Lattice and ggplot

April 29, 2016

Contents

1 Lattice

The lattice package provides a comprehensive graphical system for visualizing univariate and multivariate data. In particular, many users turn to the lattice package because of its ability to easily generate trellis graphs.

- lattice: contains code for producing Trellis graphics, which are independent of the "base" graphics system; includes functions like xyplot, bwplot, levelplot
- The lattice plotting system does not have a "two-phase" aspect with separate plotting and annotation like in base plotting
- All plotting/annotation is done at once with a single function call

1.1 Lattice Functions

The lattice package provides a wide variety of functions for producing univariate (dot plots, kernel density plots, histograms, bar charts, box plots), bivariate (scatter plots, strip plots, parallel box plots), and multivariate (3D plots, scatter plot matrices) graphs.

- xyplot: this is the main function for creating scatterplots
- bwplot: box-and-whiskers plots ("boxplots")
- histogram: histograms
- stripplot: like a boxplot but with actual points
- dotplot: plot dots on "violin strings"
- splom: scatterplot matrix; like pairs in base plotting system
- levelplot, contourplot: for plotting "image" data

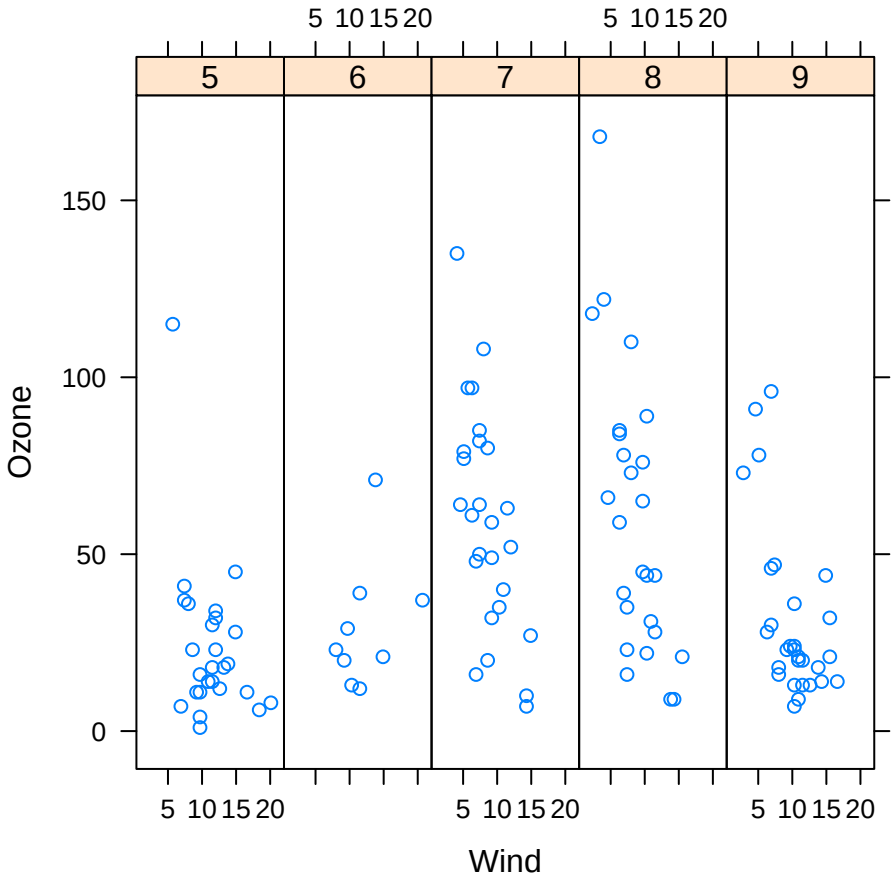
Lattice functions generally take a formula for their first argument, usually of the form

```
xyplot(y ~ x | f * g, data, option)
```

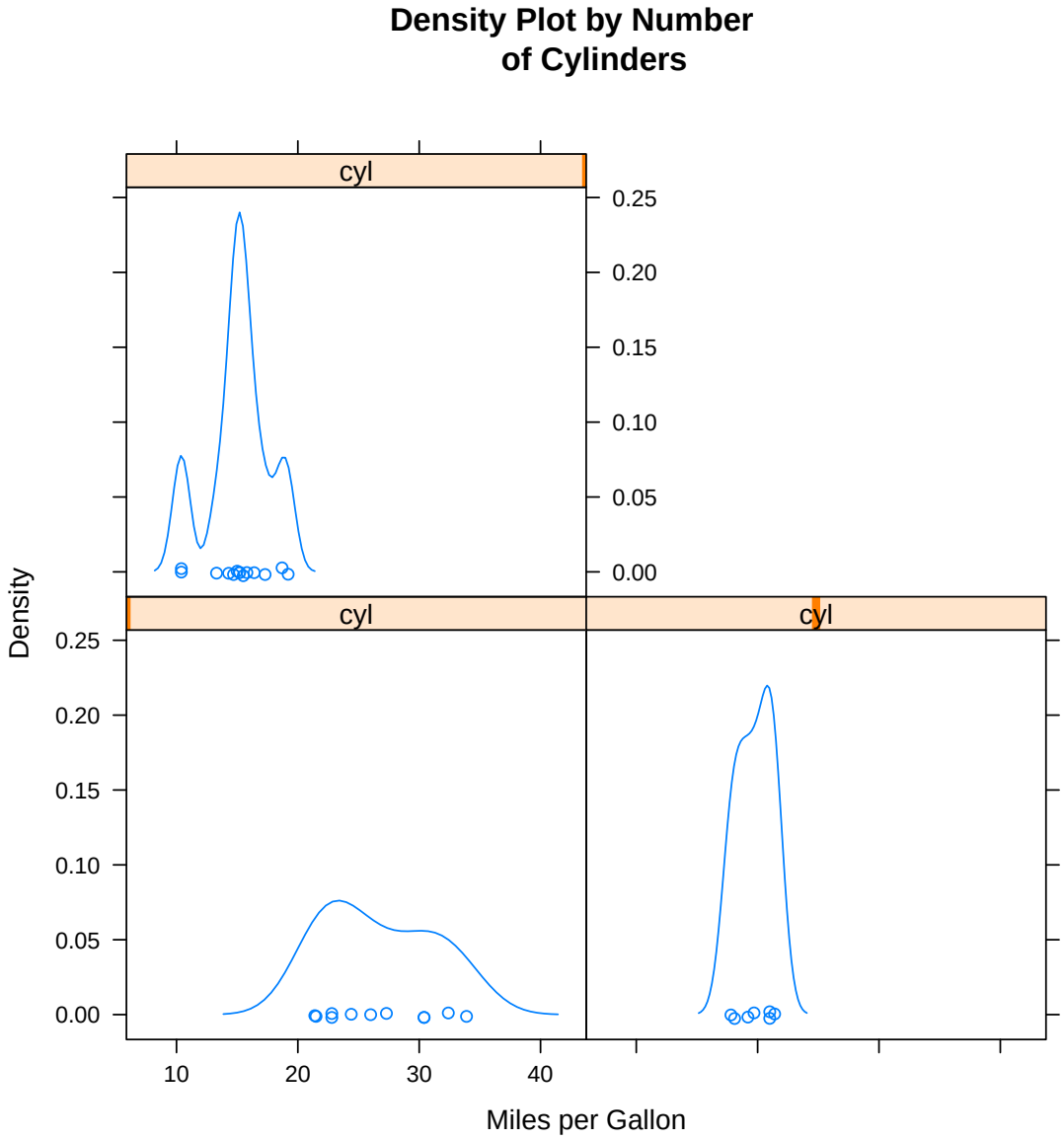
- We use the formula notation here, hence the ~.
- On the left of the ~ is the y-axis variable, on the right is the x-axis variable
- f and g are conditioning variables — they are optional
  - the * indicates an interaction between two variables
- The second argument is the data frame or list from which the variables in the formula should be looked up
  - If no data frame or list is passed, then the parent frame is used
- If no other arguments are passed, there are defaults that can be used.

lattice plot examples

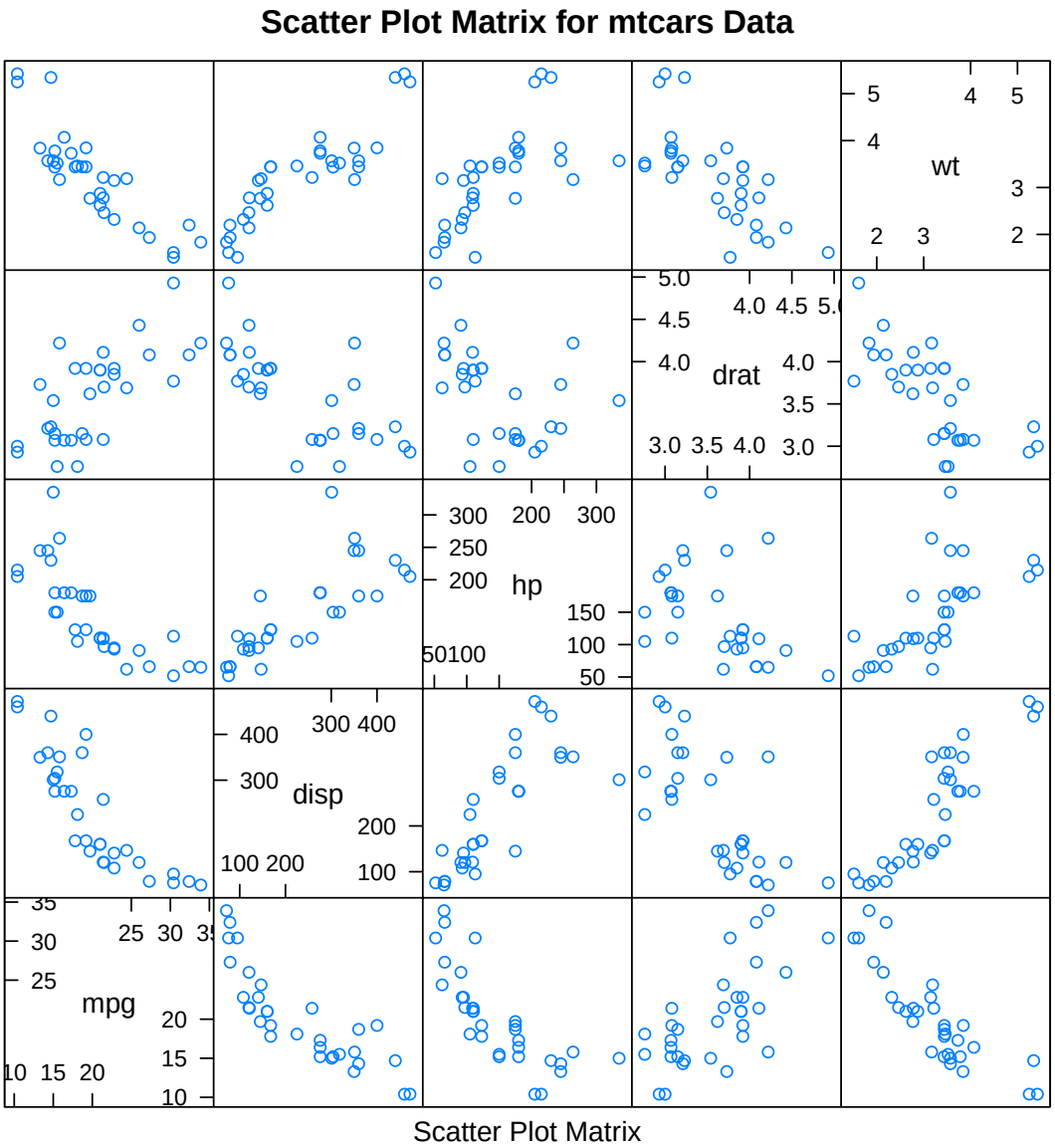
```
library(datasets)
library(lattice)
## Convert 'Month' to a factor variable
airquality <- transform(airquality, Month = factor(Month))
xyplot(Ozone ~ Wind | Month, data = airquality, layout = c(5, 1))
```



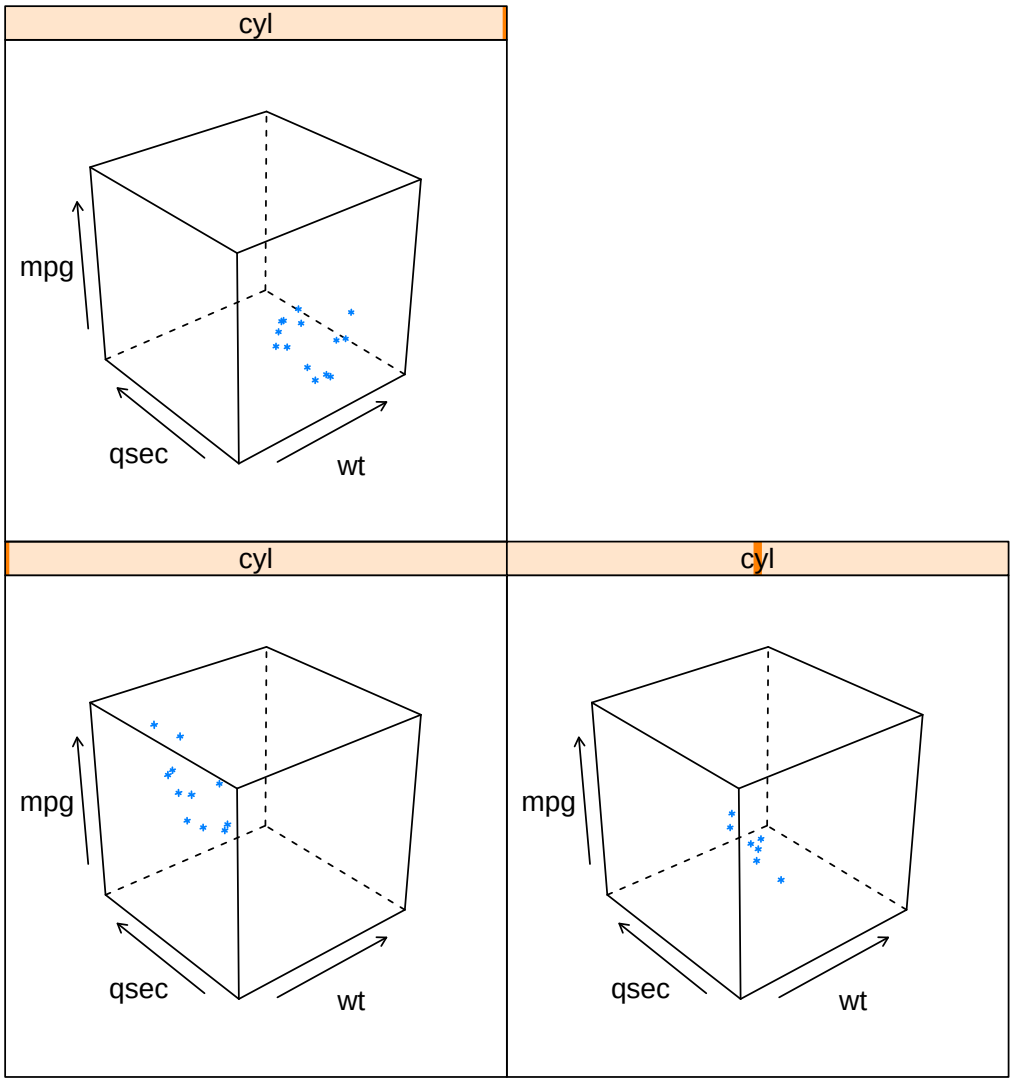
```
densityplot(~mpg | cyl, data=mtcars, main = "Density Plot by Number
    of Cylinders", xlab = "Miles per Gallon")
```



```
splom(mtcars[c(1, 3, 4, 5, 6)], main = "Scatter Plot Matrix for mtcars Data")
```

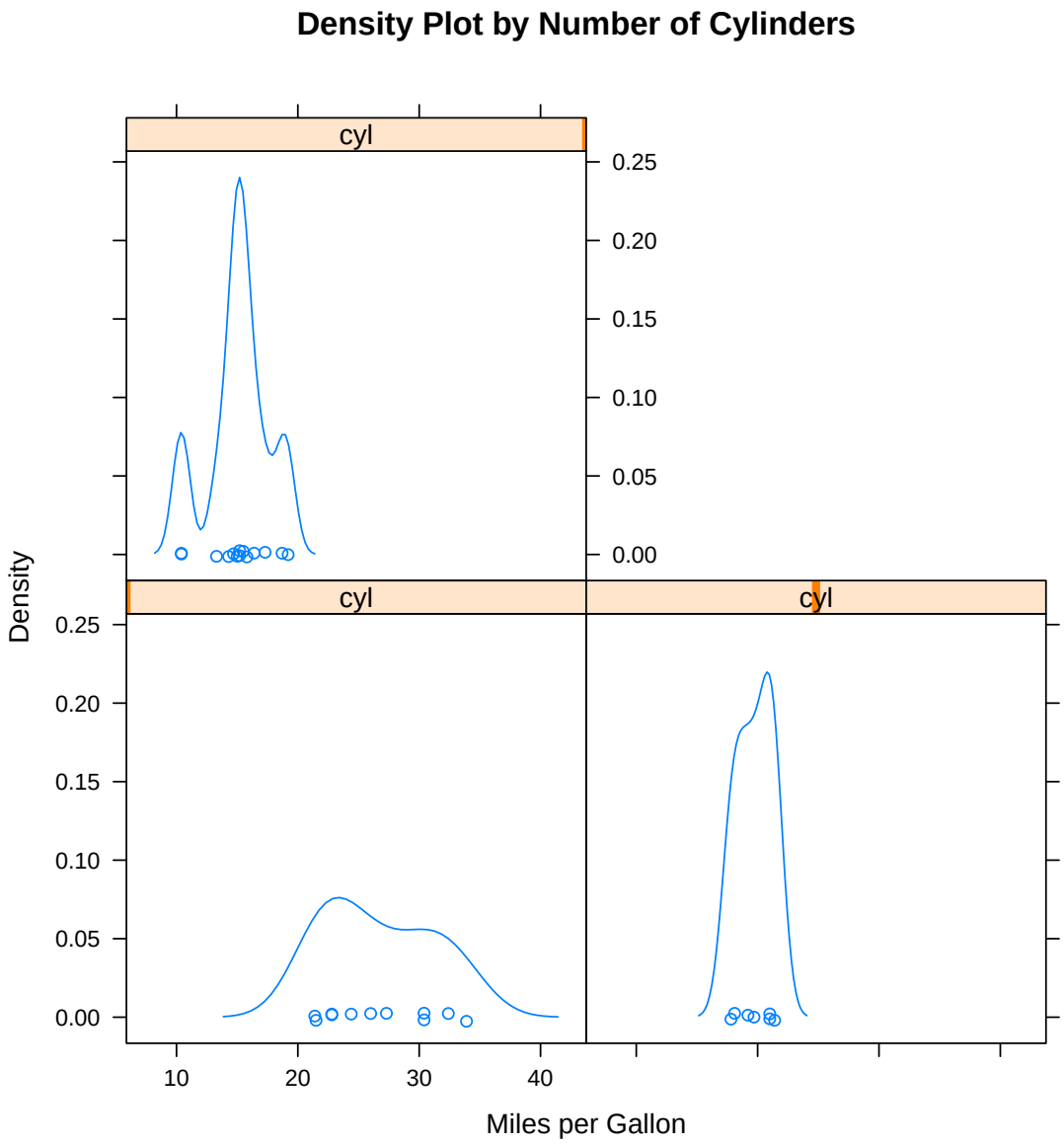


```
cloud(mpg ~ wt * qsec | cyl, data=mtcars, main = "3D Scatter Plots by Cylinders")
```

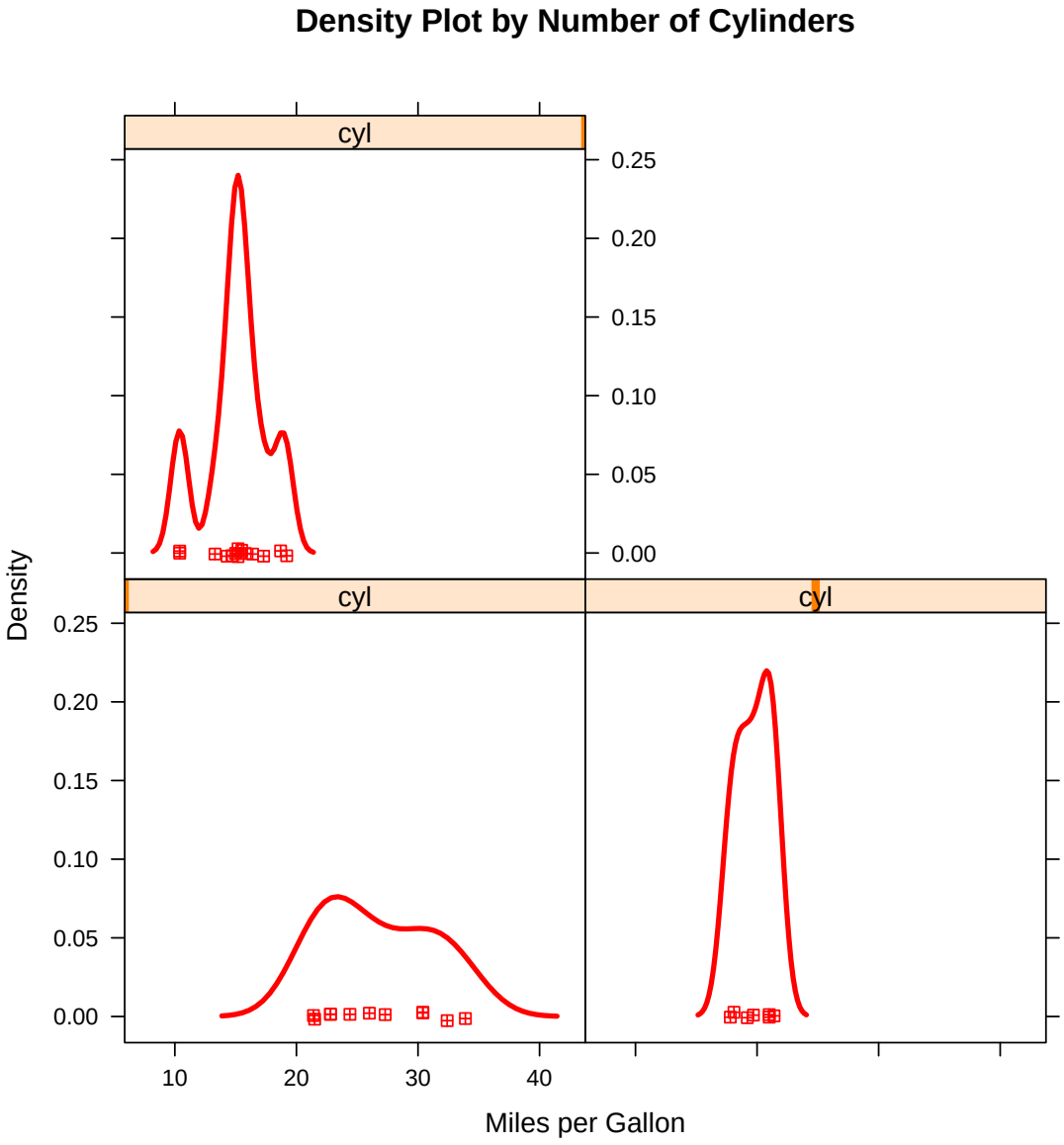


You can also use the update() function to modify a lattice graphic object.

```
library(lattice)
mygraph<-densityplot(~mpg | cyl, data=mtcars,
    main = "Density Plot by Number of Cylinders",
    xlab = "Miles per Gallon")
mygraph
```




```
update(mygraph, col="red", pch=12, cex=0.8, lwd=3)
```



1.2 Conditioning variables

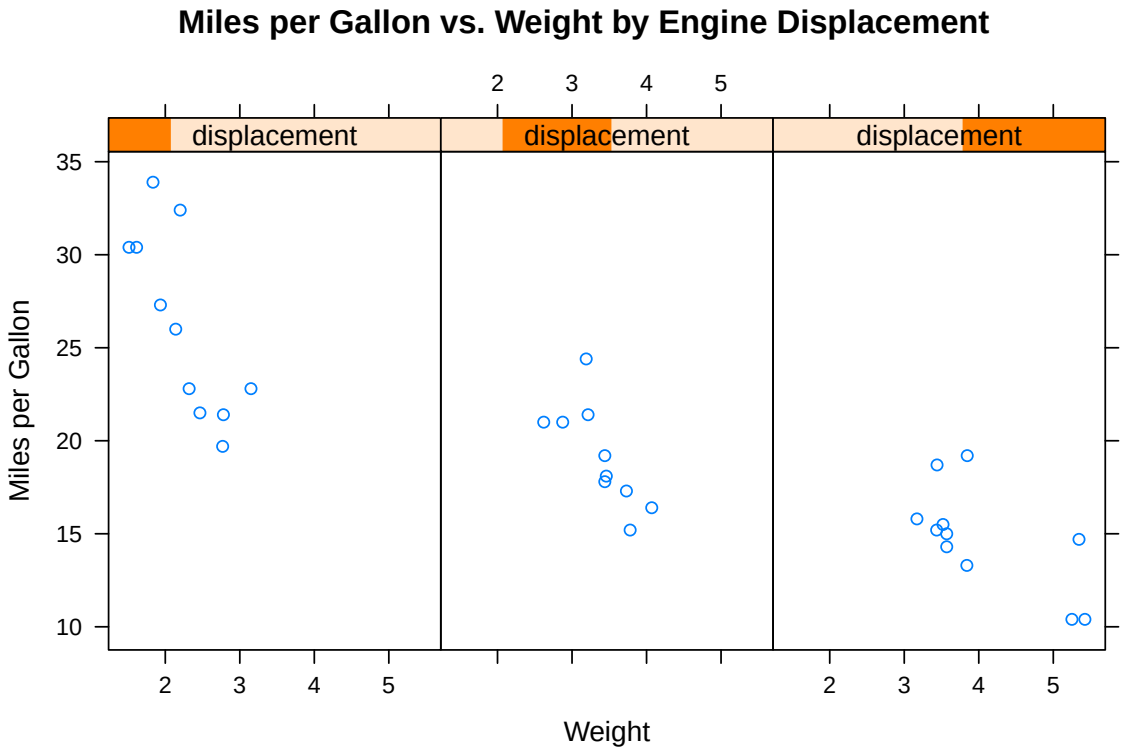
As you've seen, one of the most powerful features of lattice graphs is the ability to add conditioning variables. If one conditioning variable is present, a separate panel is created for each level. If two conditioning variables are present, a separate panel is created for each combination of levels for the two variables. It's rarely useful to include more than two conditioning variables.

Typically, conditioning variables are factors. But what if you want to condition on a continuous variable? One approach would be to transform the continuous variable into a discrete variable using R's cut() function . Alternatively, the lattice package provides functions for transforming a continuous variable into a data structure called a shingle. Specifically, the continuous variable is divided up into a series of (possibly) overlapping ranges.

```
myshingle <- equal.count(x, number=#, overlap=proportion)
```

will take continuous variable x and divide it up into # intervals, with proportion overlap, and equal numbers of observations in each range, and return it as the variable myshingle (of class shingle).

```
library(lattice)
displacement <- equal.count(mtcars$disp, number=3, overlap=0)
xyplot(mpg~wt|displacement, data=mtcars,
    main = "Miles per Gallon vs. Weight by Engine Displacement",
    xlab = "Weight", ylab = "Miles per Gallon",
    layout=c(3, 1), aspect=1.5)
```



Because engine displacement is a continuous variable, it has been converted to three nonoverlapping shingles with equal numbers of observations.

1.3 Graphic parameters

In previously section, we learned how to view and set default graphics parameters using the par() function . Although this works for graphs produced with R's native graphic system, lattice graphs are unaffected by these settings. Instead, the graphic defaults used by lattice functions are contained in a large list object that can be accessed with the trellis.par.get() function and modified through the trellis.par.set() function . The show.settings() function can be used to display the current graphic settings visually.

```
show.settings()
mysettings <- trellis.par.get()
mysettings$superpose.symbol
mysettings$superpose.symbol$pch <- c(1:15)
trellis.par.set(mysettings)
show.settings()
```

To learn more about lattice graphs, take a look the excellent text by Sarkar (2008) and its supporting website at http://lmdvr.r-forge.r-project.org. The Trellis Graphics User's Manual (http://cm.bell-labs.com/cm/ms/departments/sia/doc/trellis.user.pdf) is also an excellent source of information.

2 ggplot2

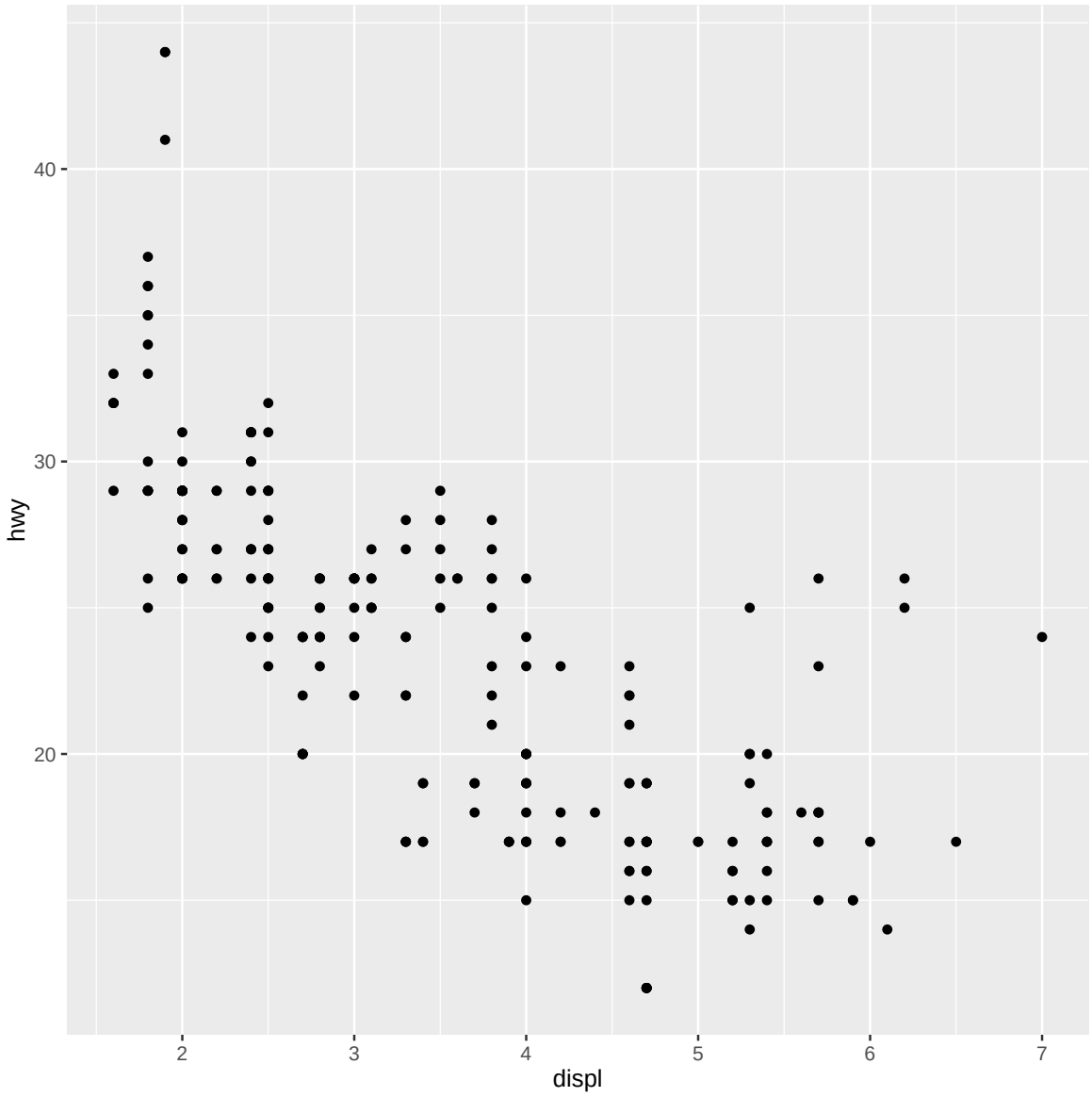
The ggplot2 package implements a system for creating graphics in R based on a comprehensive and coherent grammar. This provides a consistency to graph creation often lacking in R, and allows the user to create graph types that are innovative and novel.

2.1 qplot

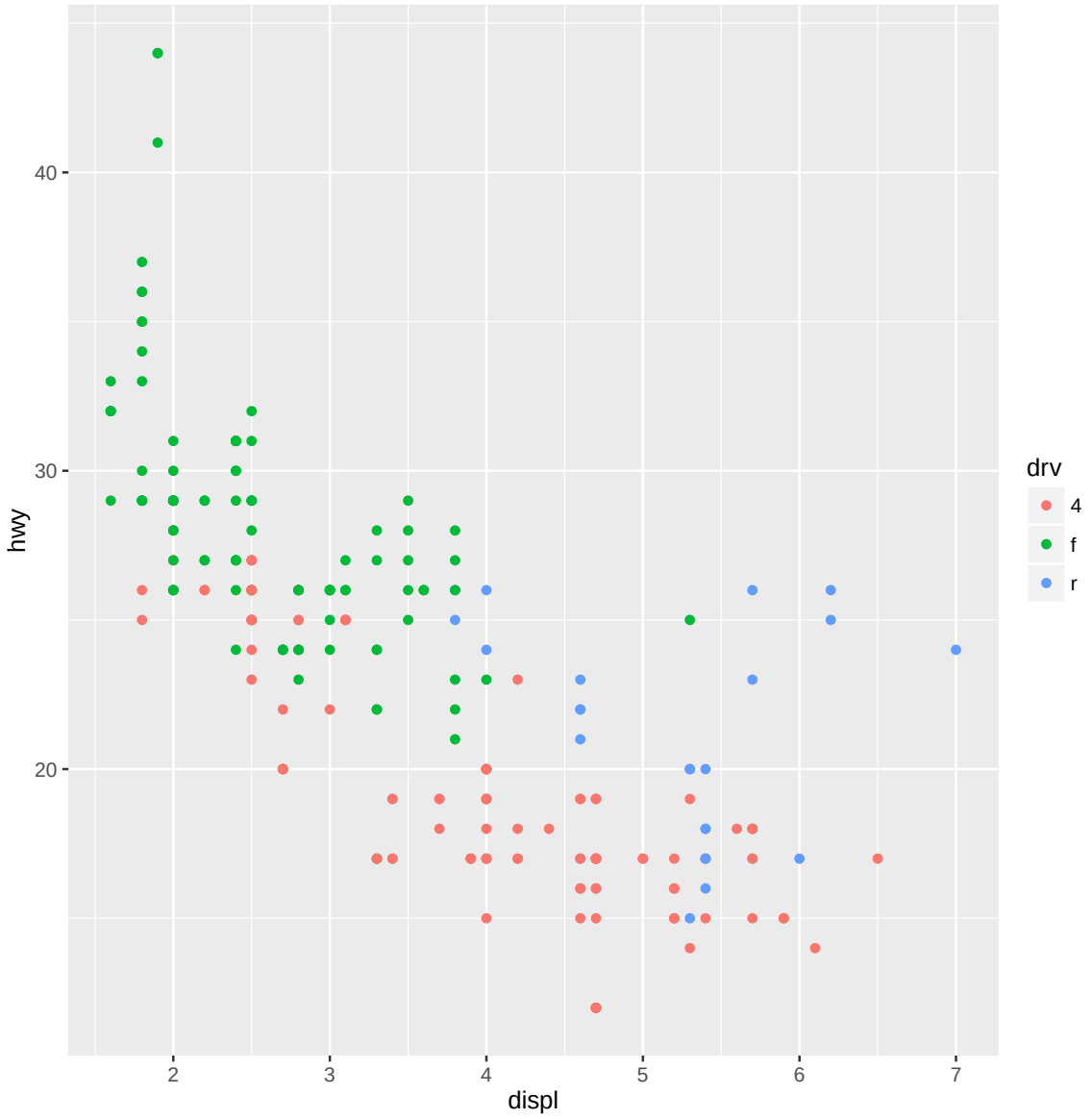
An example

The simplest approach for creating graphs in ggplot2 is through the qplot() or quick plot function.

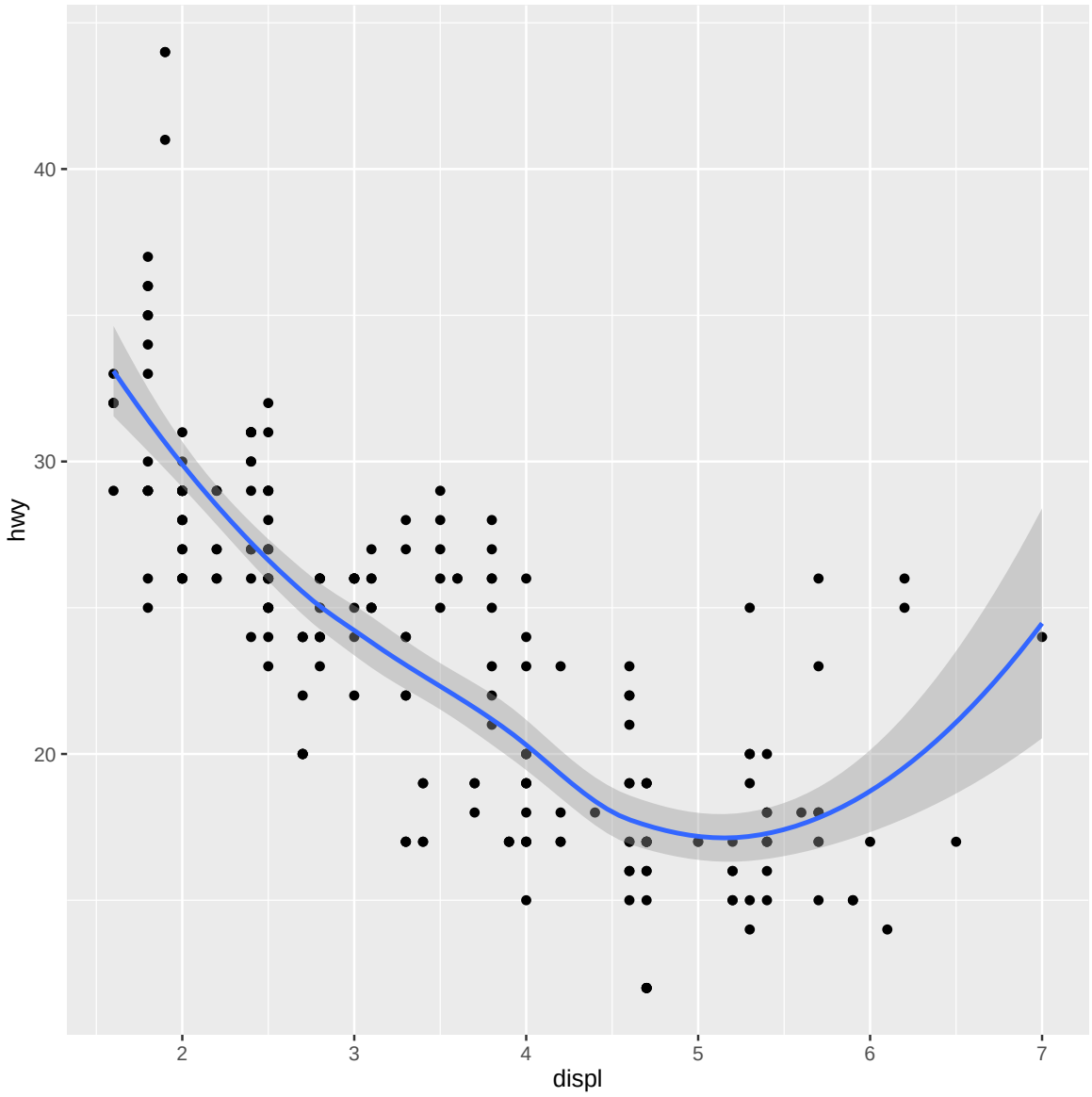
```
library("ggplot2")

## Warning:  package 'ggplot2' was built under R version 3.2.4

str(mpg)

## Classes 'tbl_df', 'tbl' and 'data.frame': 234 obs. of  11 variables:
##  $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
##  $ model       : chr  "a4" "a4" "a4" "a4" ...
##  $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr  "f" "f" "f" "f" ...
##  $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr  "p" "p" "p" "p" ...
##  $ class       : chr  "compact" "compact" "compact" "compact" ...

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




```
qplot(displ,hwy,data=mpg,color=drv)
```

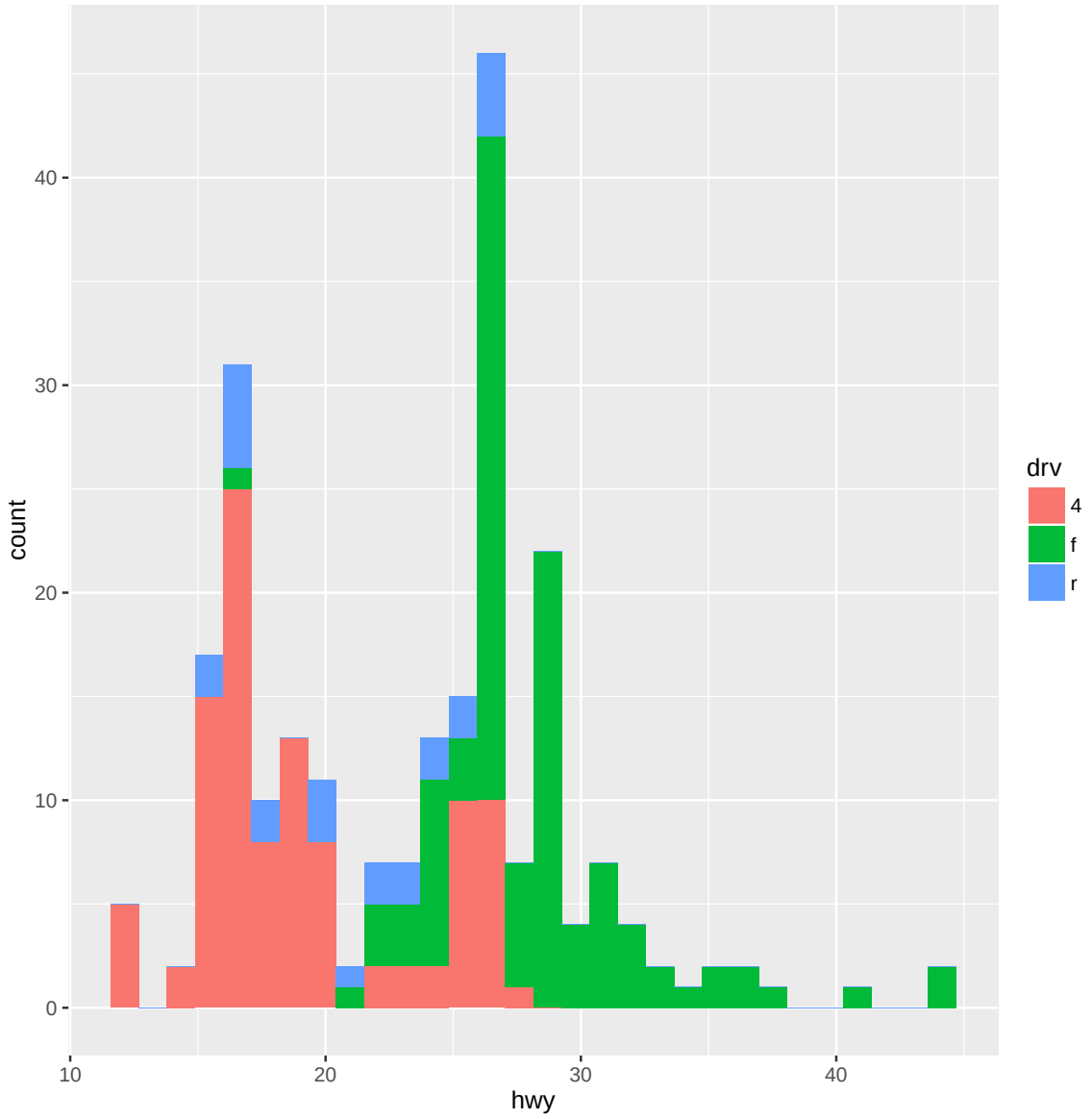



```
qplot(displ, hwy, data = mpg, geom = c("point", "smooth"))
```

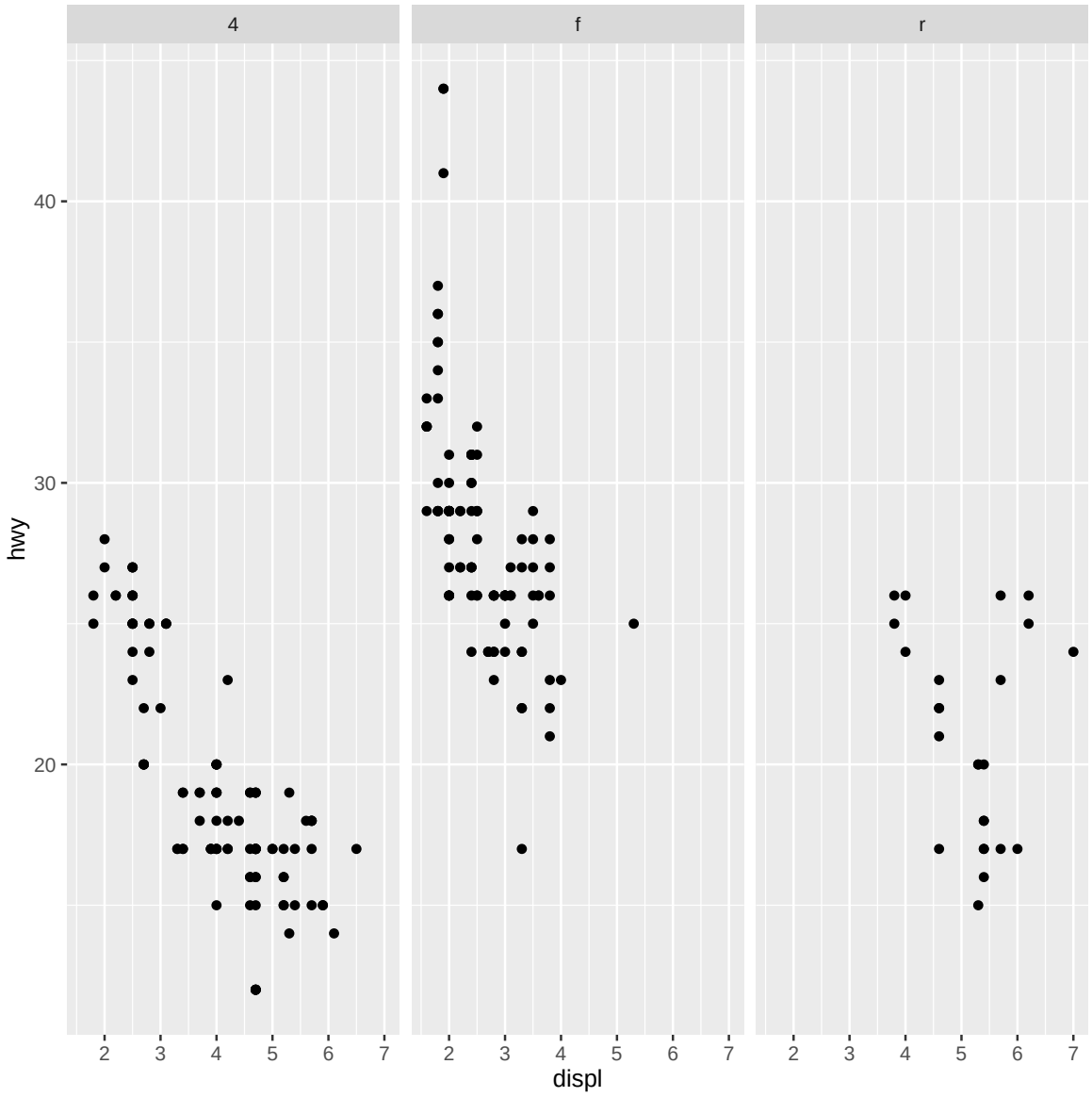



```
qplot(hwy, data = mpg, fill = drv)
```

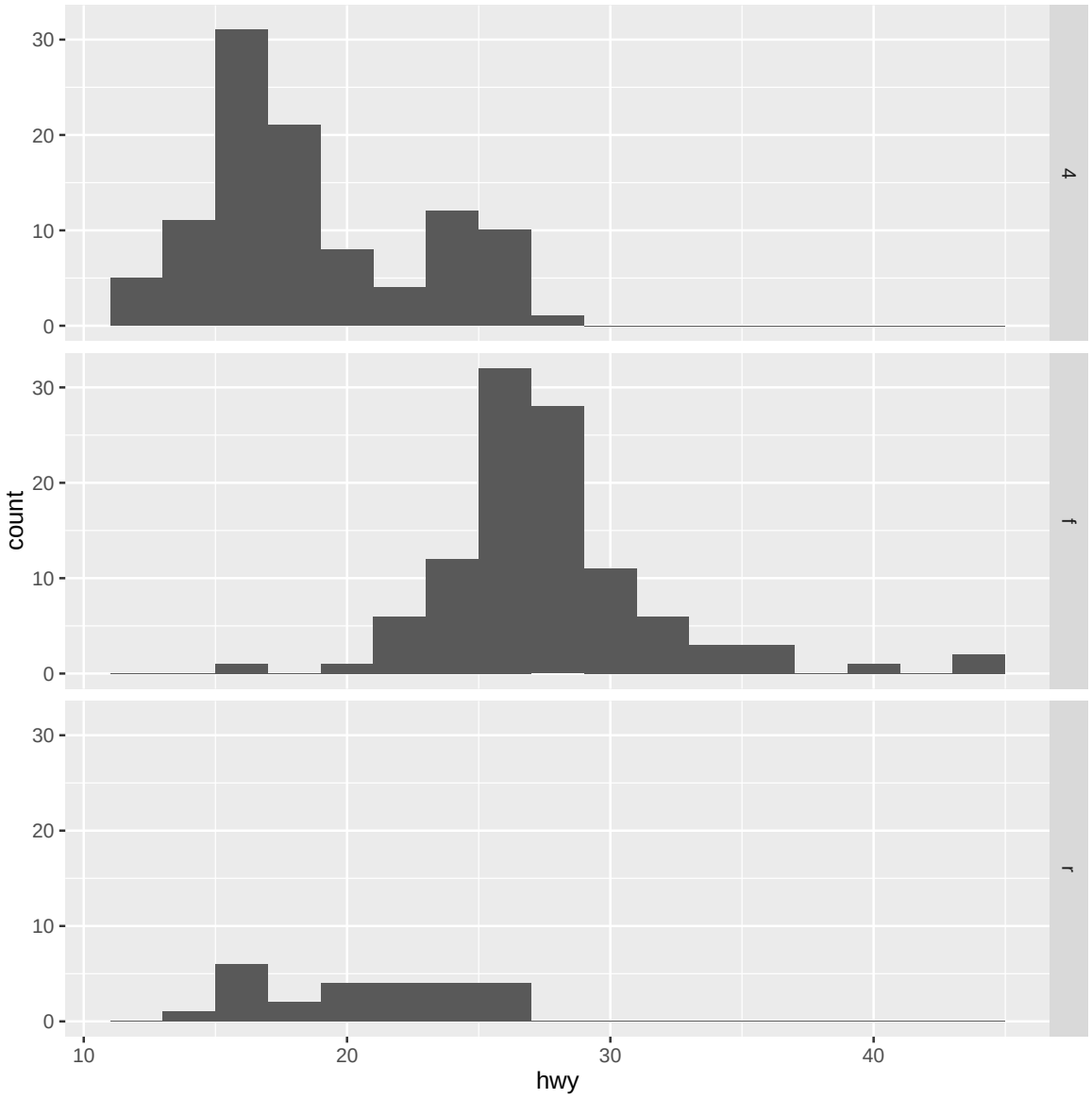
```
## `stat_bin()` using `bins = 30`.  Pick better value with `binwidth`.
```



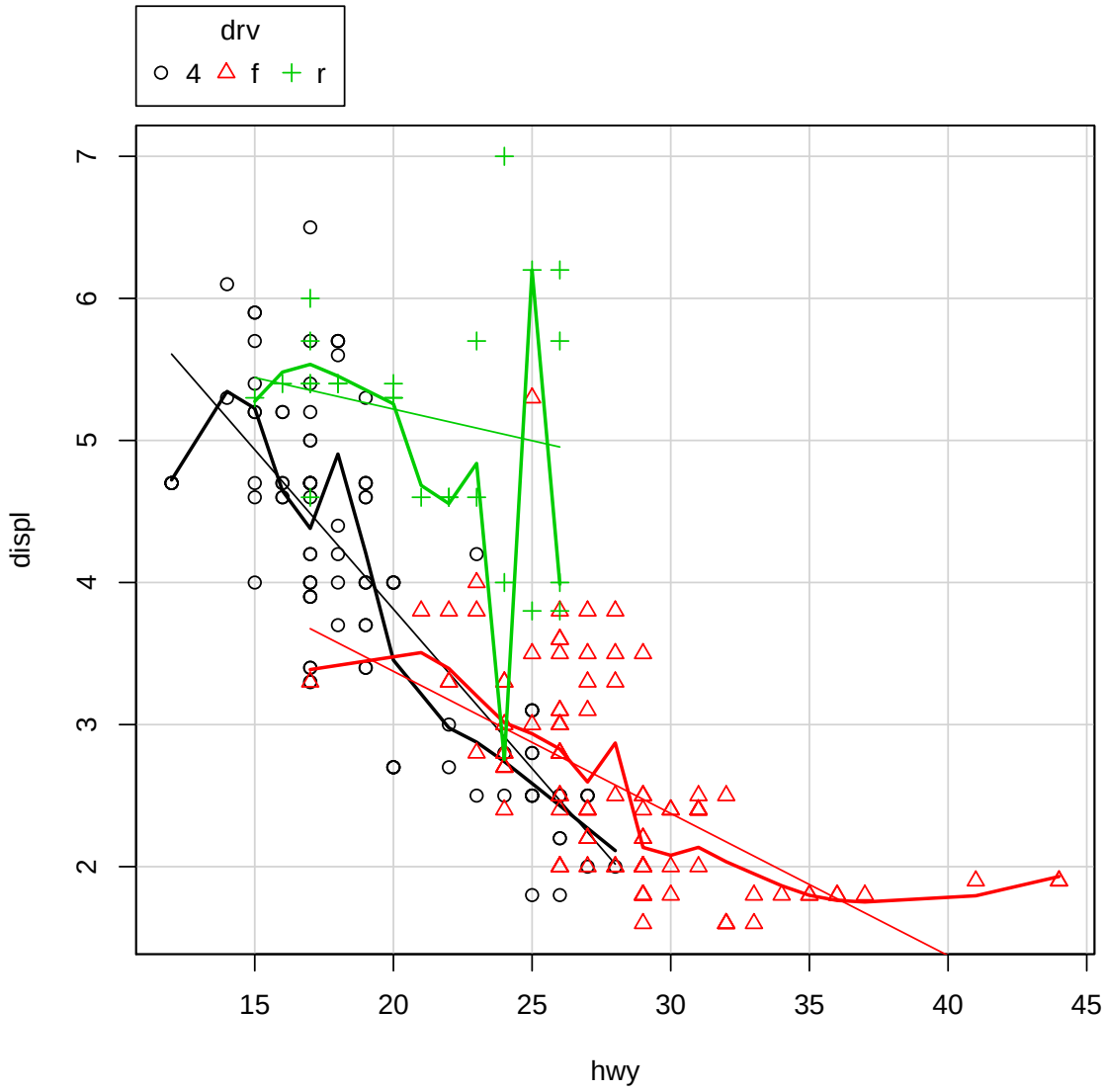

```
qplot(displ, hwy, data = mpg, facets = . ~ drv)
```



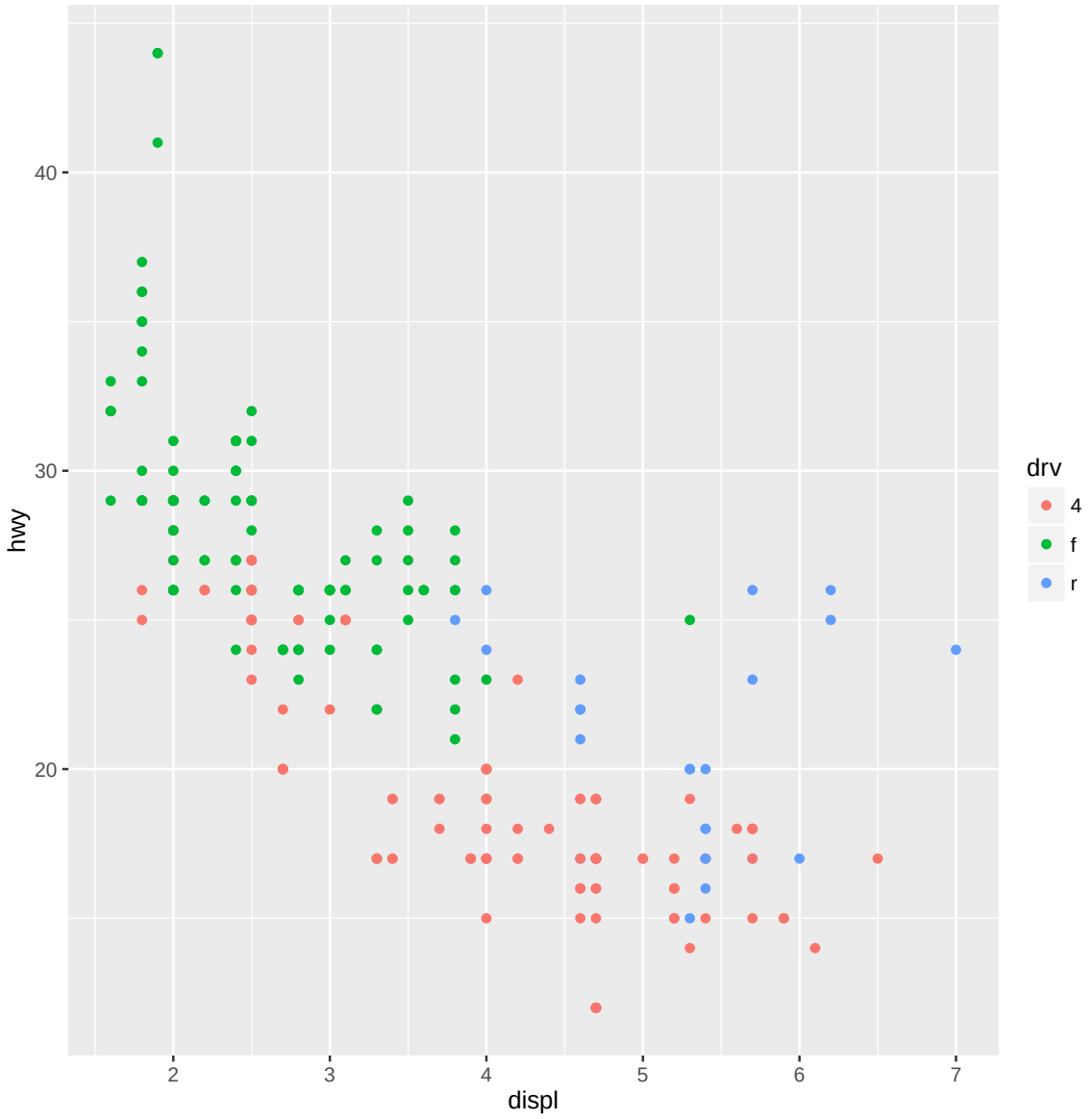

```
qplot(hwy, data = mpg, facets = drv ~ ., binwidth = 2)
```



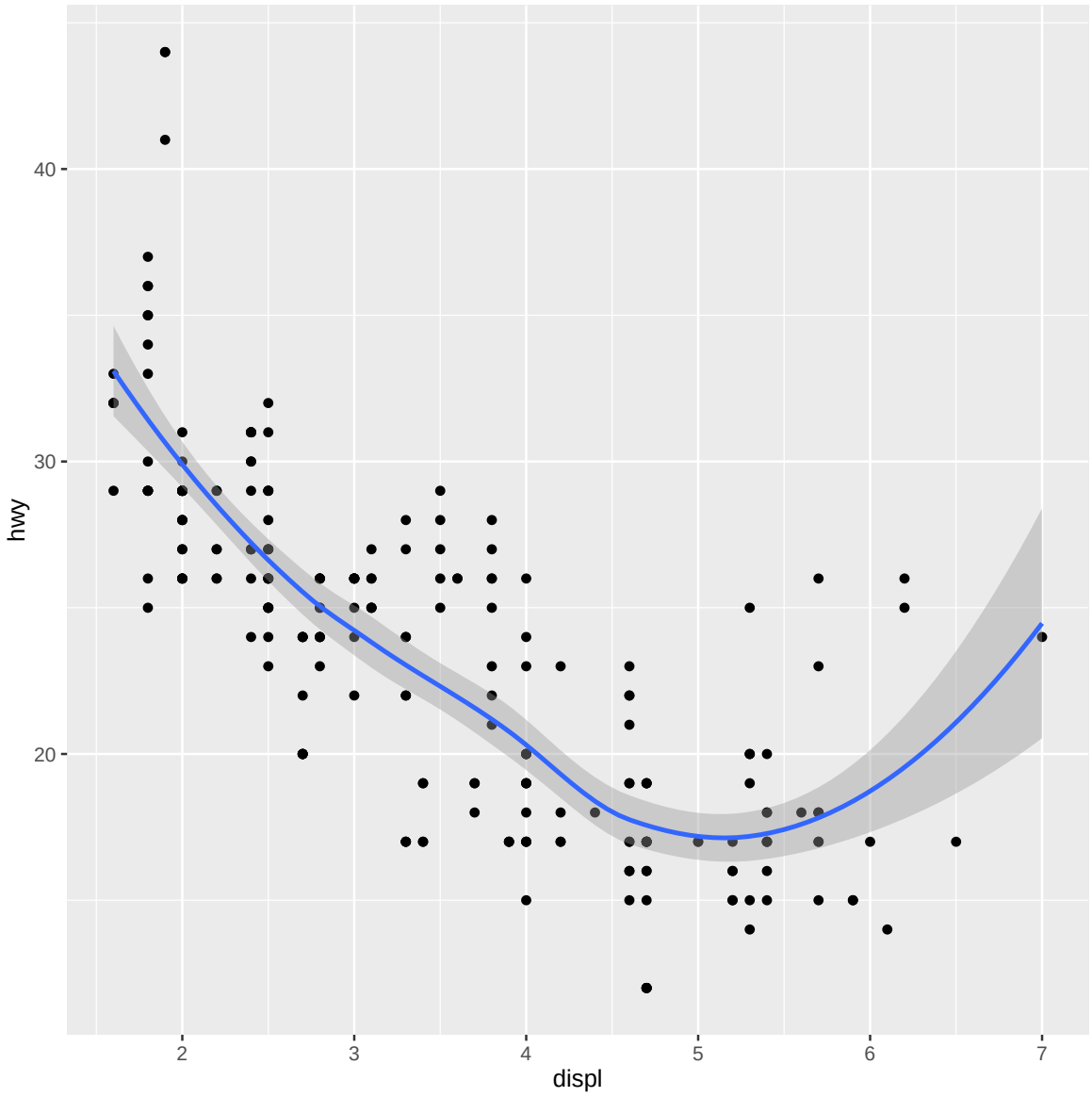

```
library(car)
scatterplot(displ~hwy|drv,data=mpg)
```



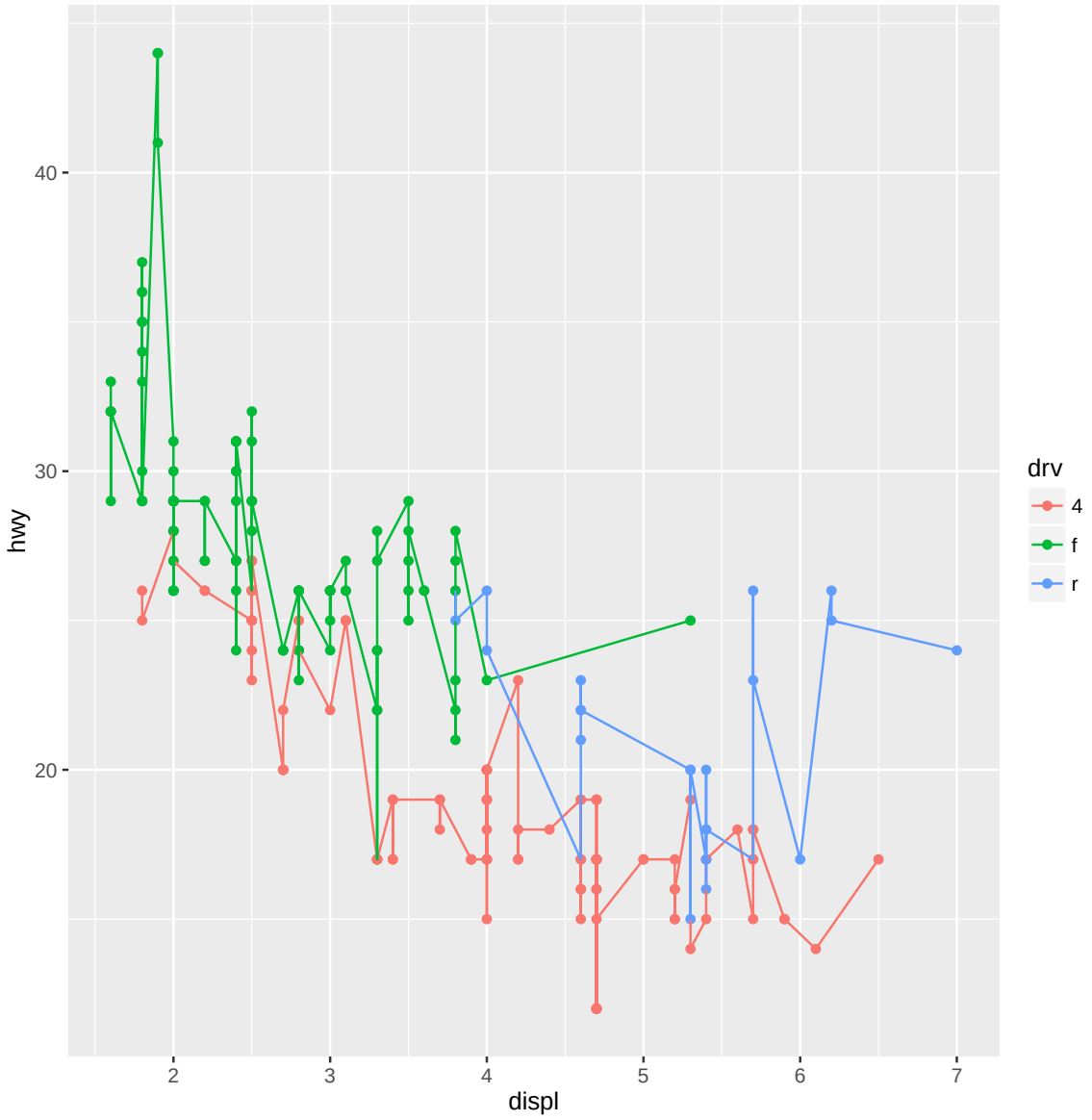

```
qplot(displ,hwy,data=mpg,color=drv)
```



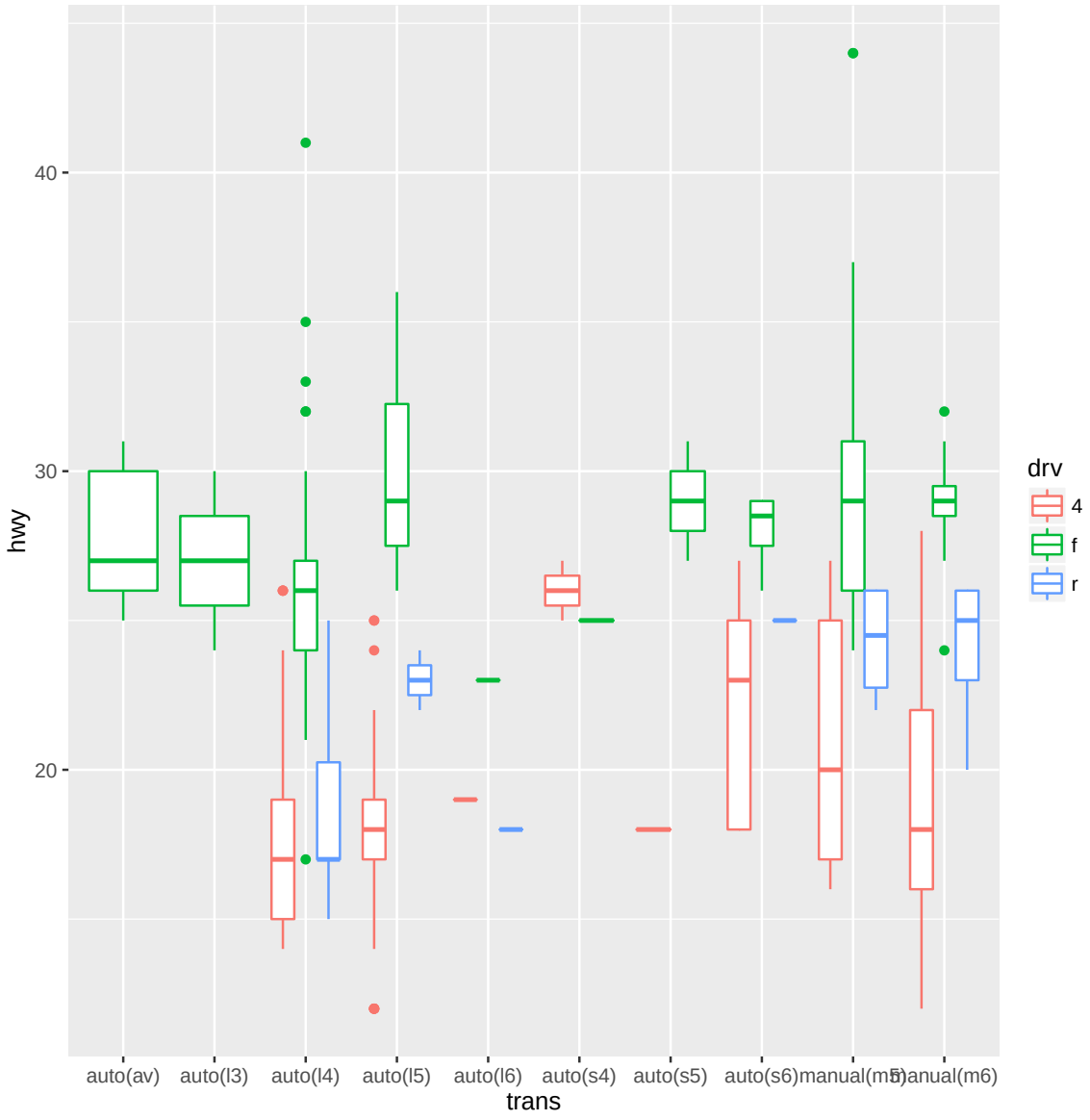

```
###point+line
qplot(displ, hwy, data = mpg, geom = c("point", "smooth"))
```



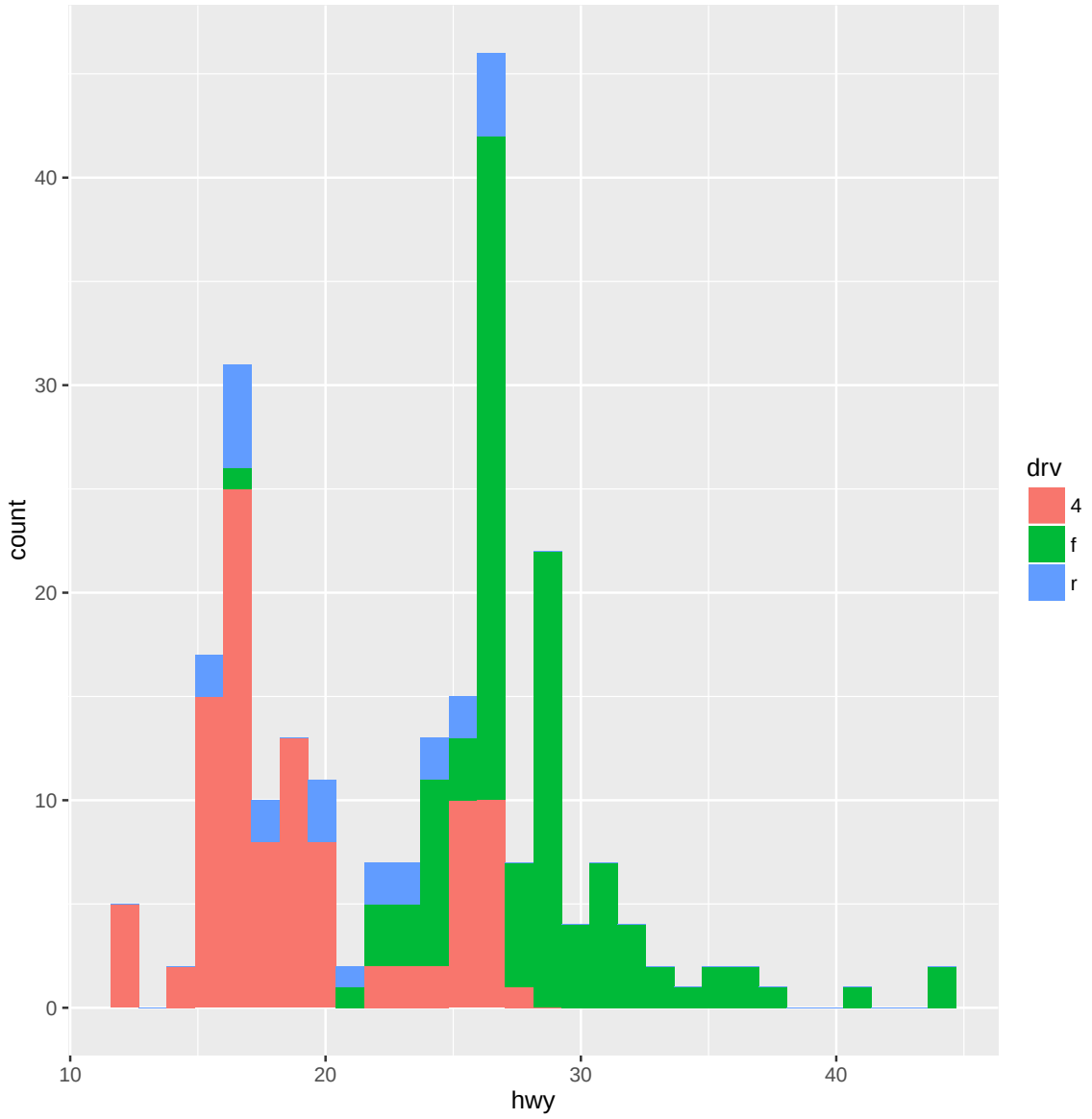

```
qplot(displ, hwy, data = mpg, color= drv, geom = c("point", "line"))
```



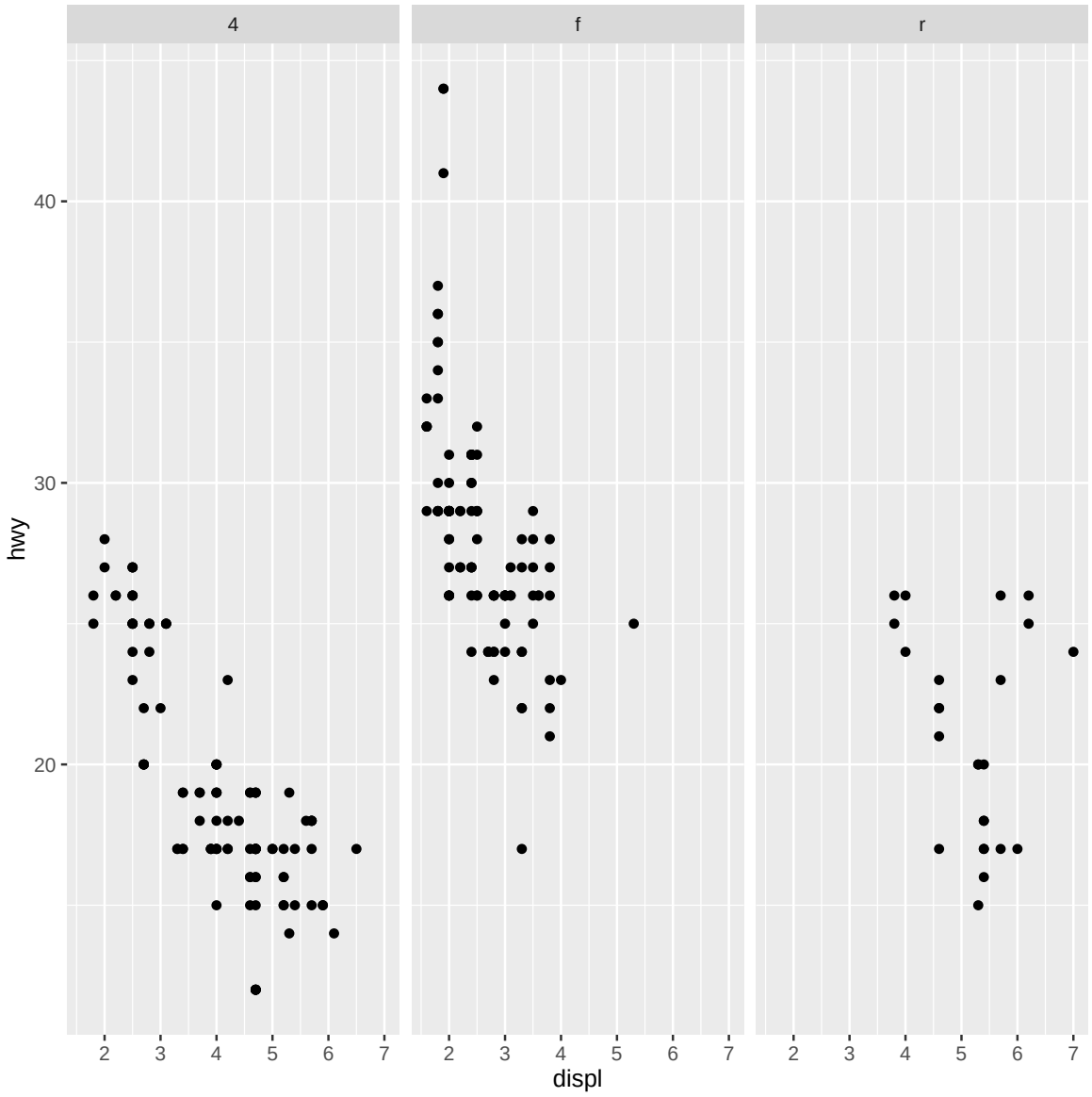

```
#boxplot
qplot(trans, hwy, data = mpg, color= drv, geom = "boxplot")
```



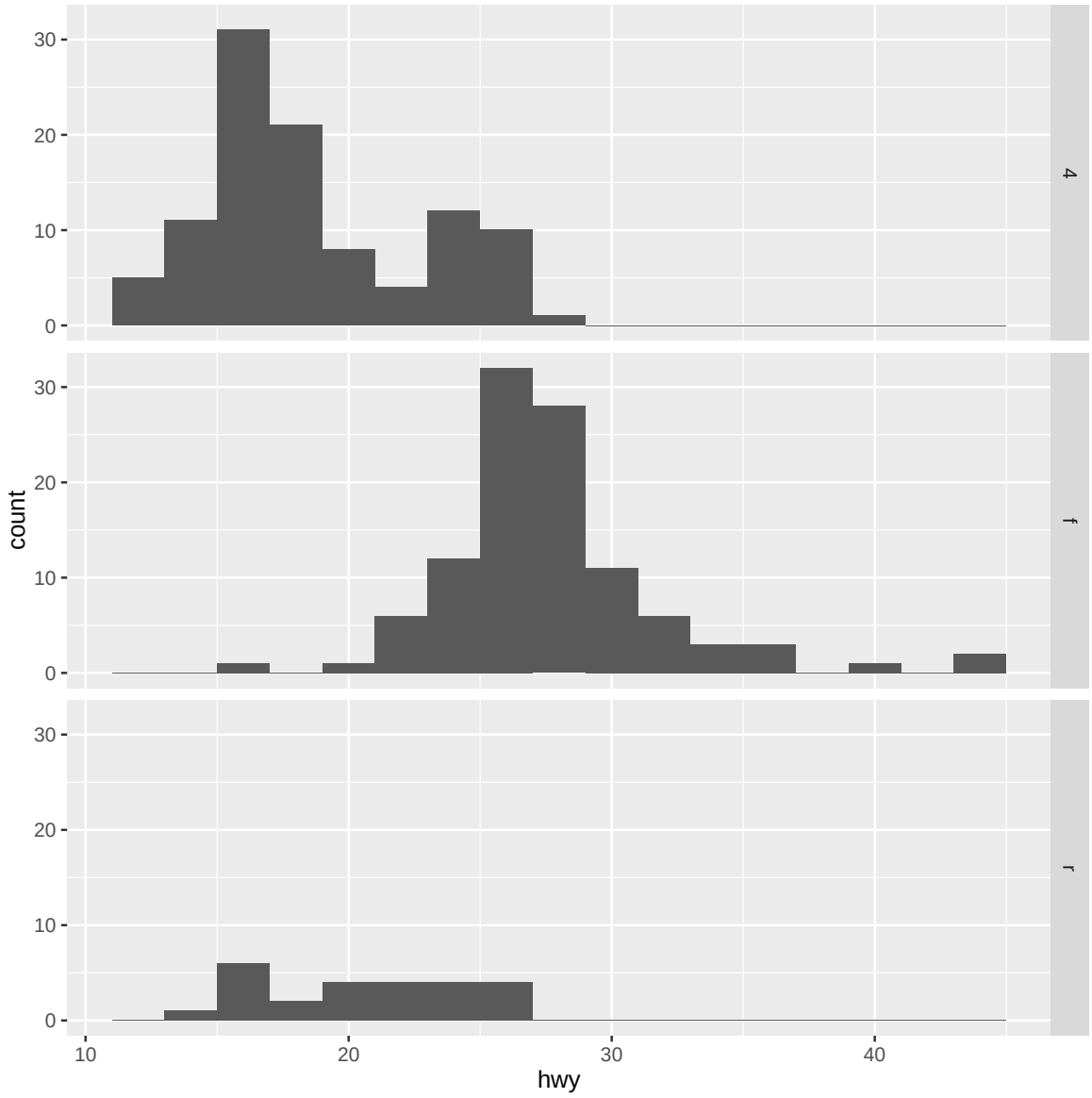

```
#Histogram
qplot(hwy, data = mpg, fill= drv, geom = "histogram")

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




```
#qplot(hwy, data = mpg, fill = drv)
#Density qplot(hwy, data = mpg, color= drv, geom = "density")
#multi graph
qplot(displ, hwy, data = mpg, facets = . ~ drv)
```




```
qplot(hwy, data = mpg, facets = drv ~ ., binwidth = 2)
```




```
# save image of plot on disk
#ggsave(file="test.pdf")
#ggsave(file="test.jpeg", dpi=72)
#ggsave(file="test.svg", plot=p.tmp, width=10, height=5)
```

2.2 ggplot

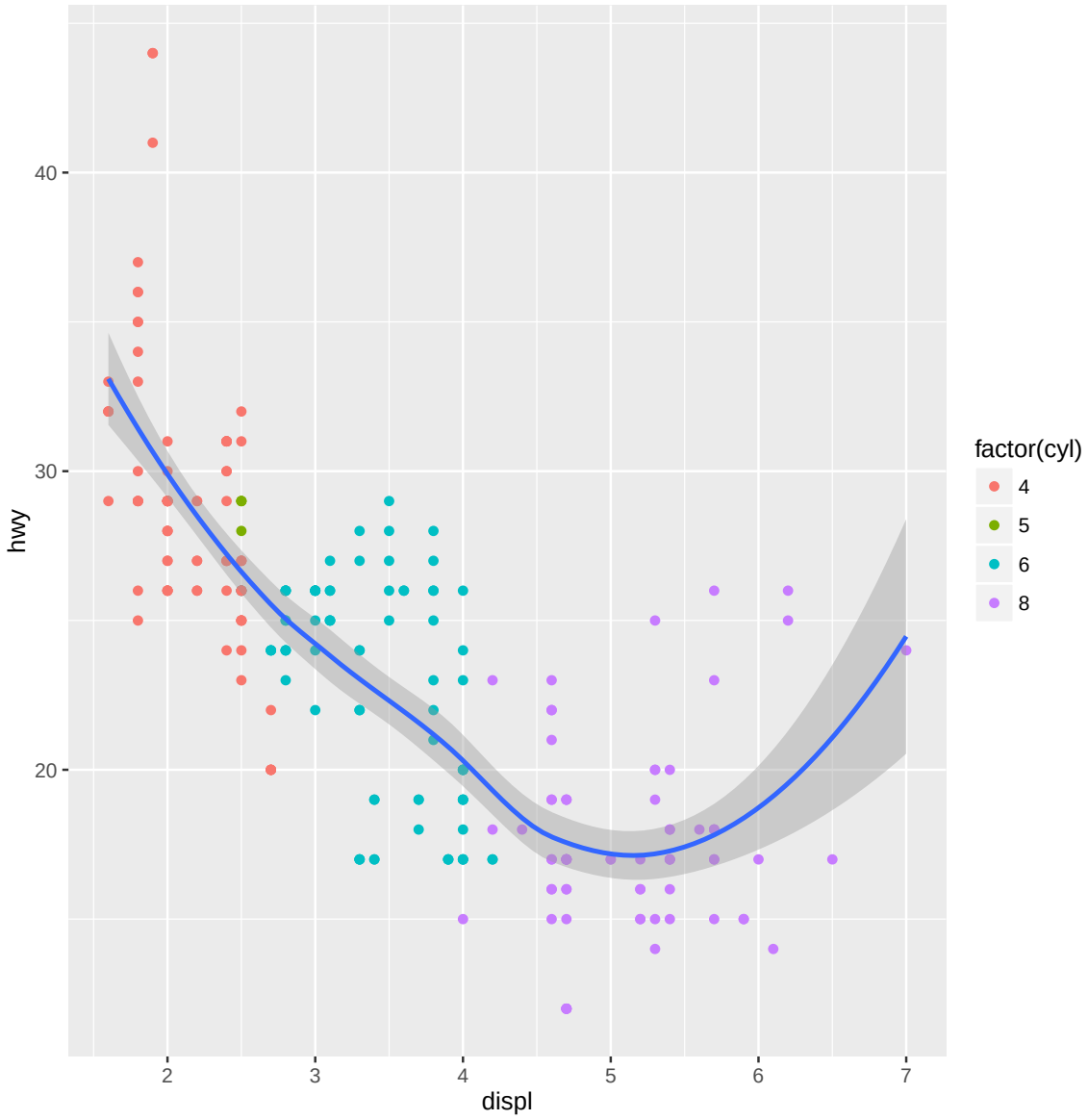
Compare qplot and ggplot

```
library("ggplot2")
### comparison qplot vs ggplot
# qplot histogram
```

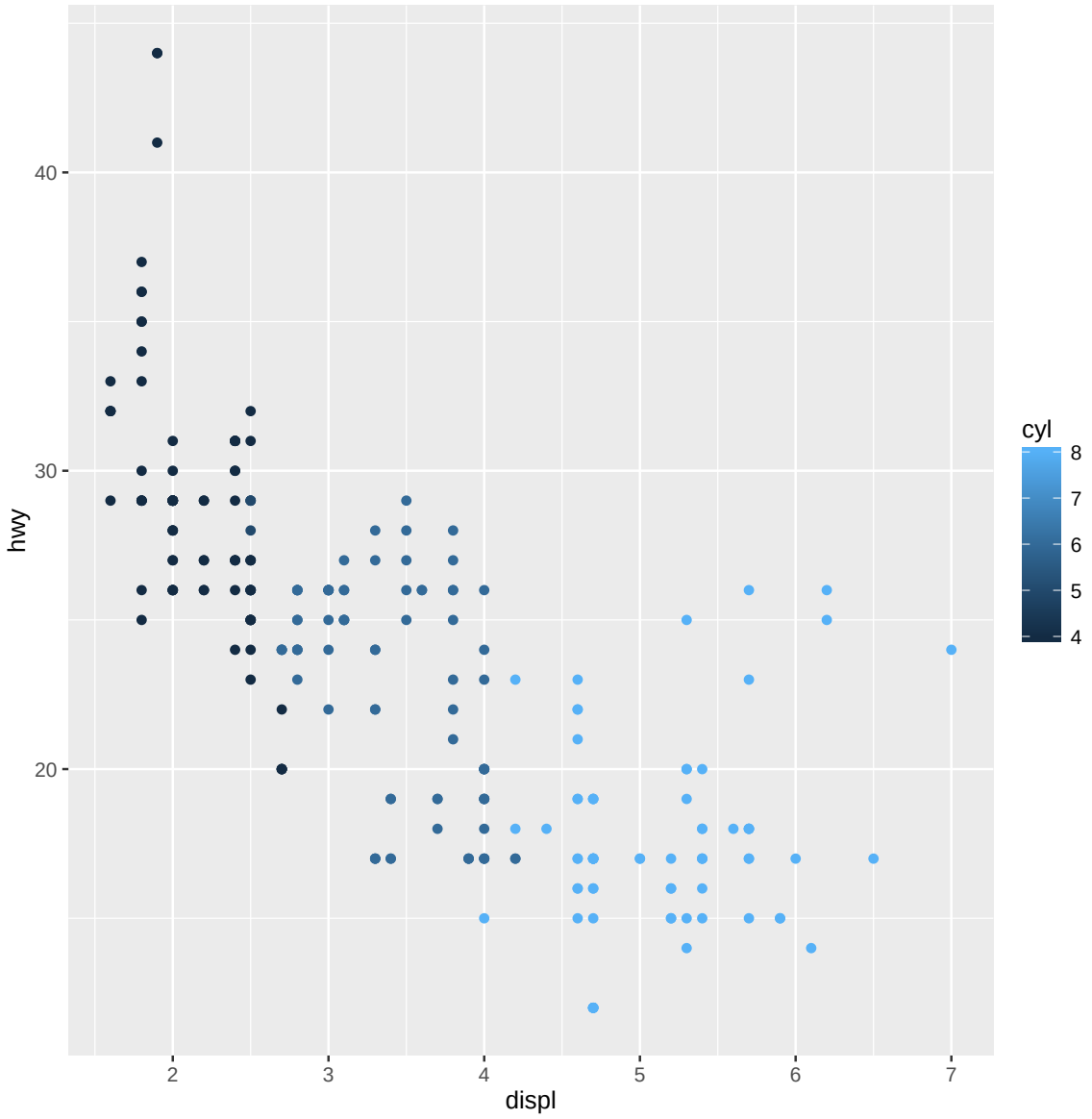
```
qplot(clarity, data=diamonds, fill=cut, geom="bar")
# ggplothistogram -> same output
ggplot(diamonds, aes(clarity, fill=cut)) + geom_bar()
```

```
library(ggplot2)
#1
p <- ggplot(data=mpg,aes(x=displ,y=hwy,colour=factor(cyl)))
p + geom_point() + geom_smooth()
```


```
#2
p <- ggplot(mpg,aes(x=displ,y=hwy))
p + geom_point(aes(colour=factor(cyl))) + geom_smooth()
```

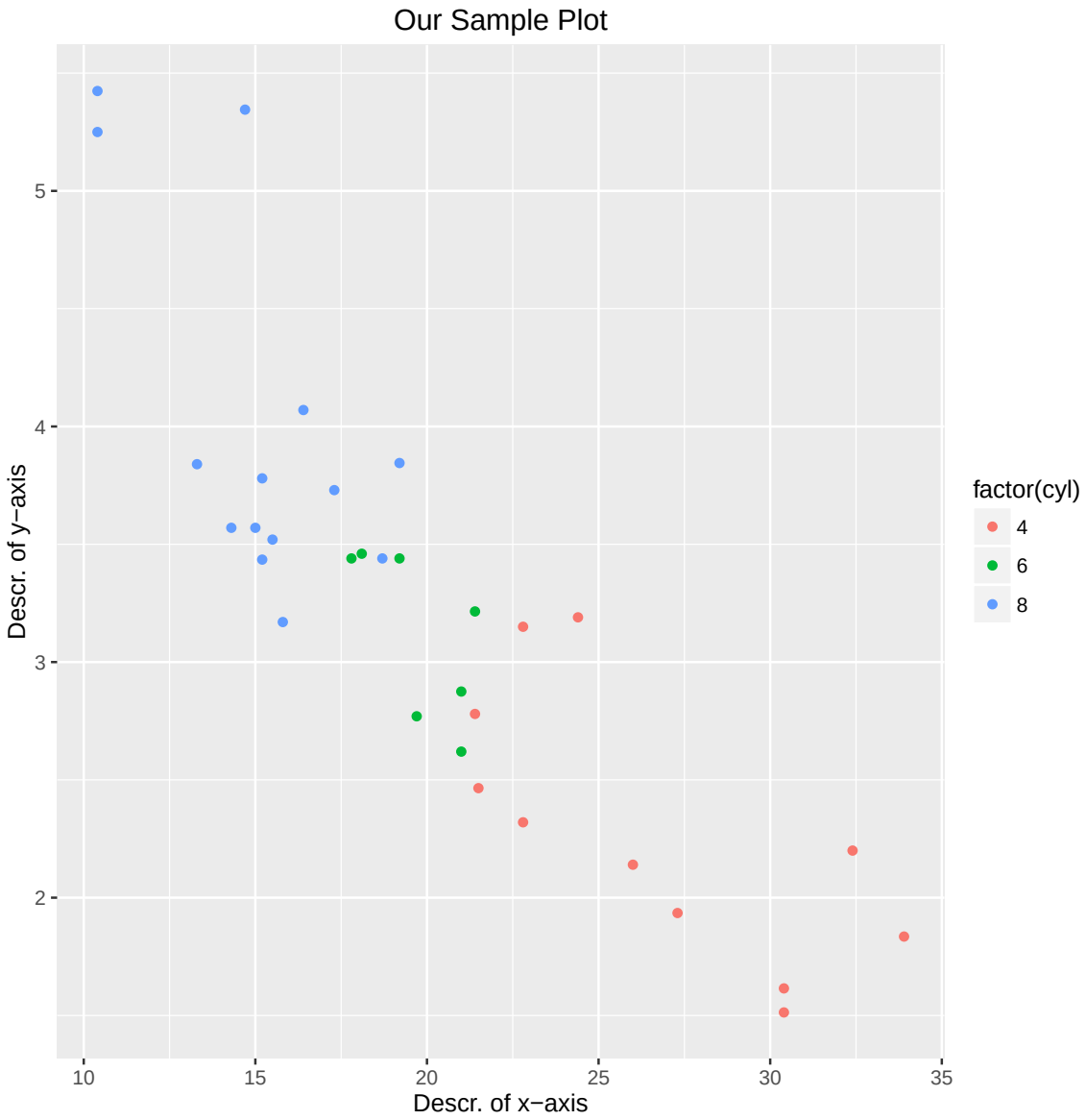



```
#3
p <- ggplot(mpg, aes(x=displ,y=hwy))
p + geom_point(aes(colour = cyl))
```

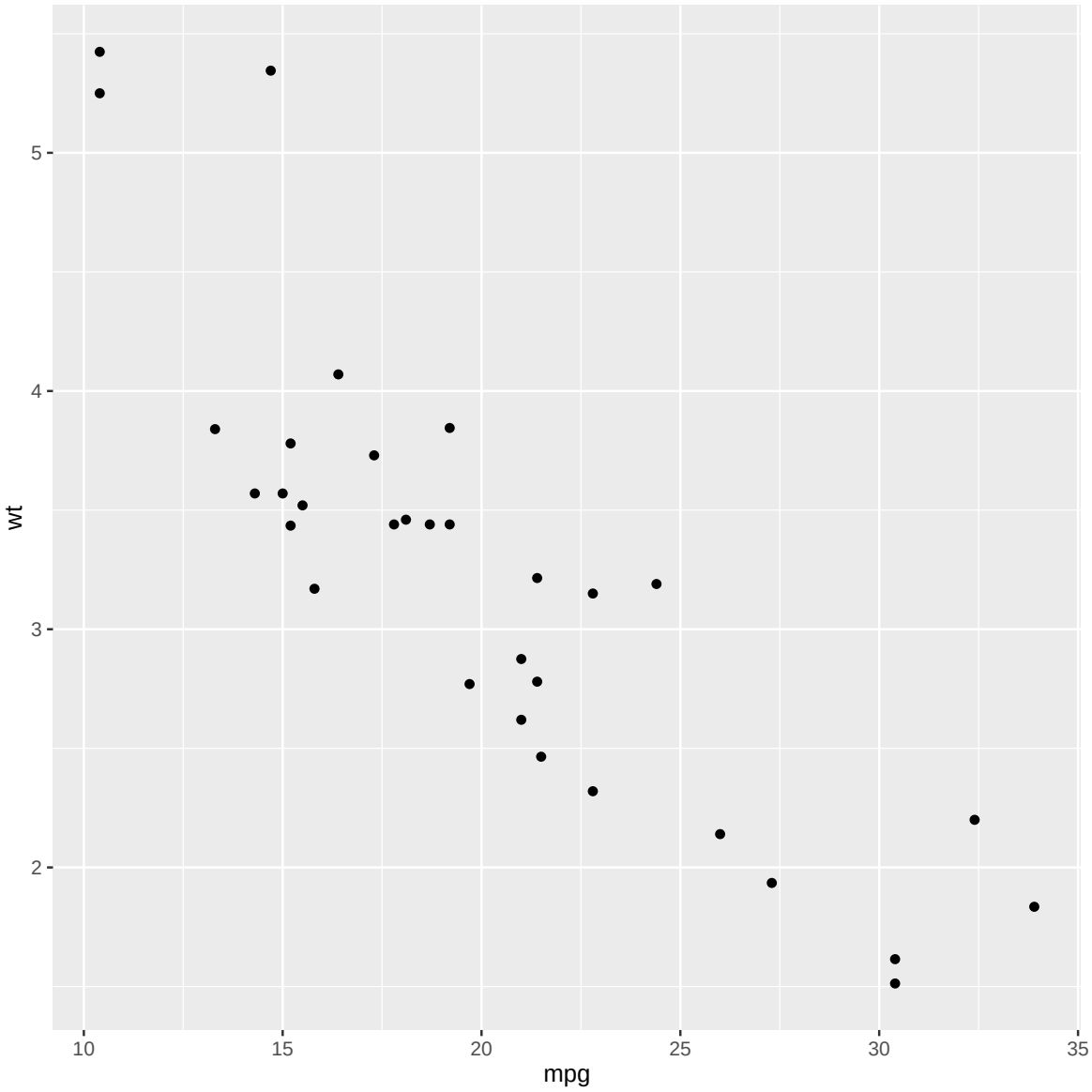


2.3 Other features

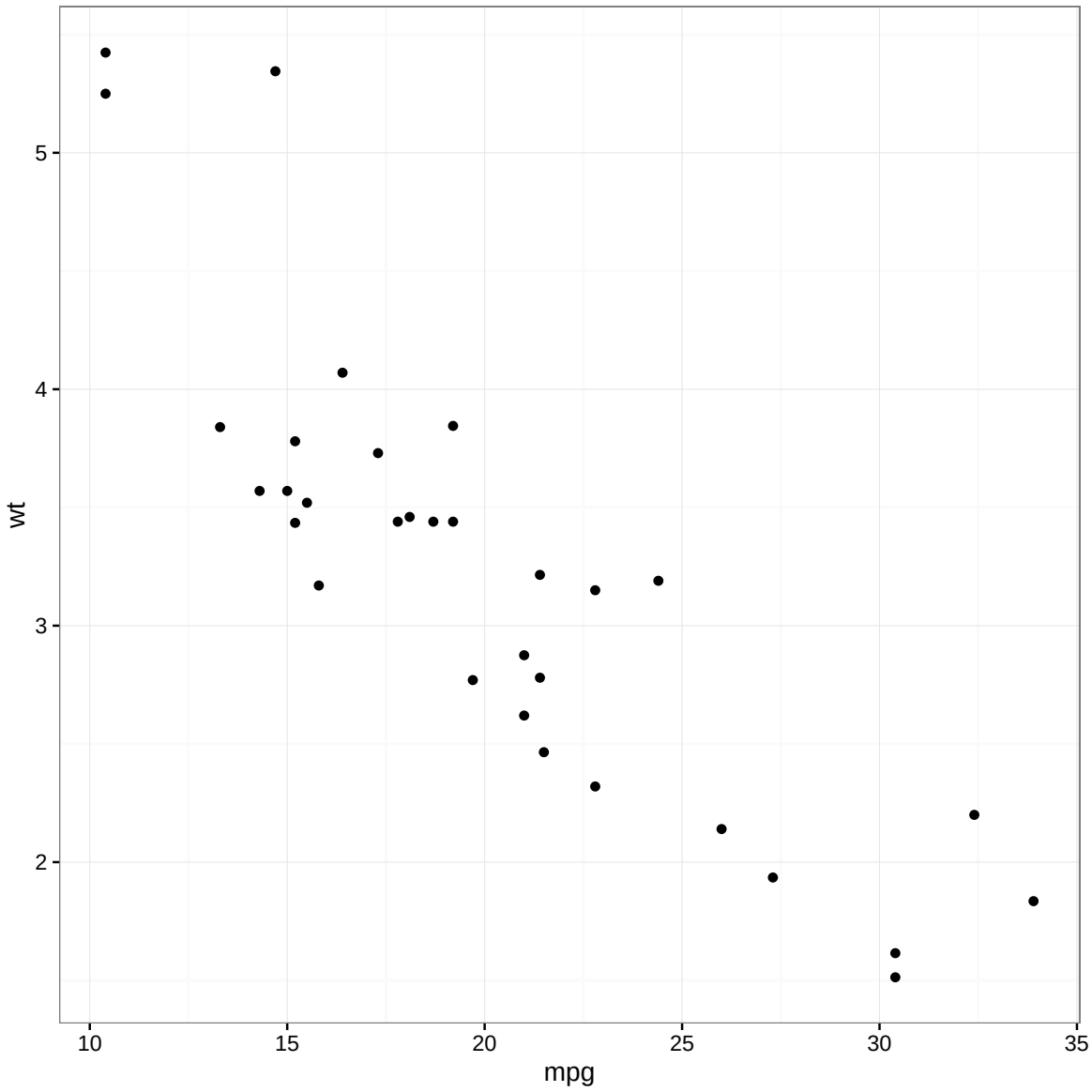
```
# changing text (directly in qplot / additional shortcut)
qplot(mpg, wt, data=mtcars, colour=factor(cyl), geom="point", xlab="Descr. of x-axis", ylab="Descr. o
```



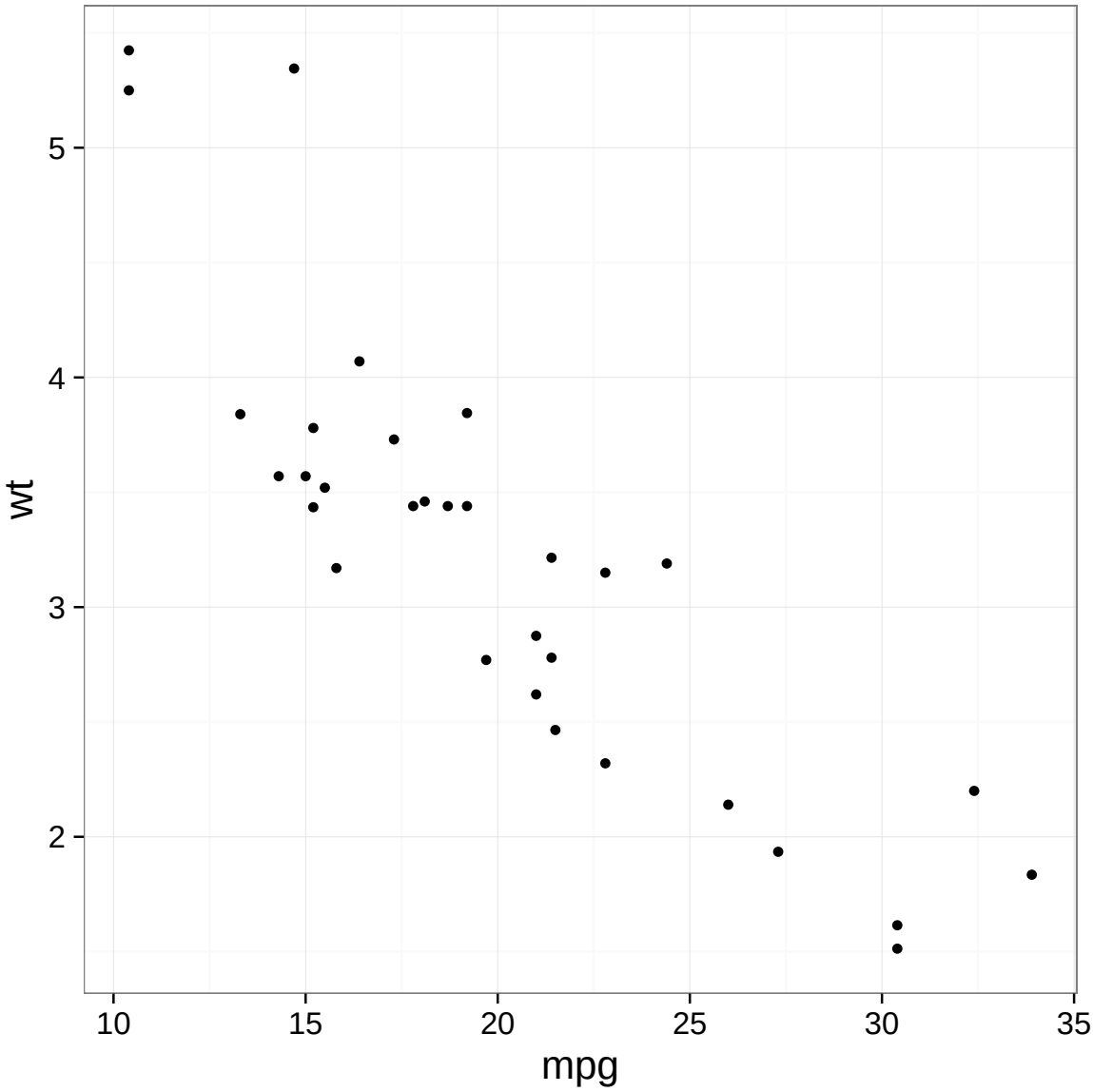

```
# use theme for plot only
qplot(mpg, wt, data=mtcars, geom="point")
```




```
qplot(mpg, wt, data=mtcars, geom="point") + theme_bw()
```




```
# change font-size for all labels (change base_size)
qplot(mpg, wt, data=mtcars, geom="point") + theme_bw(18)
```


wt
5
4
3
2
10
15
20
25
30
35
mpg