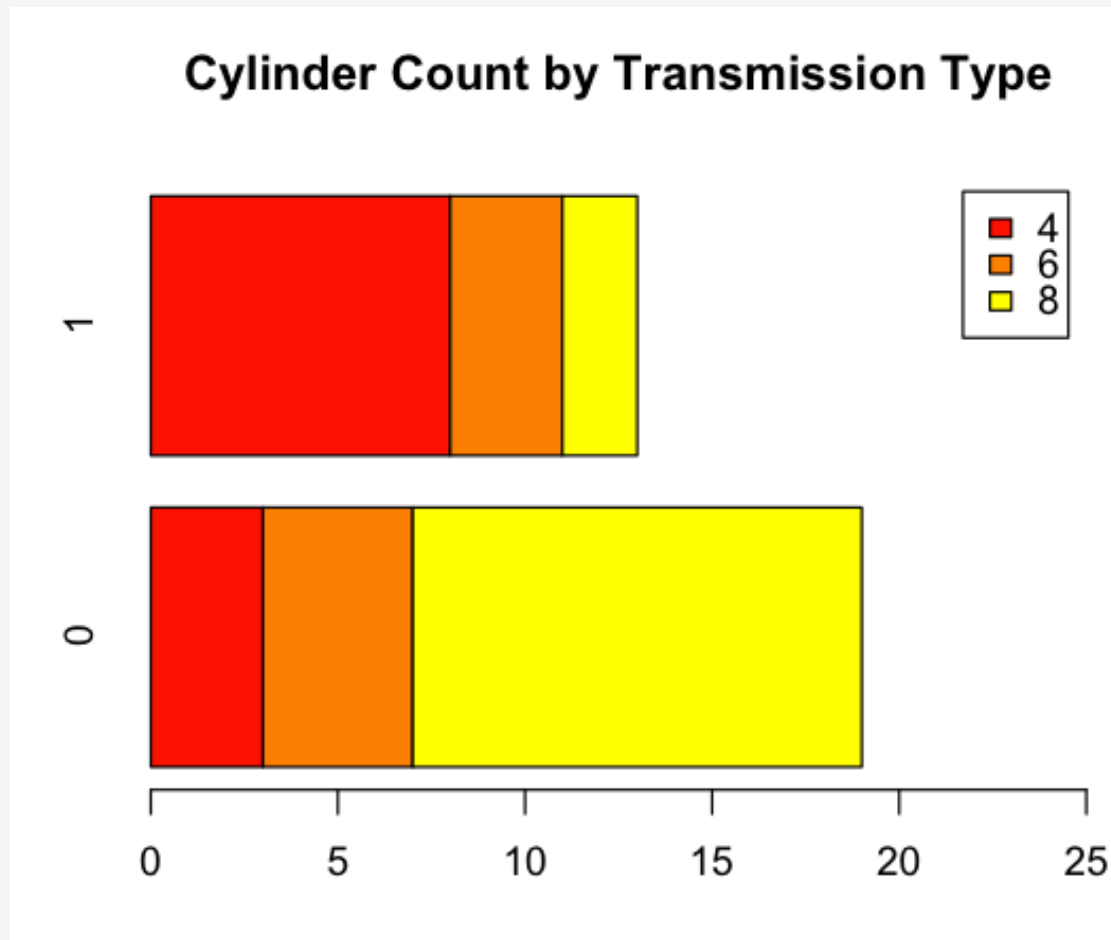
# Graphics: Base: Barplot

```
barplot(table(mtcars$cyl,mtcars$am),legend = T,  
        beside = F, col=heat.colors(3),  
        main='Cylinder Count by Transmission Type',ylim=c(0,25))
```



# Graphics: Base: Barplot

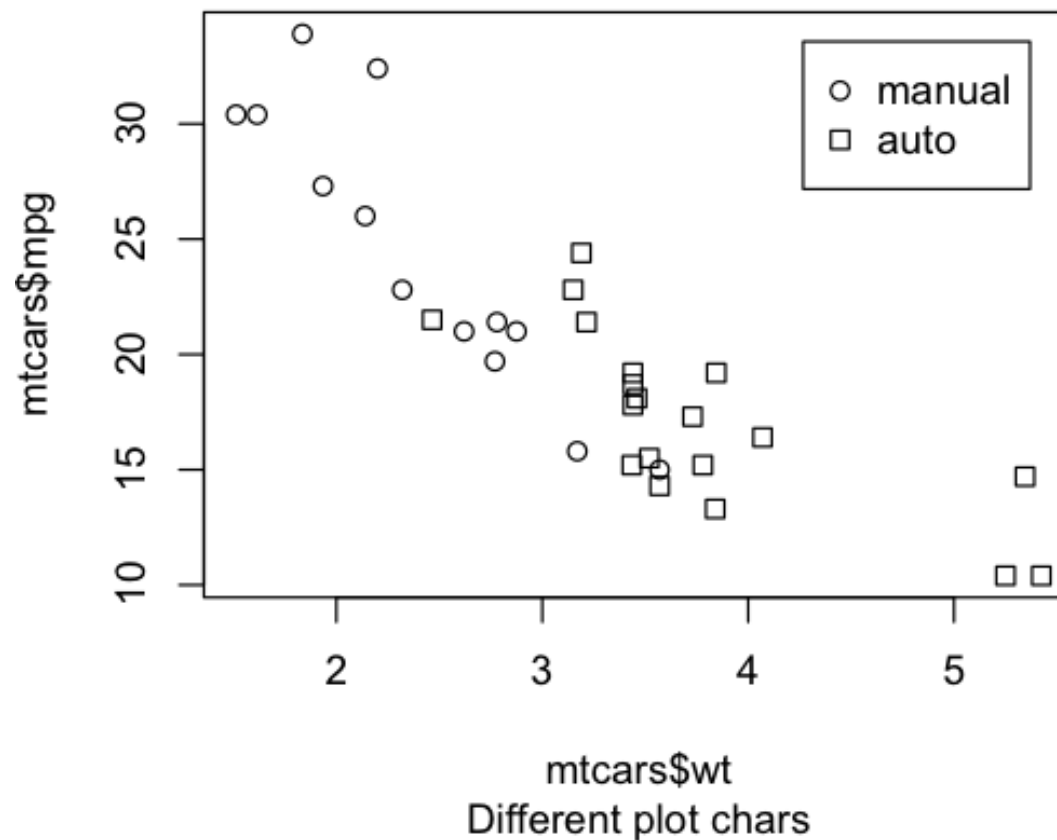
```
barplot(table(mtcars$cyl,mtcars$am),legend = T,  
        beside = F, col=heat.colors(3),  
        main='Cylinder Count by Transmission Type',  
        xlim=c(0,25),horiz=T)
```



# Graphics: Base: Layered Plot

Remember this example ? We used different plot characters to denote manual transmissions vs. automatic. Could we do the same with color ? Of course

**MPG vs. Weight**



# Graphics: Colors

```
mycols <- rainbow(2)
```

```
mycols
```

```
[1] "#FF0000FF" "#00FFFFFF"
```

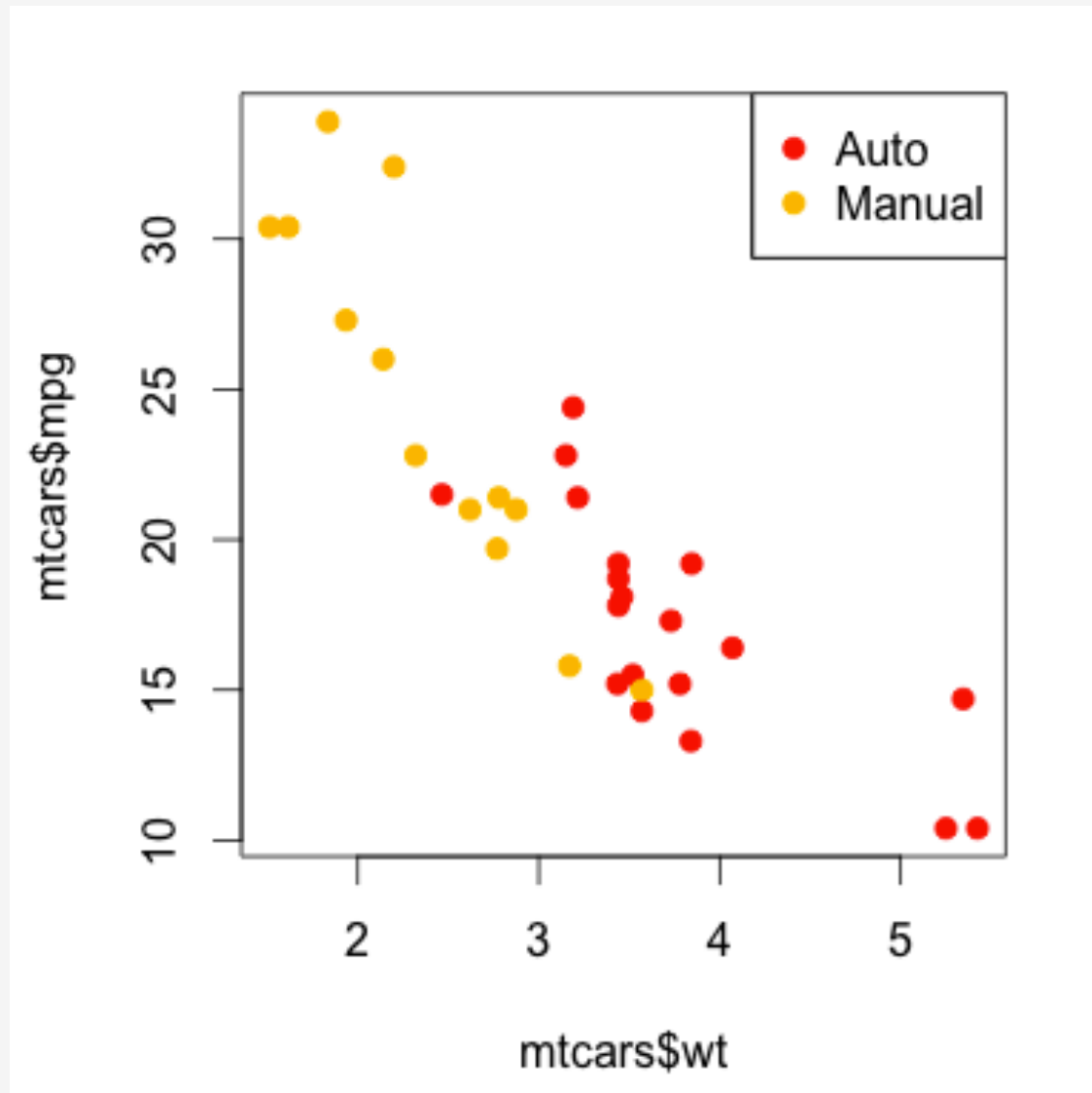
```
# Remember that the transmission types are indicated by a 0 (auto) or  
# 1 (manual). We need to take this into account when indexing into the  
# mycols vector.
```

```
plot(mtcars$wt, mtcars$mpg, col = mycols[mtcars$am+1], pch=19)
```

```
legend("topright", c("Auto", "Manual"), col=mycols, pch=19)
```



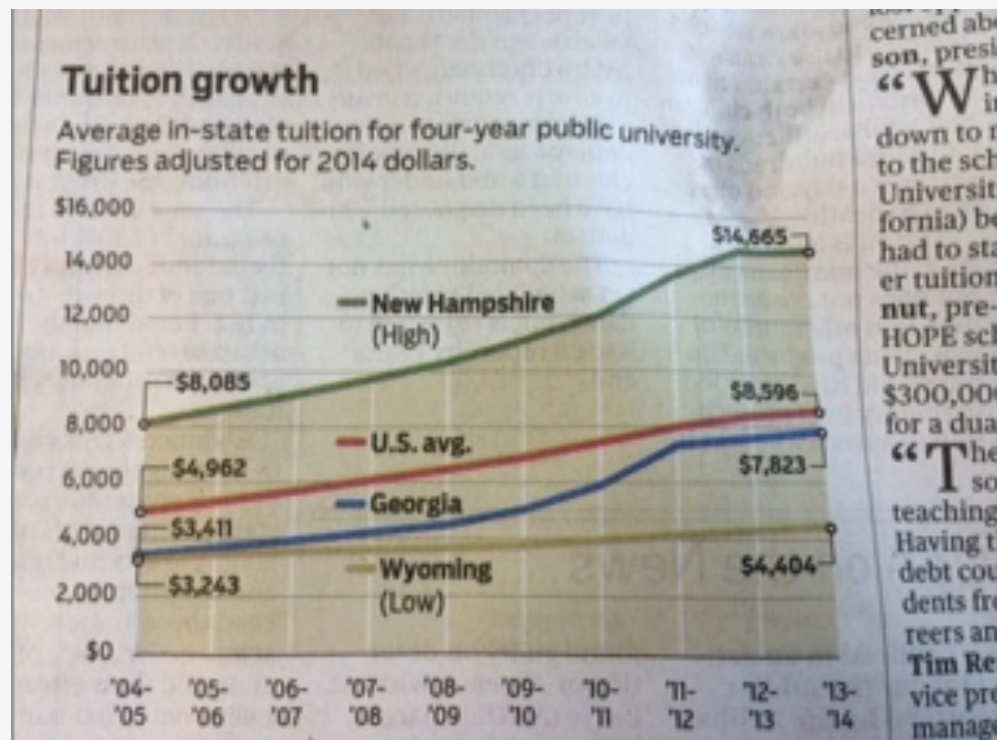
# Graphics: Colors



# Graphics: Reproducing Consumer Graphics

Consumer graphs, like those found in newspapers or news magazines, have lots of "junk" attached to them, which, for a statistician, is unnecessary.

Here is an example found in a copy of the Atlanta Journal Constitution newspaper from some time last year. You can find these in many magazines and papers.



# Graphics: Reproducing Consumer Graphics

I wrote some R code using Base graphics to approximate this.

This took a lot of work since the chart relies on intersecting lines, different colors, custom axes, arrows, and text annotations.

I don't enjoy doing things like this at all. A good plot should tell the story without all the extraneous annotations.

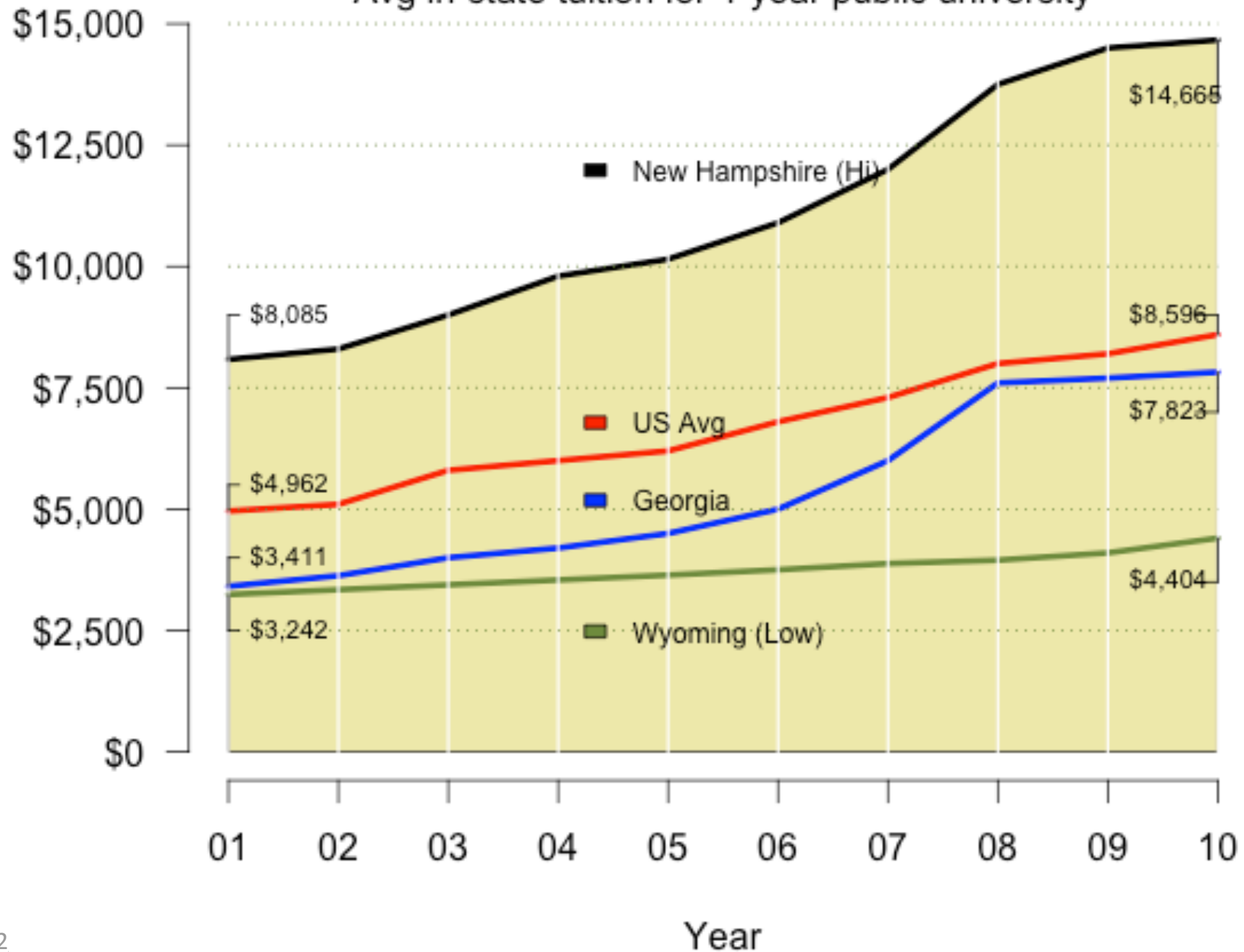
But for the public, this type of chart is standard. See the result on the next slide. It's not a perfect match. The x-axis labels need some more work but it's close enough.

The code can be found at:

<http://stevie42.bitbucket.org/bios545r/SUPP.DIR/ajc.html>

# Tuition Growth

Avg in-state tuition for 4-year public university



Figures adjusted for 2014 dollars

# Graphics: Basic Animation

We can do basic animation with Base graphics although there are dedicated packages that make this easier. We use the Central Limit Theorem as an example.

So we want sample repeatedly from some distribution that is not normal.

We then take the mean of the sample and append it to a vector.

We then plot the histogram of the vector containing the averages

We then call the 'Sys.sleep' function to stop briefly which is what gives us the illusion of animation.

We repeat this some number of times, (timestosamp), and what we will see is a histogram of the sampling distribution of the means, which will be normal.

So even though the starting population from which we sampled is NOT normal the distribution of the sampled means is !

# Graphics: Basic Animation

Here is a function that samples from the uniform distribution repeatedly, computes the mean of the sampled values, adds it to a vector, and then creates a histogram

```
x <- runif(1000000,-3,3) # Get one million vals from a uniform distro

myhist <- function(pop,timestosamp, numtosamp, sleep=0.25) {

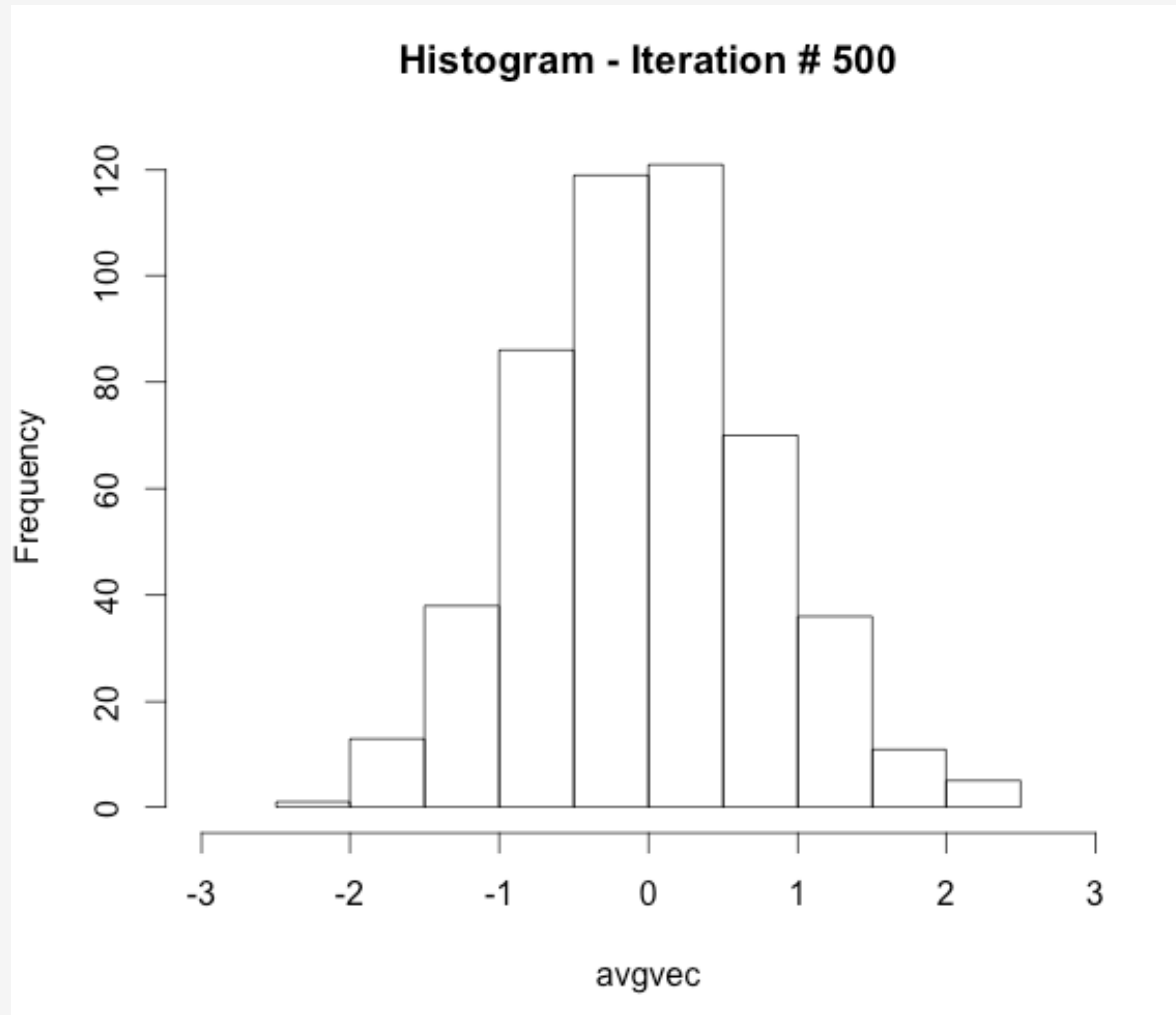
  avgvec <- vector()
  length(avgvec) <- timestosamp

  for (ii in 1:timestosamp) {
    avgvec <- c(avgvec,mean(sample(pop,numtosamp)))
    hist(avgvec,main=paste("Histogram - Iteration #",ii,sep=" "),
         xlim=c(-3,3))
    Sys.sleep(sleep)
  }
}

myhist(x,500,5,sleep=.10)
```

# Graphics: Basic Animation

We can do basic animation with Base graphics although there are dedicated packages that make this easier.



# Graphics: Supplemental: Colors

The 'Indometh' data frame has 66 rows and 3 columns of data on the pharmacokinetics of indometacin (or, older spelling, 'indomethacin').

```
head(Indometh)
```

	Subject	time	conc
1	1	0.25	1.50
2	1	0.50	0.94
3	1	0.75	0.78
4	1	1.00	0.48
5	1	1.25	0.37
6	1	2.00	0.19

```
unique(Indometh$Subject)  # We have six Subjects so generate 6 plots  
[1] 1 2 3 4 5 6  
Levels: 1 < 4 < 2 < 5 < 6 < 3
```