A Tour of Visualization Methods

Steve Pittard

February 2016

What Mode are you in today ?

Most people operate in two modes (at least) when working with data: Exploration and Publication.

In exploration mode the plots do **NOT** have to look pretty. In exploration mode you should focus on the properties of the data and the relationships between the variables - NOT getting the color schemes, fonts, and legends perfect. People waste endless hours perfecting plots that don't really tell a story to begin with.

Exploration Mode

- You get some data and maybe have to clean it up some (guaranteed with "real" data")
- You use the **str()** function on your data frame to see the types of data you have
- You convert things to the desired format (e.g. convert character dates to real dates)
- You do summary statistics on one or more attributes/columns
- You draw some plots
- You come up with some hypotheses
- You do lots of testing and analysis and more plots

Production / Publication Mode

- You have made conclusions supported by rigorous statistical analysis
- You have the results of various tests (p-values) and Odds Rations, RR, etc
- You need to make "pretty plots" to convince the world you are great
- You then try to emulate plots found in academic literature
- You then suffer endlessly trying to get the right colors, annotations, legends, etc

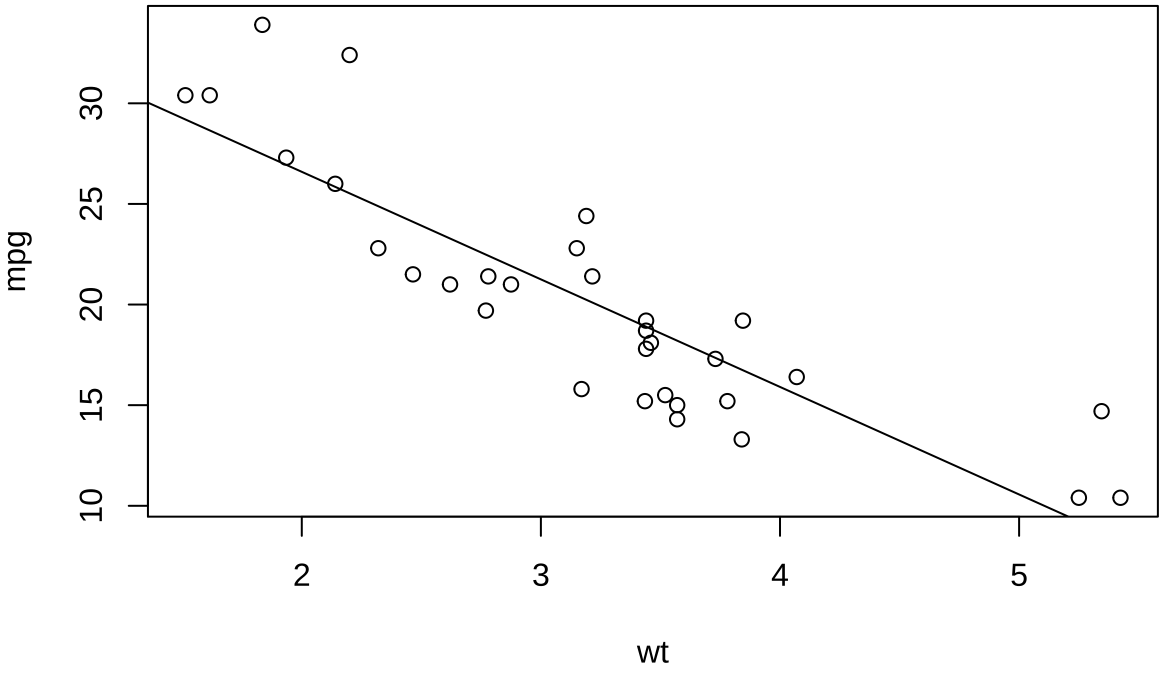
From before - I repeat that in exploration mode the plots do not have to look pretty. As an example here is a very primitive plot that tells a story quite nicely without any annotation.

It is important to know what chart or plot types are appropriate for your data. A full discussion of this is beyond the scope of this document because there are many things you could do to your data to transformit prior to plotting. In any case here is a basic set of guidelines:

Data Description	Appropriate Plot Type(s)
plot(x,y) where x and y are continuous	X/Y scatterplot, pairs, sunflower plots
plot(x,[y]) where x and y are categories (Y is optional)	dotplot, barplot, stacked bar plot
plot(x) where x is a single continuous variable	dotplot, barplot, stripchart, boxplot, desnity, 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

First Plot

```
data(mtcars)
plot(mpg~wt, data=mtcars)

#  As the weight of the automobile goes up the MPG goes down. It looks to be linear
#  which means pehaps we can then do some regression.

mylm <- lm(mpg~wt, data=mtcars)
abline(mylm)
```



Decisions, Decisions, Decisions

There are 4 graphics packages for use with R: Base, Lattice, Grid, and ggplot2. We won't discuss Grid at all because it is a low level language that works behdind the scenes in lattice and ggplot2.

Consider that Base is the oldest followed by lattice followed by ggplot2. lattice and ggplot2 try to improve on ease of use while trying to conform to established principles for visualization.

We'll get more into these but we'll look at Base graphics first because, despite its comparaively primitive nature, it is still quite powerful. Moroever it is quite likely that if you do Google searches for R graphics then you will get back information relating to Base graphics since it has been around the longest.

According to the author of the ggplot2 R package, Hadley Wickham: "Base graphics is good for drawing pictures. ggplot2 is good for understanding data". Well one can understand data in Base or lattice graphics also but I will agree that ggplot2 represents a very good approach.

The strengths of Base Graphics are:

1. It has both high and low level capability which makes it good for developing custom plots.
2. It is very fast.
3. There is lots of support for it to be found when Googling.

4. It’s the oldest and perhaps the most commonly used graphics system

Some weaknesses include:

1. There are a seemingly infinite number of arguments to the plot command that are hard to understand until you experiment A LOT !
2. There are no guiding principles or philosophy at work with Base Graphics. You pick the plot type you want and go to work. Find examples and adapt them to your purpose.
3. Creating “grouped plots” or “paneled plots” can be very involved which in part motivated the creation of packages like lattice and ggplot2

Triage the Data

While it’s possible to dive right in it is best to understand what the variables are in your data in terms of category vs continuous. Generally speaking we like to summarize continuous variables in terms of the factors/categories which allows us to form hypotheses. Of course we can consider the relationships between continuous variables or a even a single continuous variable. Even better we can “cut” up a continuous variable into intervals and treat it like a factor. Common examples include salary ranges or age groups in surveys. We may not need to know someone’s exact age but maybe just their age range.

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
# This next code segment will show us how many unique values there are in each
# column. mtcars is the most used dataset in education

mtcars
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
sapply(mtcars, function(x) length(unique(x)))
```

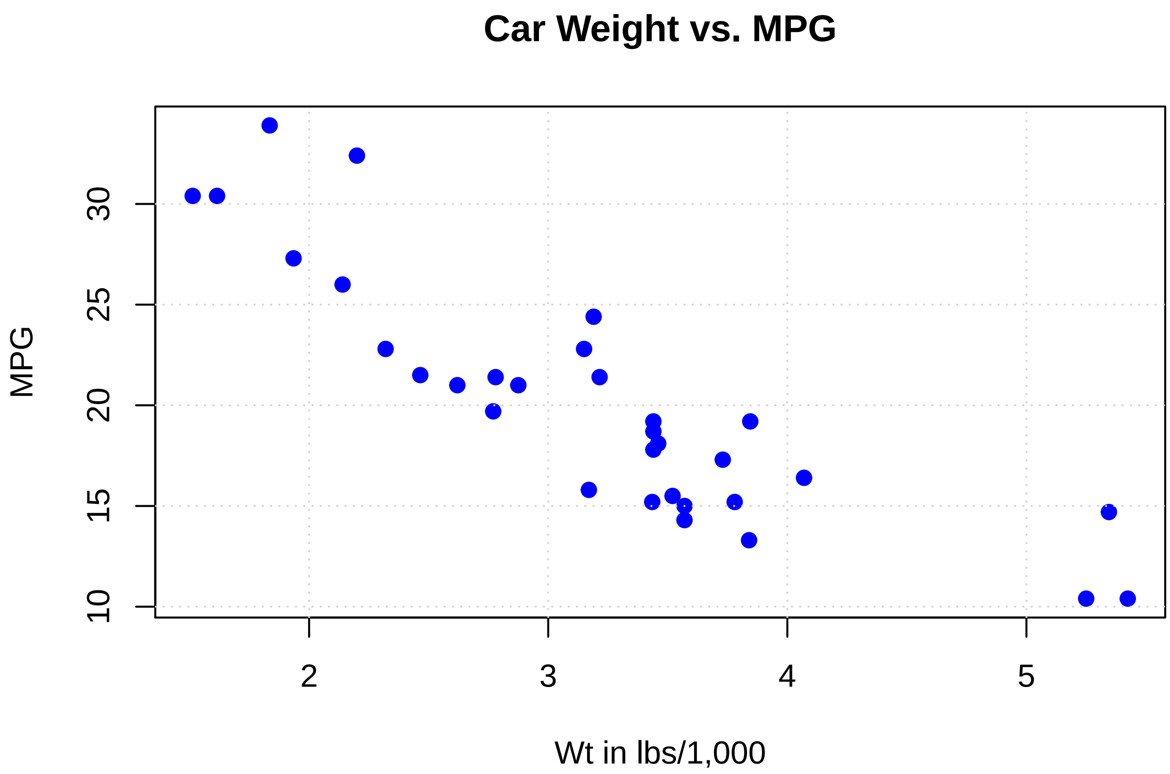
```
##  mpg  cyl disp   hp drat   wt qsec   vs   am gear carb 
##   25    3   27   22   22   29   30    2    2    3    6
```

So what are the factors in this data frame ? Candidates would include any variable that assumes several unique values.

History - Don't Forget The Past.

Base graphics is old but it's amazingly flexible

```r
data(mtcars)     # The most used dataset in R Education !

# Setup some of the annotation strings for later use

title <- "Car Weight vs. MPG"
xlab <- "Wt in lbs/1,000"
ylab <- "MPG"

# Do the actual plot

plot(mtcars$wt,mtcars$mpg,pch=19,
     col="blue",main=title,xlab=xlab,ylab=ylab)

grid()  # Draw a grid
```

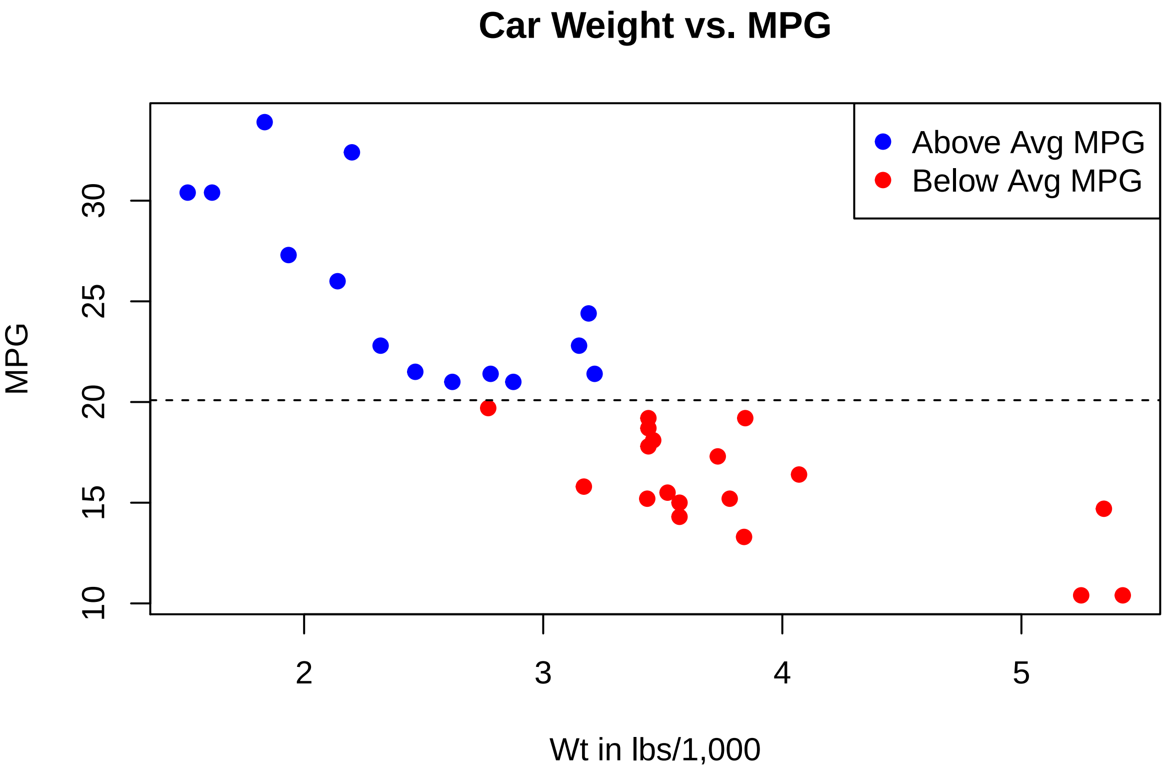


Let's do the same plot except we'll plot the points where the MPG exceeds the mean MPG using the color blue. Those points below the mean MPG will be displayed in red. This is known as creating a **group** plot where specific groups of data within the plot are represented using different colors, point sizes, or print characters.

We can generally find some information within the data frame itself to help us do the grouping.

```
data(mtcars)     # The most used dataset in R Education !
title <- "Car Weight vs. MPG"
xlab <- "Wt in lbs/1,000"
ylab <- "MPG"

# First we setup a blank plot - we do everythig EXCEPT put up the points

plot(mtcars$wt, mtcars$mpg, main=title,xlab=xlab,ylab=ylab, type="n")

# Find just the rows where mpg is >= the mean MPG for the whole data set
# Then use the points function to draw them with color blue

aboveavg <- mtcars[mtcars$mpg >= mean(mtcars$mpg),]
points(aboveavg$wt,aboveavg$mpg,col="blue",pch=19)

# Find the rows where mpg is < the mean MPG for the whole data set
# Use points to draw them with color red

belowavg <-  mtcars[mtcars$mpg < mean(mtcars$mpg),]
points(belowavg$wt,belowavg$mpg,col="red",pch=19)

# Draw a line that represents the average MPG
abline(h=mean(mtcars$mpg),lty=2)
```

```
# Now we draw a legend to identify what color matches which group

legend("topright",c("Above Avg MPG","Below Avg MPG"),pch=19,col=c("blue","red"))
```

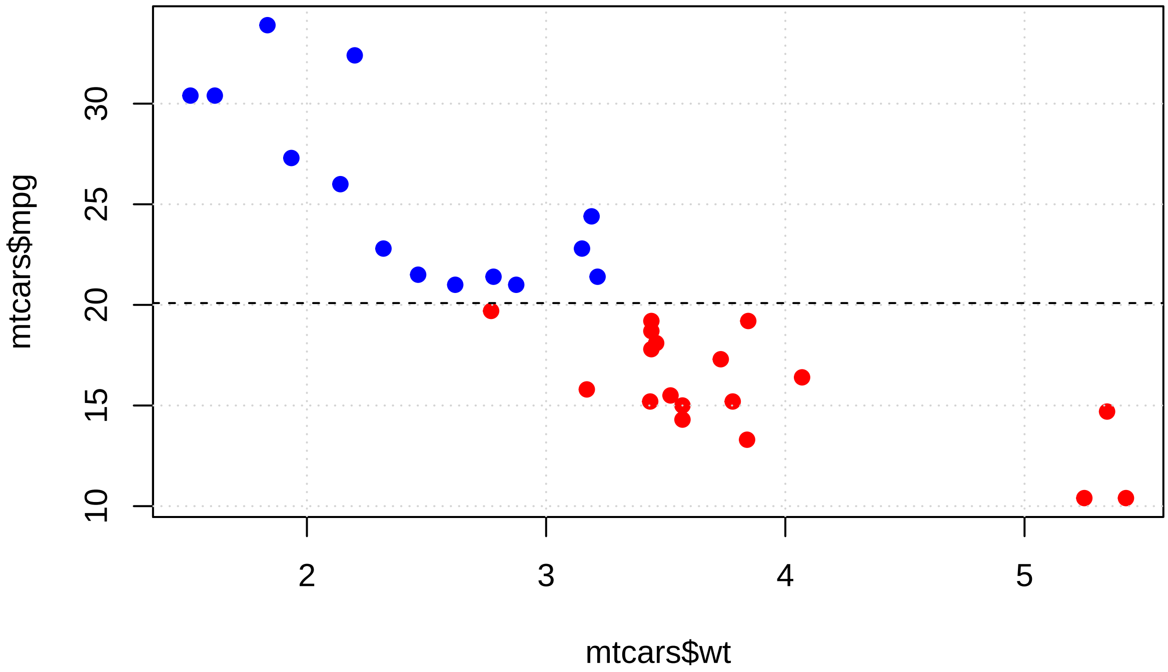


Another way to do this is to use the *ifelse* function to create a color label for each row based on the value of mpg for that row

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

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

```
plot(mtcars$wt, mtcars$mpg, col= colvec, pch=19)
grid()
abline(h=mean(mtcars$mpg),lty=2)
```



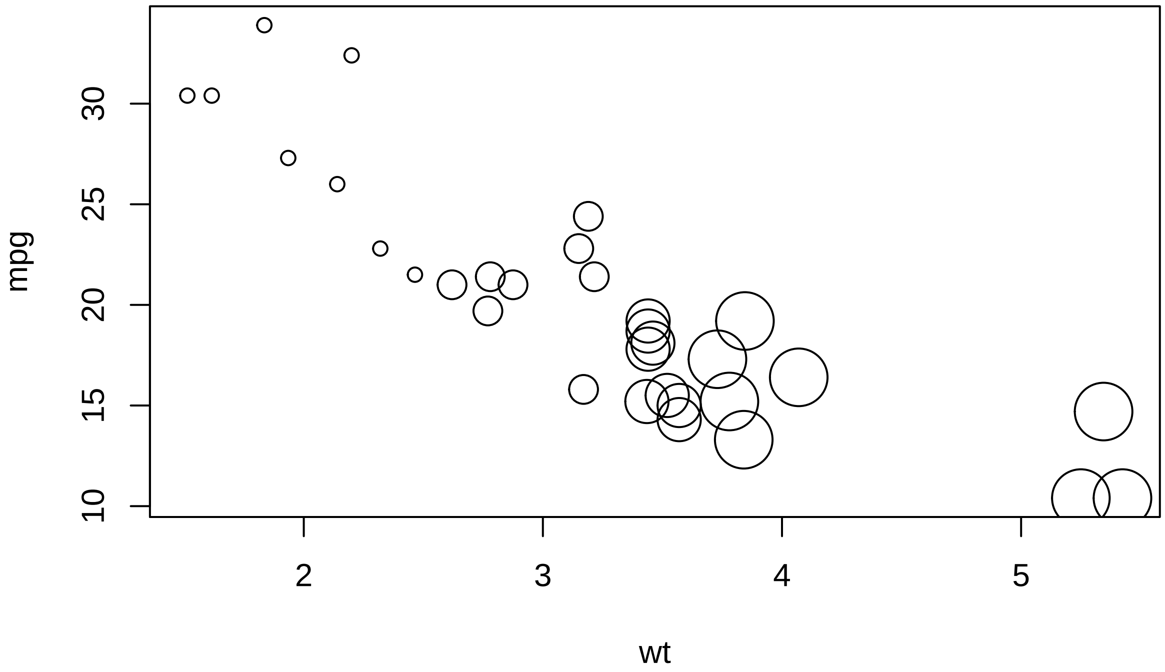
Using Factors

We can create factors out of continuous quantities such as wt. Let's create 4 categories out of the car weight using the quantile function. This is easy. If we label the intervals carefully we can get them to correspond to numbers that are suitable for use with the *cex* option in the plot command which impacts the size of the point being plotted. A cex value of 1 does nothing to the point size. A cex value of less than 1 will shrink the point size. A cex value greater than 1 will increase the size. These are things you don't know until you see them in action.

```
# Chop the wts up based on where the fall into the buckets
# returned by the quantile function

# We then label them using numbers to be given to the cex parameter
# which controls the size of the points being plotted

sizes <- cut(mtcars$wt,breaks=quantile(mtcars$wt),
             include.lowest=TRUE,label=c(0.5,1.0,1.5,1.9))

sizes
```

```
##  [1] 1   1   0.5 1   1.5 1.5 1.5 1   1   1.5 1.5 1.9 1.9 1.9 1.9 1.9 1.9
## [18] 0.5 0.5 0.5 0.5 1.5 1.5 1.9 1.9 0.5 0.5 0.5 1   1   1.5 1
## Levels: 0.5 1 1.5 1.9
```

```
plot(mpg~wt, data=mtcars, cex=as.numeric(sizes))
```



```
# Chop the wts up based on where the fall into the buckets
# returned by the quantile function

# We then label them using numbers to be given to the cex parameter
# which controls the size of the points being plotted

library(RColorBrewer)

sizes <- cut(mtcars$wt,breaks=quantile(mtcars$wt),
             include.lowest=TRUE,label=c(0.5,1.0,1.5,1.9))

# Pick some nice colors because I'm close to being color blind

mycols <- brewer.pal(4,"Set2")

# Okay let's do some tricks here to get the sizes to correspond
# to values 1,2,3,4 so we can index into the mycols vector

idxcolors <- round(as.numeric(sizes))

idxcolors
```

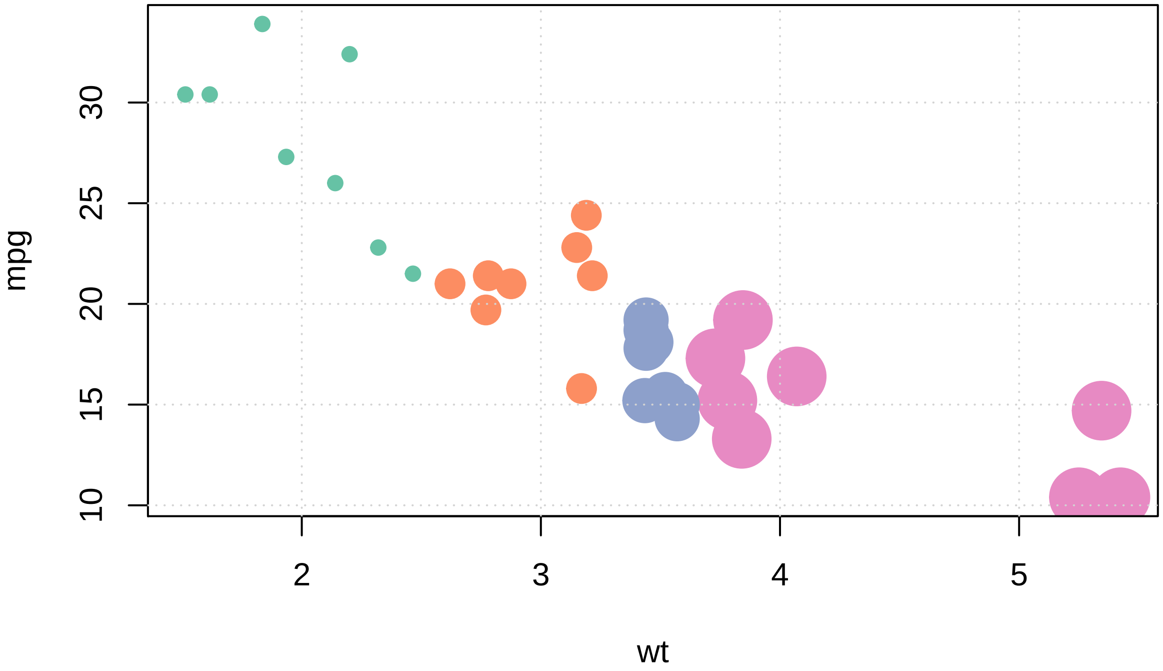
```
##  [1] 2 2 1 2 3 3 3 2 2 3 3 4 4 4 4 4 4 1 1 1 1 3 3 4 4 1 1 1 2 2 3 2
```

```
mycols[idxcolors]
```

```
##  [1] "#FC8D62" "#FC8D62" "#66C2A5" "#FC8D62" "#8DA0CB" "#8DA0CB" "#8DA0CB"
##  [8] "#FC8D62" "#FC8D62" "#8DA0CB" "#8DA0CB" "#E78AC3" "#E78AC3" "#E78AC3"
## [15] "#E78AC3" "#E78AC3" "#E78AC3" "#66C2A5" "#66C2A5" "#66C2A5" "#66C2A5"
## [22] "#8DA0CB" "#8DA0CB" "#E78AC3" "#E78AC3" "#66C2A5" "#66C2A5" "#66C2A5"
## [29] "#FC8D62" "#FC8D62" "#8DA0CB" "#FC8D62"
```

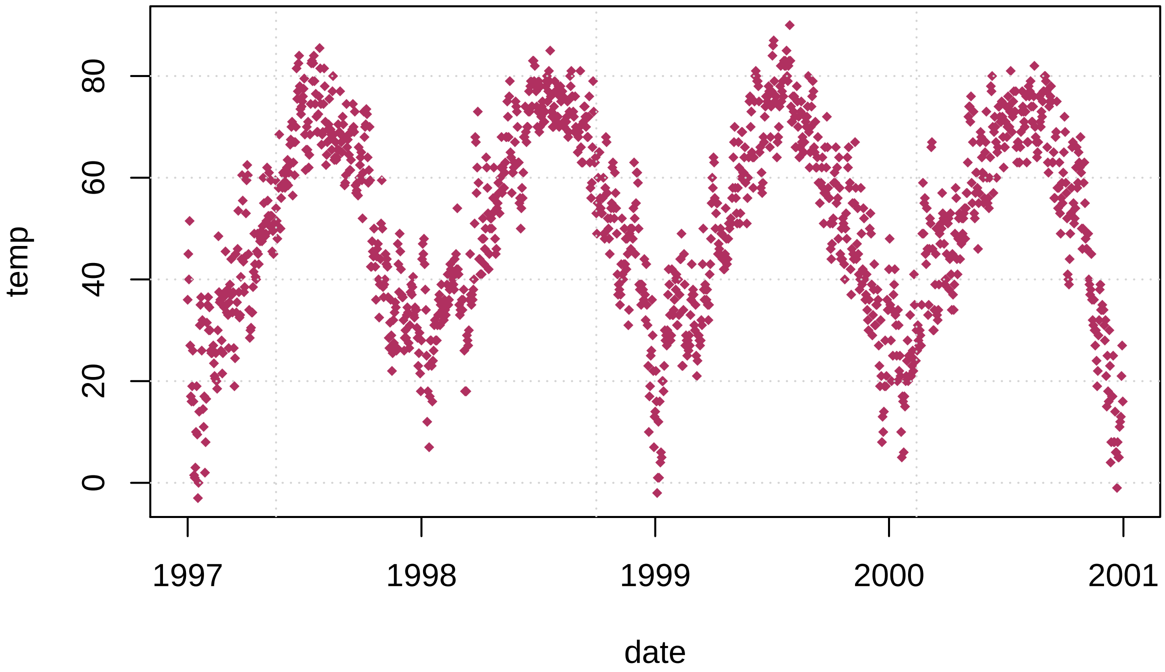
```
plot(mpg~wt, data=mtcars, cex=as.numeric(sizes),
     col=mycols[idxcolors],pch=19)
```

```r
grid()
```



Time Series

```r
# Motivated by
# http://zevross.com/blog/2014/08/04/beautiful-plotting-in-r-a-ggplot2-cheatsheet-3/
# Data comes from National Morbidity and Mortality Air Pollution Study (NMMAPS)

nm <- read.csv("http://zevross.com/blog/wp-content/uploads/2014/08/chicago-nmmaps.csv", as.is=TRUE)

# We need to convert the date strings into actual Dates

nm$date <- as.Date(nm$date)

# We pull out the records after 12/31/96

nm <- nm[nm$date > as.Date("1996-12-31"),]

# All we care about is the year in XXXX format

nm$year <- substring(nm$date,1,4)

# So we plot it

plot(temp~date,data=nm,pch=18,cex=0.7,col="maroon")
grid()
```



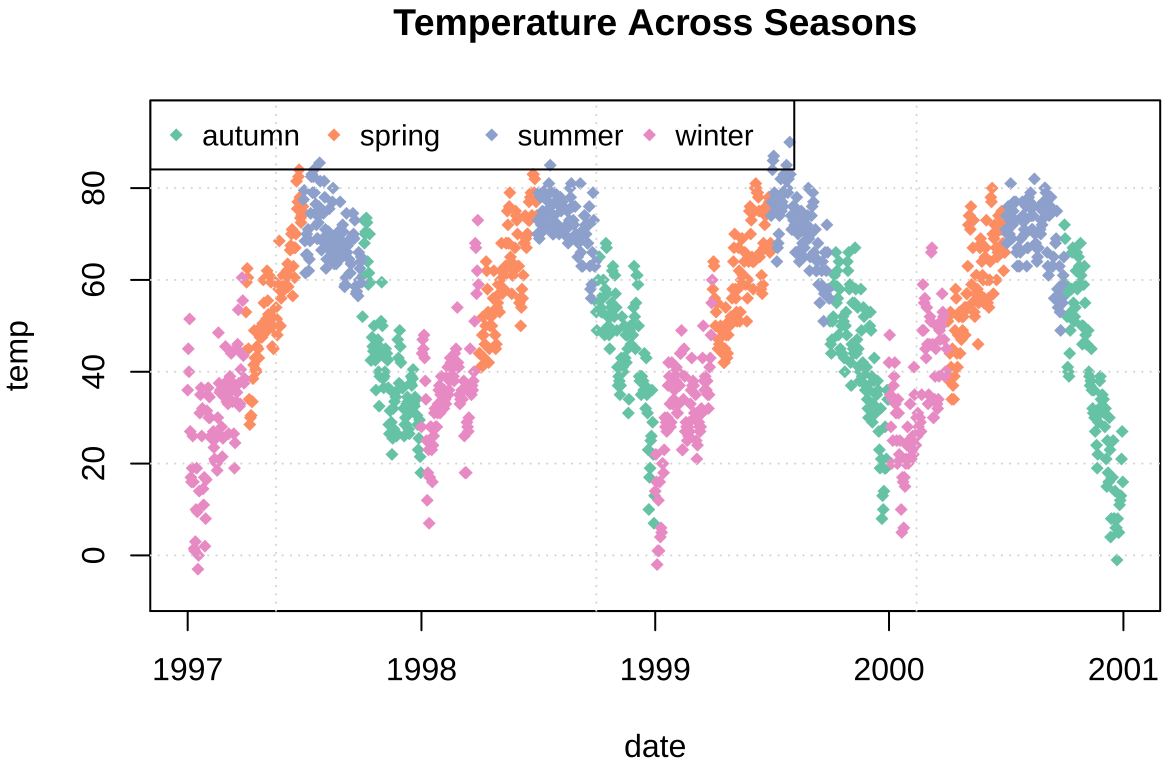
If we want to use different colors for each of the four seasons we have to do this manually. That is, we:

1. have to pick the colors ourslves, setup a null plot
2. use the split command to partition the data frame into 4 groups for each of the four seasons.
3. then loop through the four "splits" and use the *points* function to draw the points for a given season while indexing into the color vector.

```
library(RColorBrewer)

# Need to make some extra room for the legend

ylim <- c(min(nm$temp)-5,max(nm$temp)+5)

plot(temp~date,data=nm,pch=18,cex=0.7,type="n",ylim=ylim)
mycols <- brewer.pal(4,"Set2")

# Split the nm data frame on season labels

splits <- split(nm,nm$season)

str(splits,1)    # The str() function let's us peek at the structure
```

```
## List of 4
##  $ autumn:'data.frame':  368 obs. of  10 variables:
##  $ spring:'data.frame':  364 obs. of  10 variables:
##  $ summer:'data.frame':  368 obs. of  10 variables:
##  $ winter:'data.frame':  361 obs. of  10 variables:
```

```
# Loop through the splits and use the points function to
# plot the temperature per season


for (ii in 1:length(mycols)) {
  points(splits[[ii]]$date,splits[[ii]]$temp,
         col=mycols[ii],pch=18,cex=0.9)
}
title(main="Temperature Across Seasons")
legend("topleft",names(splits),pch=18,col=mycols,horiz=TRUE,cex=0.9)
grid()
```

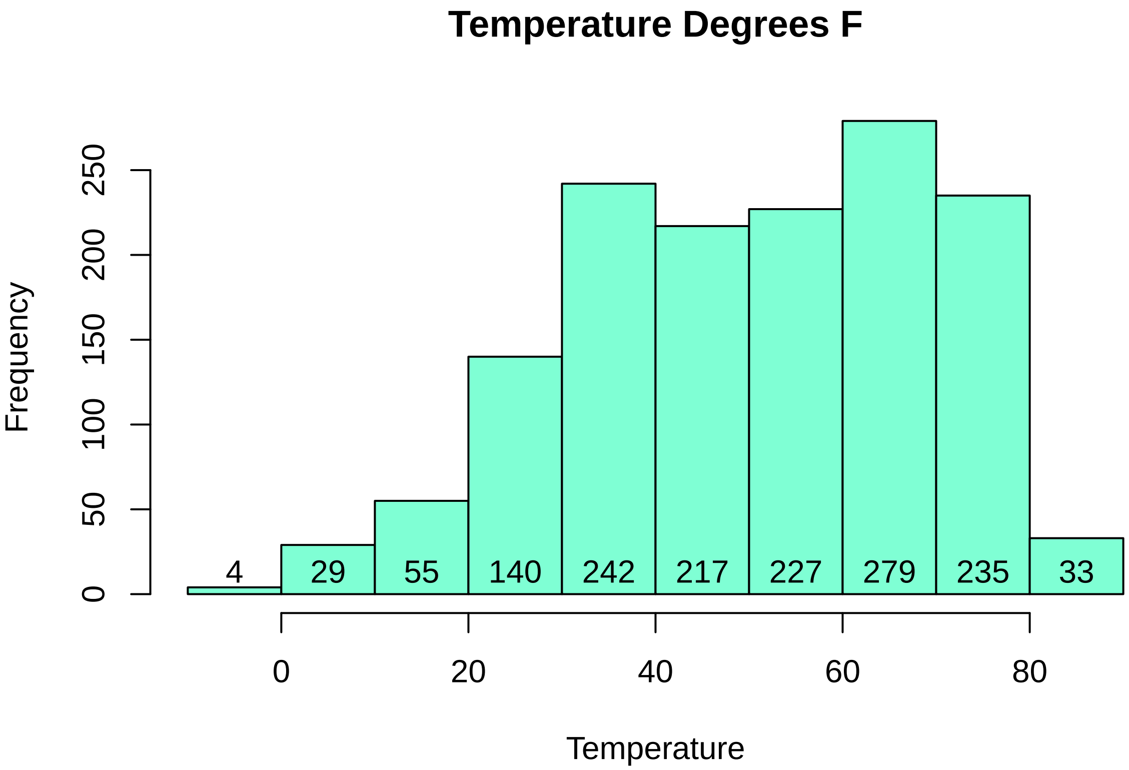


That was a lot of work. Or was it ?

At a minimum you have to know something about R programming to get this done. Of course having knowledge of programming techniques is never a bad thing although, for beginners, when one is visualizing data it is usually best to use tools that don't require alot of effort to view the data in interesting ways.

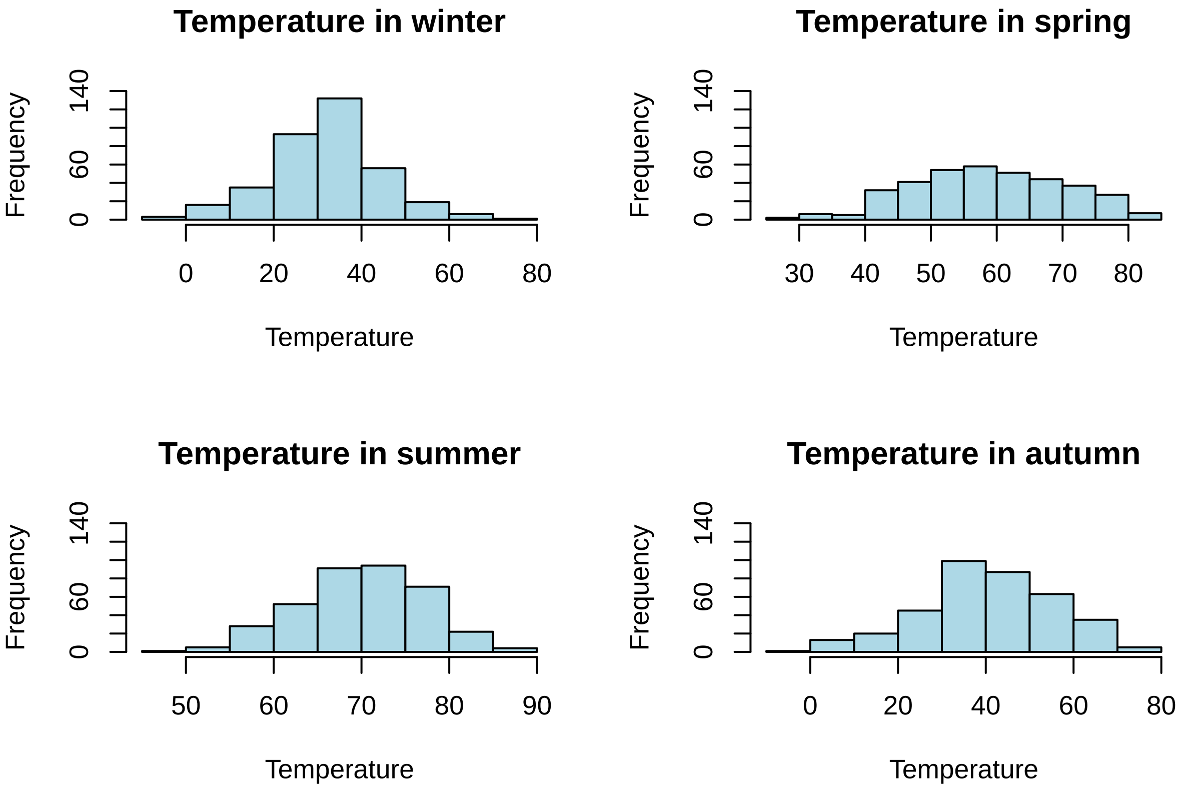
Each chart type has it's own function with it's own arguments that influence that particular chart outcome. You have to dig into the help pages to figure out how to do things such as putting text on the chart

```
r <- hist(nm$temp,col="aquamarine",breaks=12,main="Temperature Degrees F",xlab="Temperature")

text(r$mids, r$density, r$counts, adj = c(.5, -.5), col = "black")
```



If we wanted to look at temperatures as “conditioned”" by season we need to use the *par* function to carve out 4 panels. We try to plot such that the axes for all plots are the same to enable better comparisons.

```
par(mfrow=c(2,2))  # Get 2 rows by 2 columns

xlab <- "Temperature"
ylim <- c(0,140)
col <- "lightblue"

seasons <- unique(nm$season)
tempstr <- "Temperature in"
for (ii in 1:length(seasons)) {
  title  <- paste(tempstr,seasons[ii],sep=" ")
  hist(nm[nm$season==seasons[ii],]$temp, main=title,xlab=xlab,ylim=ylim,col=col)
}
```

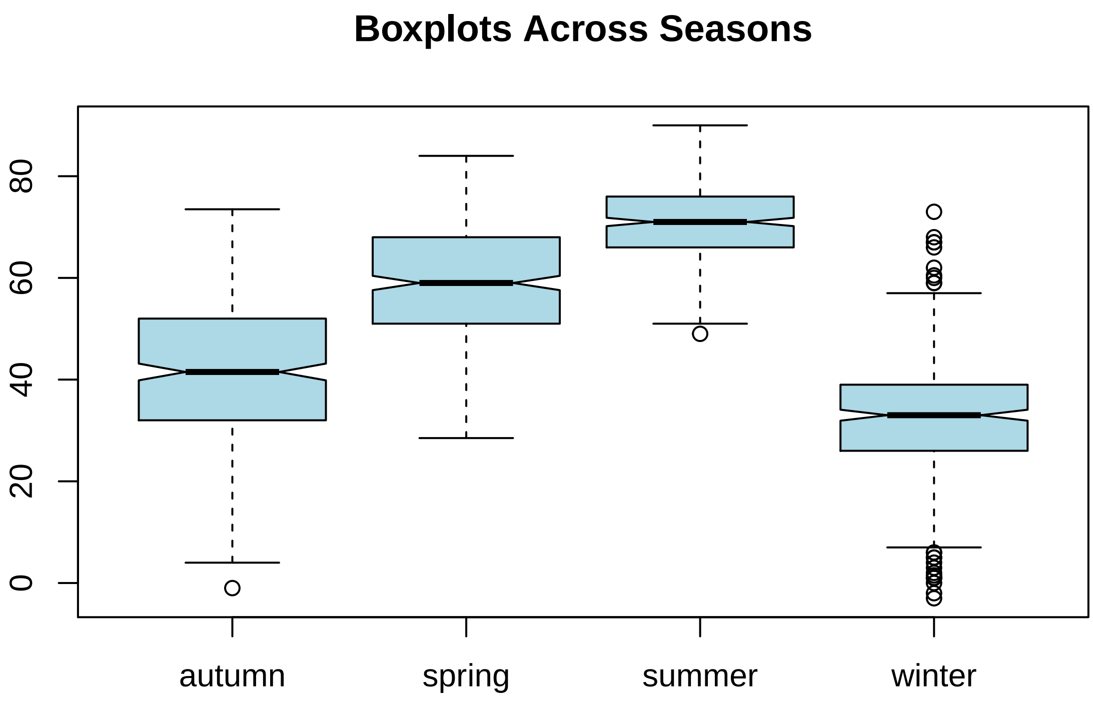


```
par(mfrow=c(1,1)) # Reset plot window to 1 row by 1 column
```

Other Base chart types

Let's look at boxplots to see what is necessary to do some comparisons across seasons. It turns out this isn't so bad at least in this case.

```
title <- "Boxplots Across Seasons"
boxplot(temp~season,data=nm,notch=TRUE,col="lightblue",main=title)
```

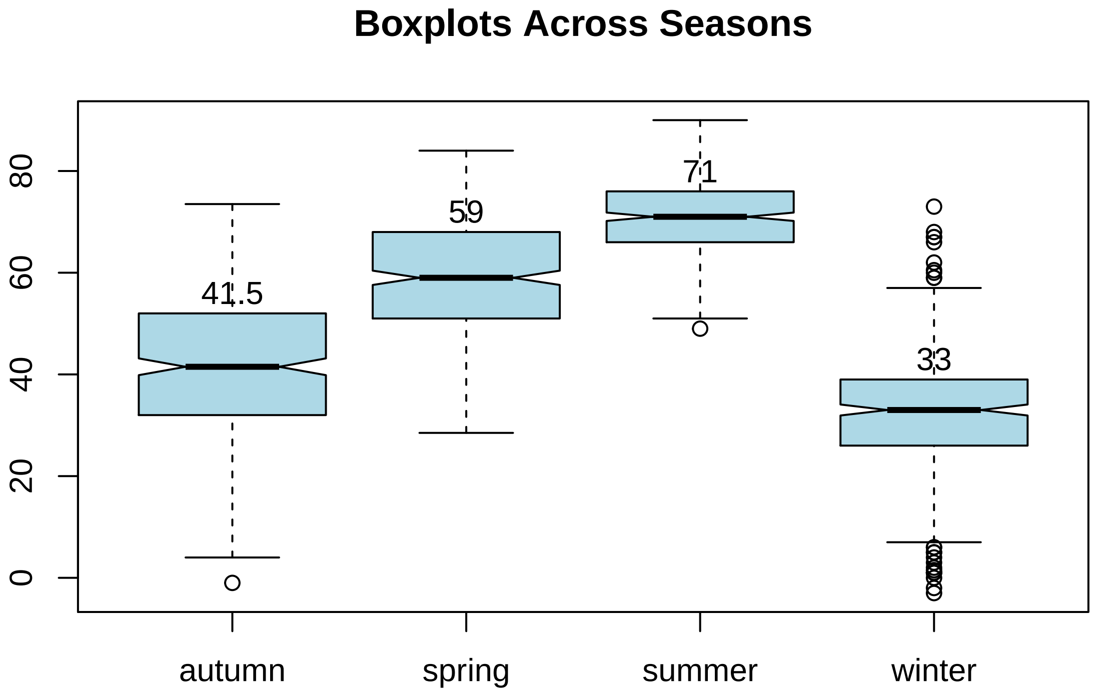


What if we want to annotate the plot say with the median value ? Note that we don't really have to do this because the graph makes it fairly obvious what the median value is. The "trick" is that **boxplot** returns information if we choose to capture it into a varable.

```
box.out <- boxplot(temp~season,data=nm,notch=TRUE,col="lightblue",main=title)

box.out
```

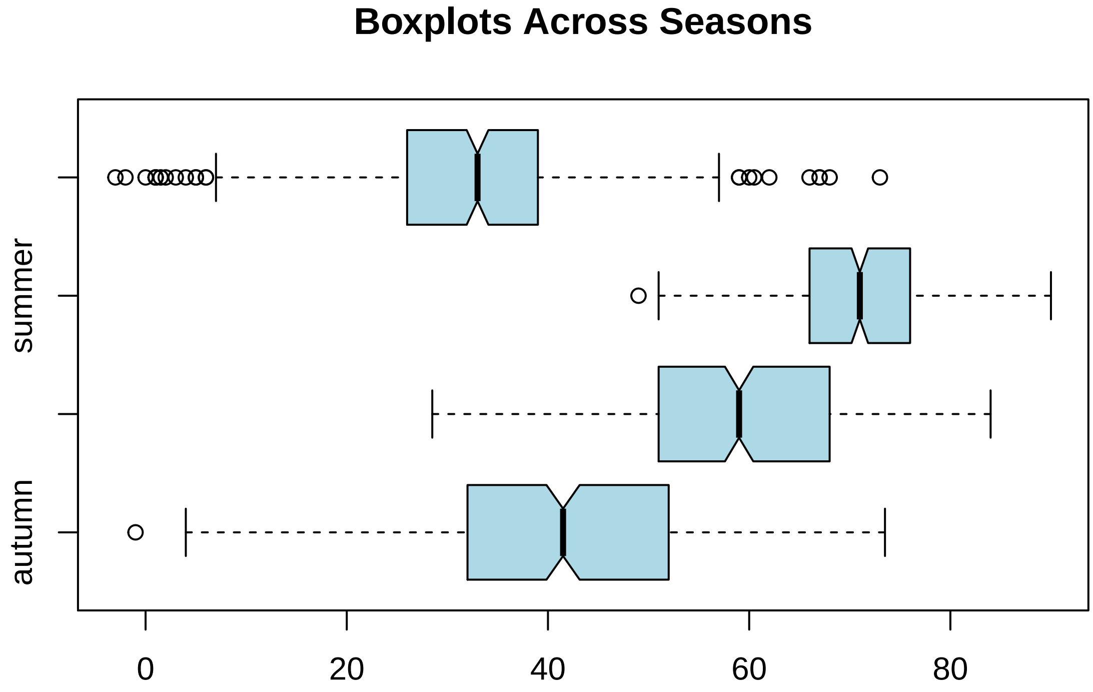
```
## $stats
##      [,1] [,2] [,3] [,4]
## [1,]  4.0 28.5   51    7
## [2,] 32.0 51.0   66   26
## [3,] 41.5 59.0   71   33
## [4,] 52.0 68.0   76   39
## [5,] 73.5 84.0   90   57
##
## $n
## [1] 368 364 368 361
##
## $conf
##          [,1]     [,2]     [,3]     [,4]
## [1,] 39.85274 57.59215 70.17637 31.91895
## [2,] 43.14726 60.40785 71.82363 34.08105
##
## $out
##  [1] -1.0 49.0  1.5  1.0  3.0 -3.0  0.0  2.0 60.5 68.0 67.0 62.0 73.0 59.0
## [15] -2.0  1.0  1.0  4.0  6.0  5.0 60.0  5.0  6.0 59.0 66.0 67.0
##
## $group
##  [1] 1 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##
## $names
## [1] "autumn" "spring" "summer" "winter"
```

```
text(1:4,box.out$stats[4,]+4,box.out$stats[3,])
```



So then we figure out how to rotate the plot after studying the help pages

```
title <- "Boxplots Across Seasons"
boxplot(temp~season,
        data=nm,
        notch=TRUE,
        col="lightblue",
        main=title,horizontal=TRUE)
```

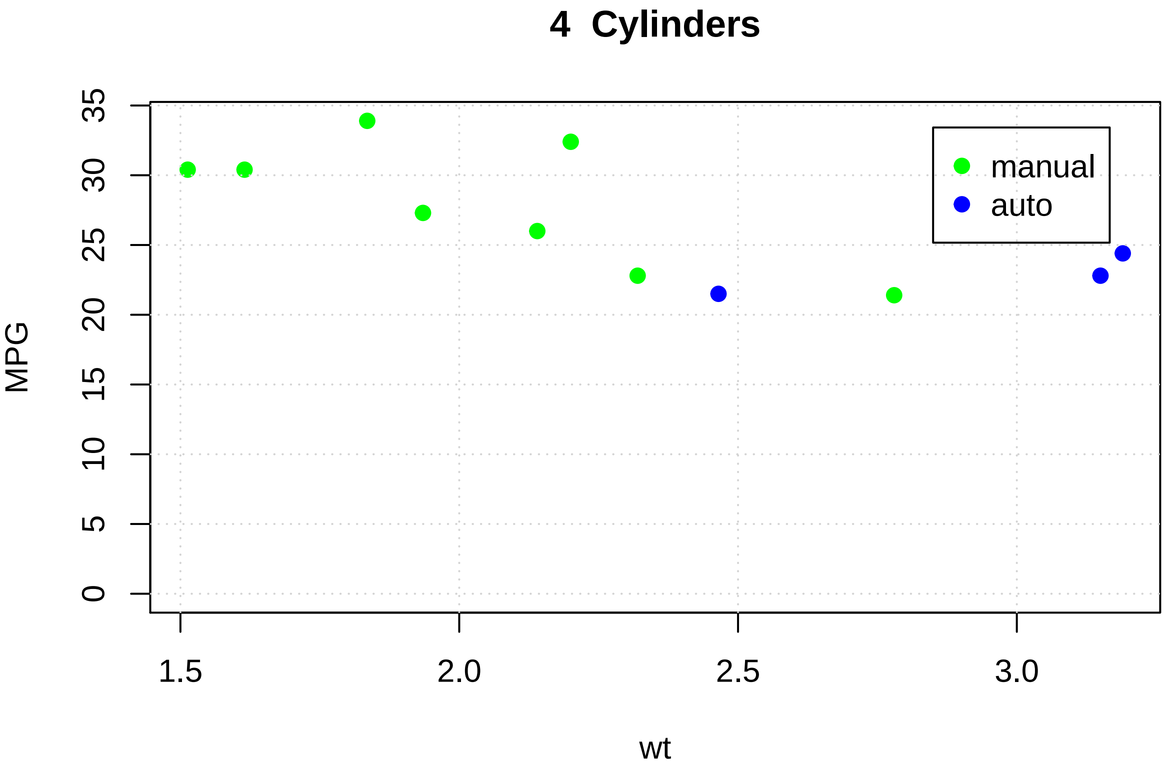


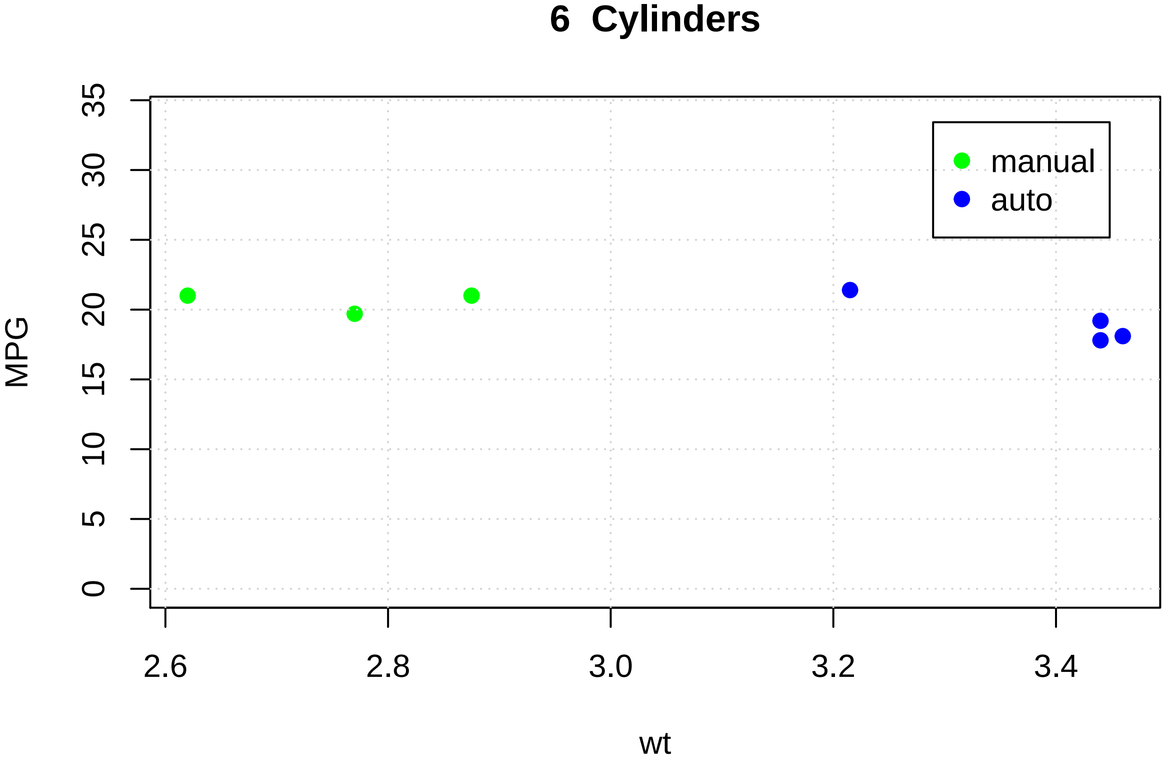
So lot's of times we want to do some plotting of a relationship, (say X/Y plot) such as MPG vs wt (using mtcars as an example) though we want to see this relationship plotted separately for each level of a factor such as cylinder group. Moreover, we want to have each the points from each group (4,6,8) separated by automatic transmission and manual transmission. We want to use a different color for each group. One way to do this is:

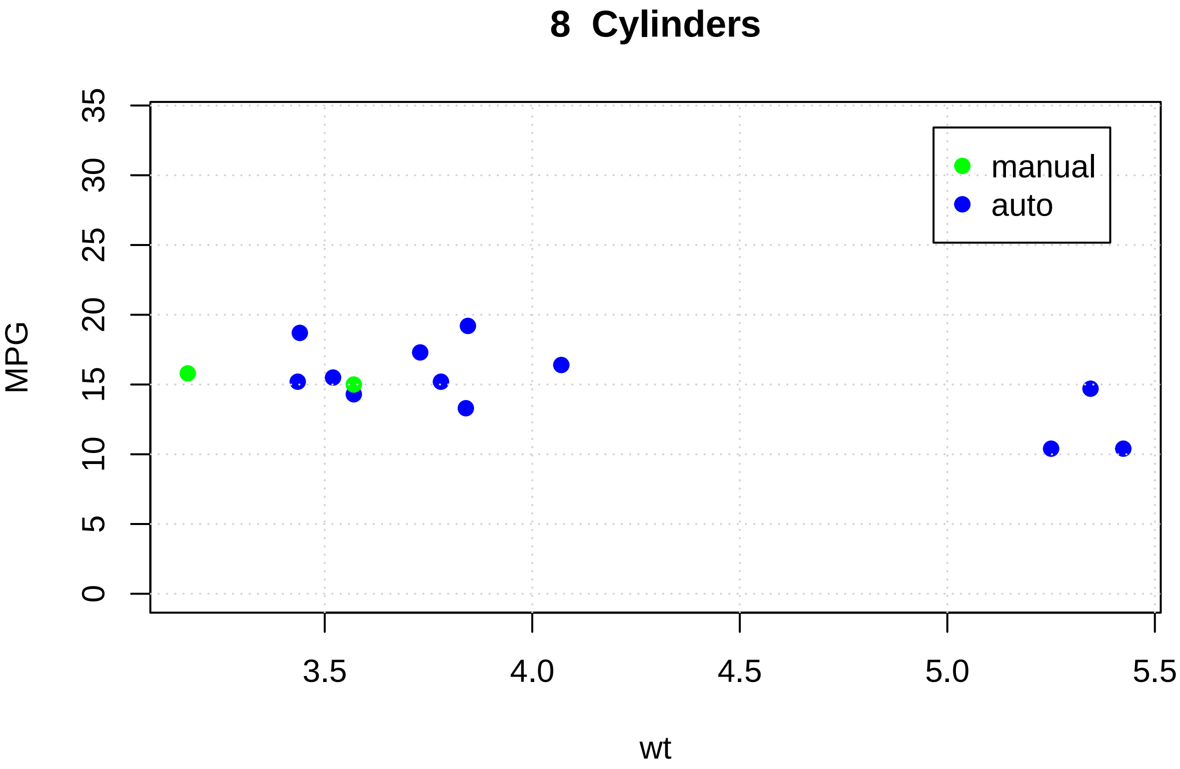
```
# Split the data frame based on cylinder values (4,6, or 8)
unique(mtcars$cyl)
```

```
## [1] 6 4 8
```

```
mysplits <- split(mtcars,mtcars$cyl)

maxmpg <- max(mtcars$mpg)   # Find the max MPG value

# Now for each element (dataframe) in mysplits initialize a plot and
# then use the points function to put up the points just for that data frame
# element

for (ii in 1:length(mysplits)) {
      tmpdf  <- mysplits[[ii]]

# Separate the automatic and manual transmission records

      auto <- tmpdf[tmpdf$am == 0,]
      man <- tmpdf[tmpdf$am == 1,]

# Setup a blank plot and use the points function to plot the points
# using different colors for auto vs manual

      plot(tmpdf$wt, tmpdf$mpg,type="n",
           main=paste(names(mysplits[ii])," Cylinders"),
           ylim=c(0,maxmpg), xlab="wt",ylab="MPG")
      points(auto$wt,auto$mpg,col="blue",pch=19)
      points(man$wt,man$mpg,col="green",pch=19)
      grid()
      legend("topright", inset=0.05, c("manual","auto"),
             pch = 19, col=c("green","blue"))
}
```


4 Cylinders
manual
auto
MPG
0
5
10
15
20
25
30
35
1.5
2.0
2.5
3.0
wt


6 Cylinders
manual
auto
MPG
0
5
10
15
20
25
30
35
2.6
2.8
3.0
3.2
3.4
wt



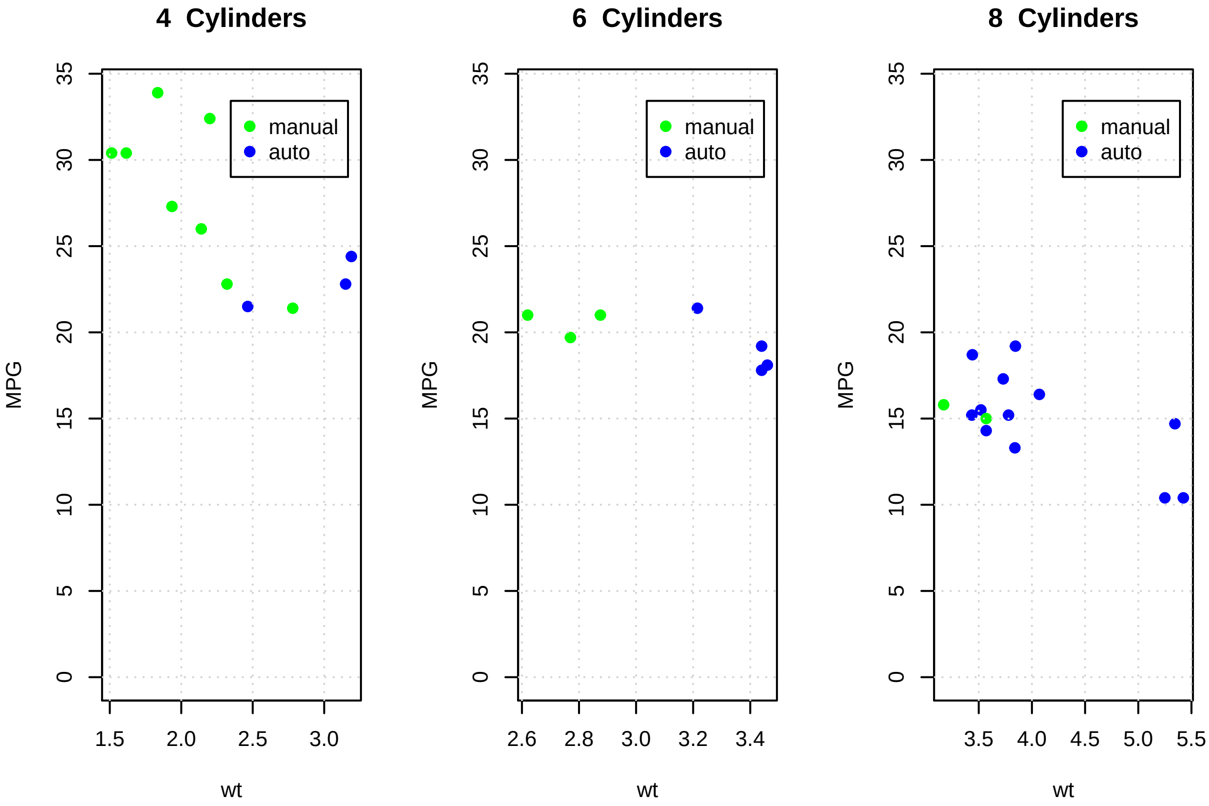
But something isn't quite right about this. If we run this from within R-Studio or R Commander we only see the last of the three plots because each plot overwrites the one before it. We want to see all three plots side by side to do some comparisons. We have to use the **par** function to do this. Here is the same example as above yet we just put a call to the par command up top.

```
par(mfrow=c(1,3))  # set the graphics device to be one row and three columns

# Split the data frame based on cylinder values (4,6, or 8)
unique(mtcars$cyl)
```

```
## [1] 6 4 8
```

```
mysplits <- split(mtcars,mtcars$cyl)

maxmpg <- max(mtcars$mpg)   # Find the max MPG value

# Now for each element (dataframe) in mysplits initialize a plot and
# then use the points function to put up the points just for that data frame
# element

for (ii in 1:length(mysplits)) {
      tmpdf  <- mysplits[[ii]]

# Separate the automatic and manual transmission records

      auto <- tmpdf[tmpdf$am == 0,]
      man <- tmpdf[tmpdf$am == 1,]

# Setup a blank plot and use the points function to plot the points
# using different colors for auto vs manual
```

```
    plot(tmpdf$wt, tmpdf$mpg,type="n",
         main=paste(names(mysplits[ii])," Cylinders"),
         ylim=c(0,maxmpg), xlab="wt",ylab="MPG")
    points(auto$wt,auto$mpg,col="blue",pch=19)
    points(man$wt,man$mpg,col="green",pch=19)
    grid()
    legend("topright", inset=0.05, c("manual","auto"),
           pch = 19, col=c("green","blue"))
}
```



```
par(mfrow=c(1,1))  # reset the graphics device to be one row and one column
```

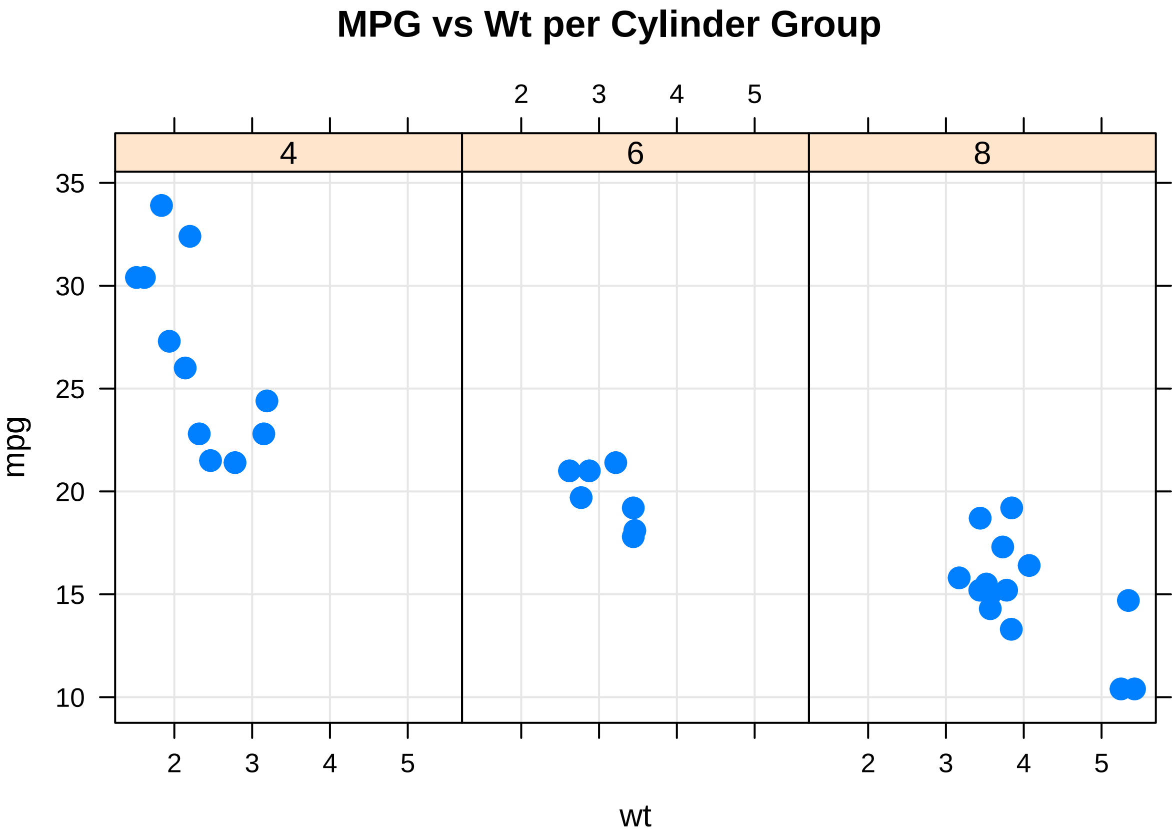
Base Graphics Summary

Base graphics are amazingly flexible but require more of a programming approach than using lattice or ggplot2. The complexity in the previous example is in part what motivated the development of lattice and ggplot2 graphics packages which attempt to simplify the idea of grouping and conditioning. Let's see some examples of this. Note that the lattice package is built in to your default installation of R. You just have to load it using a called to the library command.

Lattice Graphics to the Rescue ?

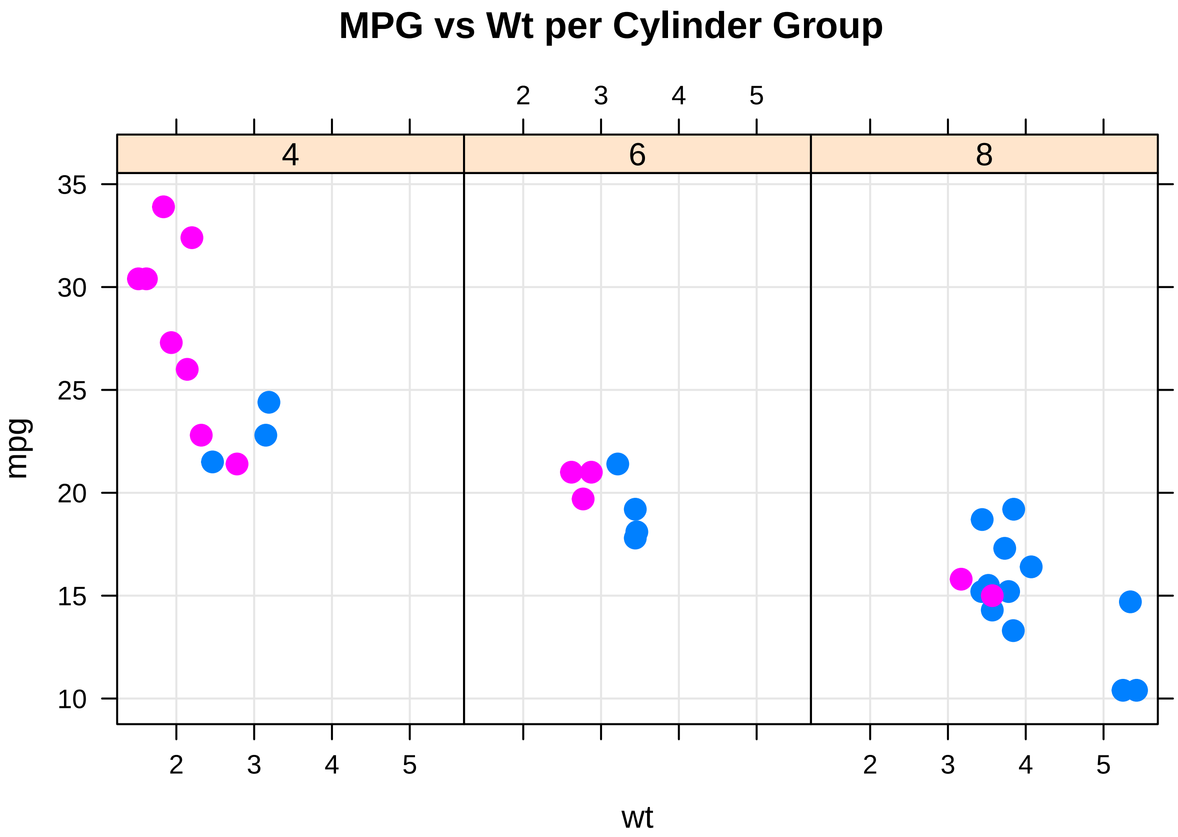
The lattice package does provide some relief from the tyranny of having to do all this programming. It implements **grouping** and **conditioning/paneling** in an easy-to-use way. Lets put up our xyplot and have it create a panel for each level of the cylinder factor.

```
library(lattice)

xyplot(mpg~wt|factor(cyl),data=mtcars, pch=19, cex=1.3,layout=c(3,1),
       type=c("p","g"),main="MPG vs Wt per Cylinder Group")
```



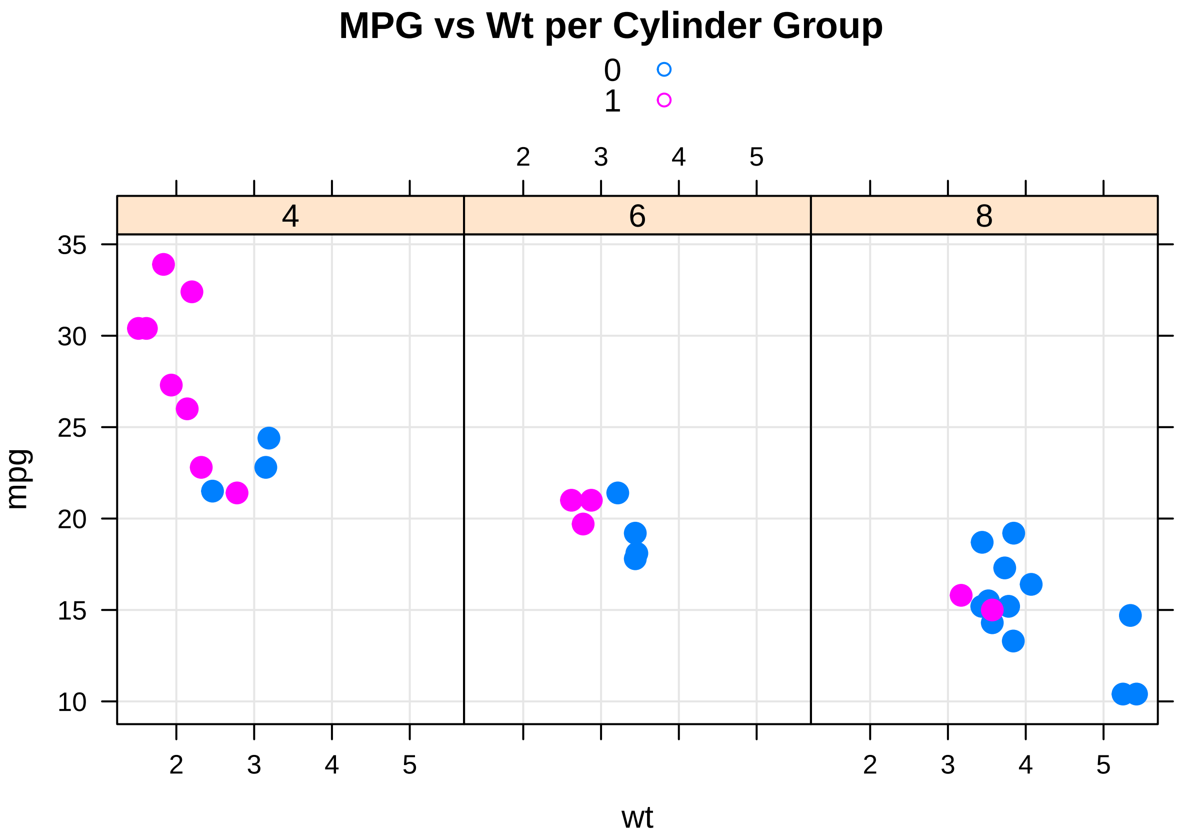
Now let's emulate what we did above with Base graphics. That is let's have the points in each panel be colored according to whether it's corresponding transmission type is automatic or manual. That was a lot of work in Base graphics. With lattice we use the **groups** argument

```
library(lattice)

xyplot(mpg~wt|factor(cyl), data=mtcars, groups=factor(am), pch=19,
       cex=1.3,layout=c(3,1),type=c("p","g"),main="MPG vs Wt per Cylinder Group")
```



But how do we know which color corresponds to which transmission type ? Add a legend

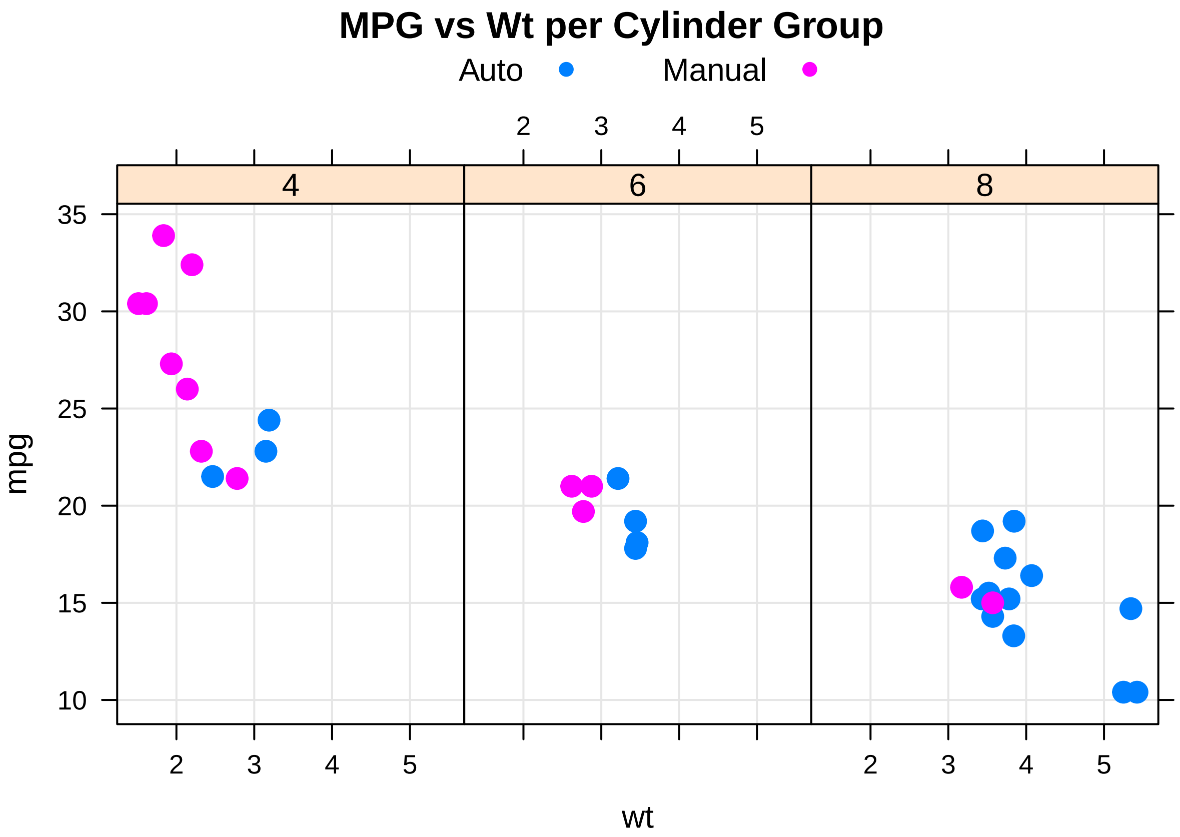
```
library(lattice)

xyplot(mpg~wt|factor(cyl), data=mtcars, groups=factor(am), pch=19,
       cex=1.3,layout=c(3,1),type=c("p","g"),main="MPG vs Wt per Cylinder Group",
       auto.key=TRUE)
```



So this is great. Lattice takes care of the panelling and coloring for us so if we are in exploration mode then don't worry about making the colors nicer or the legends perfect. We are just trying to learn about the data. However, if you are in publication mode you need to fix some things.

For example the legend, while understandable, doesn't have the same plot character as do the points. Also, we have 0 and 1 instead is automatic and manual respectively. We might also want to have the legend listed horizontally instead of vertically.

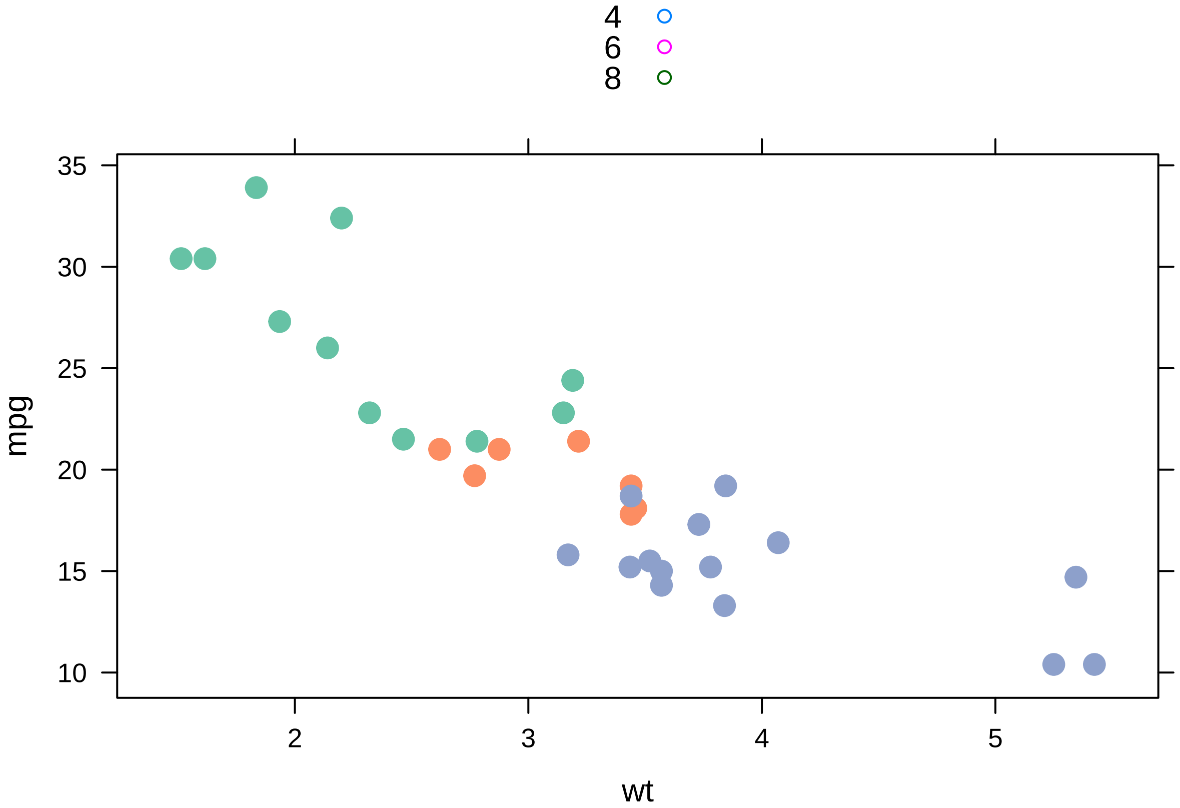
```
library(lattice)

mtcars$am <- factor(mtcars$am,labels=c("Auto","Manual"))

xyplot(mpg~wt|factor(cyl), data=mtcars, groups=am, pch=19,
       cex=1.3,layout=c(3,1),type=c("p","g"),main="MPG vs Wt per Cylinder Group",
       auto.key=list(columns=2),par.settings=list(superpose.symbol=list(pch=19)))
```



Well okay this is nice though things are starting to get more involved. If we were doing this in Base graphics we would be using separate function calls to do things. Within lattice we try to do everything withing the call to the plot function itself. It's a different paradigm though sometimes the amount of effort put forth in understanding what plot options to use can be confusing. At a minimum you find yourself doing a lot of Googling and/or searching through the help pages for a given command.

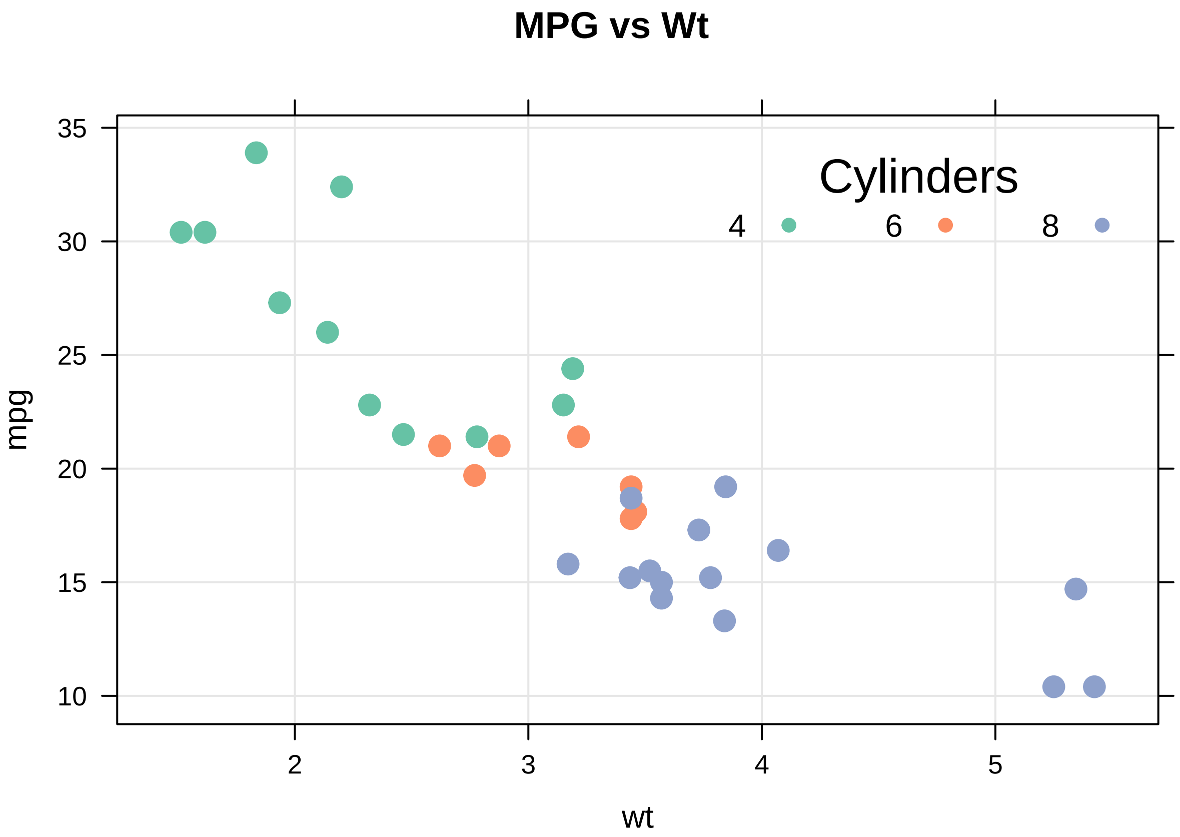
In this next example let's pick some different colors for the plot. Here we won't do the conditioning but we will group by cylinder to see the points in different colors corresponding to each level of cylinder (4,6, or 8)

```
library(lattice)

library(RColorBrewer)   # Not necessary but makes for nice colors

mycols <- brewer.pal(3,"Set2")

xyplot(mpg~wt,data=mtcars,groups=cyl,pch=19,cex=1.3,col=mycols,auto.key=TRUE)
```



So now we have to fix the legend again to get the colors to match. Let's also put the legend into the plot area itself.

```
xyplot(mpg~wt,data=mtcars,groups=cyl,pch=19,cex=1.3,col=mycols,
       auto.key=list(columns=3,corner=c(0.95,0.95),title="Cylinders"),
       type=c("p","g"),
       par.settings=list(superpose.symbol=list(col=mycols,fill=mycols,pch=19)),
       main="MPG vs Wt")
```



Lattice, like Base Graphics, work on the idea that you locate the function of interest to do the plotting. Consequently there are multiple functions corresponding to a given plot top. Thankfully, lattice did NOT name their functions to be the same as the Base Graphics functions else there would have been a real problem. Here are some of the functions as well as their Base graphics equivalent

Lattice Function	Description	Base Graphics Equivalent
barchart()	Barcharts	barplot()
bwplot()	Boxplots	boxplot()
doptlot()	Dotplots	dotchart()
histogram()	Histograms	hist()
stripplot()	Strip Plots	stripchart()
xyplot()	Scatter plot	plot()
qq	Quantile-quantile plots	qqplot()
qqmath()	Quantile-quantile plots Data set vs Data set	qqplot()
splom()	Scatterplot matrices	pairs()
levelplot()	Level Plots	image()
contourplot()	Contour Plots	contour()

Formula interface

Lattice also uses the concept of a formula when specifying the variables to be plotted. This is useful since R has a number of functions that employ a formual interface:

```
mylm <- lm(mpg ~ wt + am, data=mtcars)    # Linear Regression

xtabs(~am + cyl,mtcars)  # Cross tabulation
```

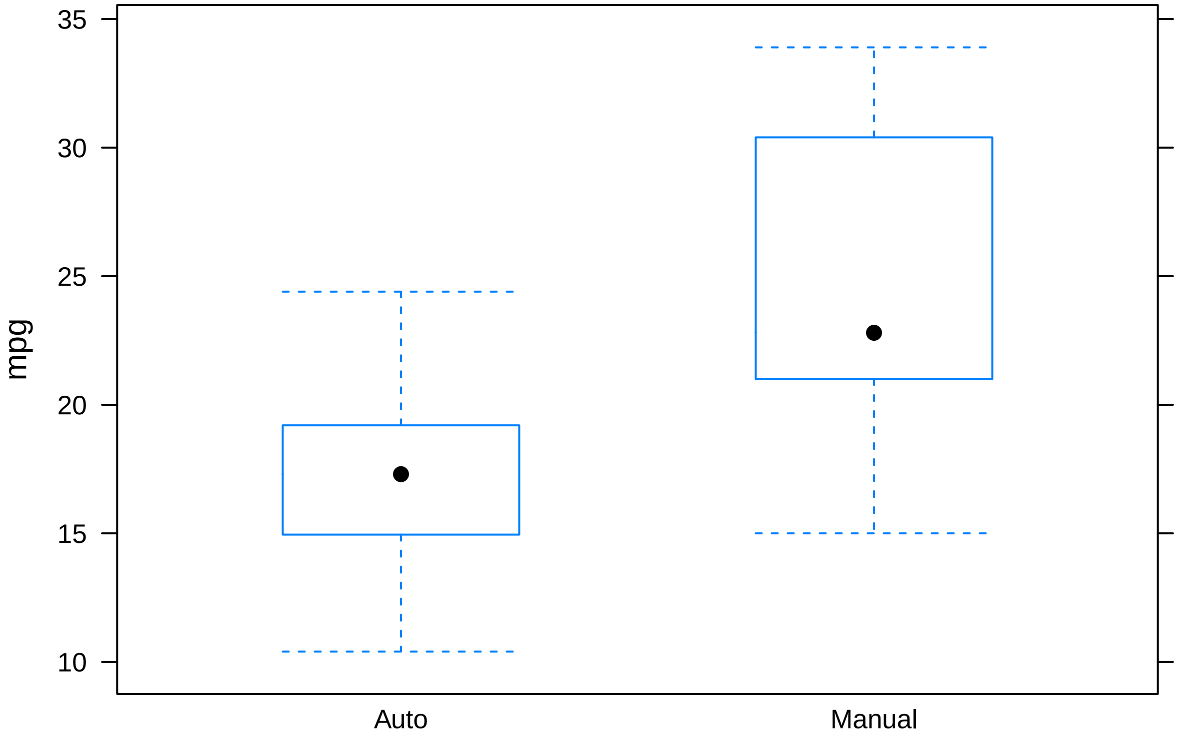
```
##          cyl
## am        4  6  8
##   Auto    3  4 12
##   Manual  8  3  2
```

```
aggregate(mpg ~ am + cyl, data=mtcars, mean)  # Aggregate
```

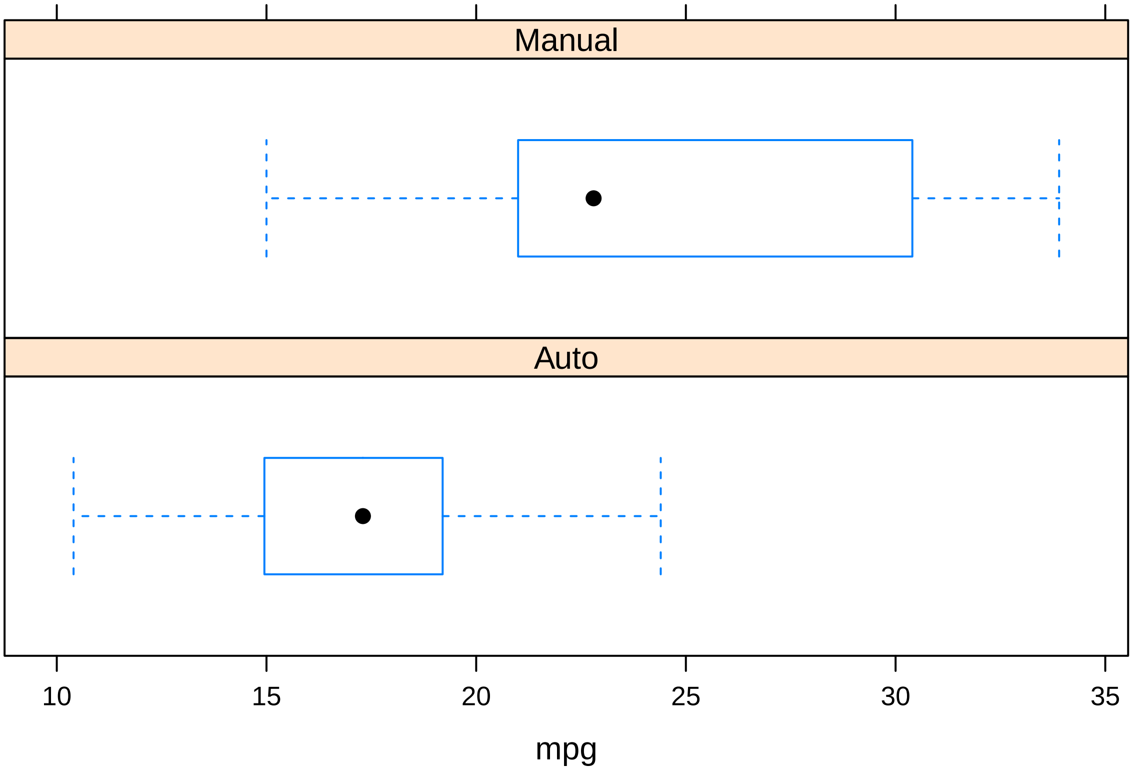
```
##       am cyl      mpg
## 1   Auto   4 22.90000
## 2 Manual   4 28.07500
## 3   Auto   6 19.12500
## 4 Manual   6 20.56667
## 5   Auto   8 15.05000
## 6 Manual   8 15.40000
```

Here is a summary of some of the formulas you might encounter when plotting with lattice graphics. In the chart below a lower case variable (e.g. “x”) implies a numeric or continuous variable. An uppercase variable (e.g. “A”) implies a factor/categorical variable.

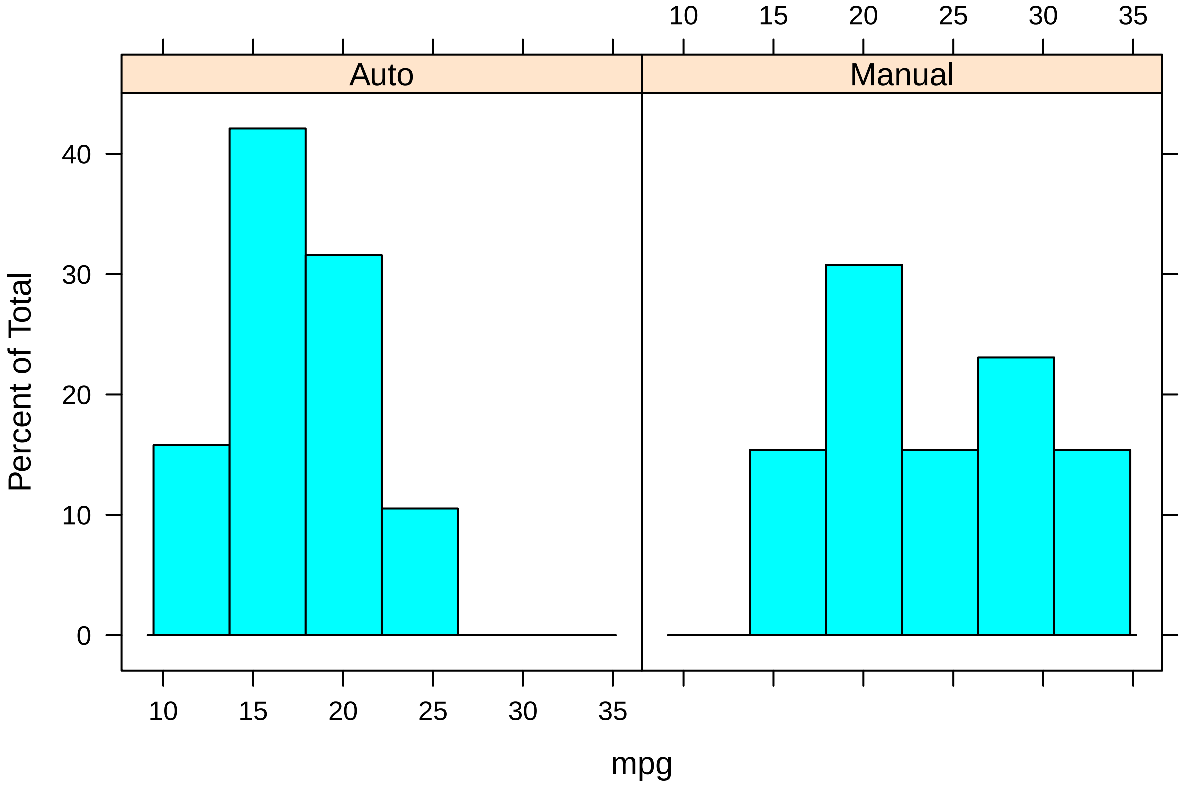
```
mtcars$am <- factor(mtcars$am,label=c("Auto","Manual"))

# Example of x~A

bwplot(mpg ~ am, data=mtcars)
```



```
# Example of ~x|A

bwplot(~mpg | am, data=mtcars,layout=c(1,2))
```

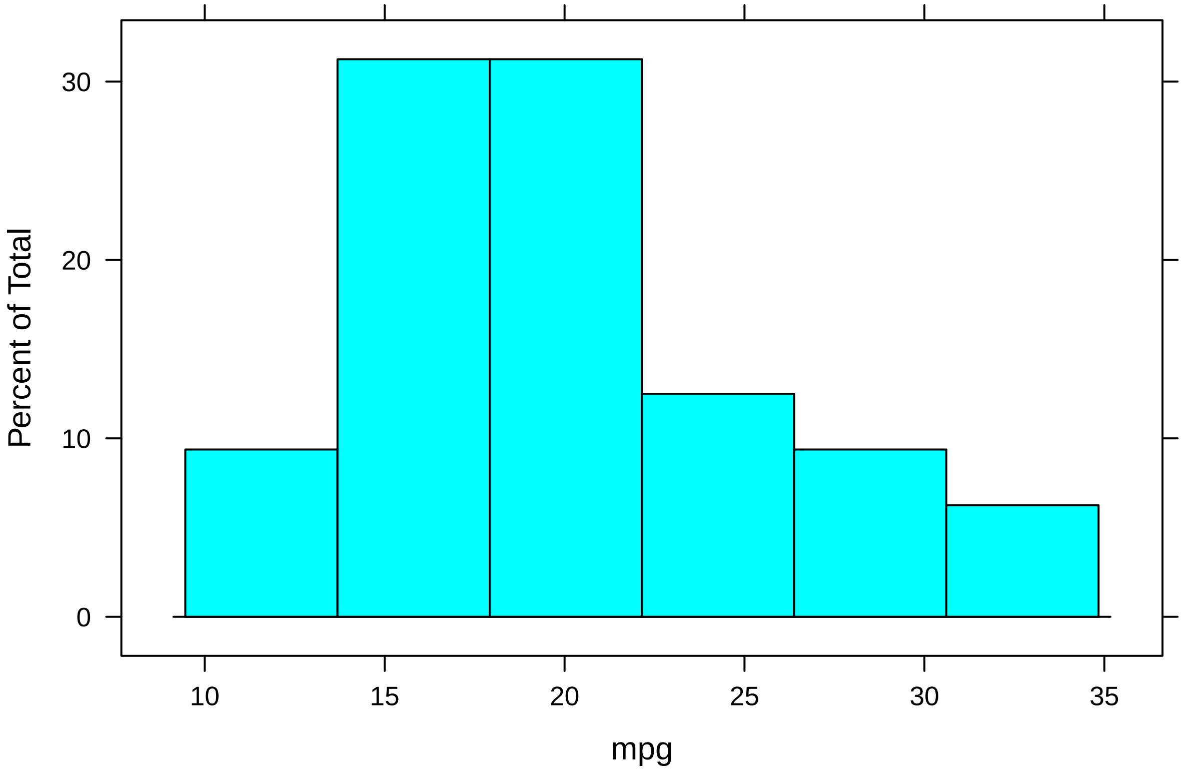


```
histogram(~mpg|am, data=mtcars)
```

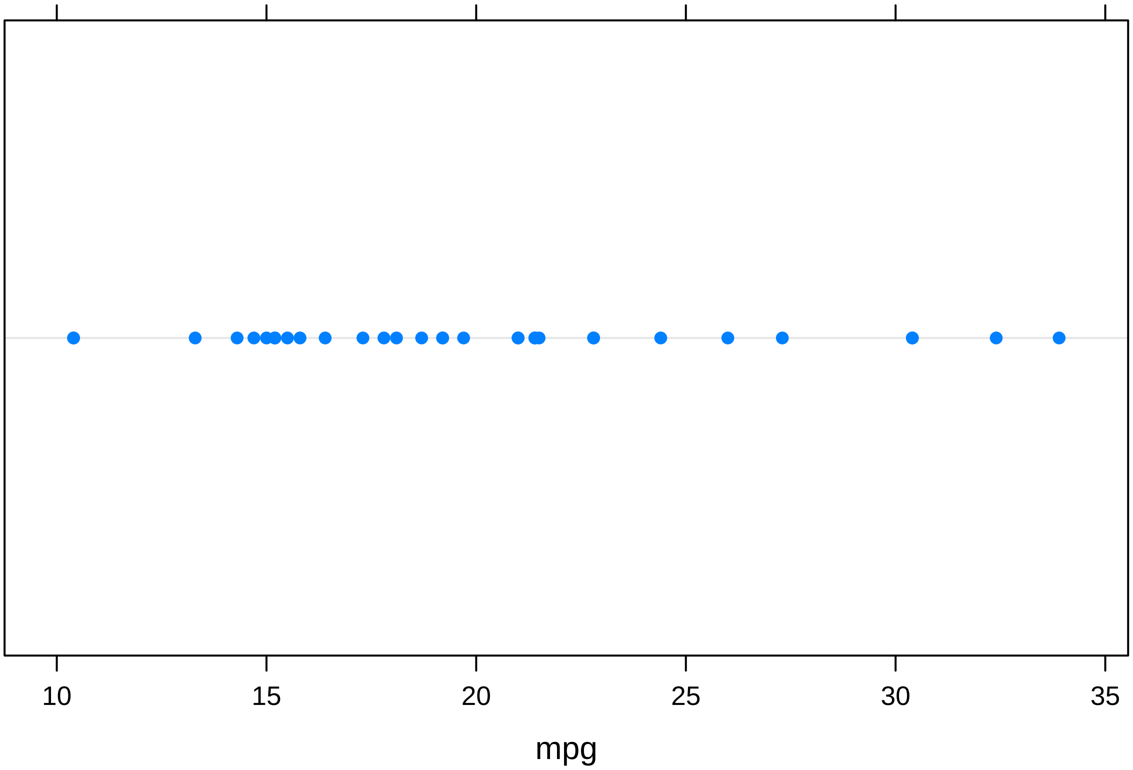


```
# Example of ~x
```

```
histogram(~mpg, data=mtcars)
```



```
dotplot(~mpg, data=mtcars)
```



graph_type	description	example formulas
barchart	barchart	x ~ A or A ~ x
bwplot	boxplot	x~A or A~x
dotplot	dotplot	~x
histogram	histogram	~x
xyplot	scatterplot	y~x
stripplot	strip plots	A~x or x~A

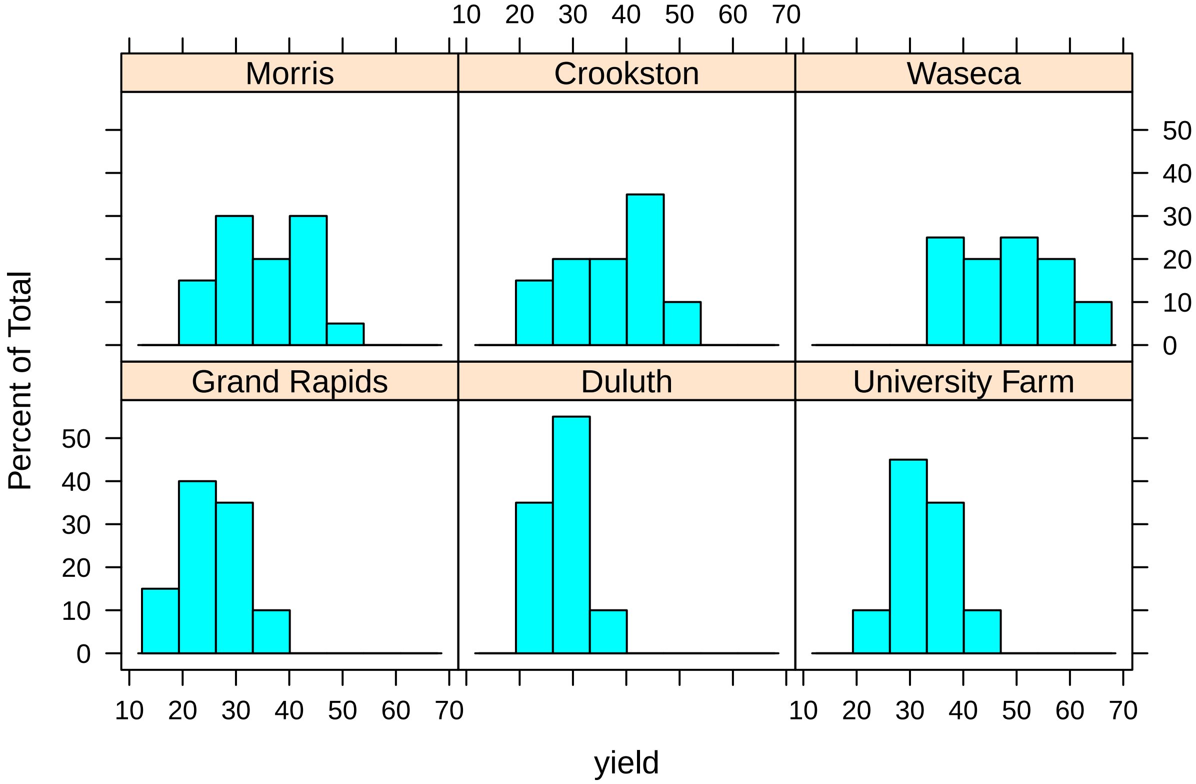
```
# Let's look at the distribution of barley yields as conditioned by location

levels(barley$site)
```

```
## [1] "Grand Rapids"    "Duluth"          "University Farm" "Morris"
## [5] "Crookston"       "Waseca"
```

```
str(barley)
```

```
## 'data.frame':    120 obs. of  4 variables:
##  $ yield  : num  27 48.9 27.4 39.9 33 ...
##  $ variety: Factor w/ 10 levels "Svansota","No. 462",..: 3 3 3 3 3 3 7 7 7 7 ...
##  $ year   : Factor w/ 2 levels "1932","1931": 2 2 2 2 2 2 2 2 2 2 ...
##  $ site   : Factor w/ 6 levels "Grand Rapids",..: 3 6 4 5 1 2 3 6 4 5 ...
```

```
histogram(~yield|site,data=barley)
```



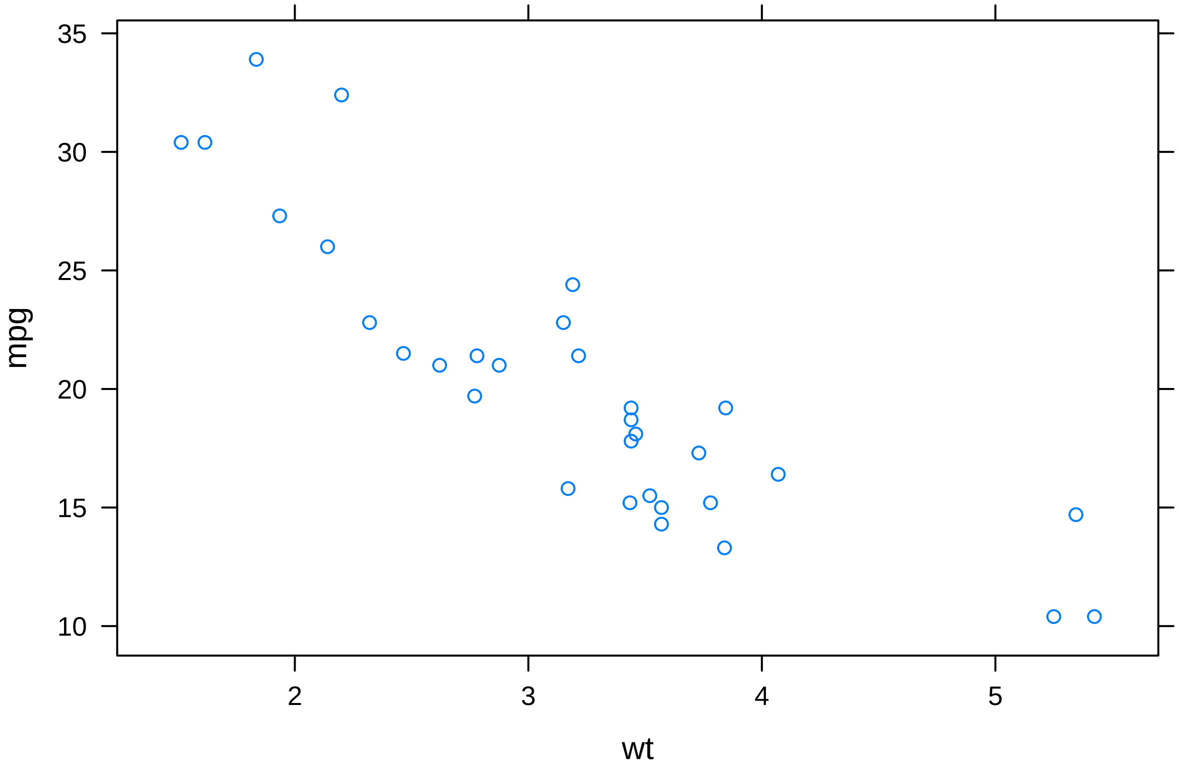
Compliance with dplyr ?

Does lattice graphics work with the chaining operators in dplyr ? In fact it does. We just have to remember that the **data** argument in lattice programs needs to be filled in with a period character which will denote the "incoming" data from the chaining or pipe character %>%

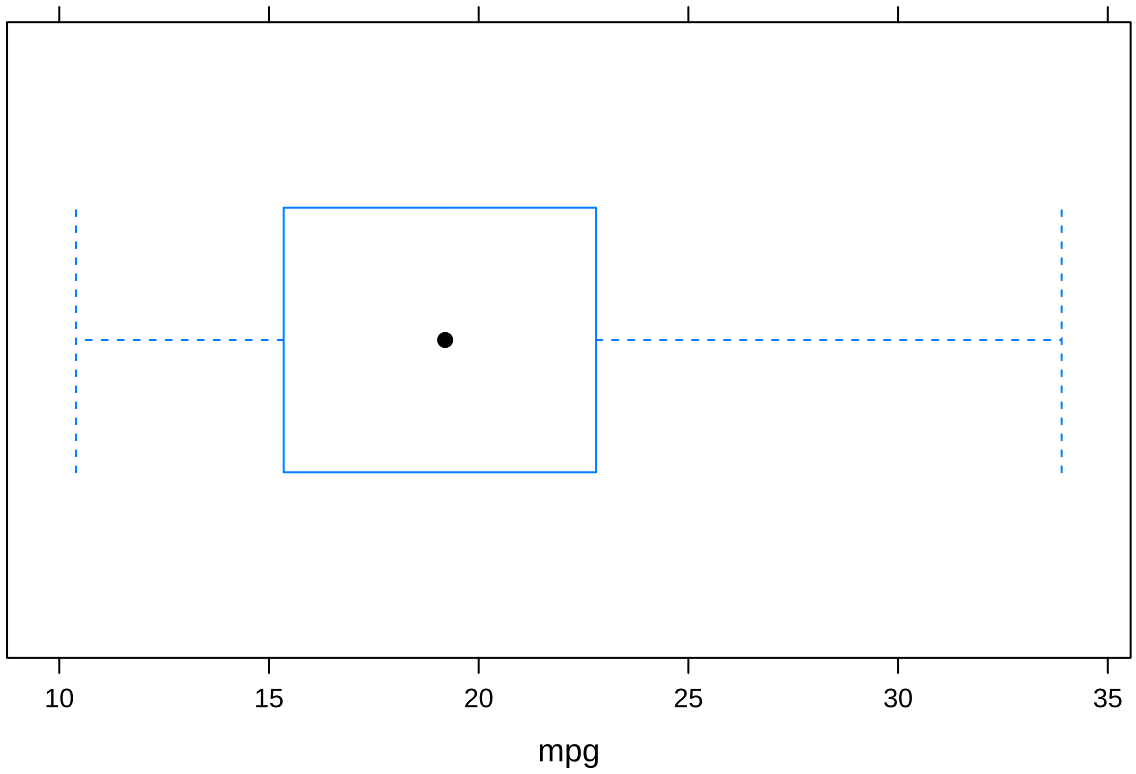
```
library(dplyr)
```

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

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

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

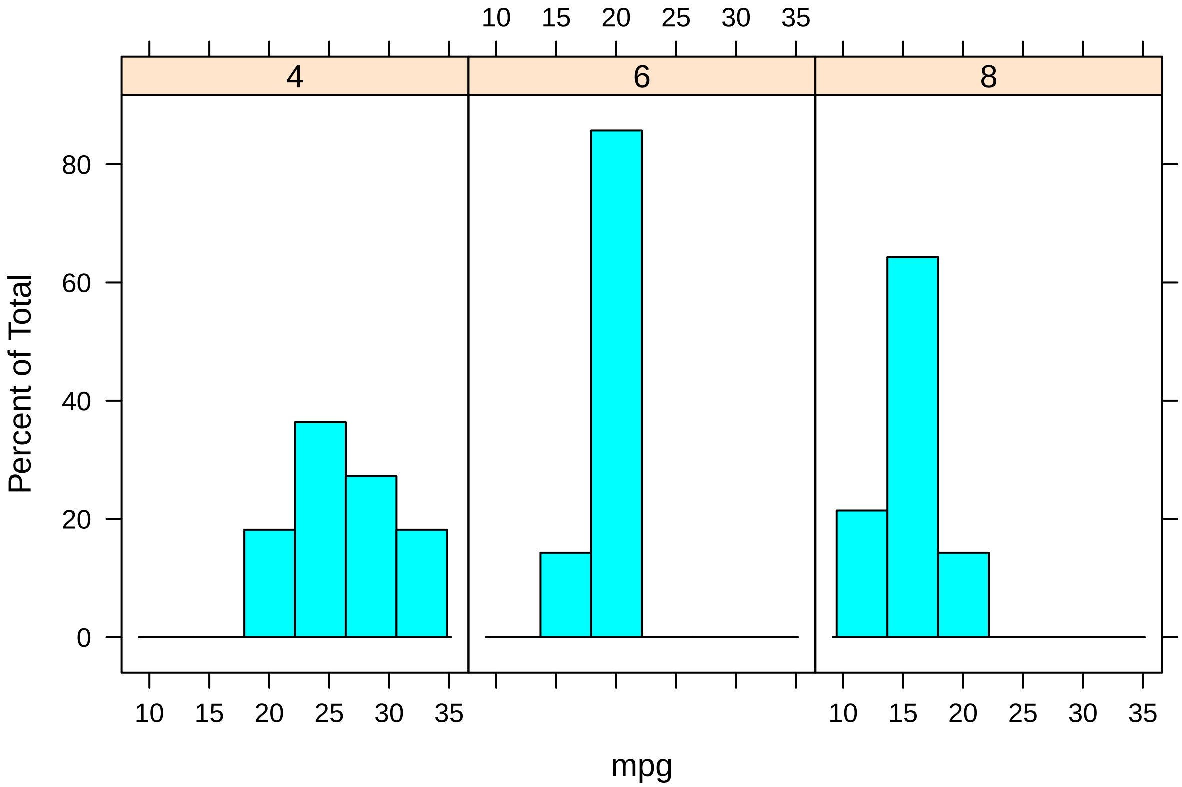
```
mtcars %>% xyplot(mpg~wt,data=.)
```



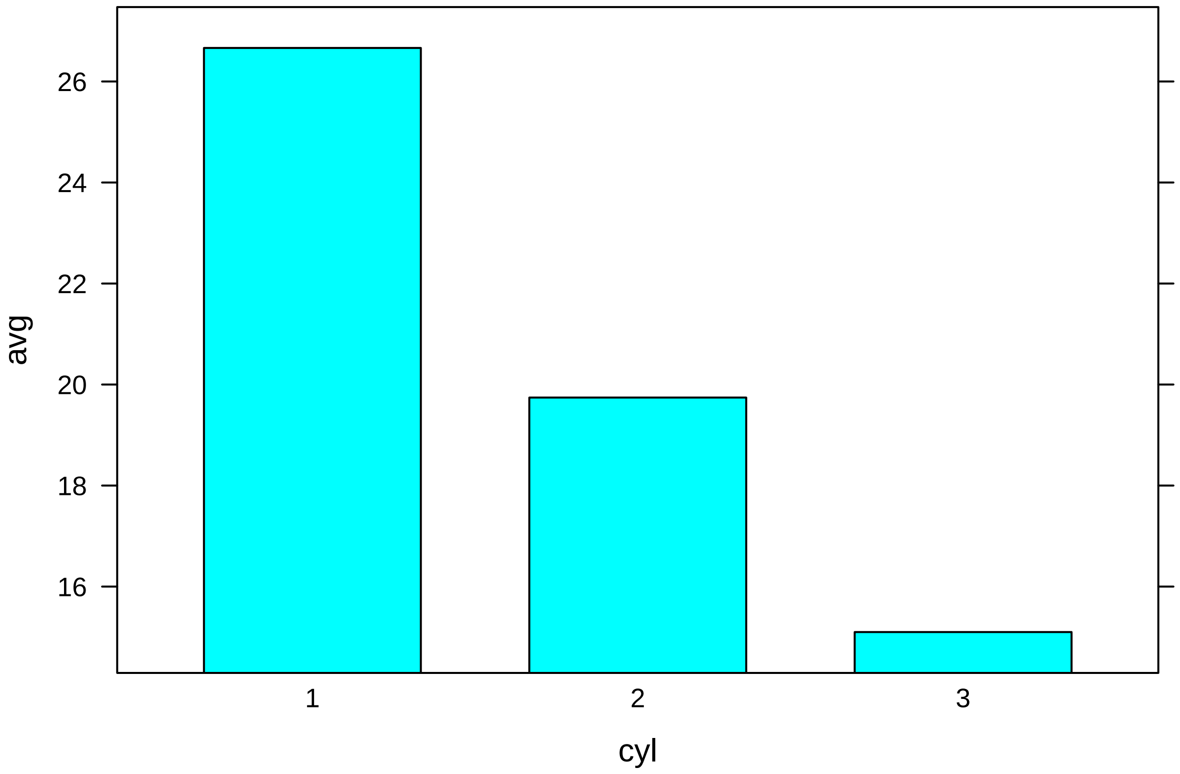
```
mtcars %>% bwplot(~mpg,data=.)
```



```
mtcars %>% histogram(~mpg|factor(cyl),data=.)
```



```
mtcars %>% group_by(cyl) %>% summarize(avg=mean(mpg)) %>% barchart(avg~cyl,data=.,horiz=F)
```



ggplot2

Now that we've laid the ground work for R graphics it's time to investigate ggplot2 which represents an innovative approach to creating plots.

Why ggplot2 ?

See the Grammar of Graphics Book by Leland Wilkinson. It outlines a theory that shows us how a very large variety of statistical graphics can be created. Let's figure out the building blocks that are common to a large number of graphics. This allows us to describe plots in terms of concepts that can then be easily implemented to generate as many plots as are necessary to arrive at a conclusion.

A **plot** is made up of one or more layers (e.g. points, reference lines). A **layer** consists of the following:

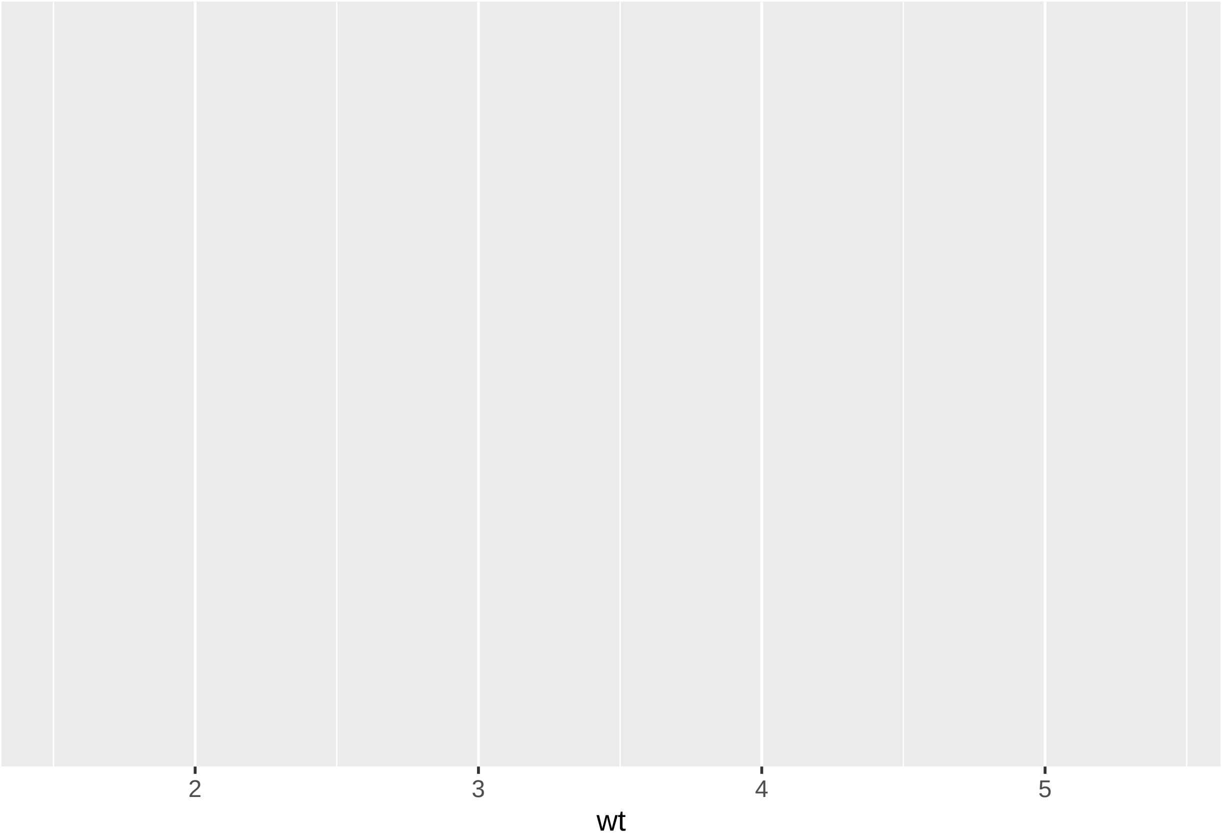
1) **data** (of course),
2) a set of **aesthetic mappings** that describes how data are mapped to size, color,etc
3) a **geometric** object - point, lines, shapes
4) a **statistical** transformation such as binning, quantiles, or smoothing
5) a **scale** that describes what elements an aesthetic mapping uses (smoker=red, nonsmoker=blue)
6) a coordinate system

A workflow using ggplot2 is to build up a plot in layers. We first plot the data and add things piece by piece as it occurs to us. We tell ggplot about the data and the basic aesthetics (what the axes are), and then summarize it (smoother, regression), and then some extra annotation or metadata

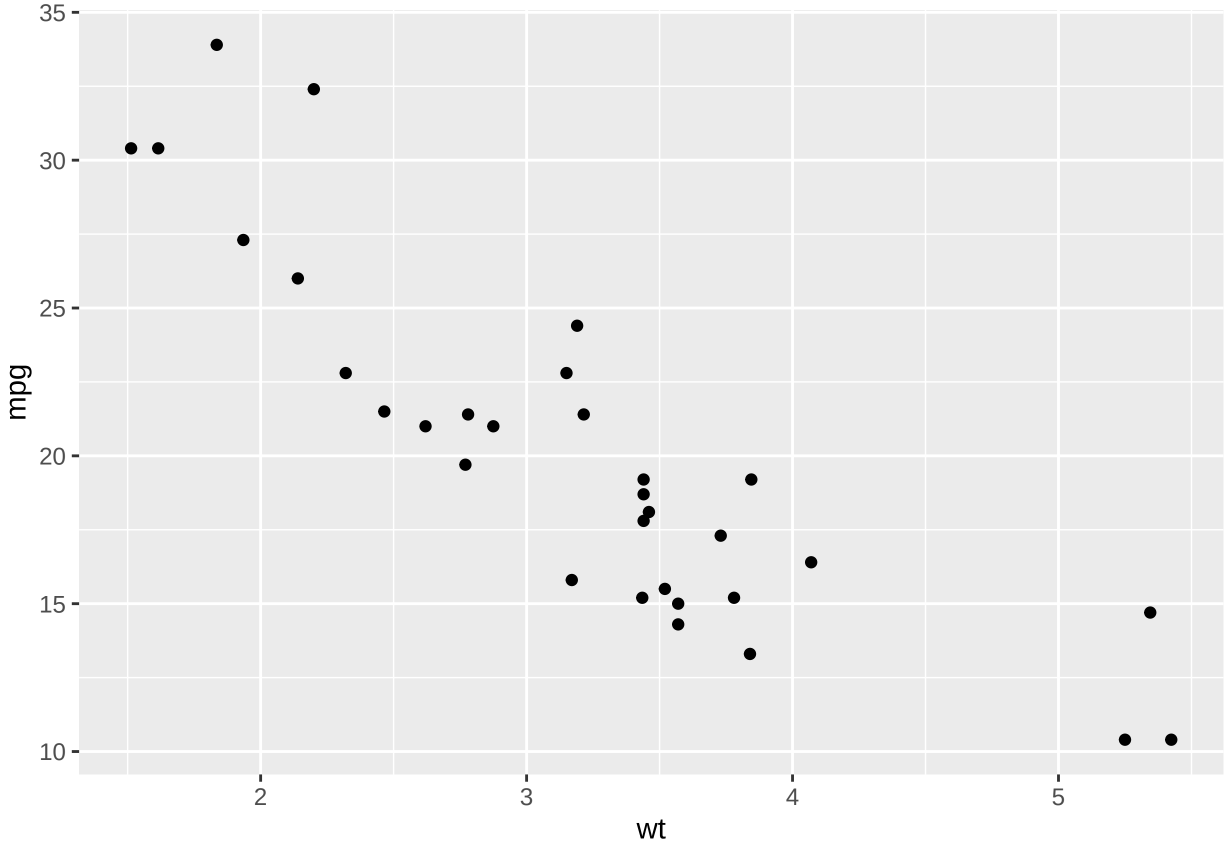
Before we start let's first reproduce some of the graphs we did earlier with lattice graphics just to make some things clear.

```r
library(ggplot2)
```

```
# We establish a relationship between the data and some basic aesthetics
#


mtcars %>% ggplot(aes(x=wt))    # Noting shows up
```

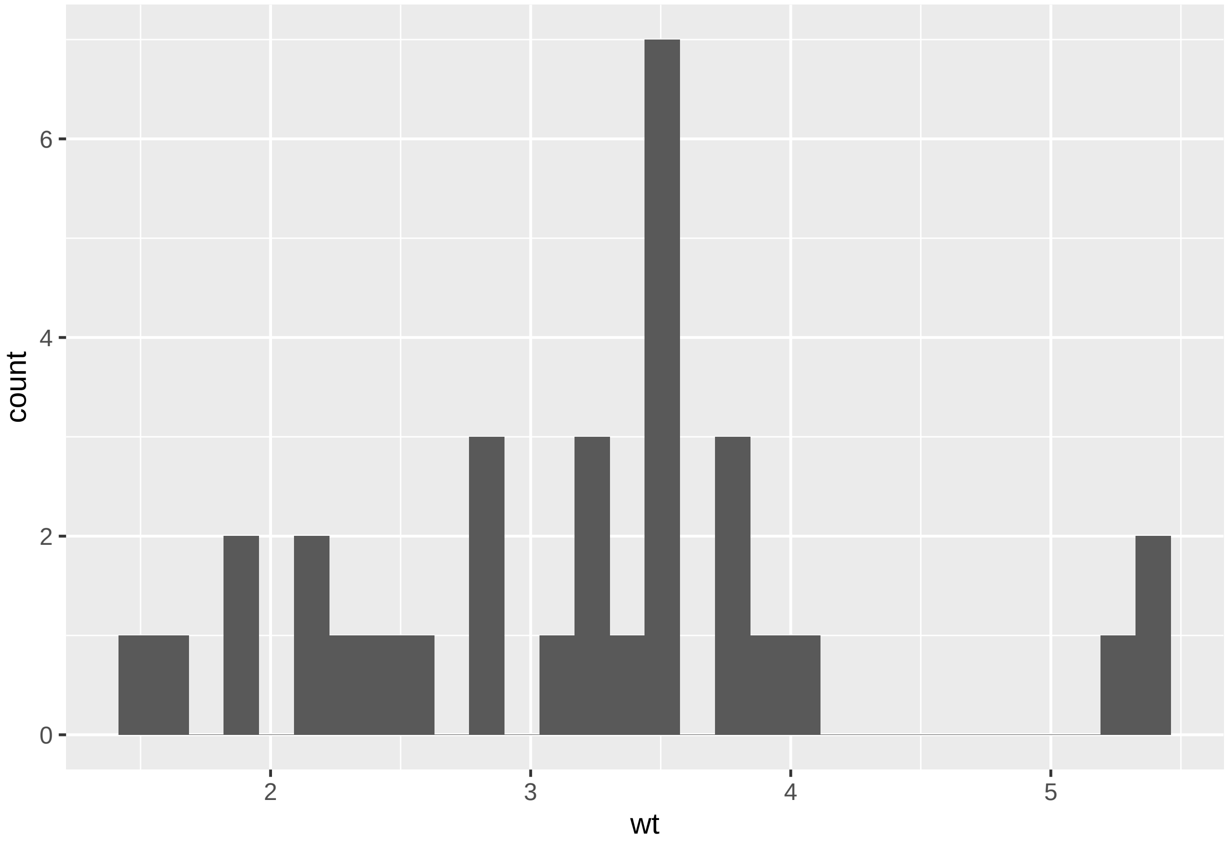


```
# We don't have to commit to a specific geometry or shape

mtcars %>% ggplot(aes(x=wt)) + geom_point(aes(y=mpg))
```



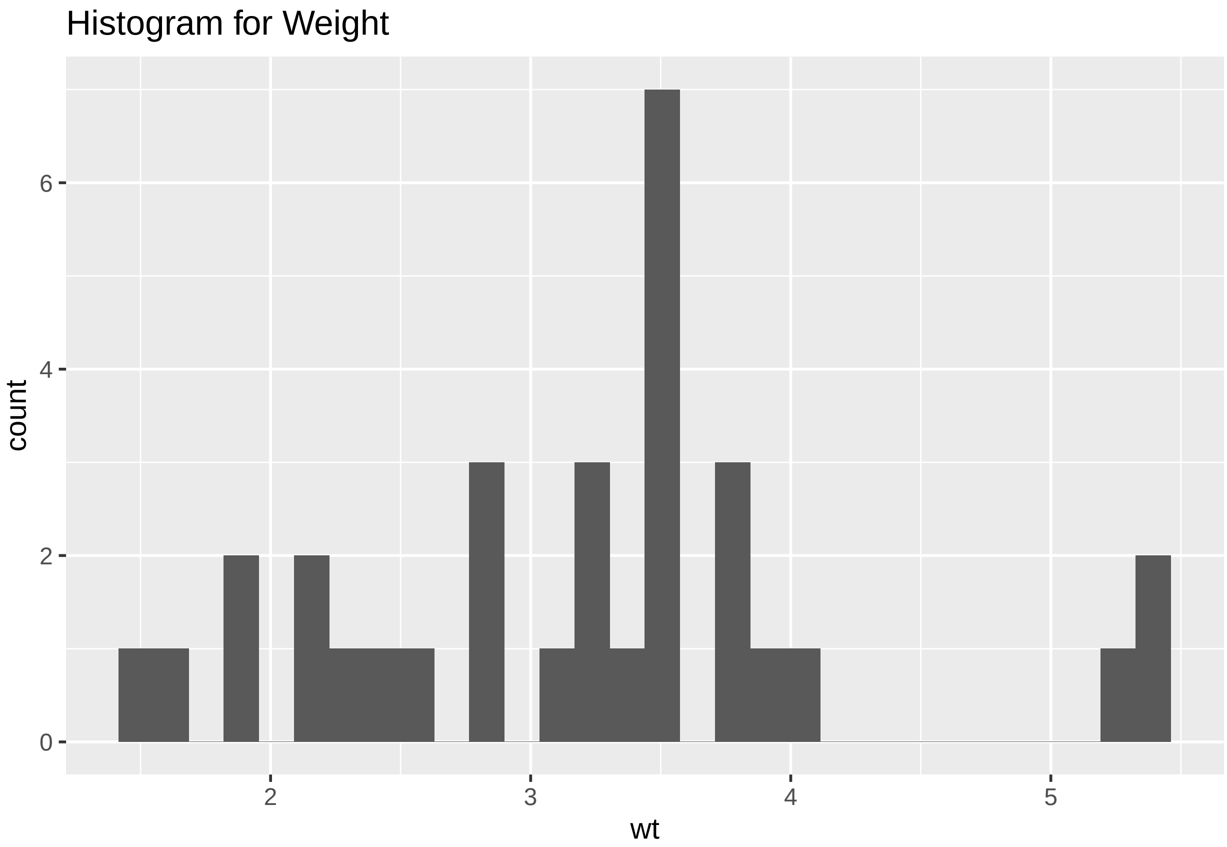
```
mtcars %>% ggplot(aes(x=wt)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

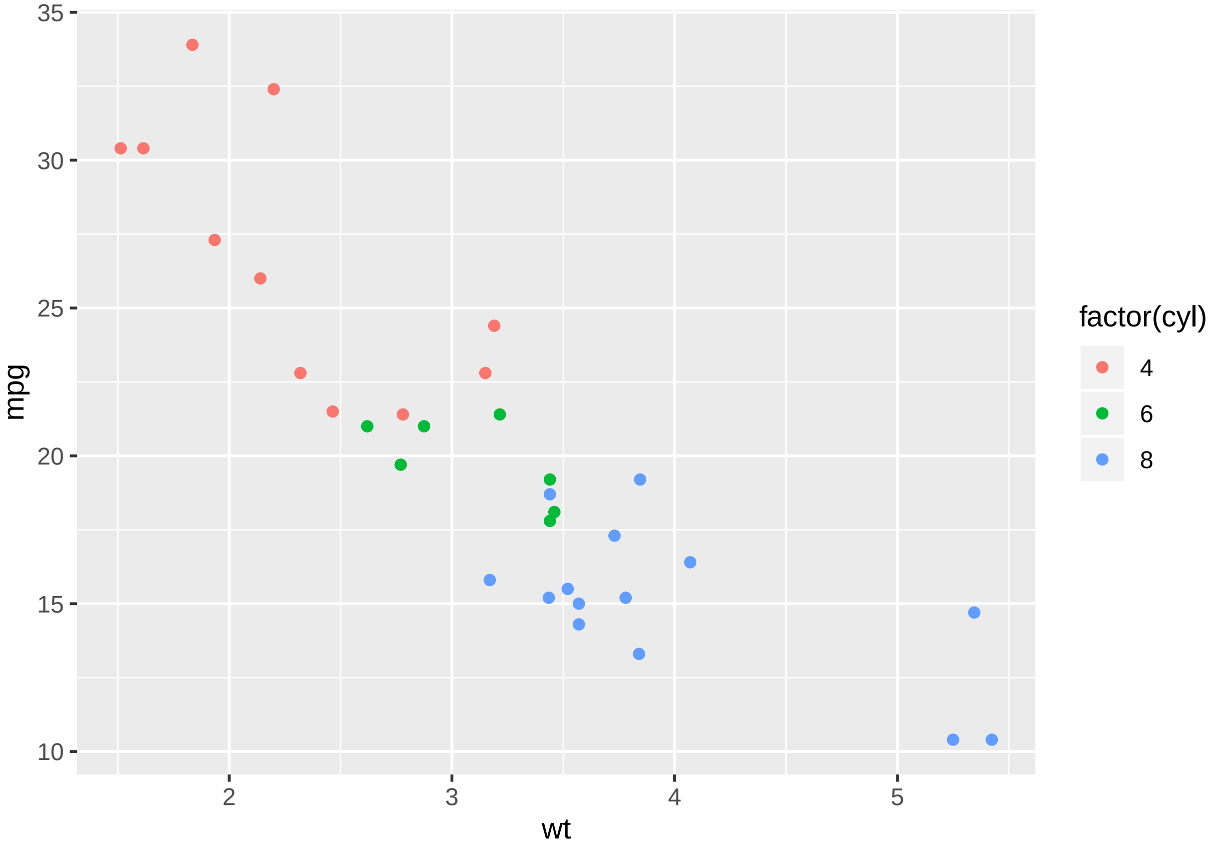


```
mtcars %>% ggplot(aes(x=wt)) + geom_histogram() + ggtitle("Histogram for Weight")
```

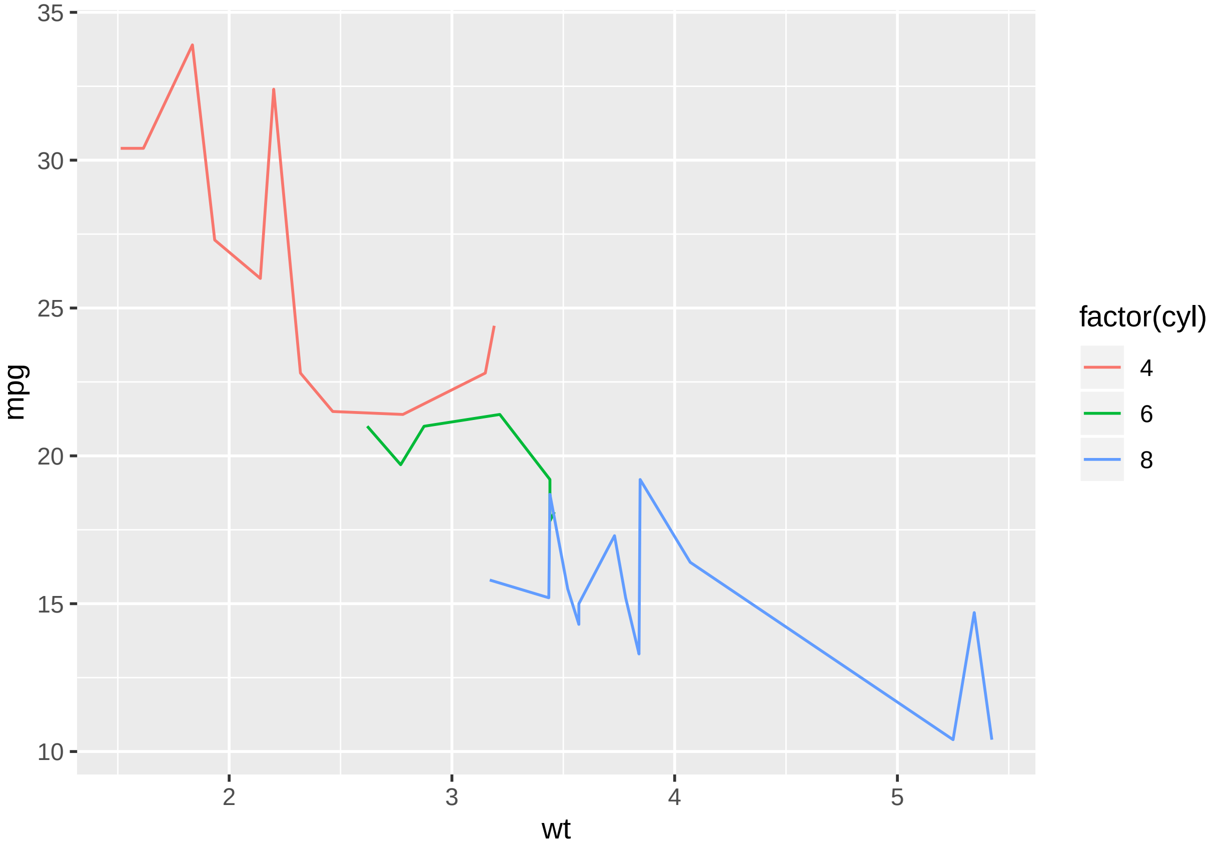
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



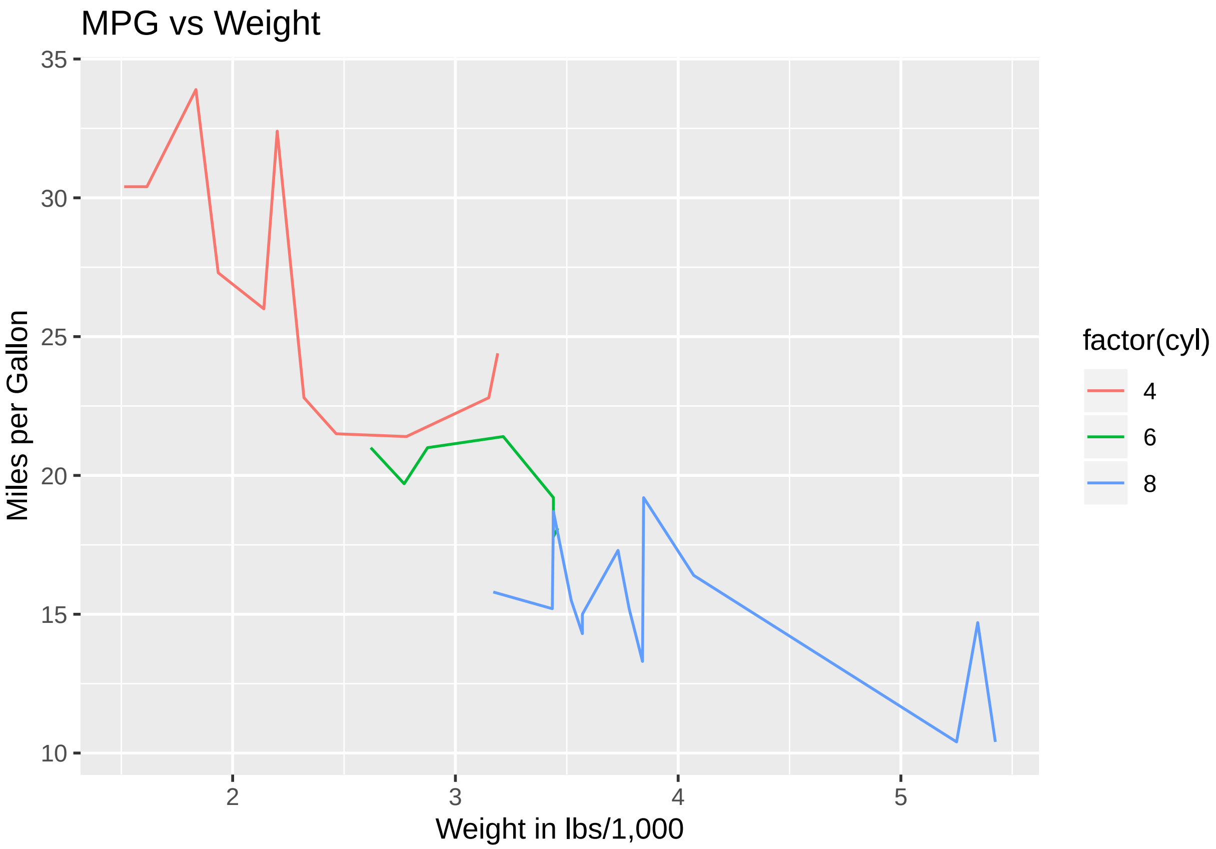
```
mtcars %>% ggplot(aes(x=wt)) + geom_point(aes(y=mpg,color=factor(cyl)))
```



```
mtcars %>% ggplot(aes(x=wt)) + geom_line(aes(y=mpg,color=factor(cyl)))
```

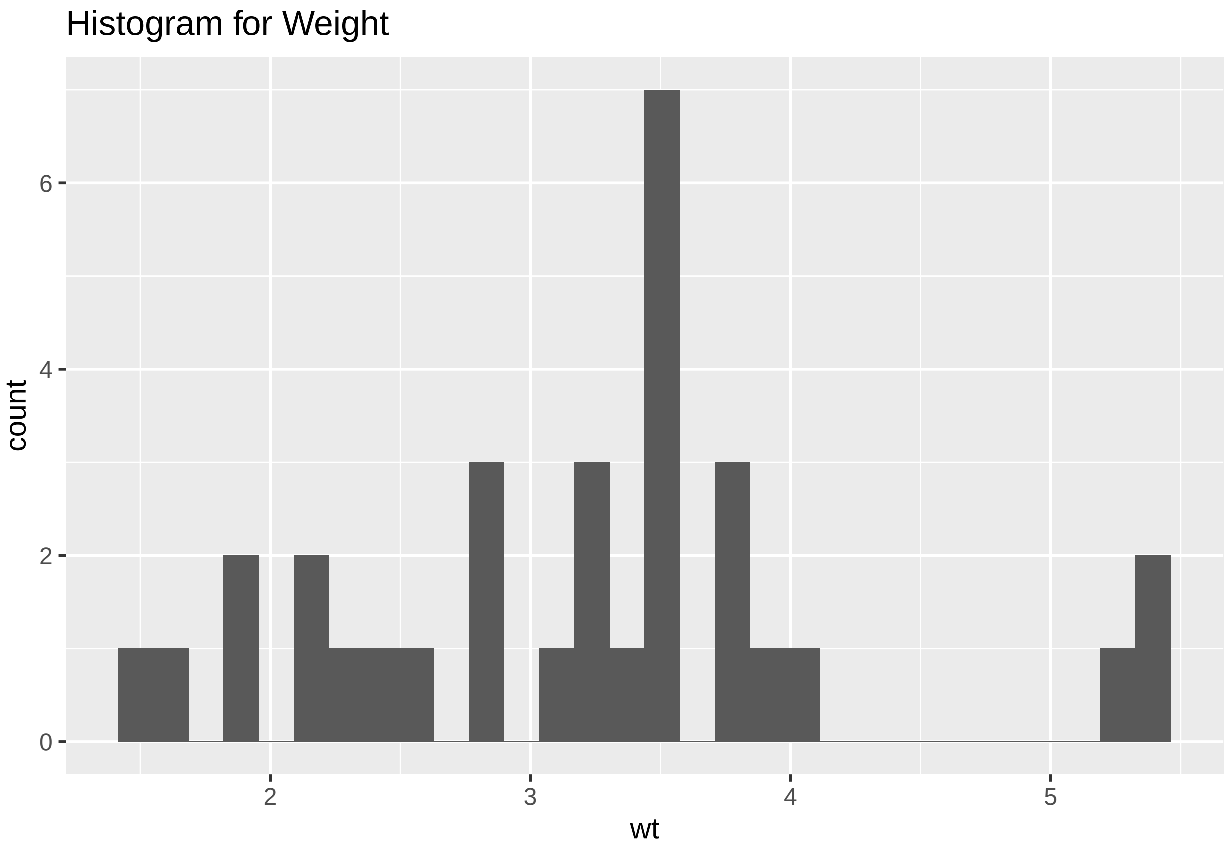


```
mtcars %>% ggplot(aes(x=wt)) + geom_line(aes(y=mpg,color=factor(cyl))) +
           xlab("Weight in lbs/1,000") + ylab("Miles per Gallon") +
           ggtitle("MPG vs Weight")
```

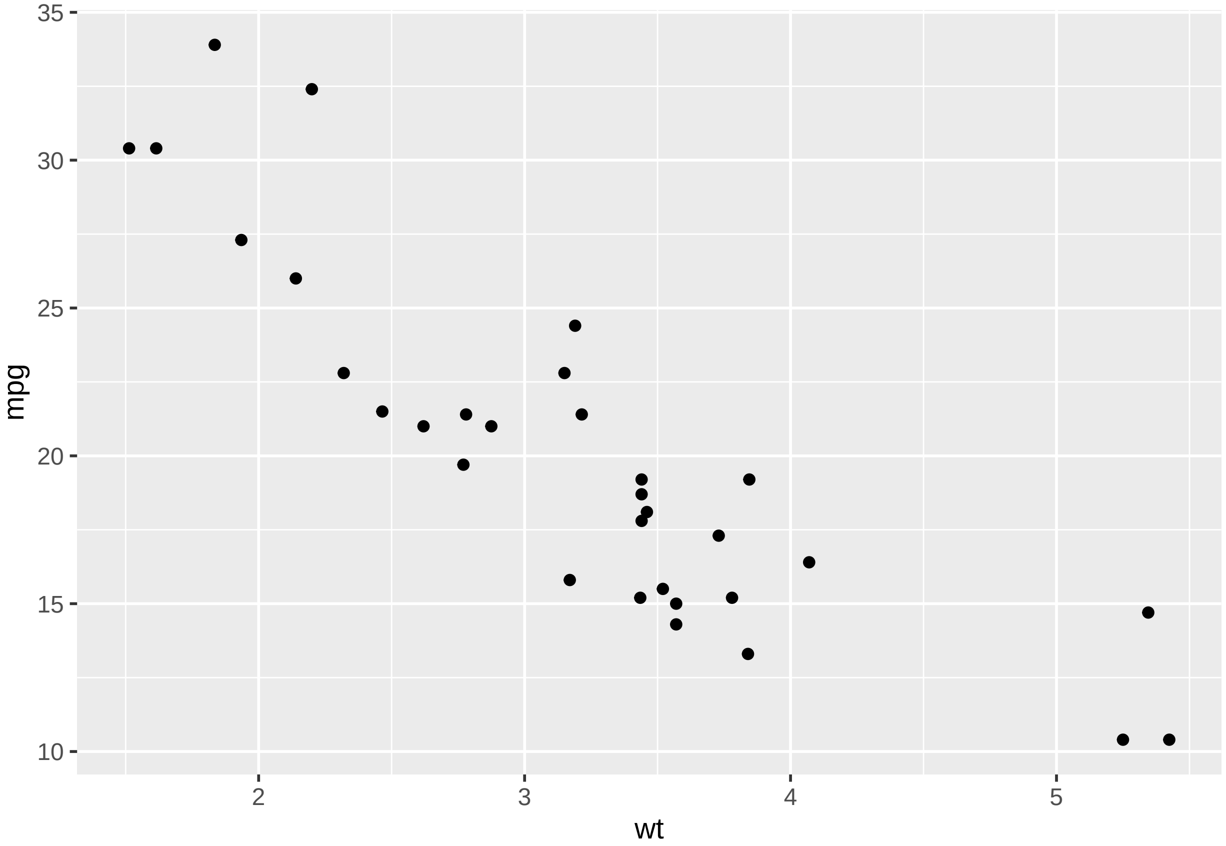


```r
# note that if we don't use the dplyr form we do something like this:

ggplot(mtcars, aes(x=wt)) + geom_histogram() + ggtitle("Histogram for Weight")
```

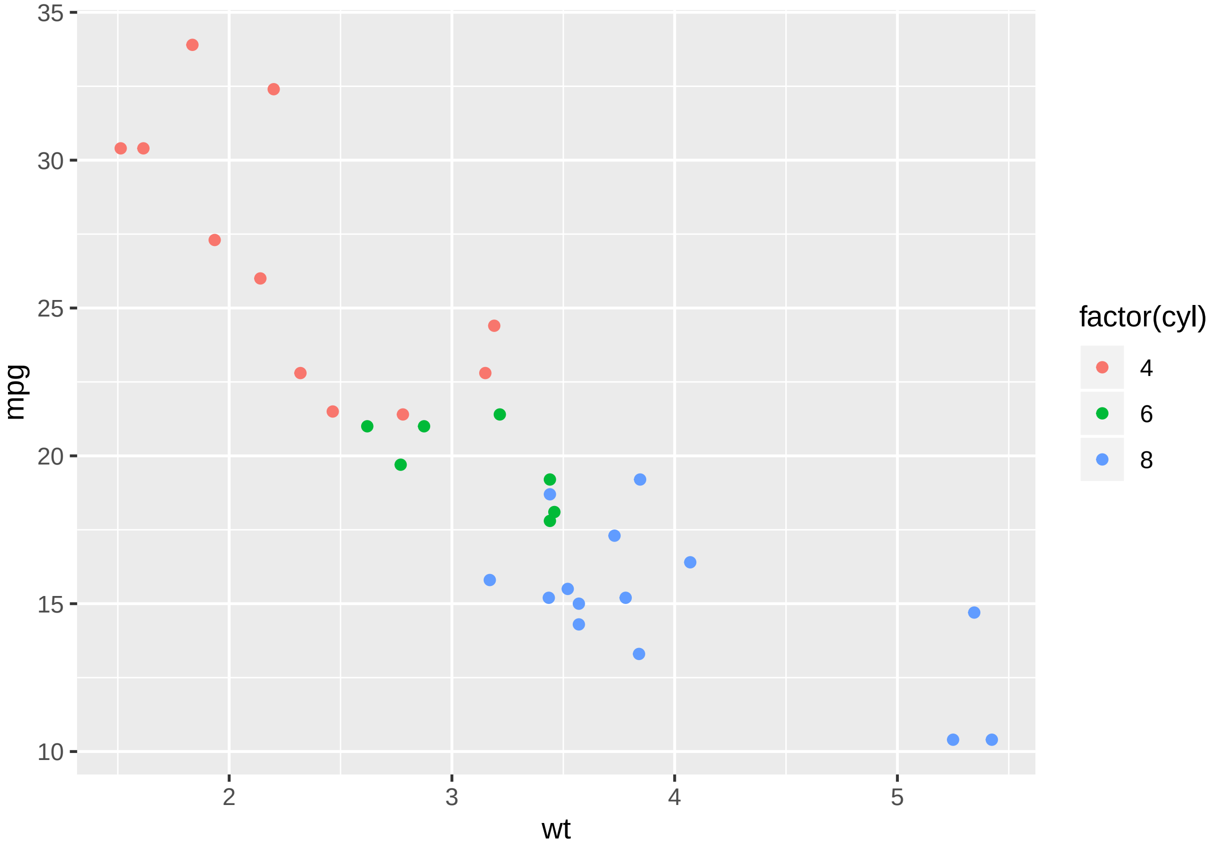
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



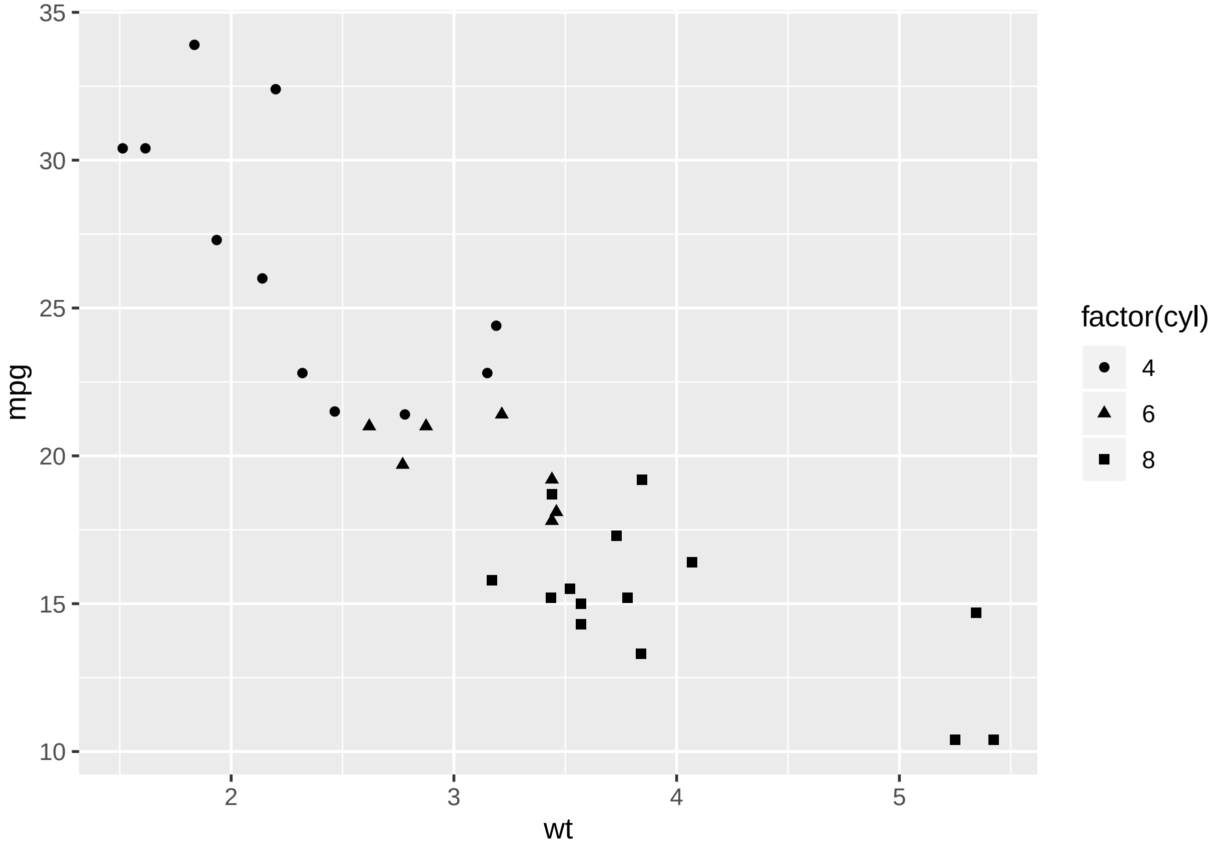
```
library(ggplot2)

# We establish a relationship between the data and some basic aesthetic
# mappings - the x and y axes

myplot <- ggplot(mtcars, aes(x=wt,y=mpg))

# Now that we've estbalished an aestehtic mapping we can put up layers and try out different
# plots.

myplot + geom_point()     # Just basic points
```



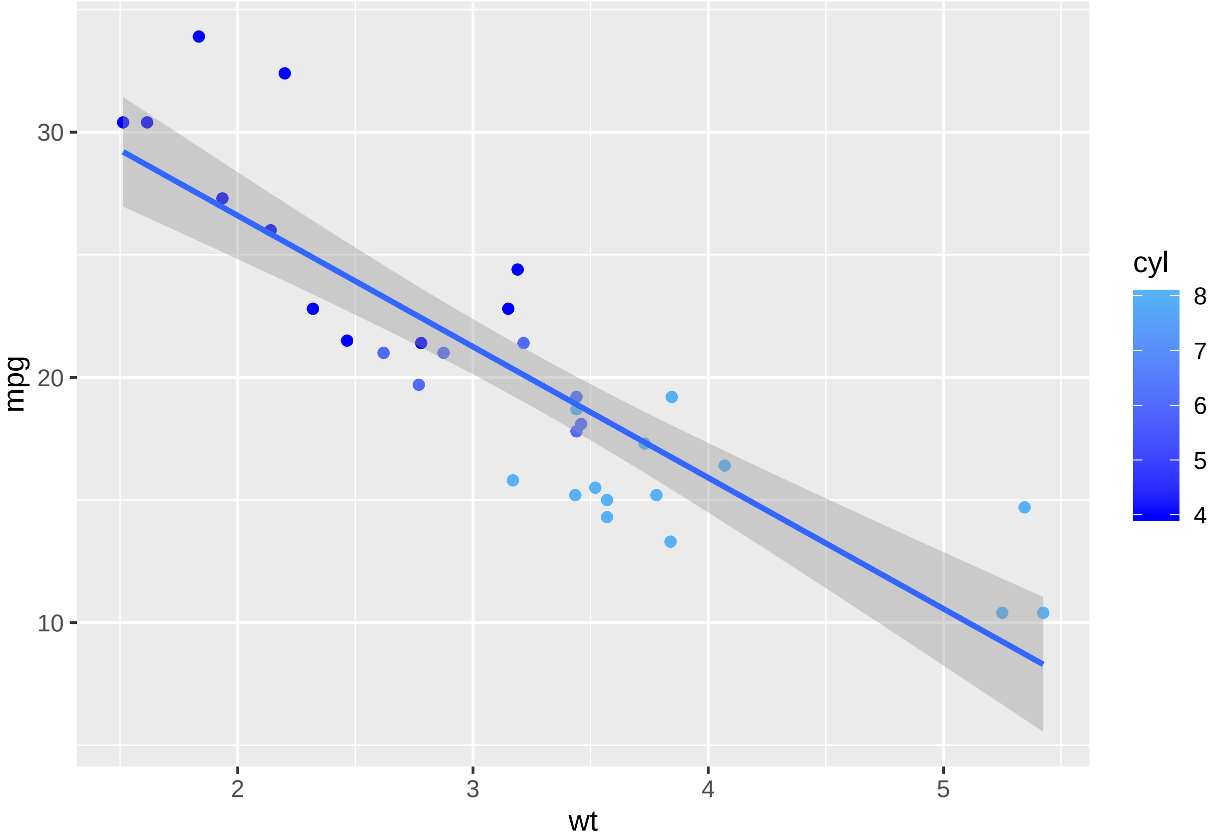
```
myplot + geom_point(aes(color=factor(cyl)))    # Similar to lattice "groups"
```



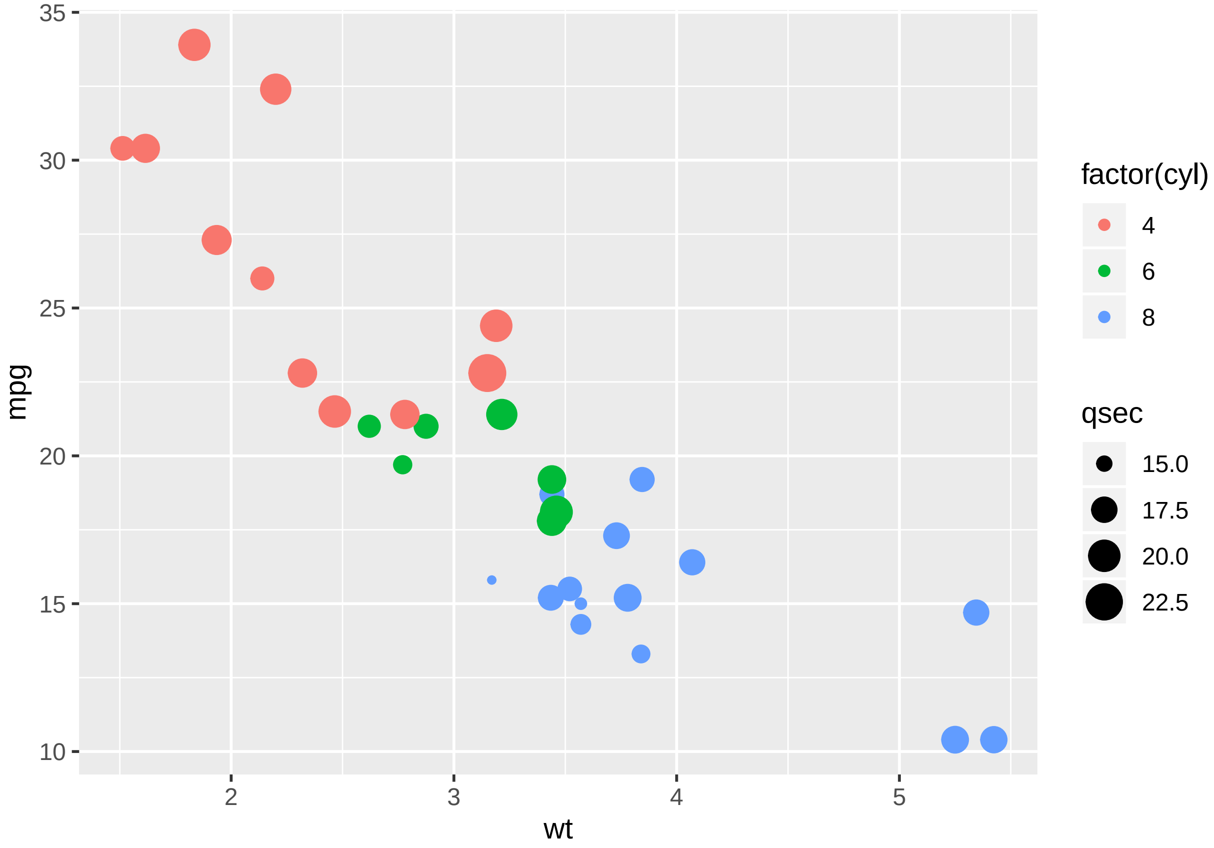
```
myplot + geom_point(aes(shape=factor(cyl)))    # Get a different shape for cyl
```



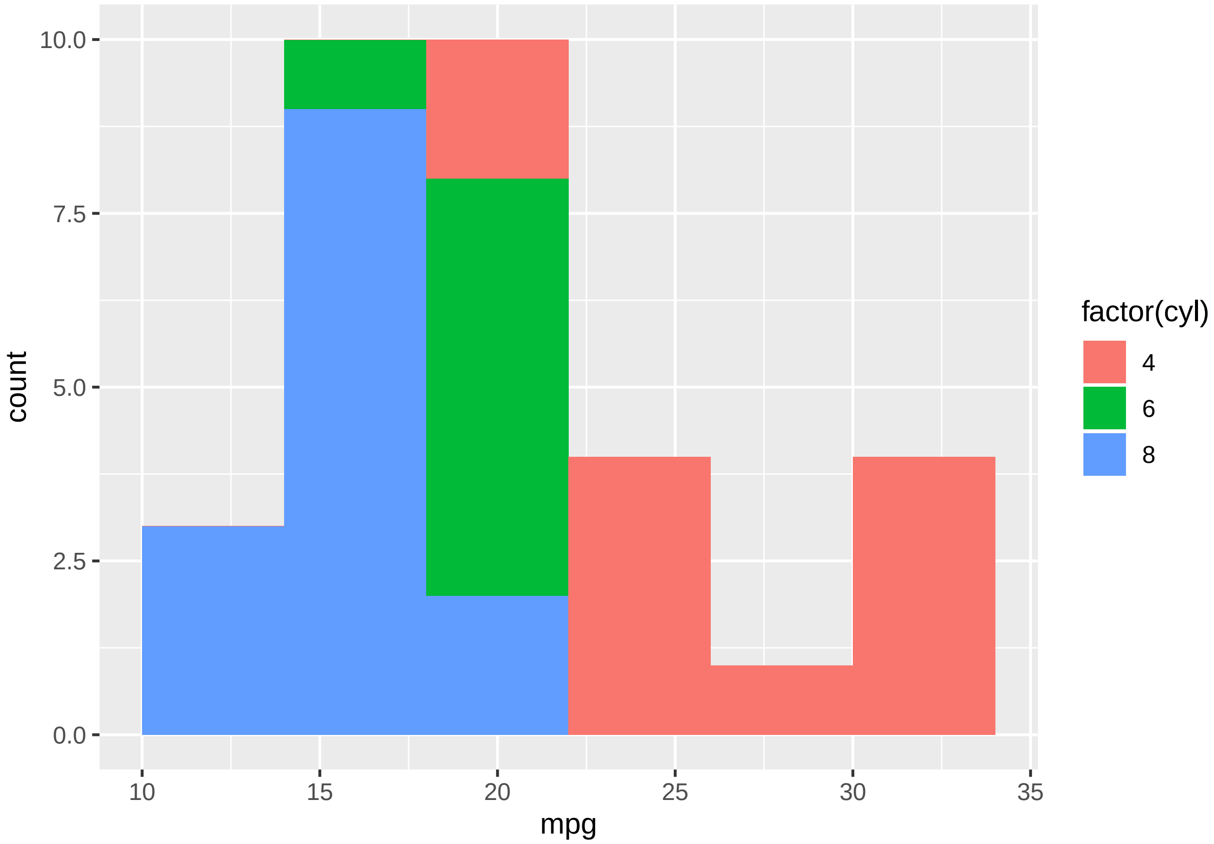
```
myplot + geom_point(aes(color = cyl)) +
  scale_colour_gradient(low = "blue") +
  geom_smooth(method="lm")
```



```
myplot +
  geom_point( aes(color=factor(cyl), size=qsec))  # Map two new aesthetics
```

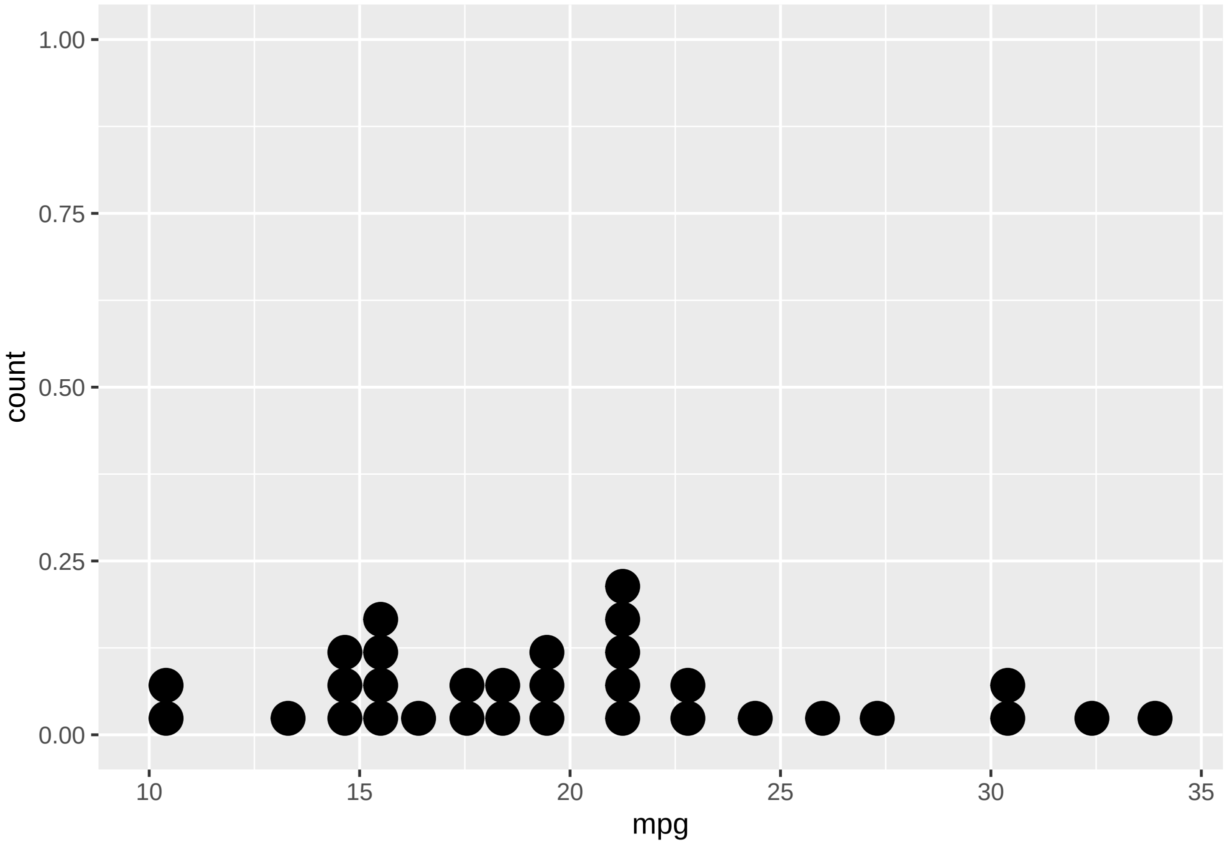


```
# Notice that we can change our basic ggplot mapping at any time

myplot <- ggplot(data=mtcars, aes(x=mpg))

myplot + geom_histogram(aes(fill=factor(cyl)),binwidth=4)
```

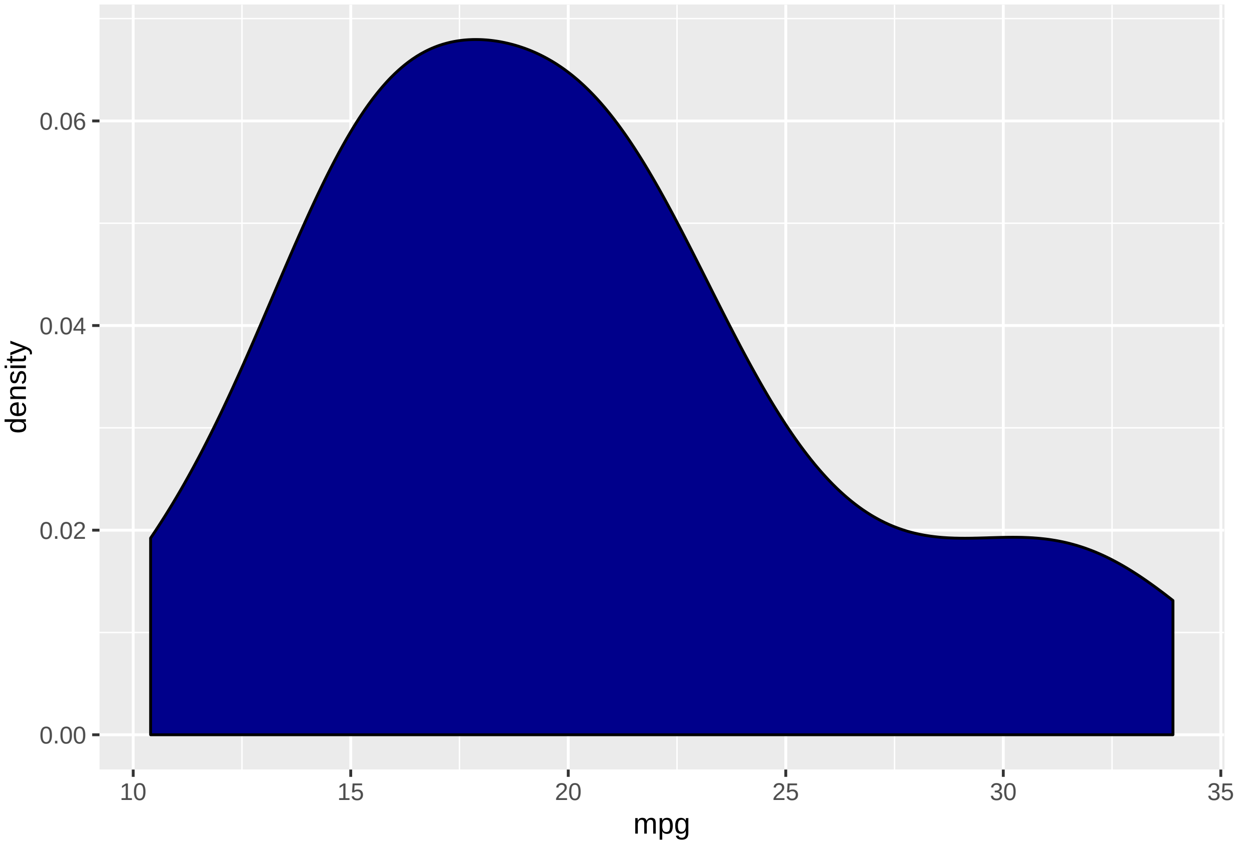


```
myplot + geom_dotplot()
```

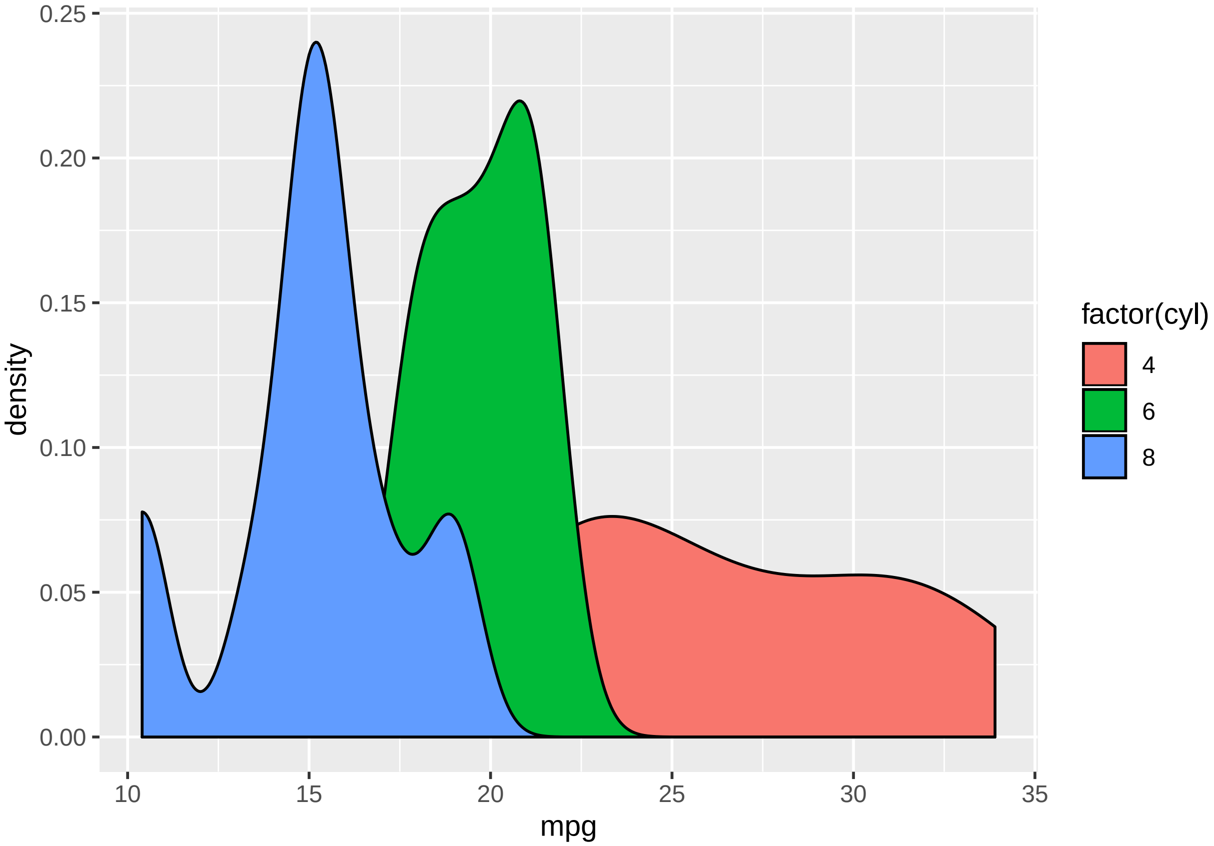
```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



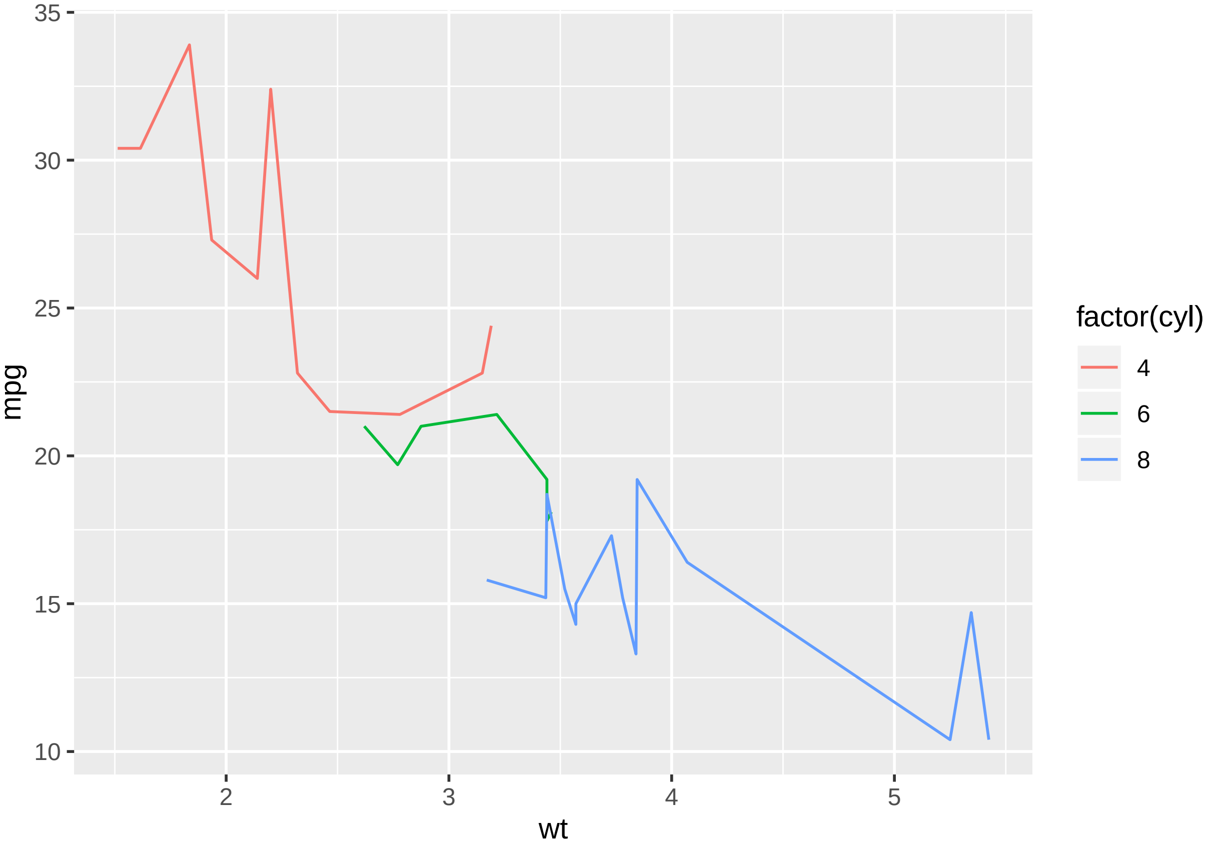
```
# Let's plot a density of the mpg variable

myplot + geom_density(fill="darkblue")
```



```
myplot + geom_density(aes(fill=factor(cyl)))
```



```
# Note that we can do groups explicitly

myplot <- ggplot(mtcars,aes(x=wt,y=mpg))
myplot  + geom_line(aes(group=factor(cyl),col=factor(cyl)))
```



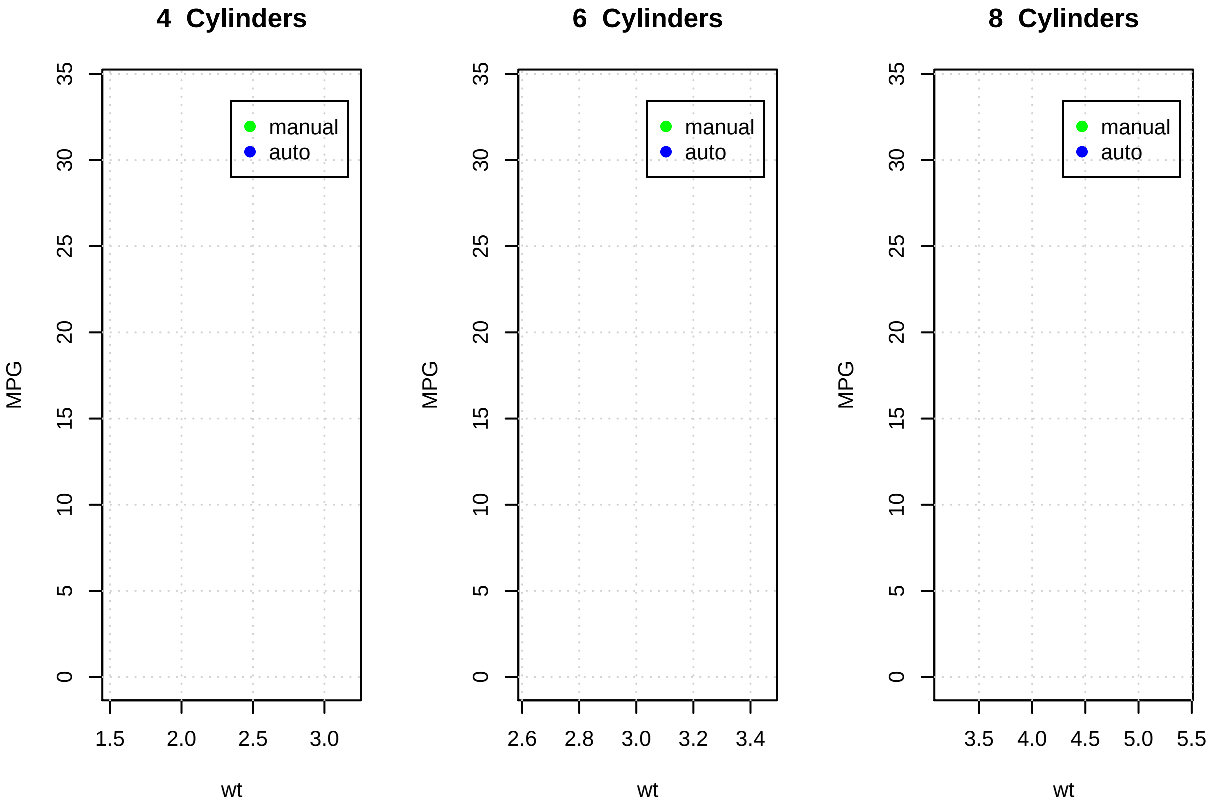
Remember this plot we created from Base graphics ? It took a fair amount of programming to pull of.

```
par(mfrow=c(1,3))  # set the graphics device to be one row and three columns

# Split the data frame based on cylinder values (4,6, or 8)
unique(mtcars$cyl)
```

```
## [1] 6 4 8
```

```
mysplits <- split(mtcars,mtcars$cyl)

maxmpg <- max(mtcars$mpg)   # Find the max MPG value

# Now for each element (dataframe) in mysplits initialize a plot and
# then use the points function to put up the points just for that data frame
# element

for (ii in 1:length(mysplits)) {
      tmpdf  <- mysplits[[ii]]

# Separate the automatic and manual transmission records

      auto <- tmpdf[tmpdf$am == 0,]
      man <- tmpdf[tmpdf$am == 1,]

# Setup a blank plot and use the points function to plot the points
# using different colors for auto vs manual
```

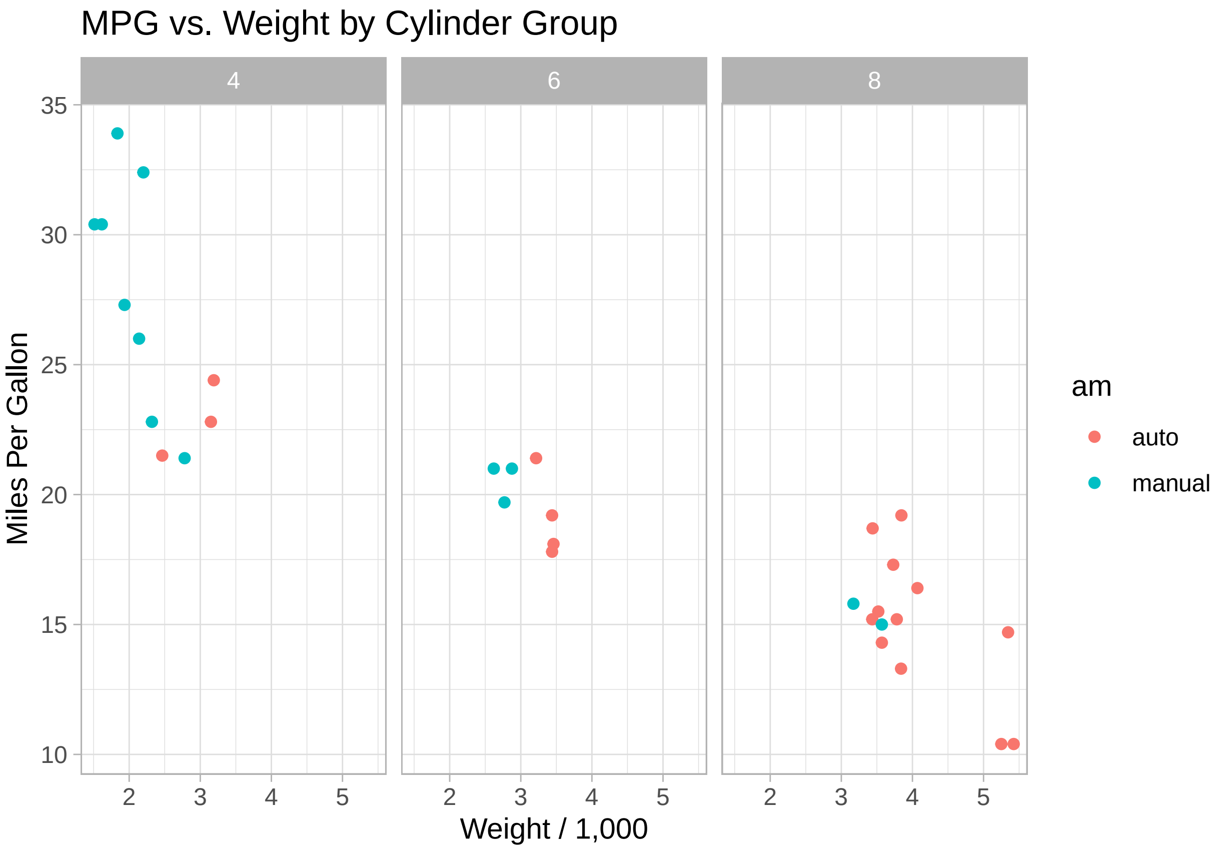
```
      plot(tmpdf$wt, tmpdf$mpg,type="n",
           main=paste(names(mysplits[ii])," Cylinders"),
           ylim=c(0,maxmpg), xlab="wt",ylab="MPG")
      points(auto$wt,auto$mpg,col="blue",pch=19)
      points(man$wt,man$mpg,col="green",pch=19)
      grid()
      legend("topright", inset=0.05, c("manual","auto"),
             pch = 19, col=c("green","blue"))
}
```



```
par(mfrow=c(1,1))  # reset the graphics device to be one row and one column
```

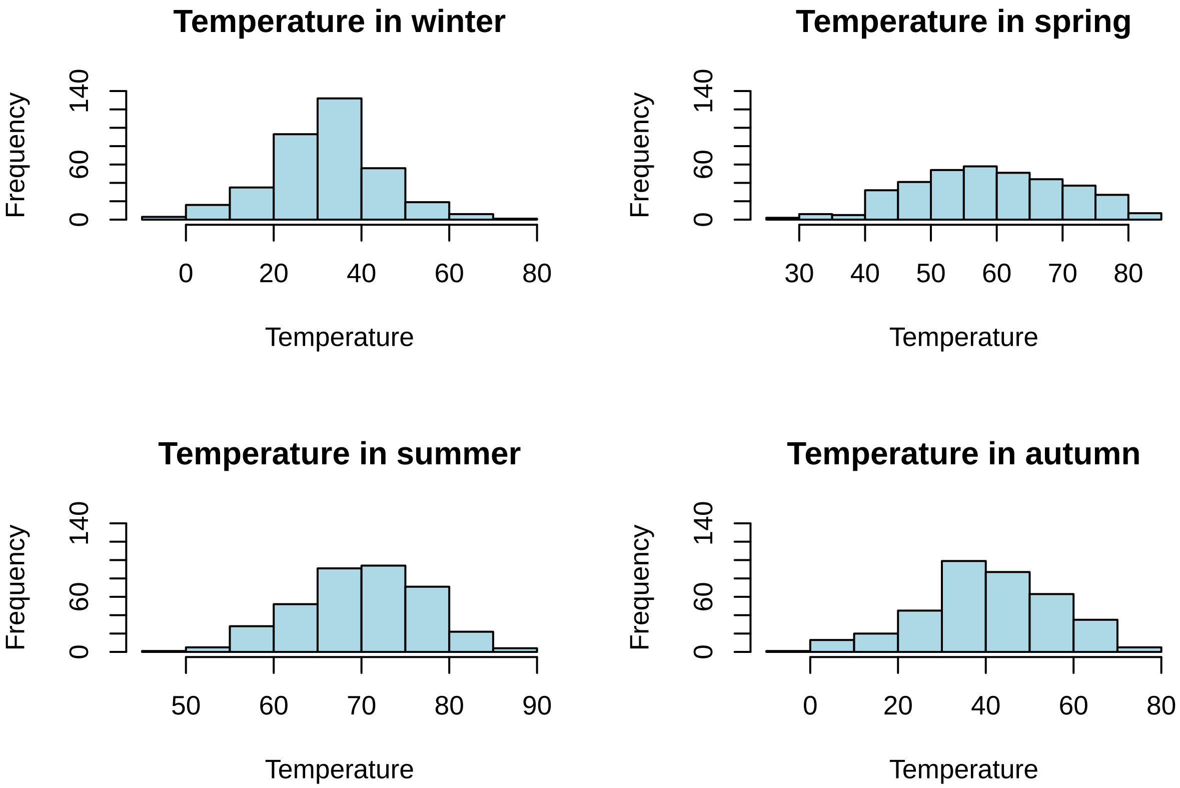
We can do this with ggplot much more easily as long as we understand some things about ggplot.

```
mtcars %>%
  mutate(am=factor(am,labels=c("auto","manual"))) %>%
  ggplot(aes(x=wt,y=mpg,color=am)) +
       geom_point() +
       ggtitle("MPG vs. Weight by Cylinder Group") +
       xlab("Weight / 1,000") +
       ylab("Miles Per Gallon") +
       theme_light() +
       facet_wrap(~cyl)
```



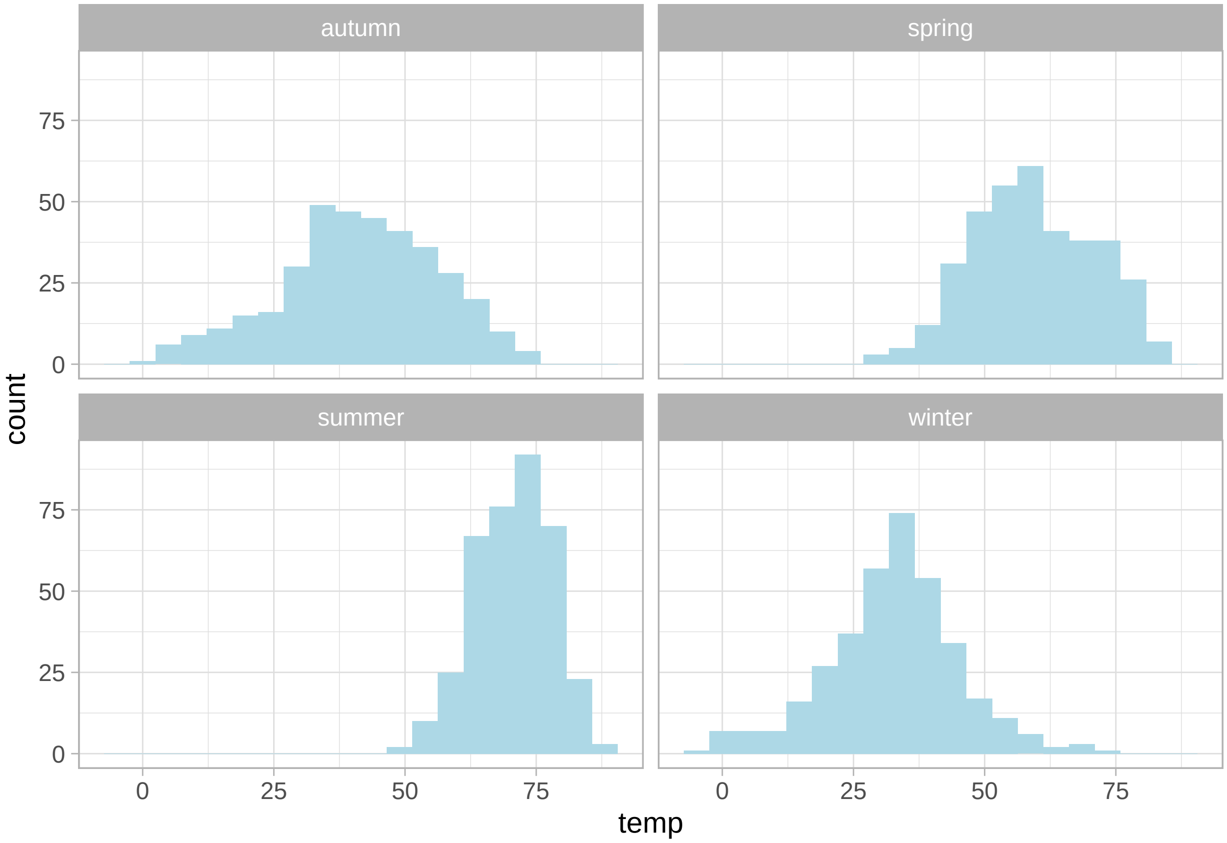
What about the histograms we made of seasonal temperatures ? Here is what it looked like in Base Graphics.

```
par(mfrow=c(2,2))  # Get 2 rows by 2 columns

xlab <- "Temperature"
ylim <- c(0,140)
col <- "lightblue"

seasons <- unique(nm$season)
tempstr <- "Temperature in"
for (ii in 1:length(seasons)) {
  title  <- paste(tempstr,seasons[ii],sep=" ")
  hist(nm[nm$season==seasons[ii],]$temp, main=title,xlab=xlab,ylim=ylim,col=col)
}
```



```
par(mfrow=c(1,1)) # Reset plot window to 1 row by 1 column
```

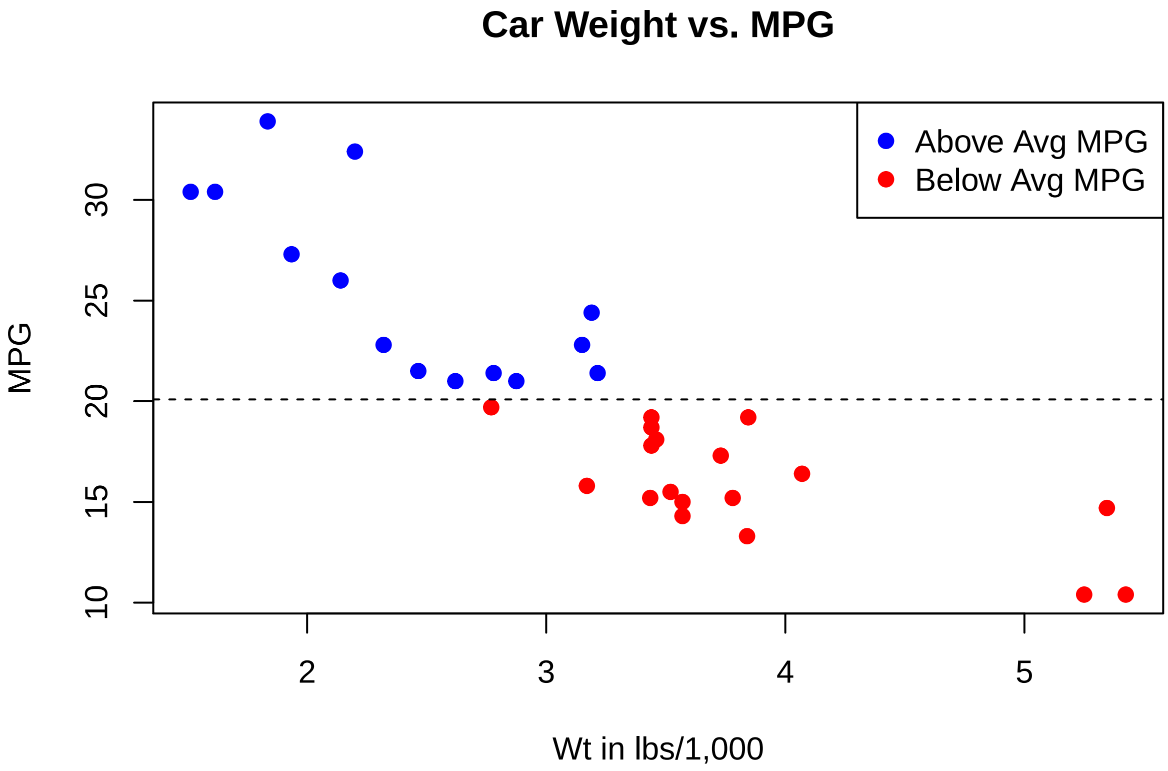
```
nm %>% ggplot(aes(x=temp,fill="lightblue")) +
  geom_histogram(bins=20) +
  facet_wrap(~season,nrow=2) +
  scale_fill_manual(values="lightblue",guide=FALSE) +
  theme_light()
```



What about this Base code ?

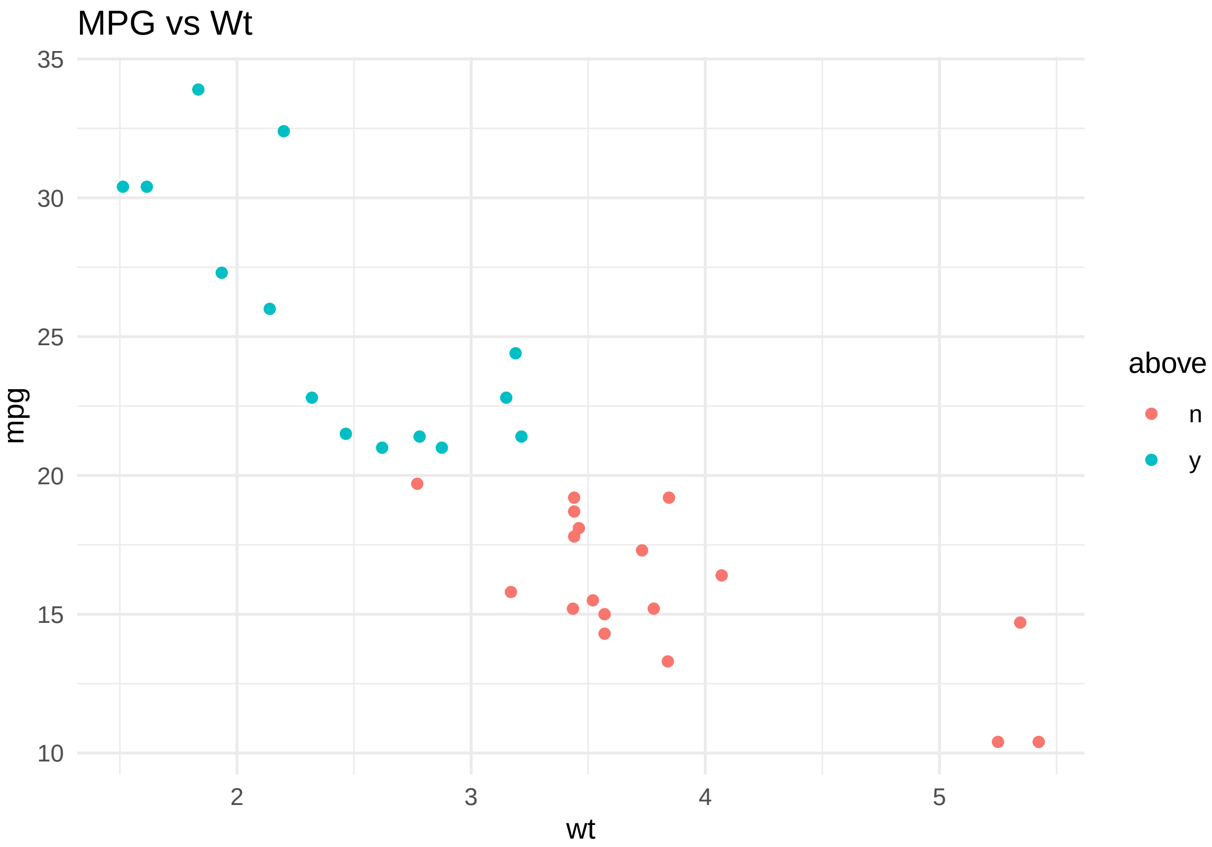
```
data(mtcars)     # The most used dataset in R Education !
title <- "Car Weight vs. MPG"
xlab <- "Wt in lbs/1,000"
ylab <- "MPG"

# First we setup a blank plot - we do everythig EXCEPT put up the points

plot(mtcars$wt, mtcars$mpg, main=title,xlab=xlab,ylab=ylab, type="n")

# Find just the rows where mpg is >= the mean MPG for the whole data set
# Then use the points function to draw them with color blue

aboveavg <- mtcars[mtcars$mpg >= mean(mtcars$mpg),]
points(aboveavg$wt,aboveavg$mpg,col="blue",pch=19)

# Find the rows where mpg is < the mean MPG for the whole data set
# Use points to draw them with color red

belowavg <-  mtcars[mtcars$mpg < mean(mtcars$mpg),]
points(belowavg$wt,belowavg$mpg,col="red",pch=19)

# Draw a line that represents the average MPG
abline(h=mean(mtcars$mpg),lty=2)

# Now we draw a legend to identify what color matches which group
```

```
legend("topright",c("Above Avg MPG","Below Avg MPG"),pch=19,col=c("blue","red"))
```



Here is how we can reproduce this with ggplot.

```
mtcars %>% mutate(above = ifelse(mpg > mean(mpg),"y","n")) %>%
  ggplot(aes(x=wt,y=mpg)) +
  geom_point(aes(color=above)) +
  theme_minimal() +
  ggtitle("MPG vs Wt")
```



The iris data

Let's look at the built in iris data for a change of pace. It's also important to know how to do faceting which is the ggplot equivalent of conditioning in Lattice graphics.

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

```
str(iris)
```

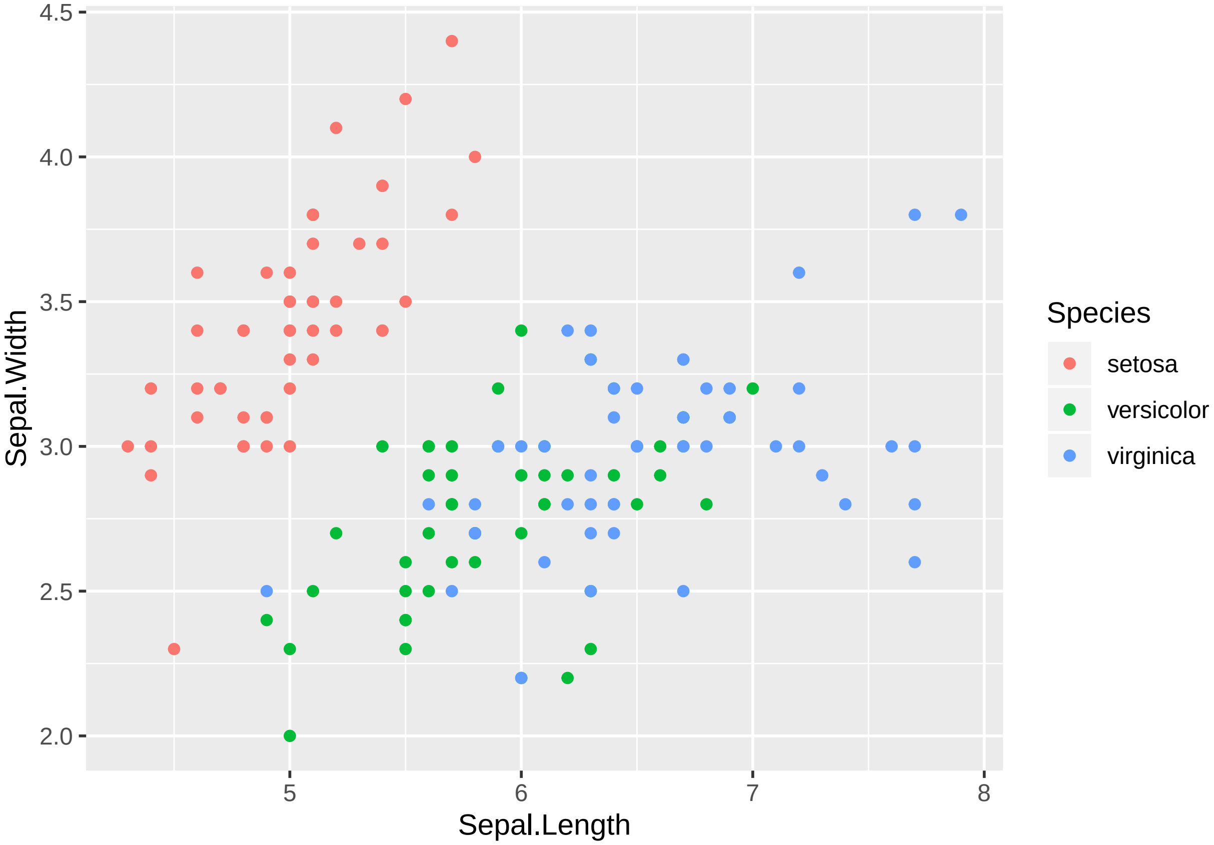
```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
head(iris)
```

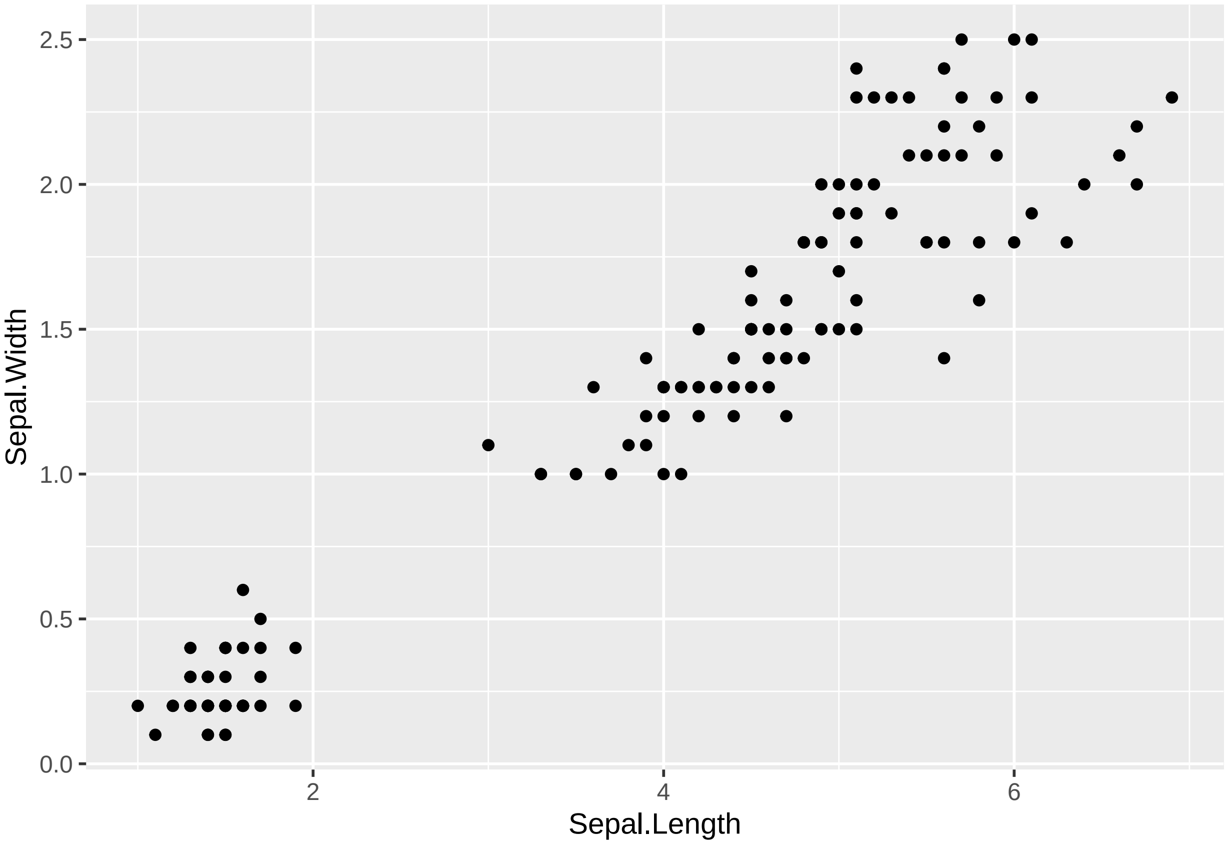
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Let's illustrate faceting

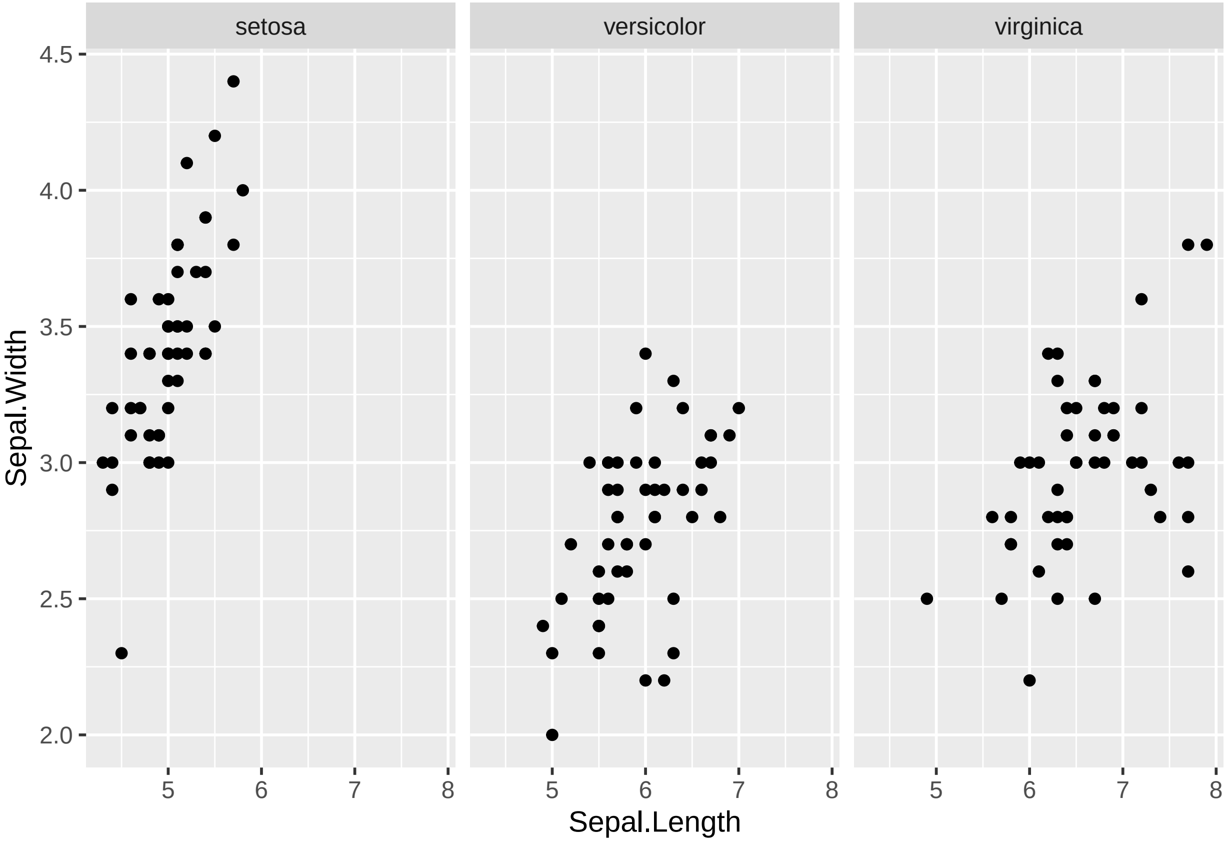
```
iris_plot <- ggplot(iris,aes(x=Sepal.Length,y=Sepal.Width))
iris_plot + geom_point(aes(color=Species))
```



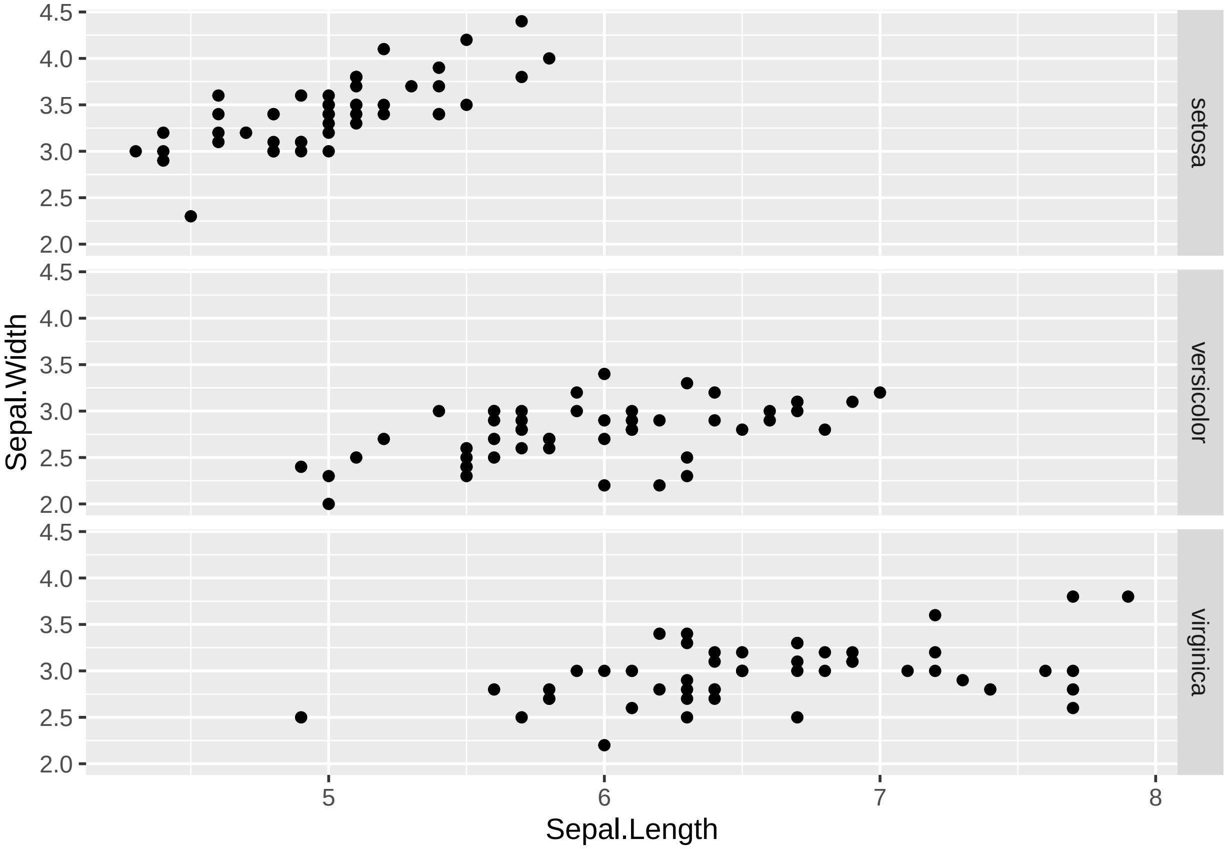
```
# Note also that we can override the x and y variables specified in the ggplot definition

iris_plot + geom_point(aes(x=Petal.Length,y=Petal.Width))
```



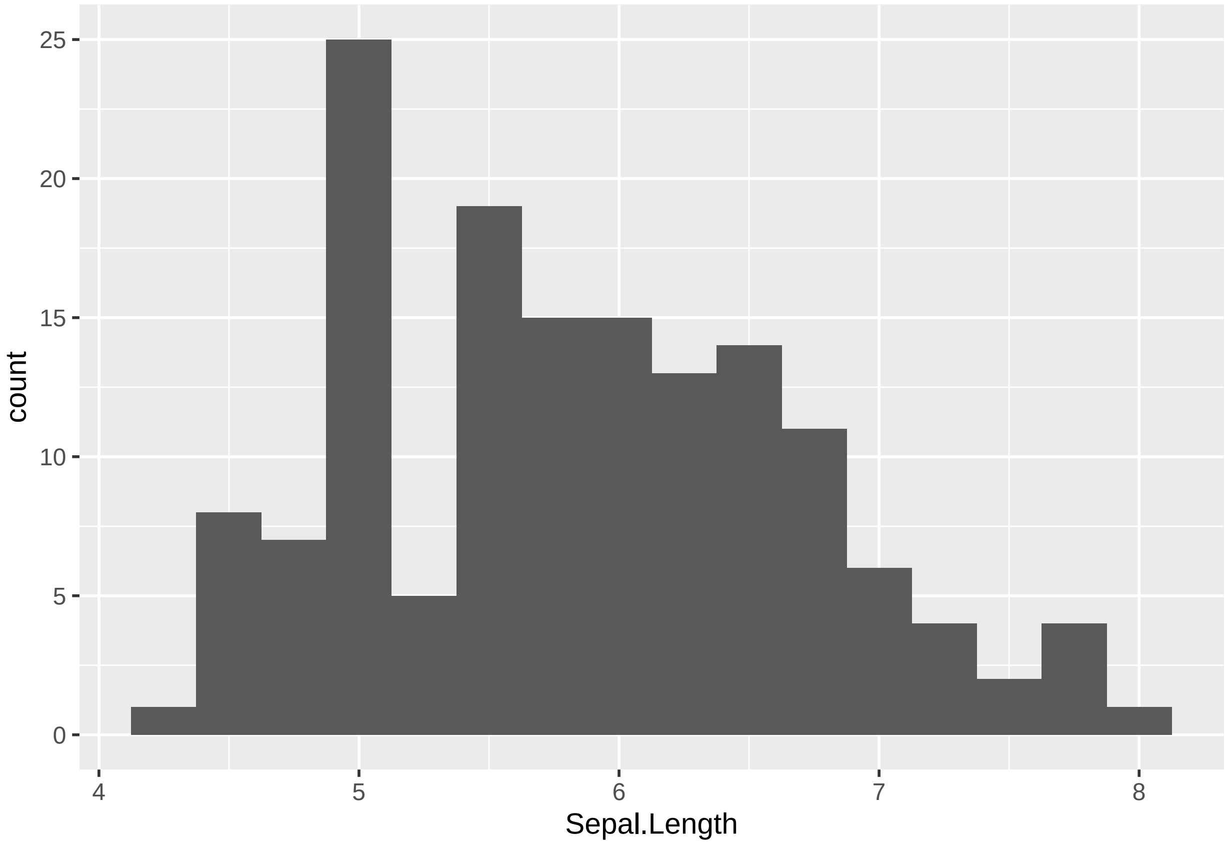
```
# Or we could break the comparion between Species into facets

iris_plot + geom_point() + facet_grid(.~Species)
```



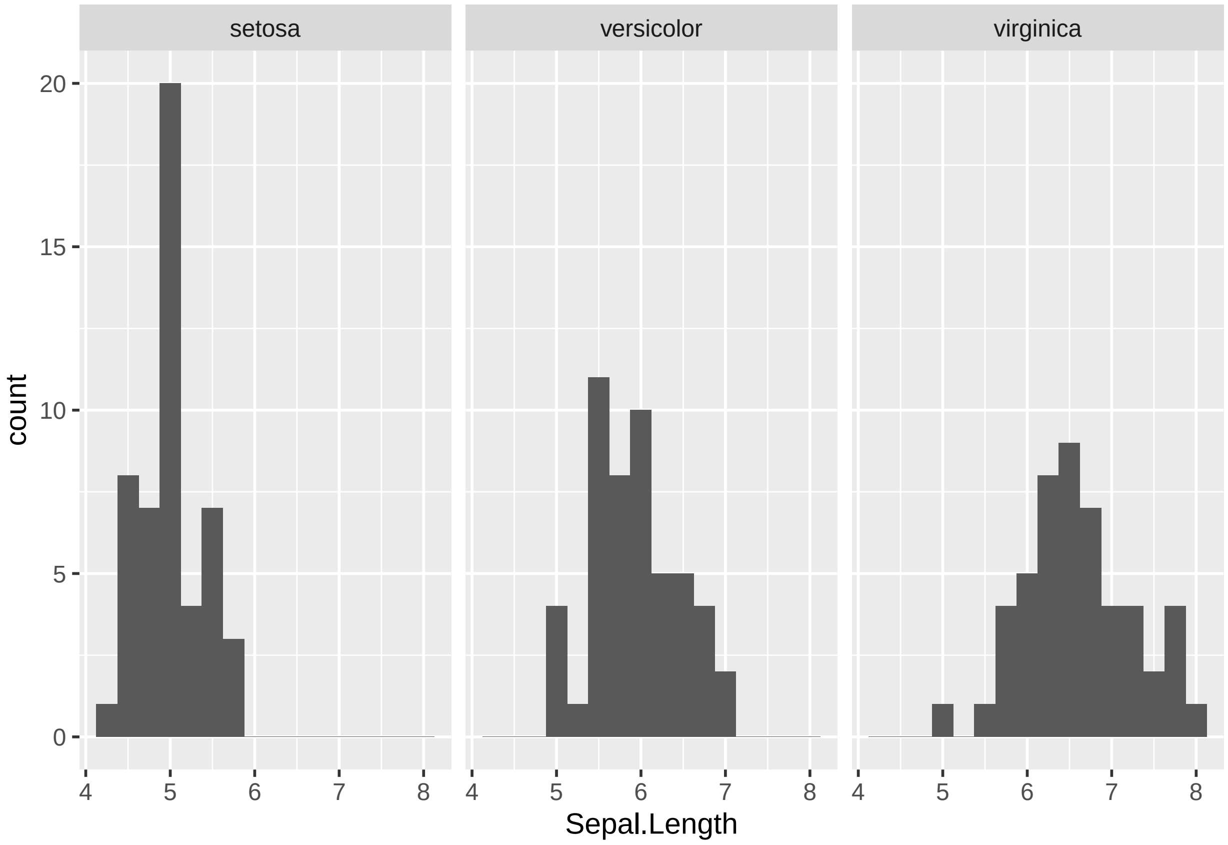
```
iris_plot + geom_point() + facet_grid(Species~.)
```



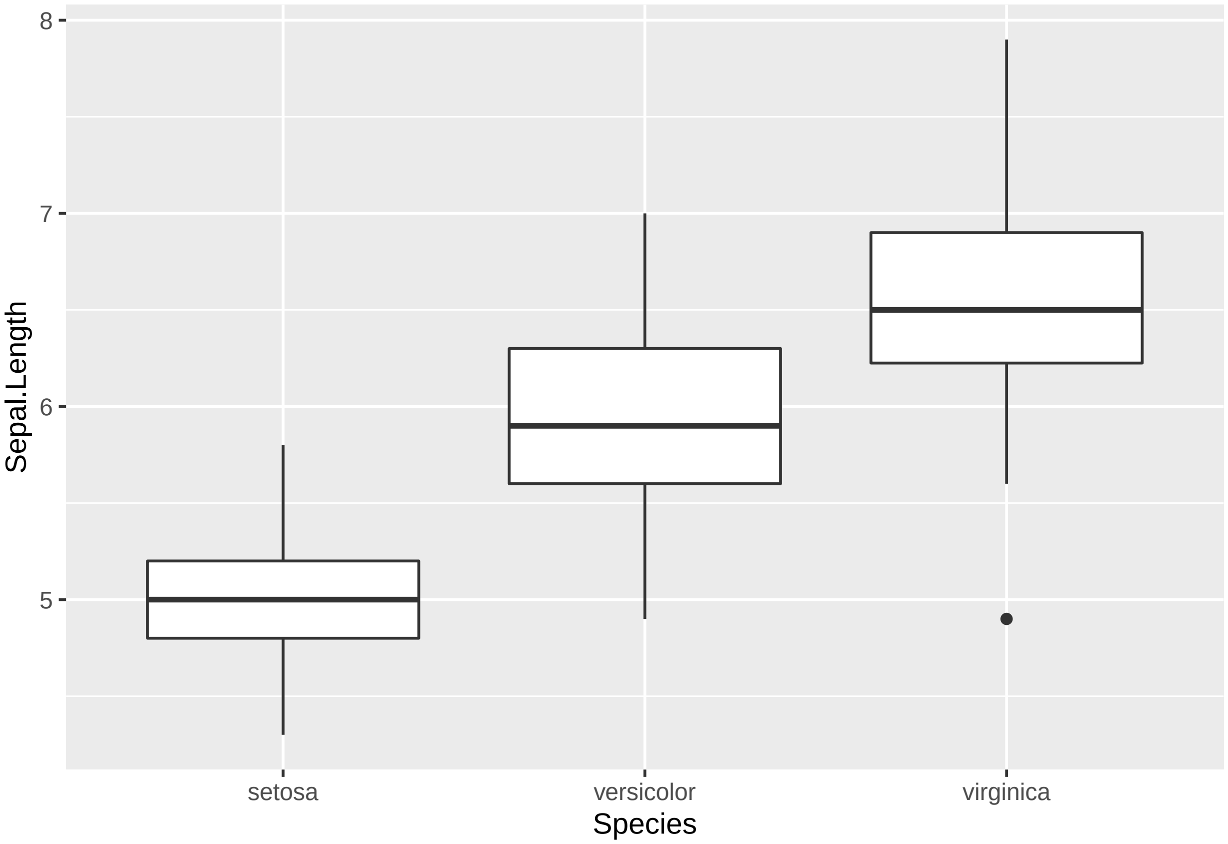
```
# Of course faceting works indepenednetly of chart type

iris_plot <- ggplot(iris,aes(x=Sepal.Length))
iris_plot + geom_histogram(binwidth=0.25)
```



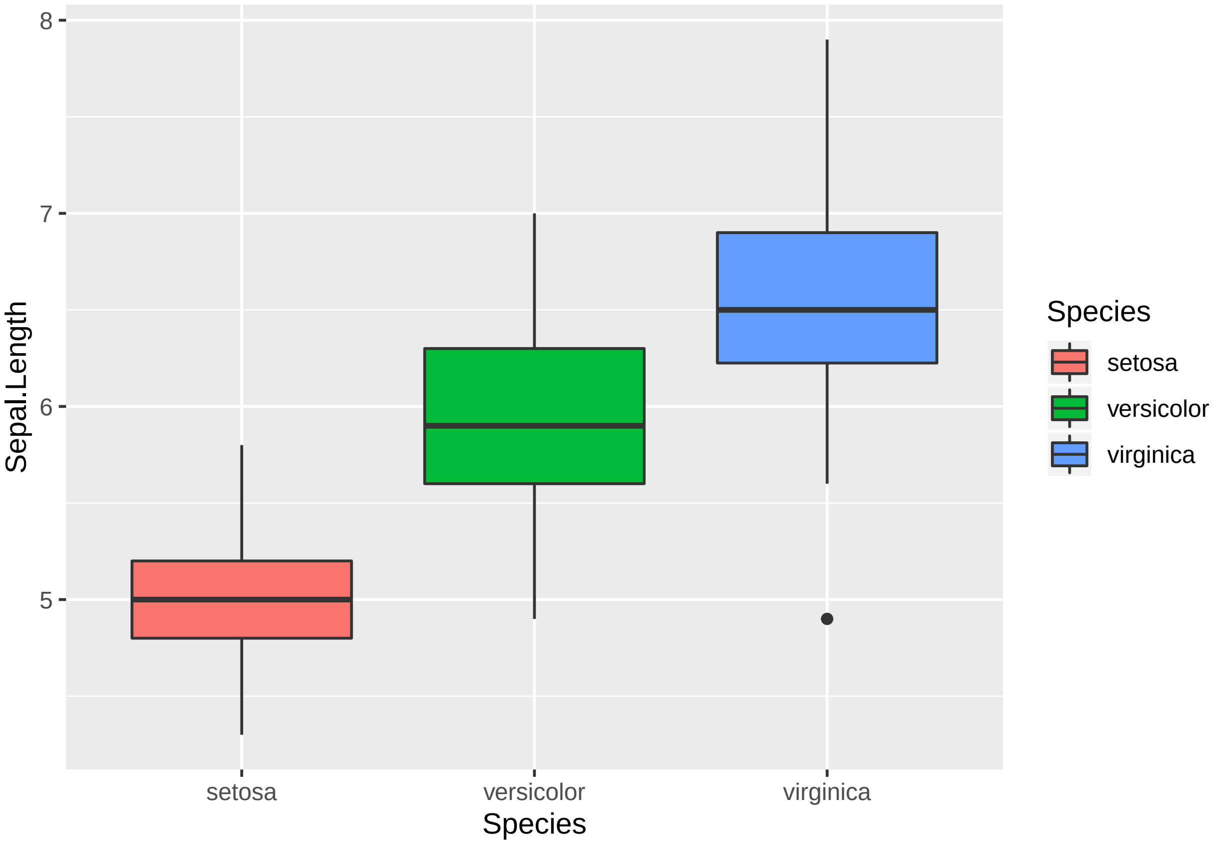
```
iris_plot + geom_histogram(binwidth=0.25) + facet_grid(.~Species)
```



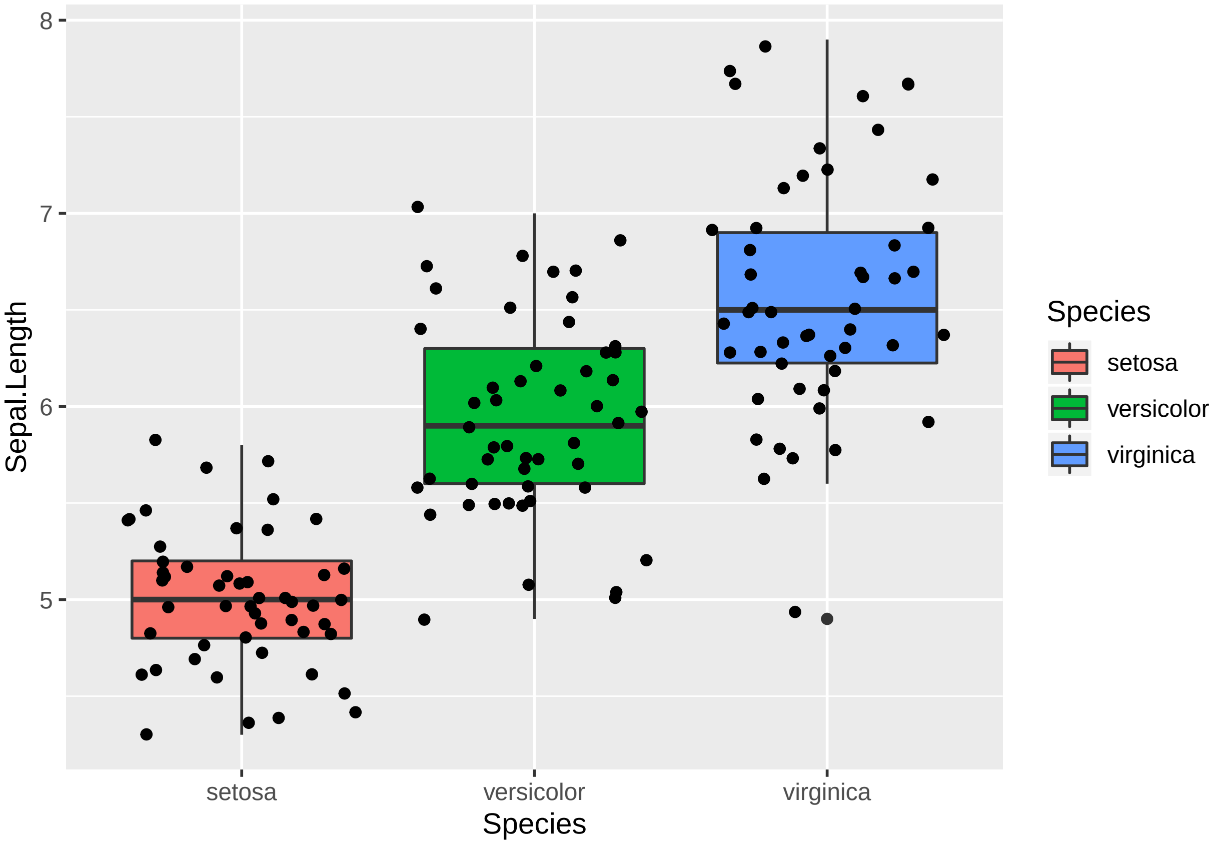
```
# Let's try some box plots

iris_plot <- ggplot(iris, aes(x=Species,y=Sepal.Length))

iris_plot + geom_boxplot()
```



```
iris_plot + geom_boxplot(aes(fill=Species)) # Not really necessary
```

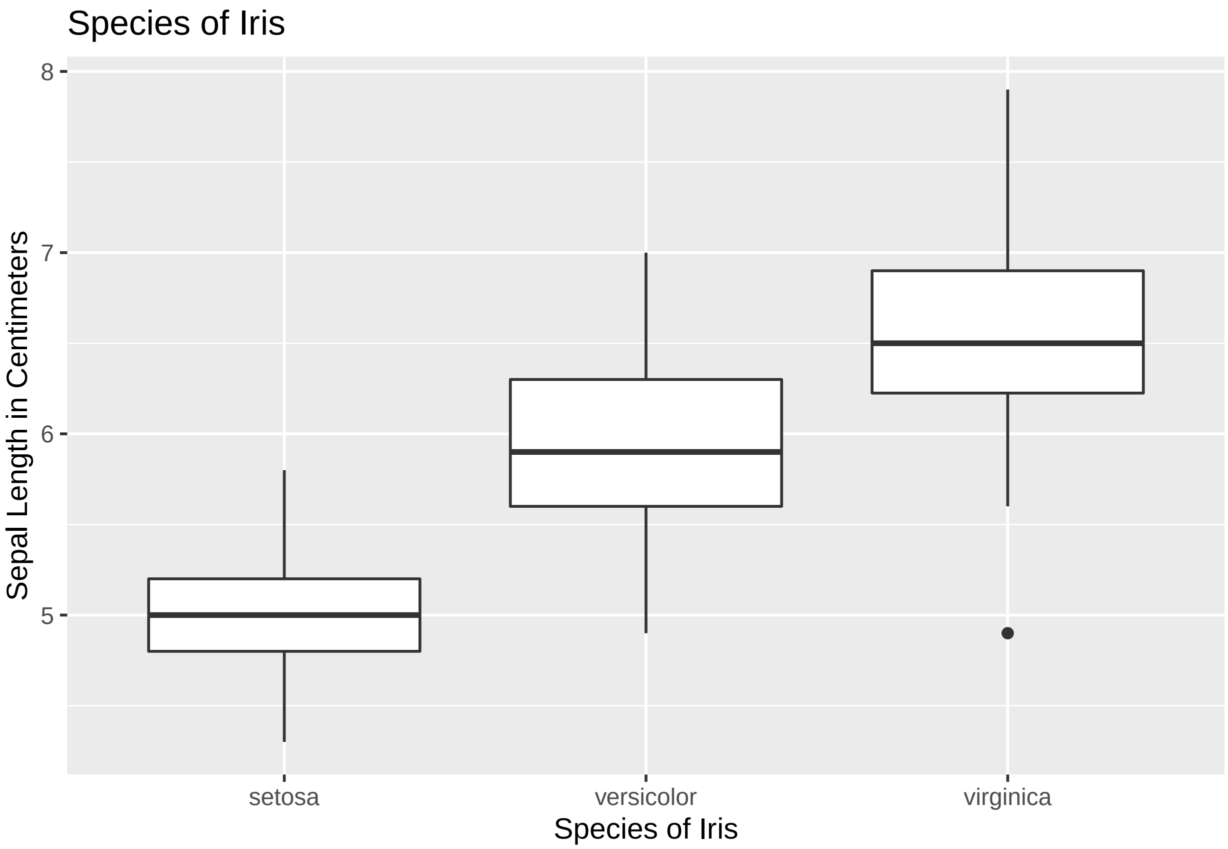


```
iris_plot + geom_boxplot(aes(fill=Species)) + geom_jitter()
```

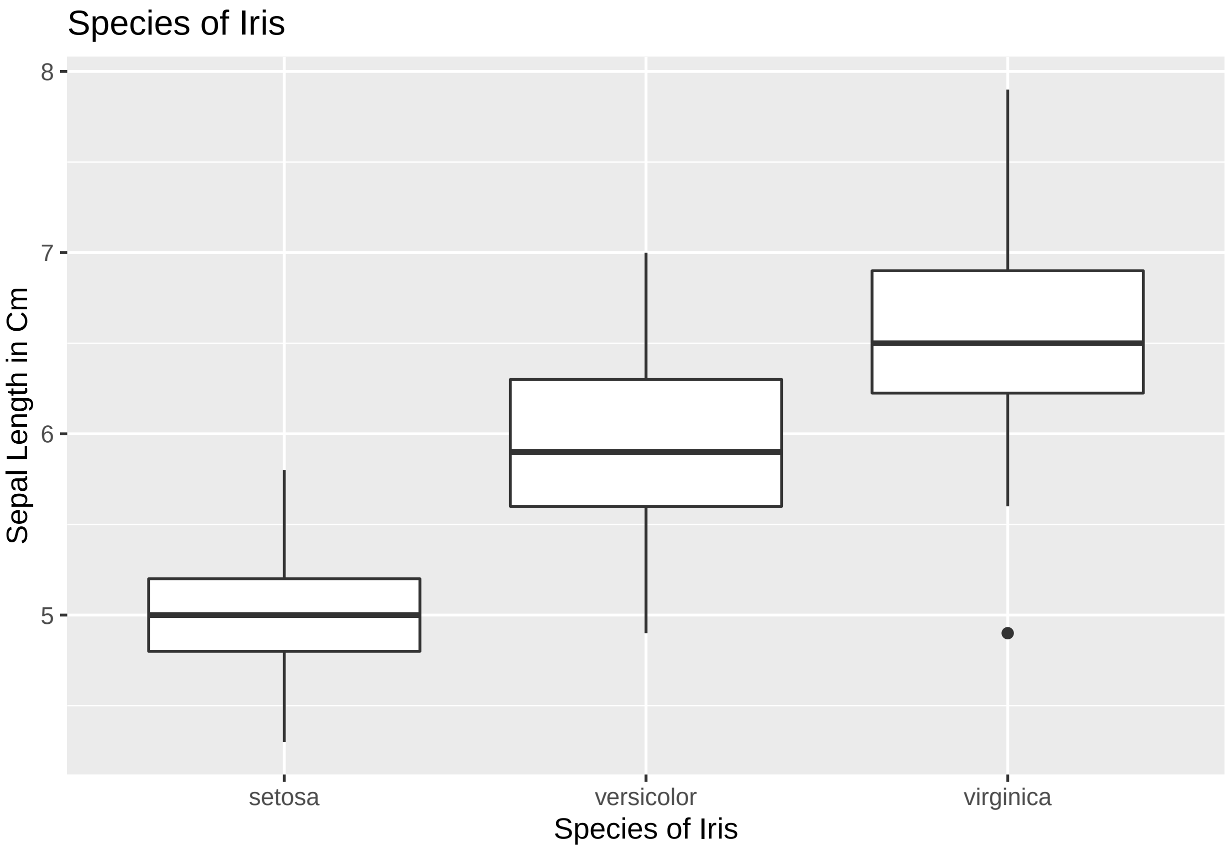


When it comes to annotation and labelling there are a number of ways to do this. You can add labels in layers just as you would plot types and new aesthetics. When you add the legends, titles, axis labels, etc is up to you. Many put up the plot first and then add in the annotation later.

```
iris_plot <- ggplot(iris,aes(x=Species,y=Sepal.Length))
iris_plot + geom_boxplot() + xlab("Species of Iris") + ylab("Sepal Length in Centimeters") + ggtitle("Sp
```



```
# This can be consolidate if you wish

iris_plot + geom_boxplot() +
        labs(x="Species of Iris", y="Sepal Length in Cm", title="Species of Iris")
```

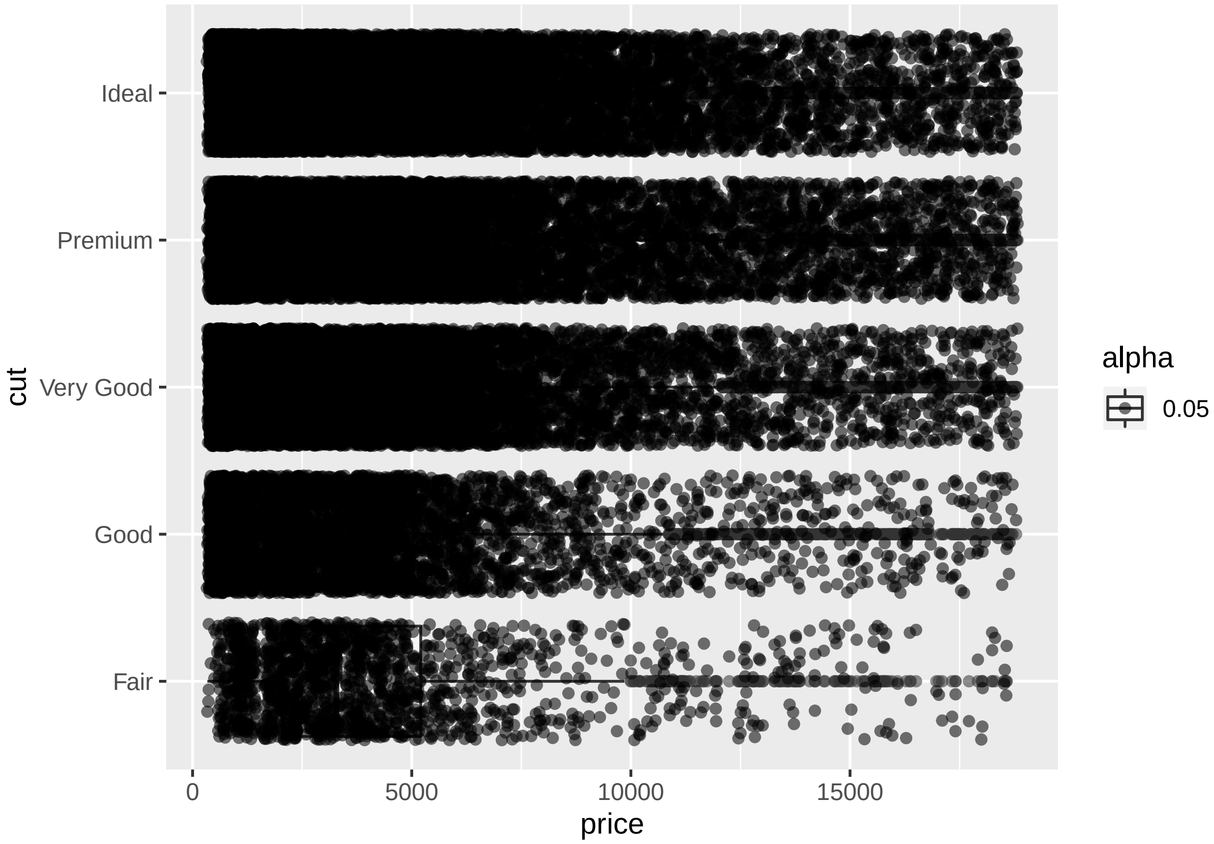


Diamonds

Let's look at some more involved data. Refer to the diamonds data frame that comes as part of the ggplot2 package. It's a dataset containing the prices and other attributes of almost 54,000 diamonds. There are 10 variables:

- price in US Dollars ($326 - $18,823)
- carat weight of the diamond (0.2 - 5.01)
- cut: quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- colour: diamond colour, from J (worst) to D (best)
- clarity: a measurement of how clear the diamond is (I1 (worst), SI1, SI2, VS1, VS2, VVS1, VVS2, IF (best))
- x: length in mm (0–10.74)
- y: width in mm (0–58.9)
- z: depth in mm (0–31.8)
- depth. total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
- table. width of top of diamond relative to widest point (43–95)

```
ggplot(diamonds,aes(x=cut,y=price,alpha=0.05)) +
  geom_boxplot() + coord_flip() +
  geom_jitter()
```



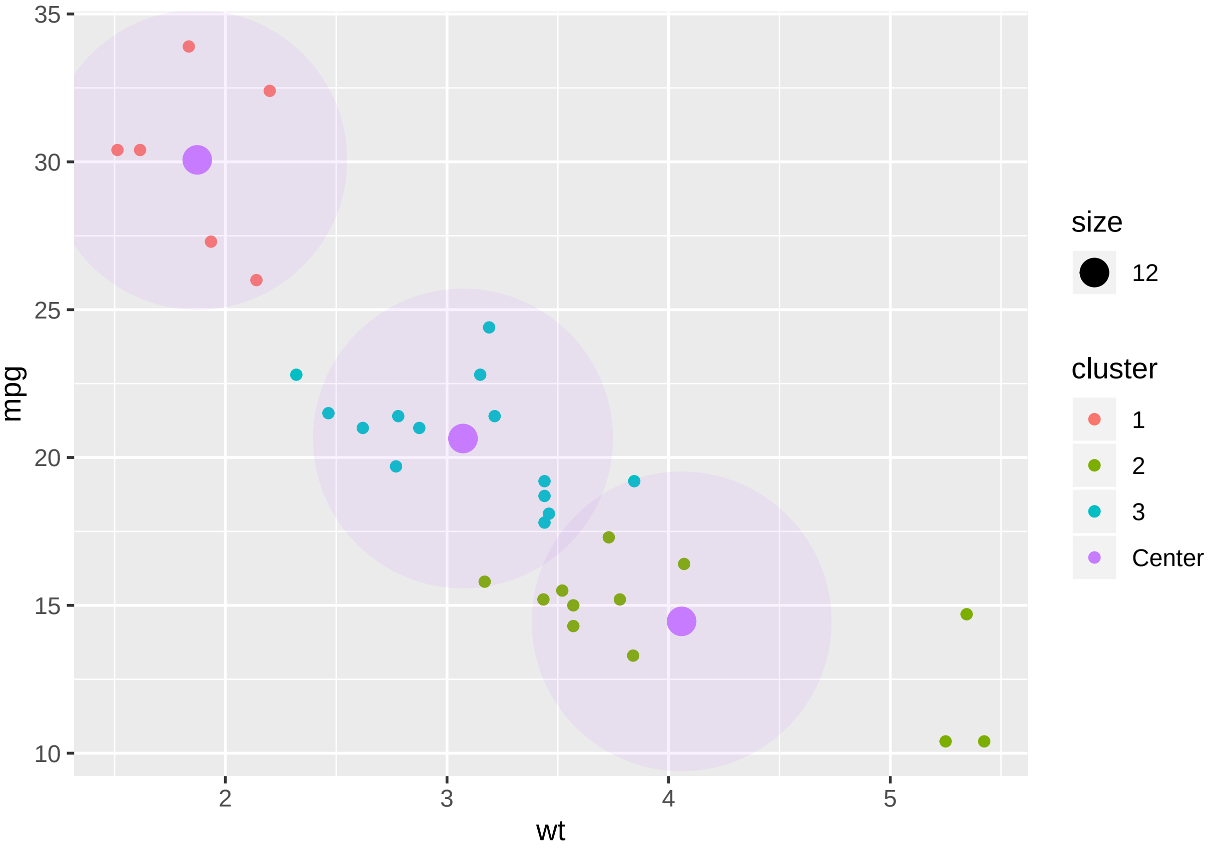
We can use ggplot to plot clusters are returned by the K-Means clustering function in R. The goal of this function is to partition the observations into a predetermined set of clusters such that each observation belongs to one of the clusters. Obviously there will be some misclassification but this helps us identify which observations might be in a group.

```
library(ggplot2)

df <- mtcars[,c(1,6)]
clus <- kmeans(df,3)
df$cluster <- factor(clus$cluster)
head(df)
```

```
##                    mpg    wt cluster
## Mazda RX4         21.0 2.620       3
## Mazda RX4 Wag     21.0 2.875       3
## Datsun 710        22.8 2.320       3
## Hornet 4 Drive    21.4 3.215       3
## Hornet Sportabout 18.7 3.440       3
## Valiant           18.1 3.460       3
```

```
centers <- as.data.frame(clus$centers)

iplot <- ggplot(data=df, aes(x=wt,y=mpg,color=cluster))

iplot + geom_point() +
  geom_point(data=centers, aes(x=wt,y=mpg, color='Center',size=12)) +
  geom_point(data=centers, aes(x=wt,y=mpg,color='Center'),size=52,alpha=.1,show_guide=FALSE)
```

```
## Warning: `show_guide` has been deprecated. Please use `show.legend`
## instead.
```


mpg
wt
size
cluster
Center