

# Chapter 5

## **PLOTTING SYSTEM**

---

DATA ANALYSIS IN R



# Outline

- The Base Plotting System
- The Lattice System
- The ggplot2 System
- Graphic Device
- Copying Plot

# Plotting System

Three systems in R:

- The base plotting system
- The lattice system
- The ggplot2 system

# The Base Plotting System

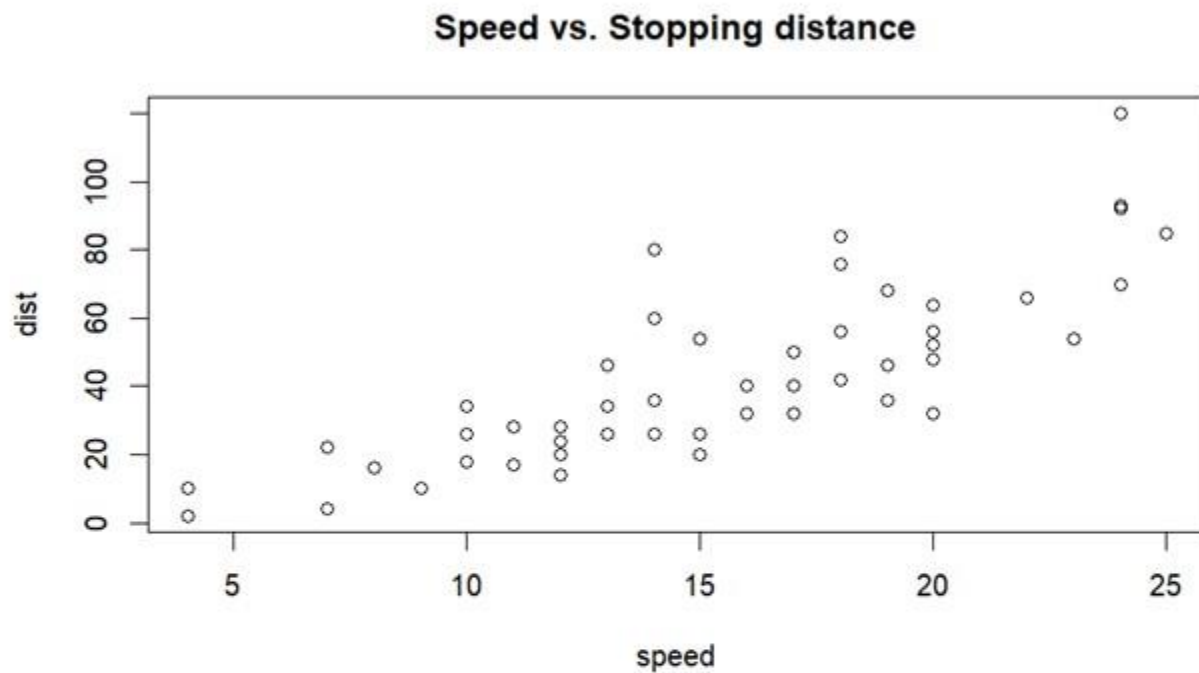
- "Artist's palette" model
- Start with blank canvas and build up from there
- Start with plot function (or similar)
- Use annotation functions to add/modify (text, lines, points, axis)
- In more R-specific terms, you typically start with plot function (or similar plot creating function) to *initiate* a plot and then *annotate* the plot with various annotation functions (text, lines, points, axis)

# Base Graphics

- Base graphics are used most commonly and are a very powerful system for creating 2-D graphics.
- There are two *phases* to creating a base plot
  - Initializing a new plot
  - Annotating (adding to) an existing plot
- Calling `plot(x, y)` or `hist(x)` will launch a graphics device (if one is not already open) and draw a new plot on the device
- If the arguments to plot are not of some special class, then the *default* method for plot is called;
  - this function has *many* arguments, letting you set the title, x axis label, y axis label, etc.
- The base graphics system has *many* parameters that can be set and tweaked; these parameters are documented in `?par`;
  - it wouldn't hurt to try to memorize this help page!

# Example

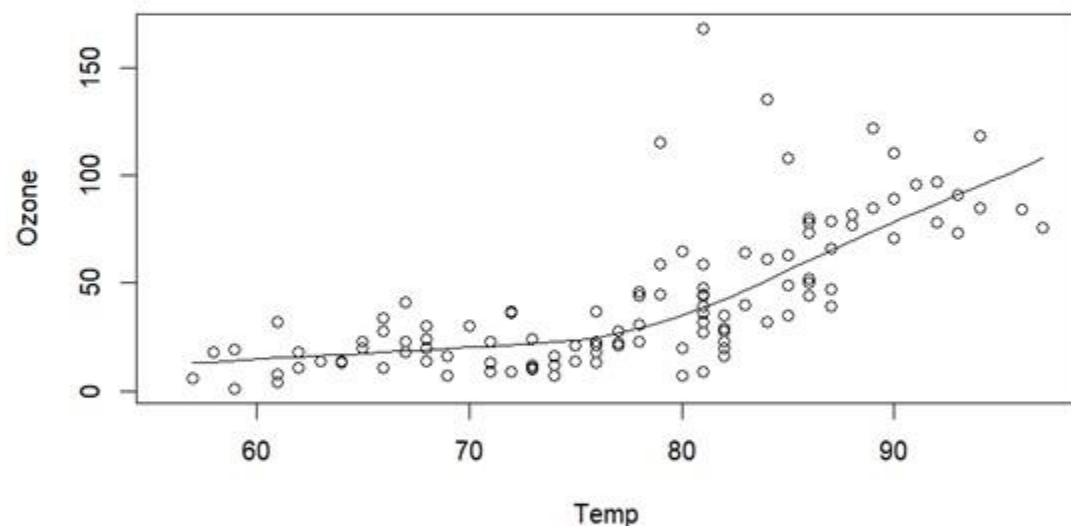
```
> data(cars)
> ## Create the plot / draw canvas
> with(cars, plot(speed,dist))
> ## Add annotation
> title("Speed vs. Stopping distance")
```



# Example

- Start with a simple scatterplot and then decide to add a linear regression line or a smoother to it to highlight the trends.

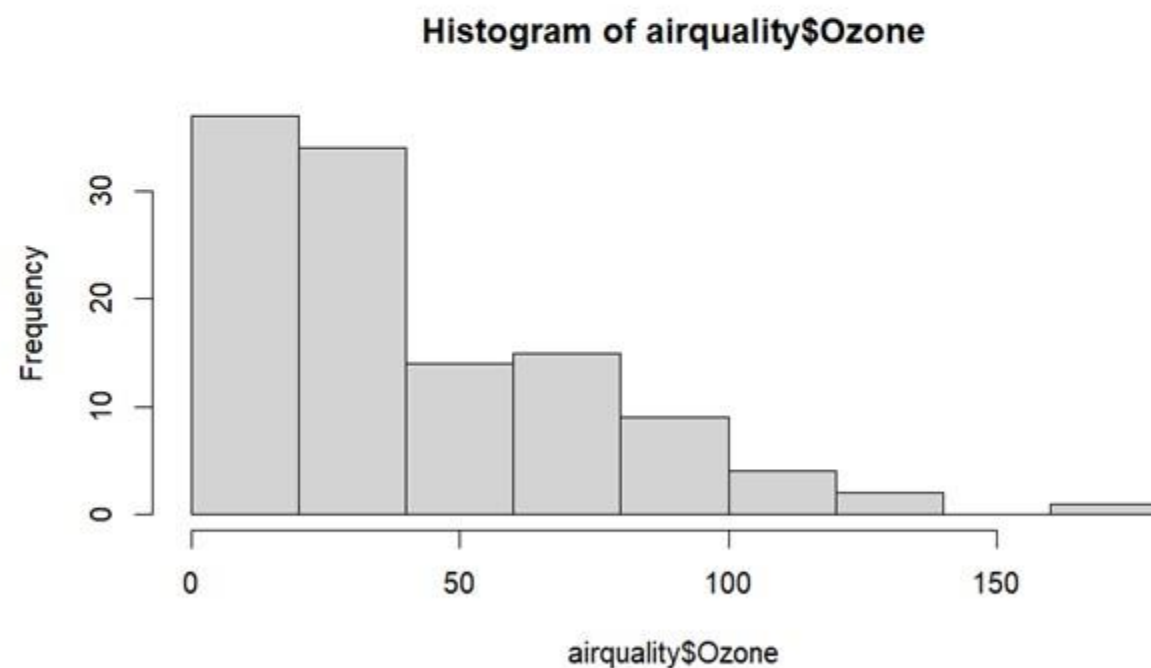
```
> data(airquality)
> with(airquality, {plot(Temp, Ozone)
+   lines(loess.smooth(Temp,Ozone))})
```



- plot function creates the initial plot and draws the points (circles) on the canvas.
- The lines function is used to annotate or add to the plot; in this case it adds a loess smoother to the scatterplot.

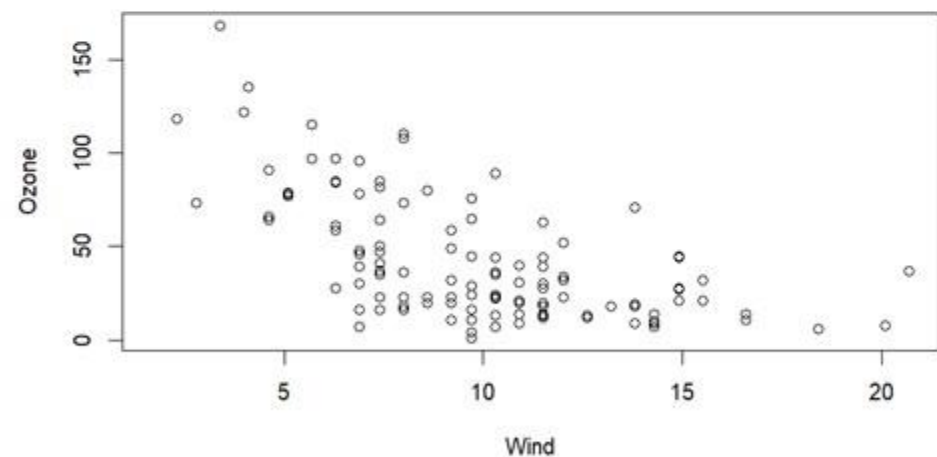
# Simple Base Graphics: Histogram

```
> hist(airquality$Ozone) ## Draw a new plot
```



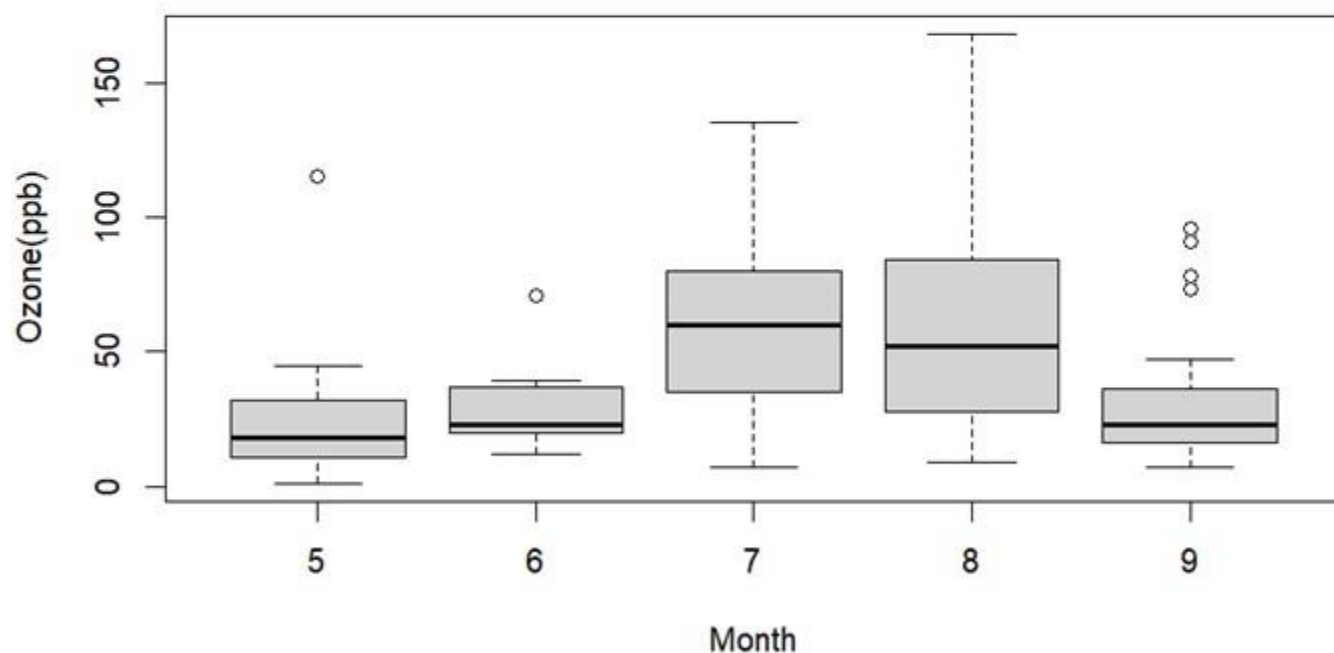
# Simple Base Graphics: Scatterplot

```
with(airquality, plot(Wind, Ozone))
```



# Simple Base Graphics: Boxplot

```
> airquality <- transform(airquality, Month= factor(Month))  
> boxplot(Ozone ~ Month, airquality, xlab = "Month", ylab = "Ozone(ppb)")
```



# Some Important Base Graphics Parameters

- **Many base plotting functions share a set of parameters. Here are a few key ones:**
- **pch:** the plotting symbol (default is open circle)
- **lty:** the line type (default is solid line), can be dashed, dotted, etc.
- **lwd:** the line width, specified as an integer multiple
- **col:** the plotting color, specified as a number, string, or hex code;
- **xlab:** character string for the x-axis label
- **ylab:** character string for the y-axis label

# Some Important Base Graphics Parameters

- The `par()` function is used to specify *global* graphics parameters that affect all plots in an R session. These parameters can be overridden when specified as arguments to specific plotting functions.
- `las`: the orientation of the axis labels on the plot
- `bg`: the background color
- `mar`: the margin size
- `oma`: the outer margin size (default is 0 for all sides)
- `mfrow`: number of plots per row, column (plots are filled row-wise)
- `mfcol`: number of plots per row, column (plots are filled column - wise)

# Some Important Base Graphics Parameters

- **Default values for global graphics parameters**

```
par("lty")  
## [1] "solid"  
par("col")  
## [1] "black"  
par("pch")  
## [1] 1
```

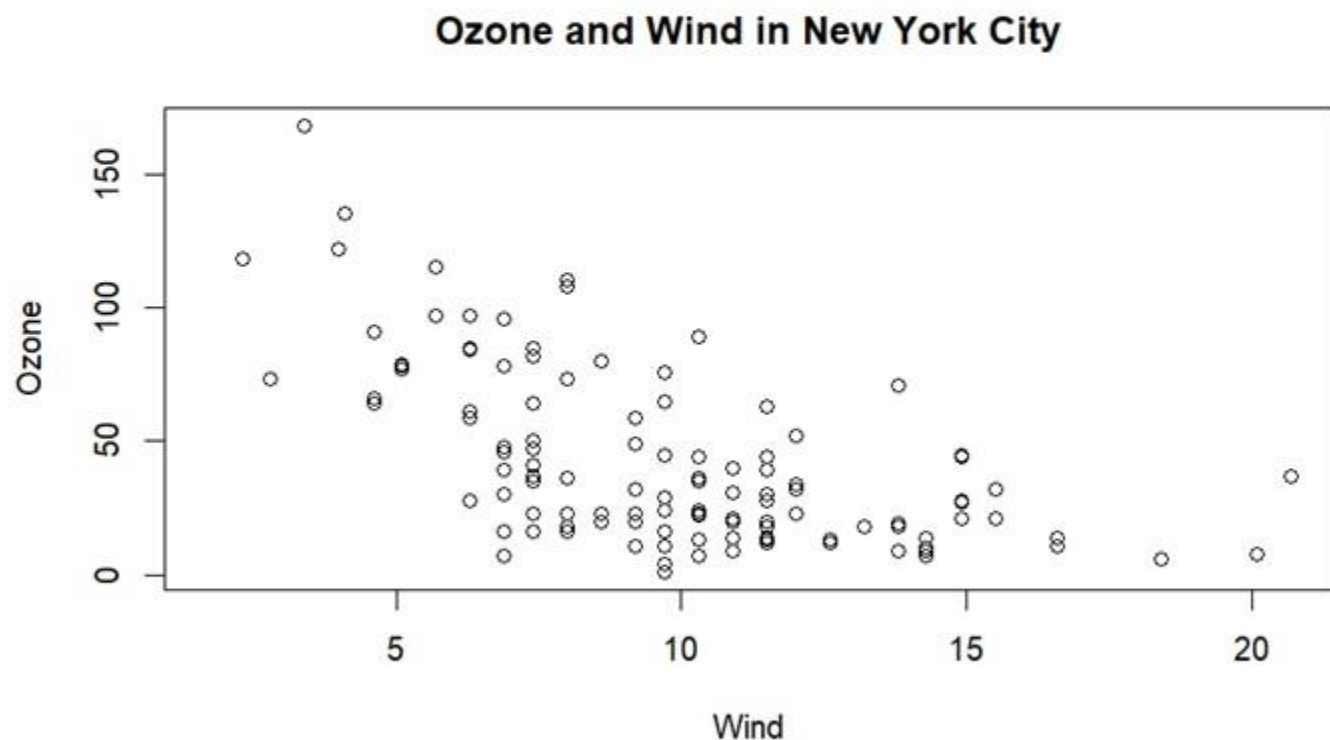
```
par("bg")  
## [1] "transparent"  
par("mar")  
## [1] 5.1 4.1 4.1 2.1  
par("mfrow")  
## [1] 1 1
```

# Base Plotting Functions

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

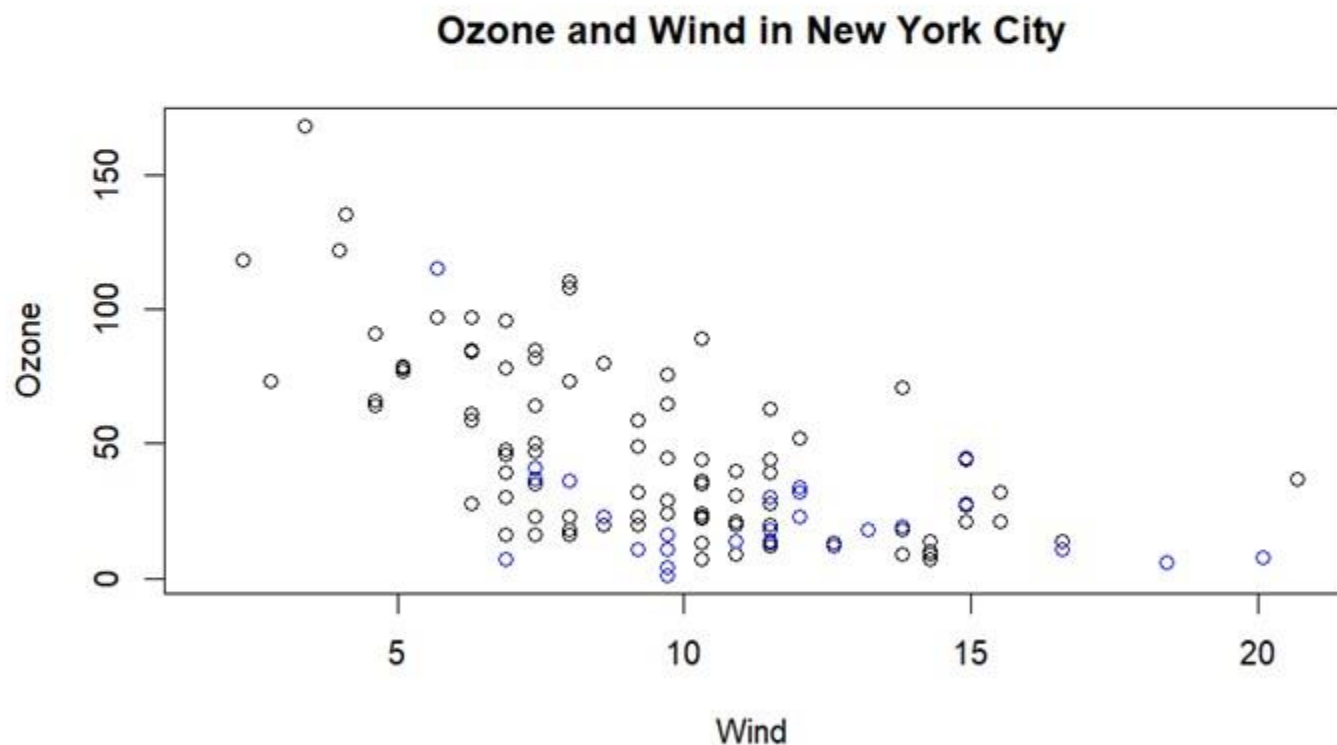
# Base Plot with Annotation

```
> with(airquality, plot(Wind,Ozone))  
> title(main = "Ozone and Wind in New York City")
```



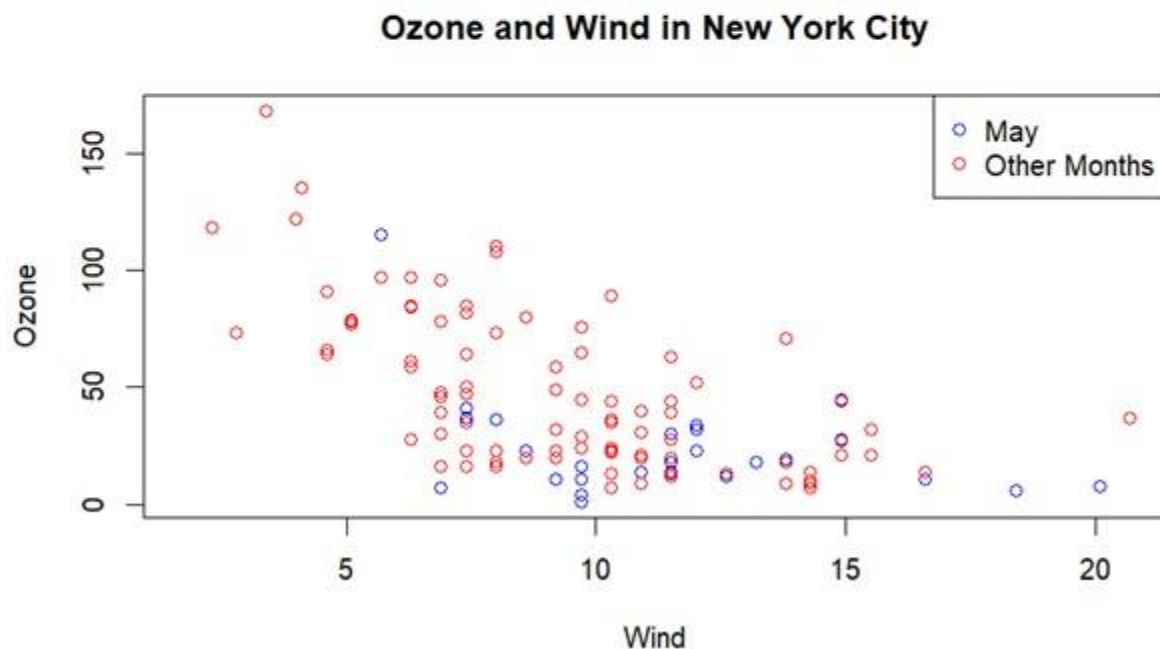
# Base Plot with Annotation

```
> with(airquality, plot(Wind, Ozone, main = "Ozone and Wind in New York City"))  
> with(subset(airquality, Month == 5), points(Wind, Ozone, col = "blue"))
```



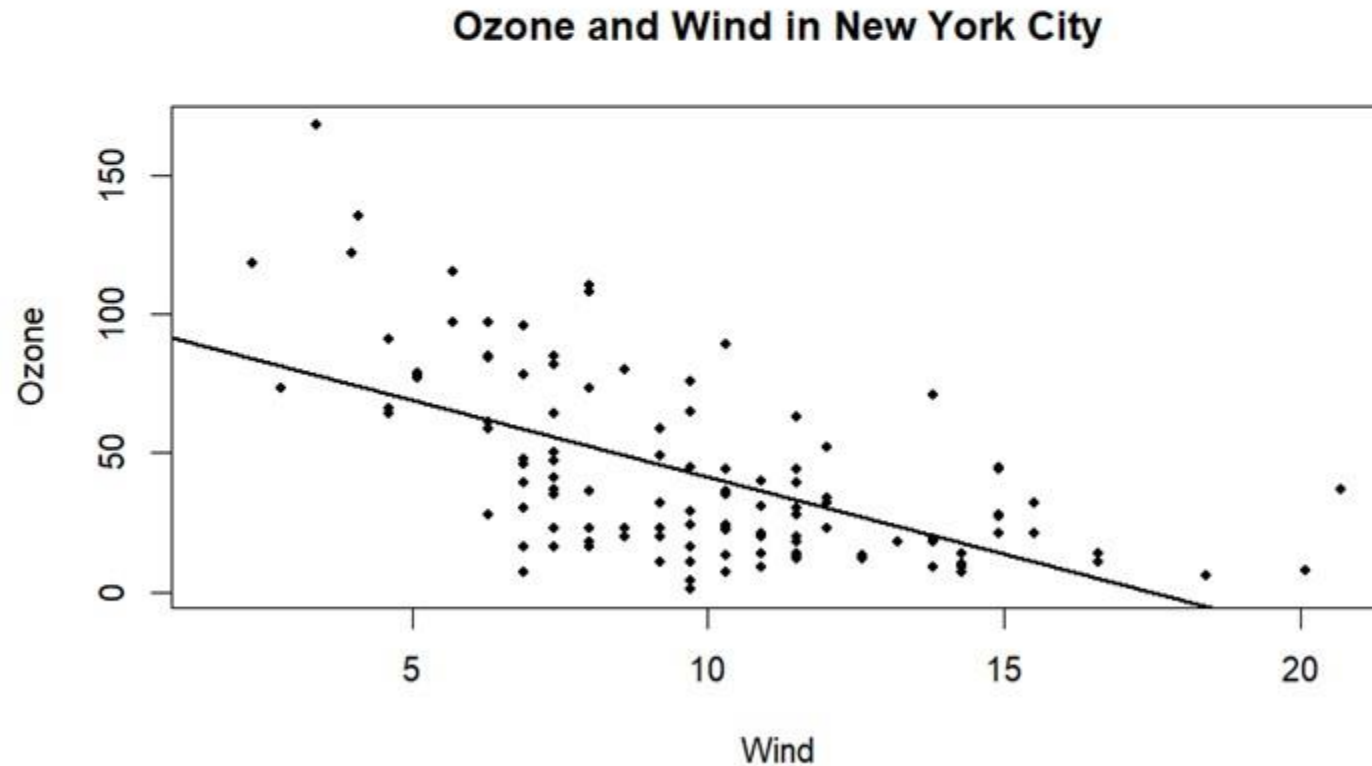
# Base Plot with Annotation

```
> with(airquality, plot(Wind, Ozone, main = "Ozone and Wind in New York City"), type = "n")  
> with(subset(airquality, Month == 5), points(Wind, Ozone, col = "blue"))  
> with(subset(airquality, Month != 5), points(Wind, Ozone, col = "red"))  
> legend("topright", pch = 1, col = c("blue", "red"), legend = c("May", "Other Months"))
```



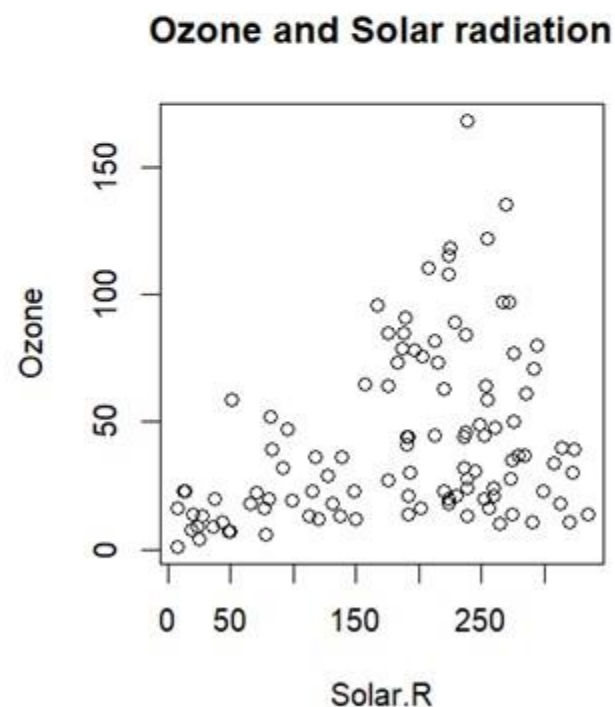
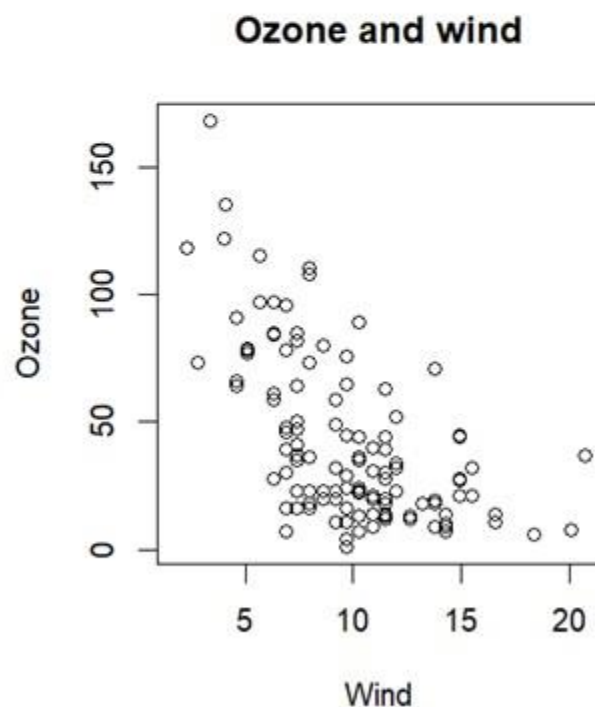
# Base Plot with Regression Line

```
> with(airquality, plot(Wind, Ozone, main = "Ozone and Wind in New York City", pch = 20))  
> model <- lm(Ozone ~ Wind, airquality)  
> abline(model, lwd=2)
```



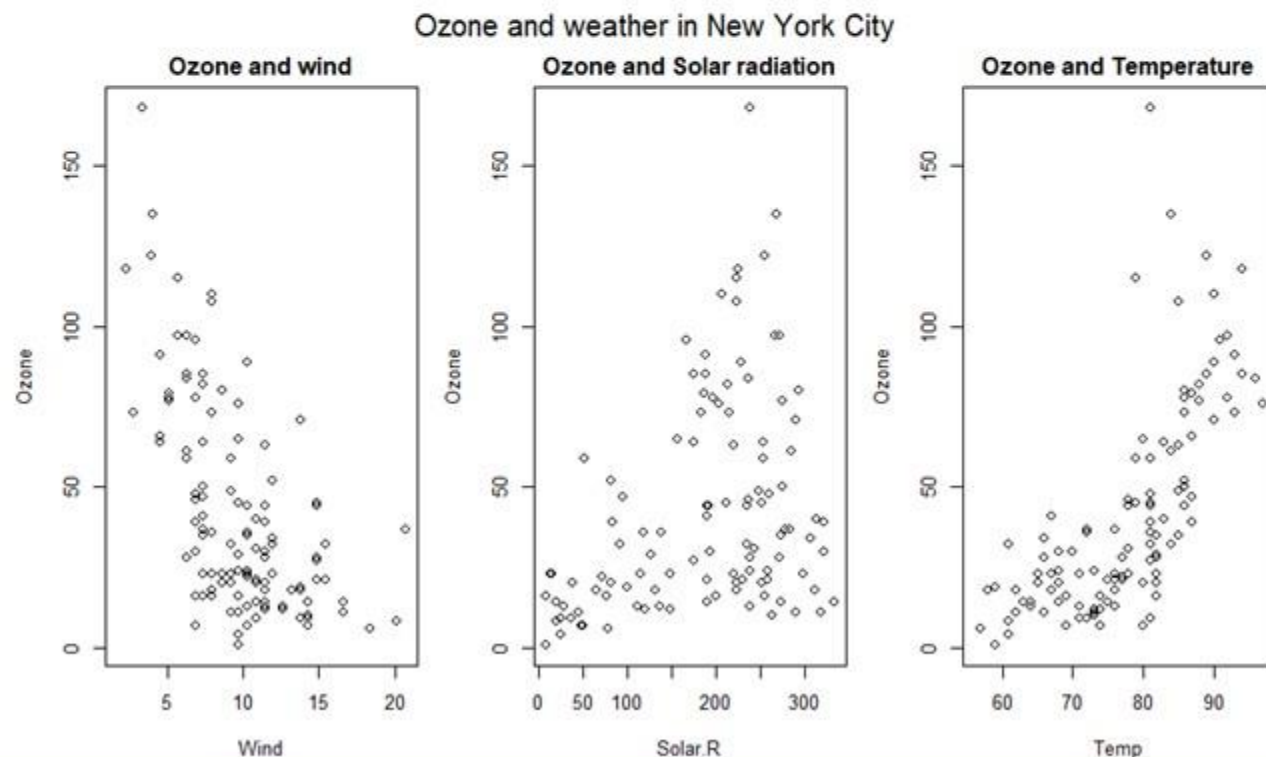
# Multiple Base Plots

```
> par(mfrow = c(1,2))  
> with(airquality, {  
+   plot(wind, Ozone, main = "Ozone and wind")  
+   plot(Solar.R, Ozone, main = "Ozone and Solar radiation")  
+ })
```



# Multiple Base Plots

```
> par(mfrow = c(1,3), mar = c(4,4,2,1), oma = c(0,0,2,0))  
> with(airquality, {  
+   plot(Wind, Ozone, main = "Ozone and wind")  
+   plot(Solar.R, Ozone, main = "Ozone and Solar radiation")  
+   plot(Temp, Ozone, main = "Ozone and Temperature")  
+   mtext("Ozone and weather in New York City", outer = TRUE)  
+ })
```



# Summary

- **Plots in the base plotting system are created by calling successive R functions to "build up" a plot**
- **Plotting occurs in two stages:**
  - Creation of a plot
  - Annotation of a plot (adding lines, points, text, legends)
- **The base plotting system is very flexible and offers a high degree of control over plotting**

# The Lattice System

- Implemented in the lattice package (comes with every installation of R).
- Load the lattice package : `library(lattice)`
- Plots are created with a single function call (`xyplot`, `bwplot`, etc.)
  - no real distinction between functions that create or initiate plots and functions that annotate plots because it all happens at once
- **Most useful for conditioning types of plots: Looking at how y changes with x across levels of z**
  - These types of plots are useful for looking at multidimensional data and often allow you to squeeze a lot of information into a single window or page.
- **Things like margins/spacing set automatically because entire plot is specified at once via single function**
- **Good for putting many plots on a screen**

# The Lattice Plotting System

The lattice plotting system is implemented using the following packages:

- ***lattice***: contains code for producing Trellis graphics, which are independent of the "base" graphics system; includes functions like `xyplot`, `bwplot`, `levelplot`
- ***grid***: implements a different graphing system independent of the "base" system; the *lattice* package builds on top of *grid*

We seldom call functions from the *grid* package directly

- 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

# Lattice Functions

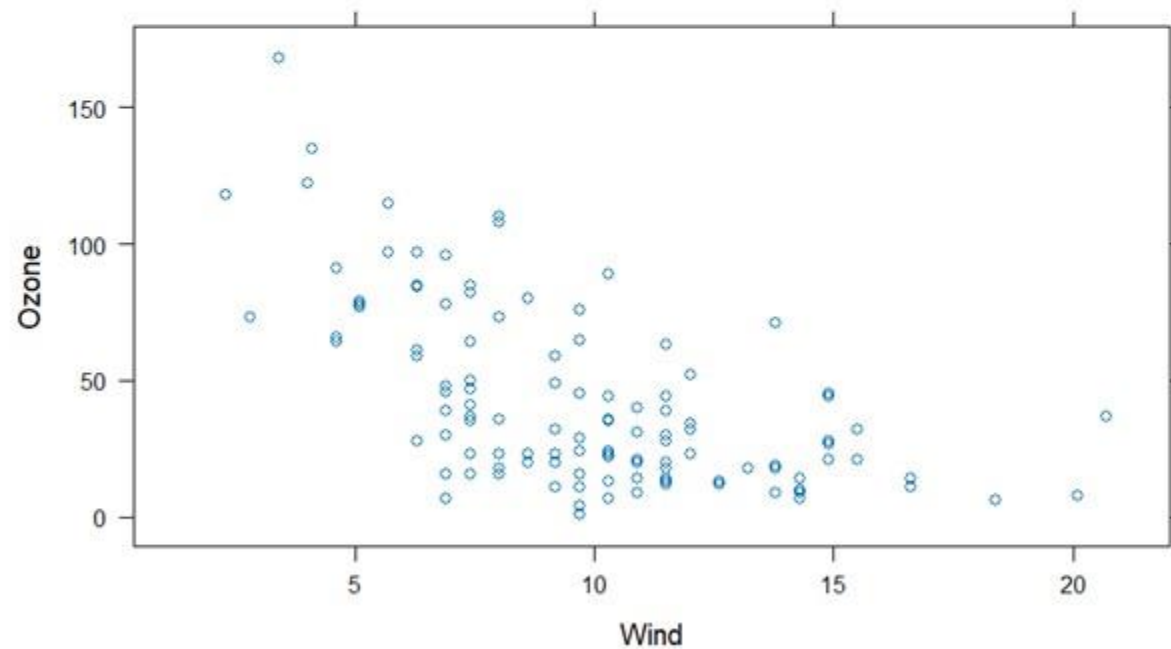
- **xyplot:** this is the main function for creating scatterplots
- **bwplot:** box-and-whiskers plots ("boxplots")
- **histogram:** histograms
- **stripplot:** like a boxplot but with actual points
- **dotplot:** plot dots on "violin strings"
- **spom:** scatterplot matrix; like pairs in base plotting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

# Lattice Functions

- Lattice functions generally take a formula for their first argument, usually of the form `xyplot(y ~ x | f * g, data)`
  - 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.*

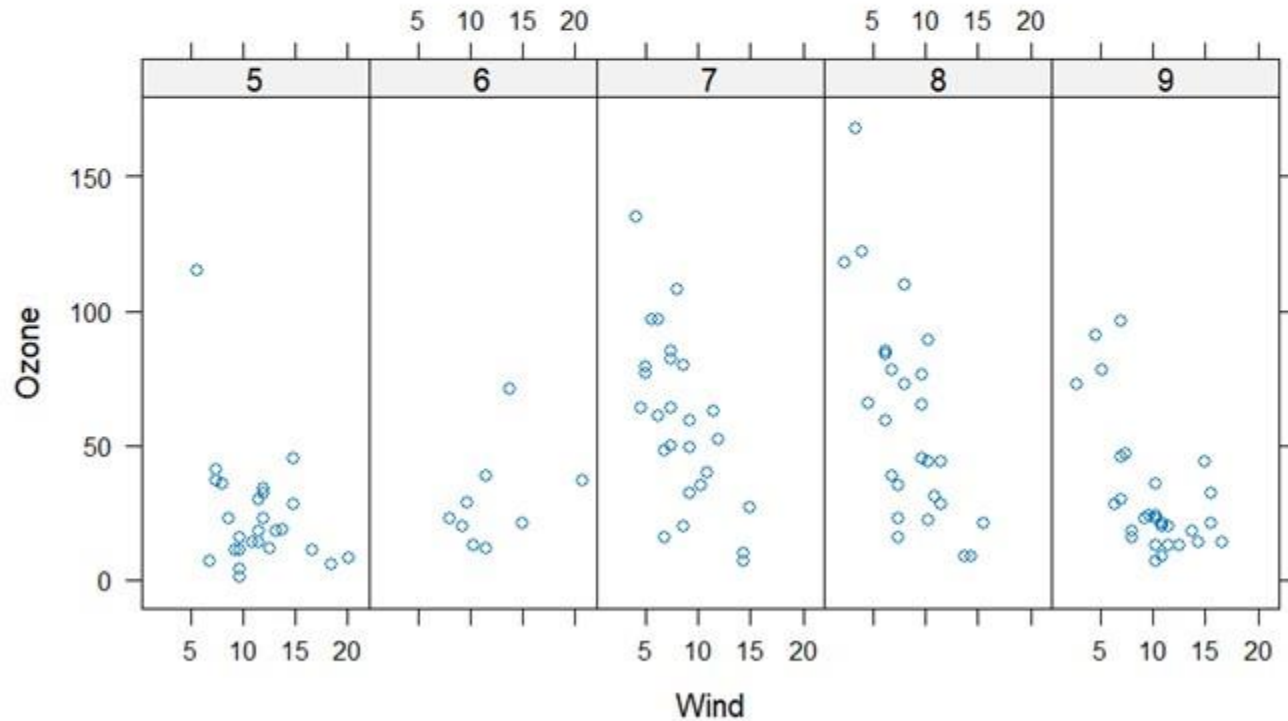
# Simple Lattice Plot

```
library(lattice)  
xyplot(Ozone~Wind, data = airquality)
```



# Simple Lattice Plot

```
> airquality <- transform(airquality, Month = factor(Month))  
> xyplot(Ozone ~ Wind|Month, data = airquality, layout = c(5,1))
```

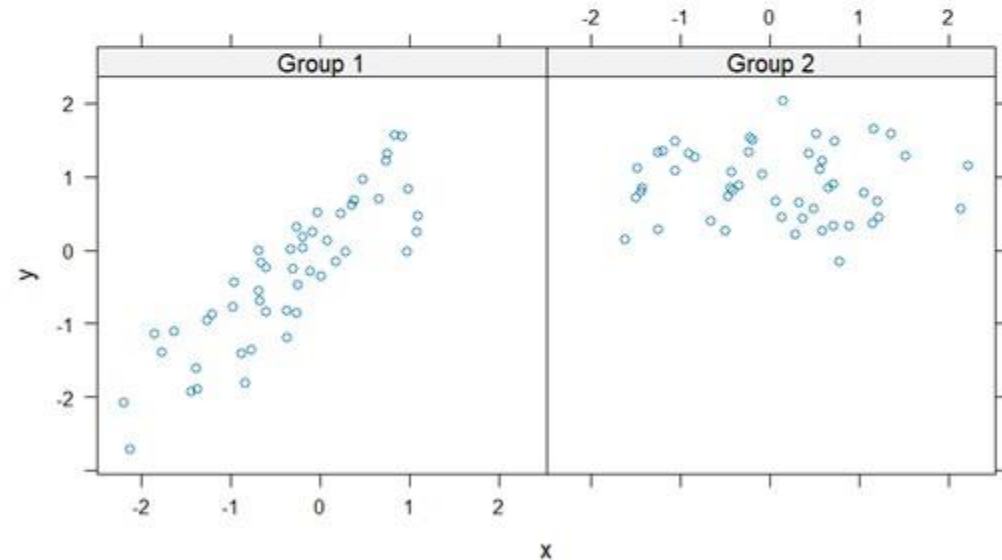


# Lattice Panel Functions

- Lattice functions have a panel function which controls what happens inside each panel of the plot.
- The *lattice* package comes with default panel functions, but you can supply your own if you want to customize what happens in each panel
- Panel functions receive the x/y coordinates of the data points in their panel (along with any optional arguments)

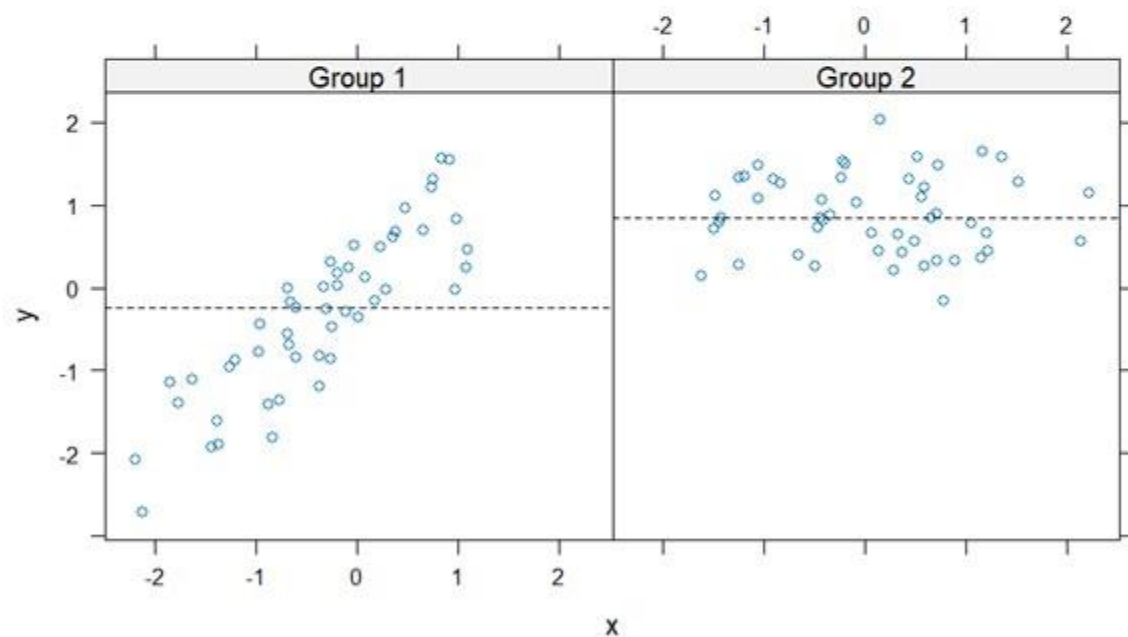
# Lattice Panel Functions

```
> set.seed(10)
> x <- rnorm(100)
> f <- rep(0:1, each = 50)
> y <- x + f - f * x + rnorm(100, sd=0.5)
> f <- factor(f, labels = c("Group 1", "Group 2"))
> xyplot(y~x|f, layout = c(2,1))
```



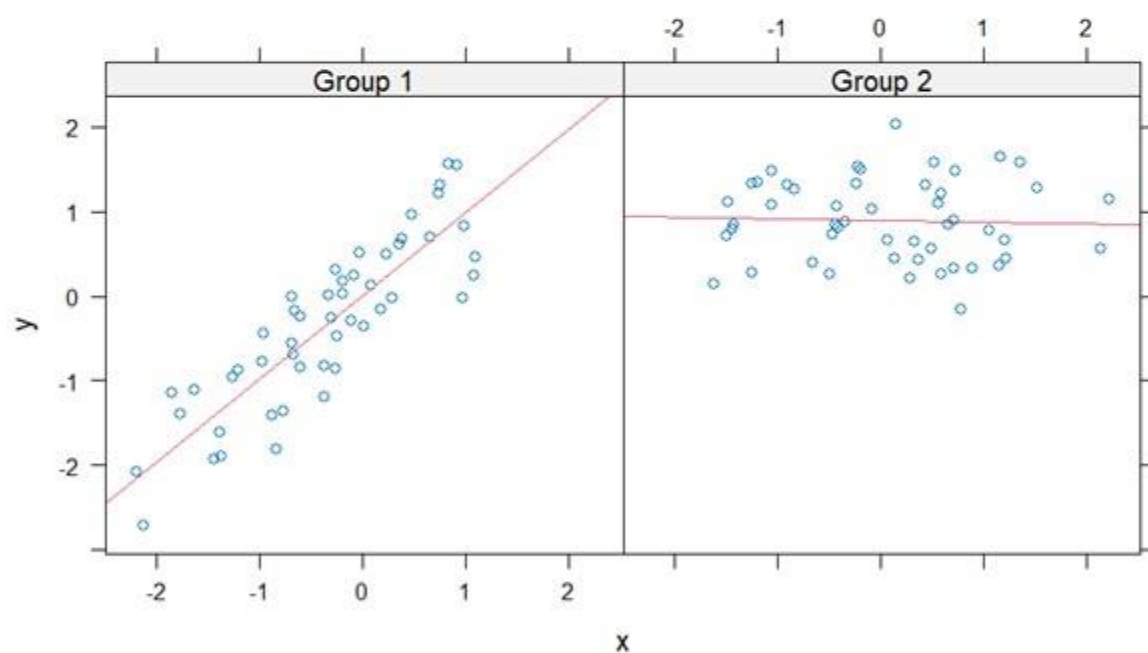
# Lattice Panel Functions

```
> xyplot(y~x|f, panel = function(x,y,...){  
+   panel.xyplot(x,y,...)  
+   panel.abline(h=median(y), lty =2)  
+ })
```



# Lattice Panel Functions: Regression line

```
> xyplot(y~x|f, panel = function(x,y,...){  
+   panel.xyplot(x,y,...)  
+   panel.lmline(x, y, col=2)  
+ })
```

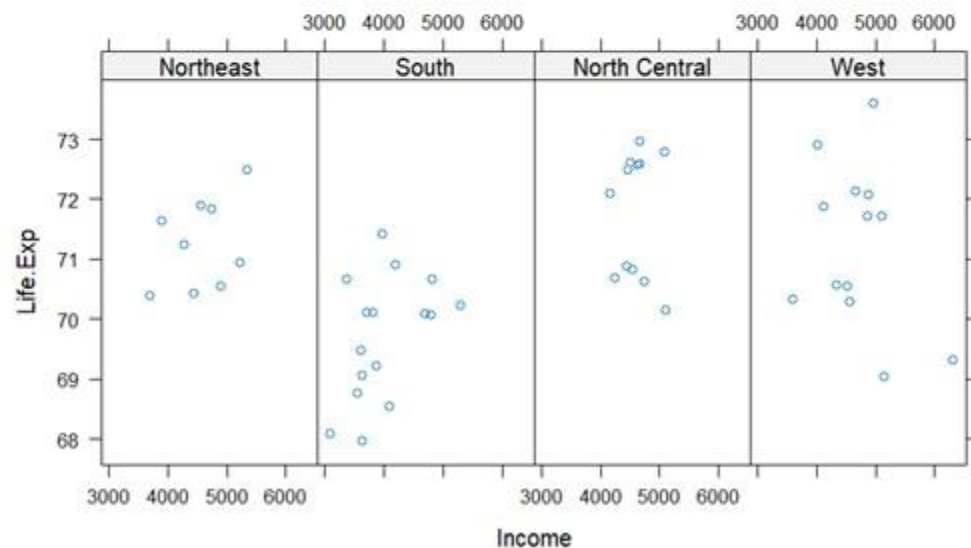


# Example

- Lattice plot that looks at the relationship between life expectancy and income and how that relationship varies by region in the United States.

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

- The plot itself contains four panels
    - one for each region- and within each panel is a scatterplot of life expectancy and income.
- The notion of *panels* comes up a lot with lattice plots because you typically have many panels in a lattice plot
- each panel typically represents a *condition* like "region".



# Lattice Plot: Summary

- Lattice plots are constructed ,with a single function call to a core lattice function (e.g. xyplot)
- Aspects like margins and spacing are automatically handled and defaults are usually sufficient
- The lattice system is ideal for creating conditioning plots where you examine the same kind of plot under many different conditions
- Panel functions can be specified/customized to modify what is plotted in each of the plot panels

# The ggplot2 System

- **Splits the difference between base and lattice in a number of ways:**  
Automatically deals with spacings text titles but also allows you to annotate by "adding" to a plot
- **Superficial similarity to lattice but generally easier/more intuitive to use**
- **Default mode makes many choices for you (but you can still customize to your heart's desire)**
- **Implemented in the ggplot2 package which is available from CRAN (it does not come with R).**

```
install.packages("ggplot2")  
#load it into R via the library function.  
> library(ggplot2)
```

# What is ggplot2?

- Grammar of graphics represents and abstraction of graphics ideas/objects
- Think "verb", "noun", "adjective" for graphics
- Allows for a "theory" of graphics on which to build new graphics and graphics objects
- "Shorten the distance from mind to page"

# Grammar of Graphics

- "In brief, the grammar tells us that a statistical graphic is a mapping from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars). The plot may also contain statistical transformations of the data and is drawn on a specific coordinate system" from *ggplot2* book

# Example Dataset

- **Among the variables in mpg are:**

displ, a car's engine size, in liters.

hwy a car's fuel efficiency on the highway in miles per gallon (mpg). A car with a low fuel efficiency consumes more fuel than a car with a high fuel efficiency when they travel the same distance.

```
> str(mpg)
tibble [234 × 11] (S3: tbl_df/tbl/data.frame)
 $ manufacturer: chr [1:234] "audi" "audi" "audi" "audi" ...
 $ model       : chr [1:234] "a4" "a4" "a4" "a4" ...
 $ displ       : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year        : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl         : int [1:234] 4 4 4 4 6 6 6 4 4 4 ...
 $ trans       : chr [1:234] "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
 $ drv         : chr [1:234] "f" "f" "f" "f" ...
 $ cty         : int [1:234] 18 21 20 21 16 18 18 18 16 20 ...
 $ hwy         : int [1:234] 29 29 31 30 26 26 27 26 25 28 ...
 $ fl          : chr [1:234] "p" "p" "p" "p" ...
 $ class       : chr [1:234] "compact" "compact" "compact" "compact" ...
```

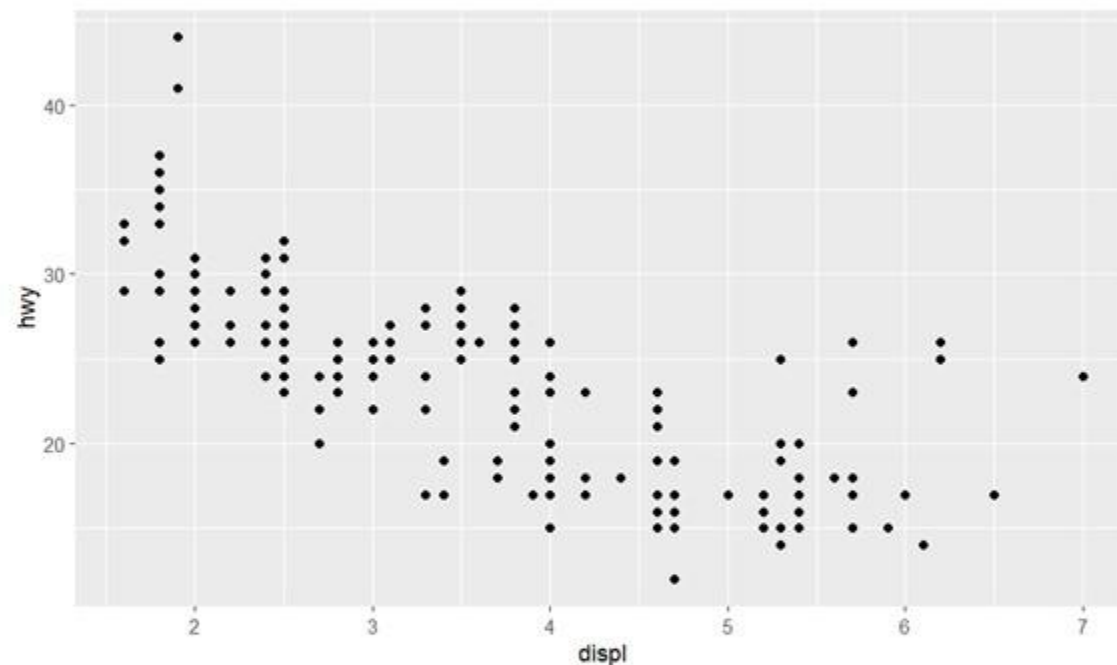
# The Basics: A Quick Plot `qplot()`!

- Works much like the `plot` function in base graphics system
- Looks for data in a data frame, similar to `lattice`, or in the parent environment
- Plots are made up of *aesthetics* (size, shape, color) and *geoms* (points, lines)
- Factors are important for indicating subsets of the data (if they are to have different properties); they should be labeled
- `ggplot()` is the core function and very flexible for doing things `qplot()` cannot do

# Example

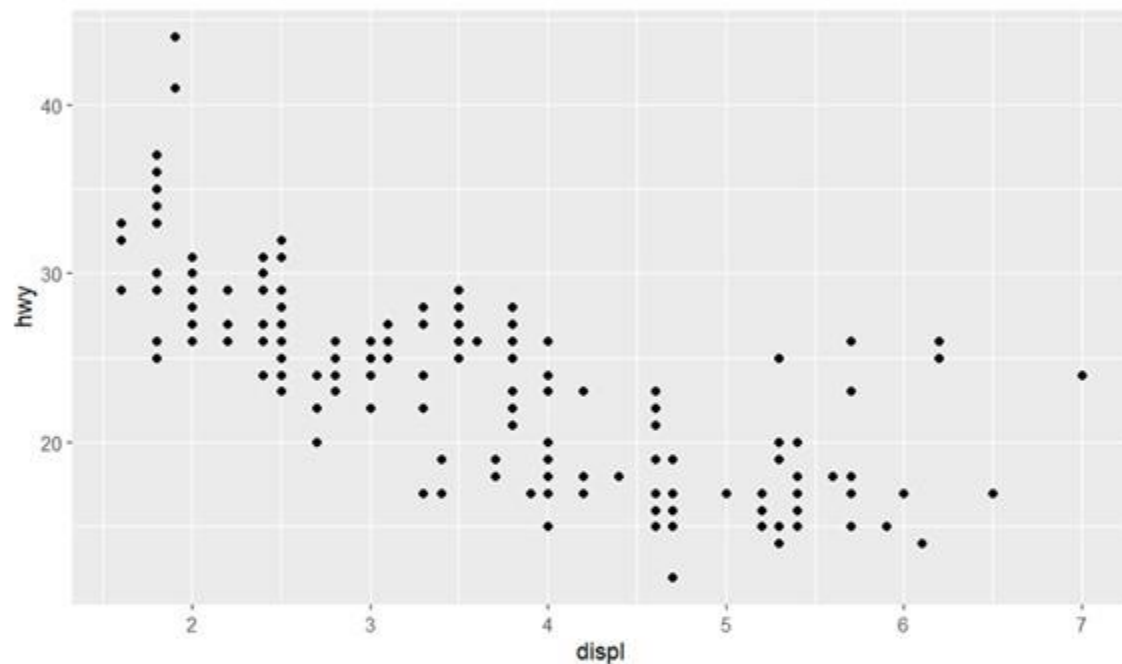
- The ggplot2 system is very useful for conditioning plots just like in Lattice.

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

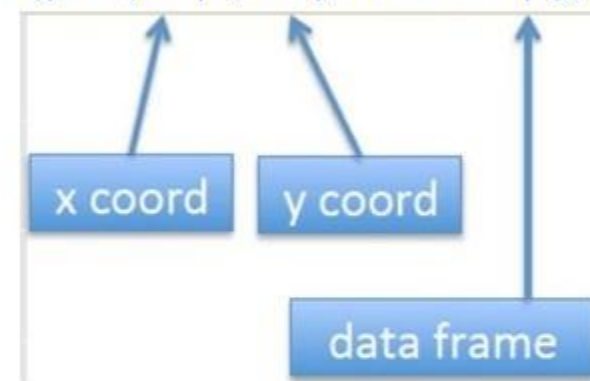


- qplot function in ggplot2 is what you use to "quickly get some data on the screen".
- ggplot2 system has a lot of defaults.
- Additional functions in ggplot2 allow you to make arbitrarily sophisticated plots.

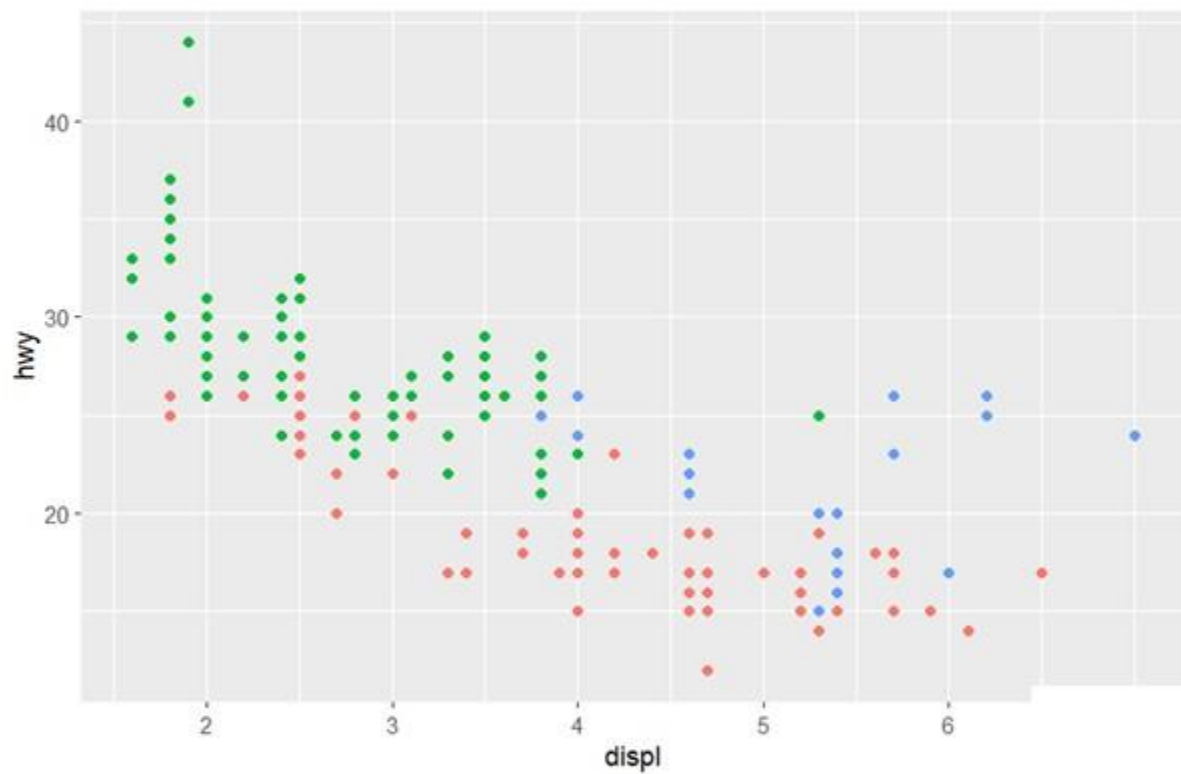
# ggplot2 "Hello, world!"



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



# Modifying aesthetics



drv

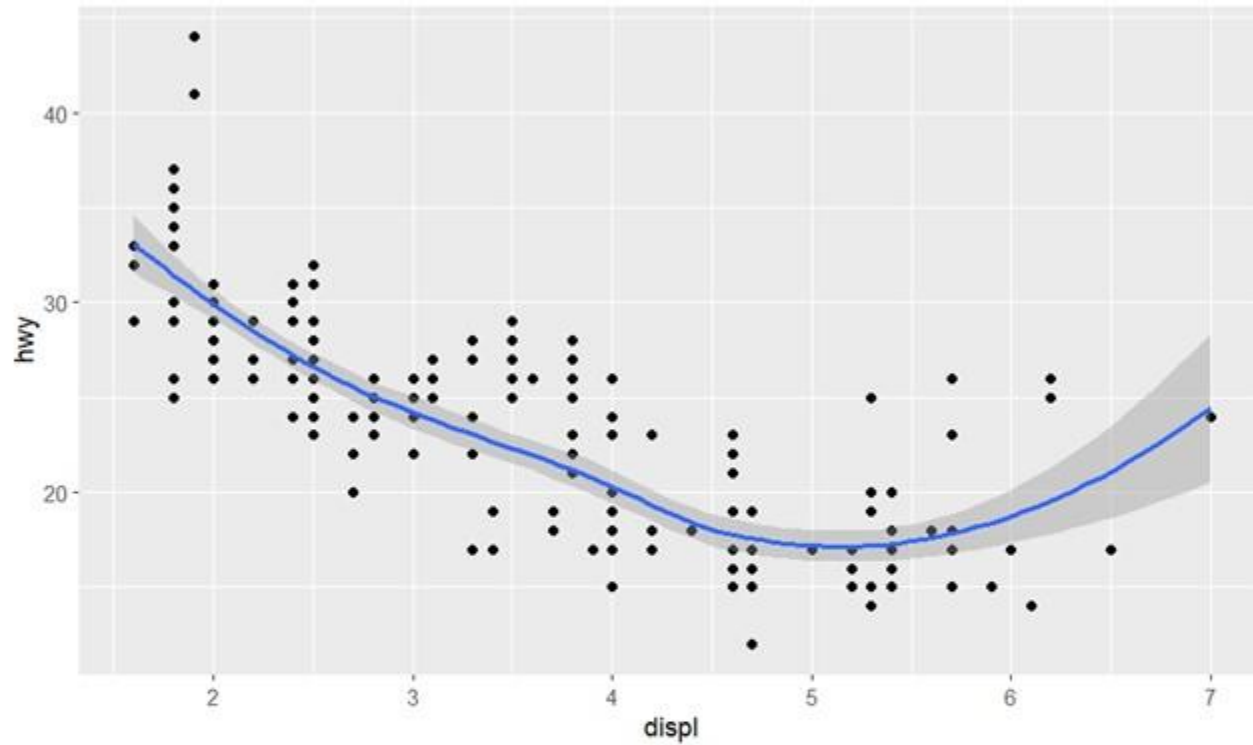
- 4
- f
- r

auto legend  
placement

color aesthetic

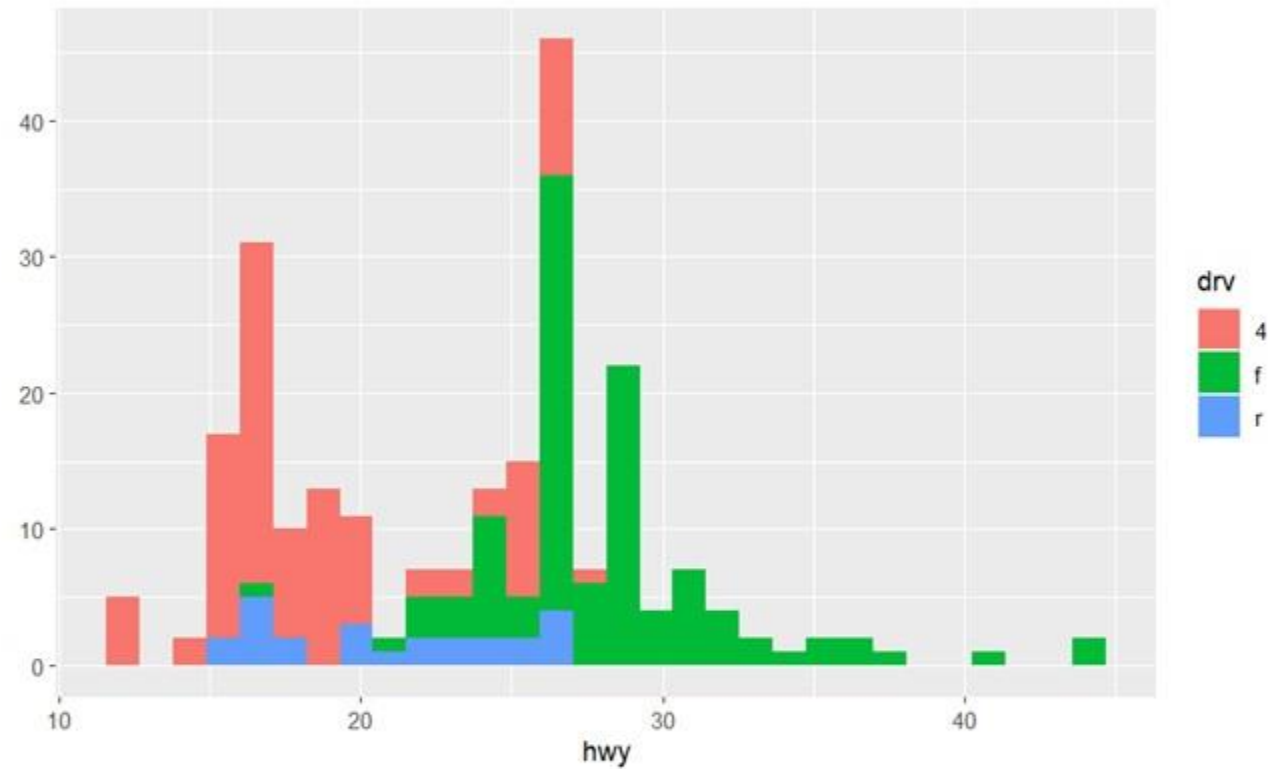
```
qplot(displ, hwy, data = mpg, color = drv)
```

# Adding a geom



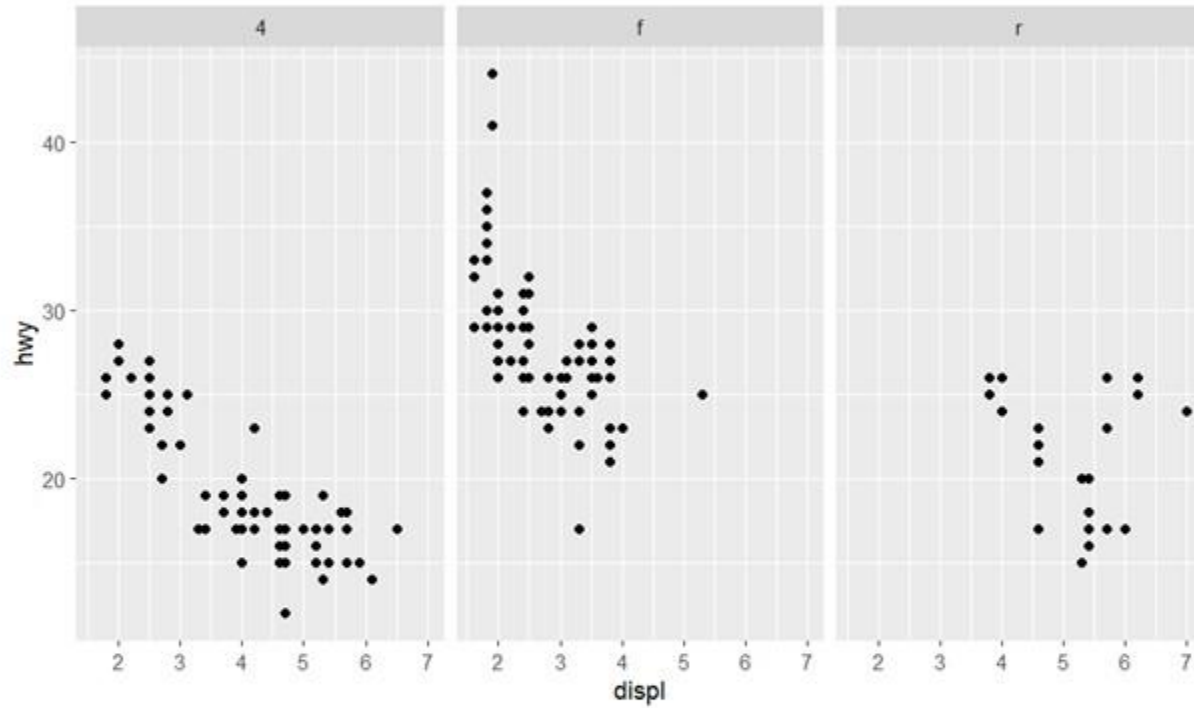
```
qplot(displ, hwy, data = mpg, geom = c("point", "smooth"))
```

# Histograms by Group

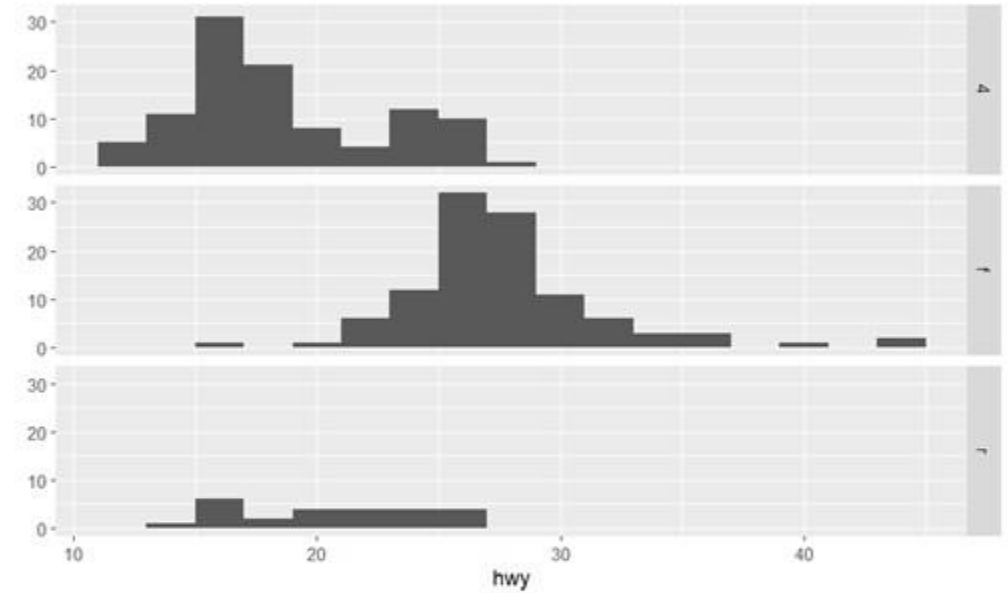


```
qplot(hwy, data = mpg, fill = drv)
```

# Facets

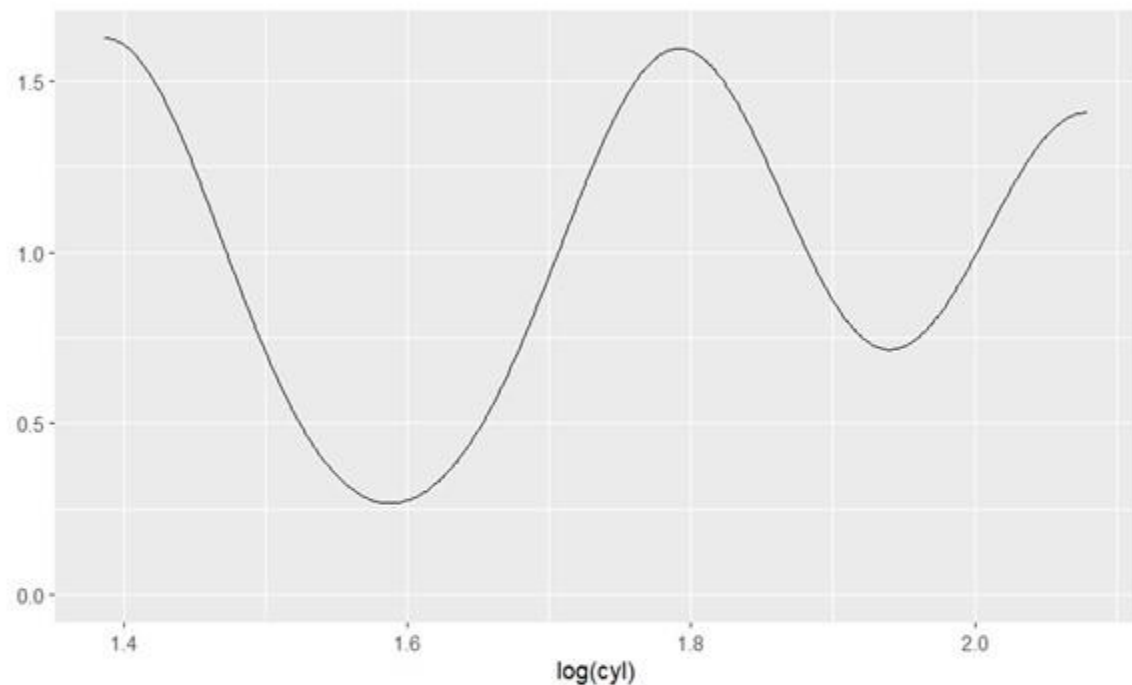


```
qplot(displ, hwy, data = mpg, facets = .~drv)
```

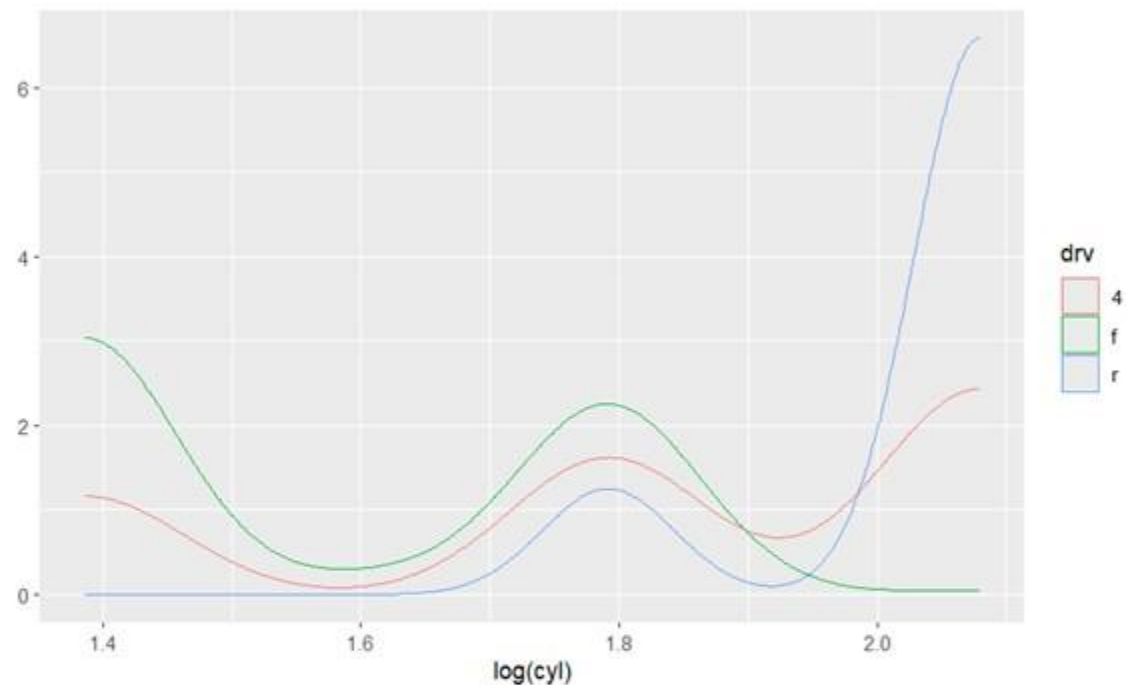


```
qplot(hwy, data = mpg, facets = drv~., binwidth = 2)
```

# Density Smooth

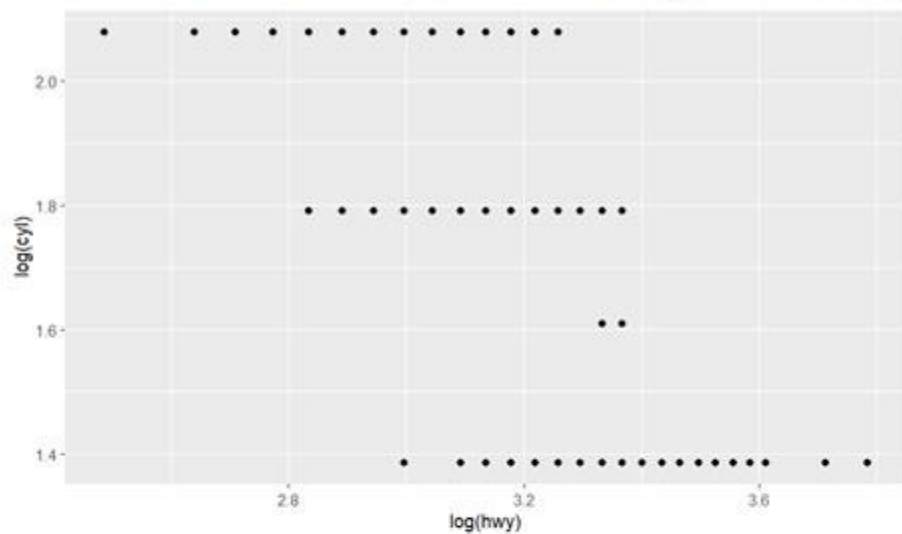


```
qplot(log(cyl), data = mpg, geom = "density")
```

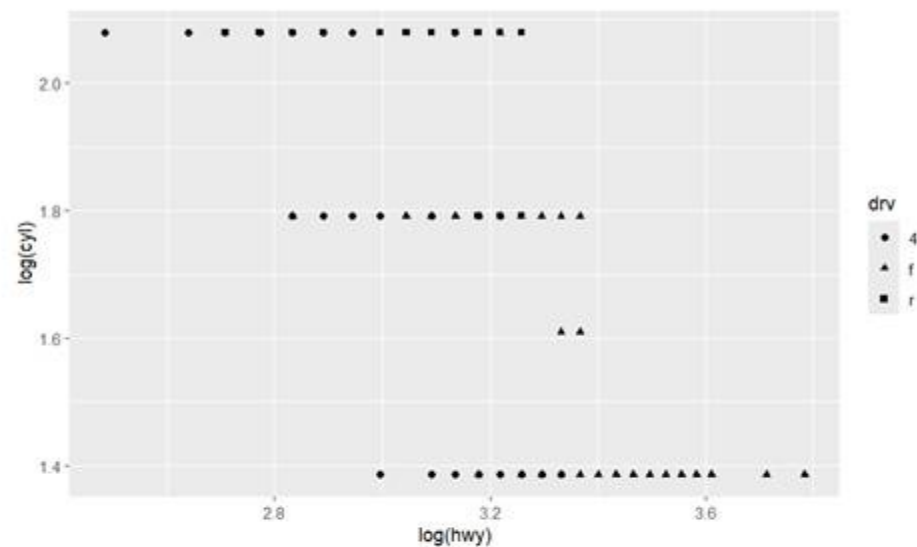
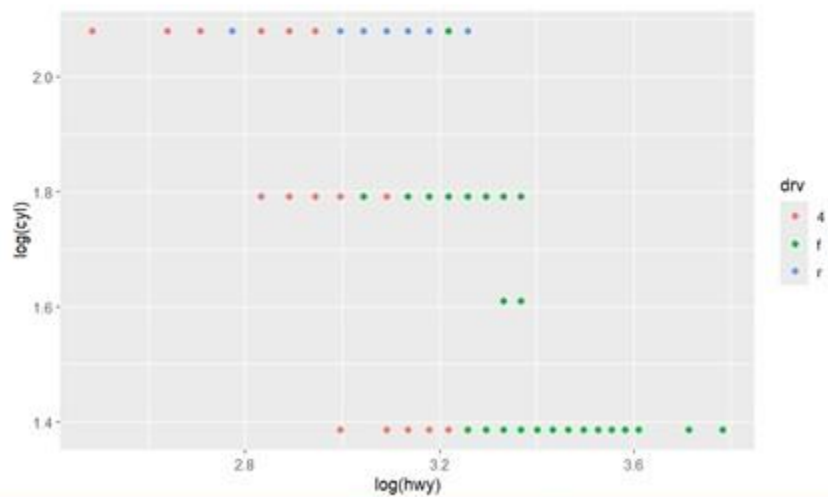


```
qplot(log(cyl), data = mpg, geom = "density", color = drv)
```

# ScatterPlot: hwy vs cyl



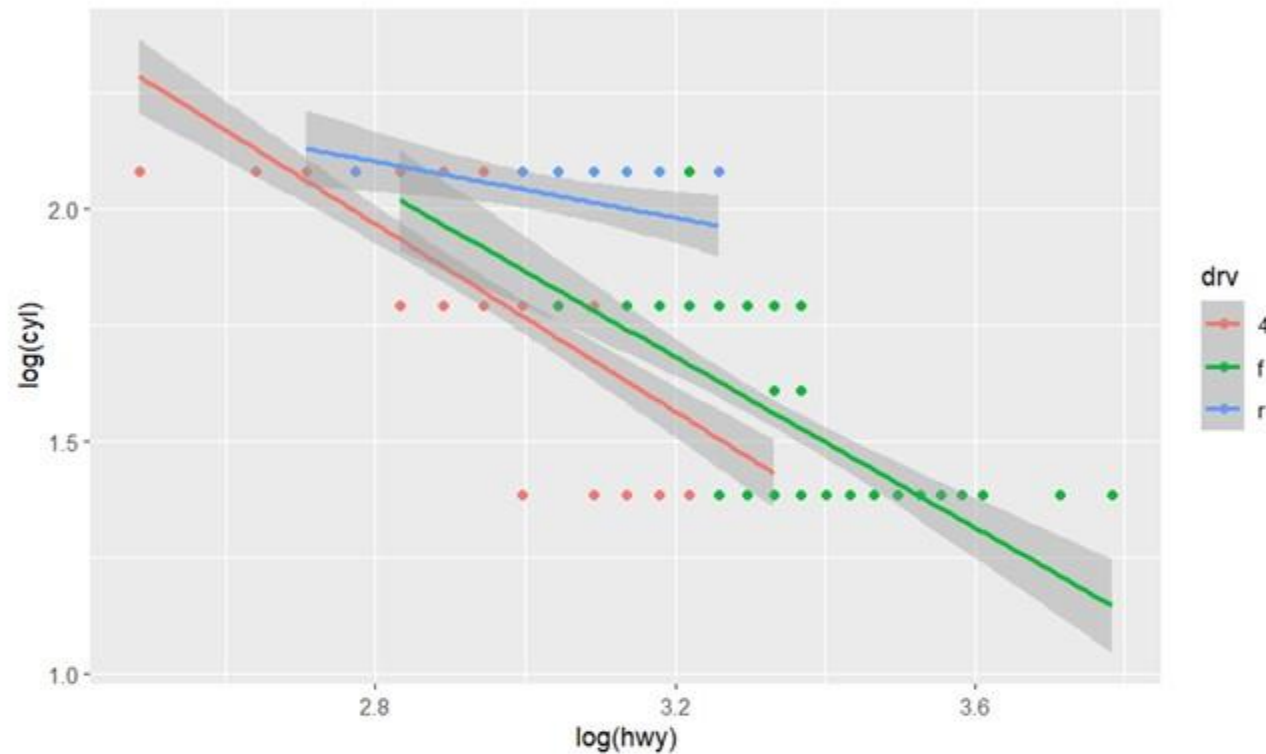
```
qplot(log(hwy), log(cyl), data = mpg)
```



```
qplot(log(hwy), log(cyl), data = mpg, shape = drv)
```

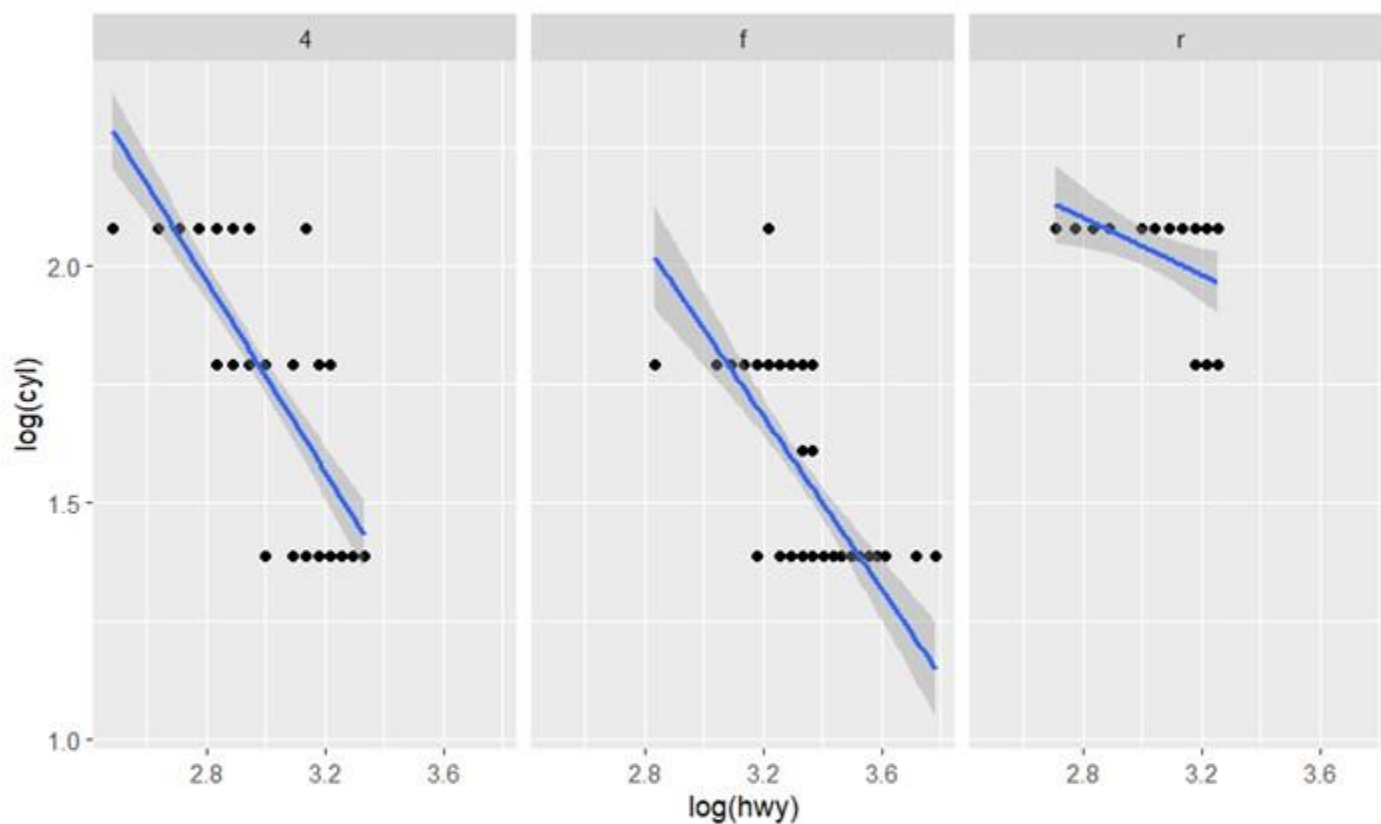
```
qplot(log(hwy), log(cyl), data = mpg, color = drv)
```

# ScatterPlot: hwy vs mpg



```
qplot(log(hwy), log(cyl), data = mpg, color = drv, geom = c("point", "smooth"), method = "lm")
```

# ScatterPlot: hwy vs mpg



```
qplot(log(hwy), log(cyl), data = mpg, geom = c("point", "smooth"), method = "lm", facets = .~drv)
```

# Summary of `qplot()`

- The `qplot()` function is the analog to `plot()` but with many built in features
- Syntax somewhere in between `base`/`lattice`
- Produces very nice graphics

# Basic components of ggplot2 Plot

- A **data frame**
- **aesthetic mappings:** how data are mapped to color, size
- **geoms:** geometric objects like points, lines, shapes.
- **facets:** for conditional plots.
- ....

# Building plots in ggplot2

- When building plots in ggplot2 (rather than using qplot) the "artist's palette" model may be the closest analogy
- Plots are built up in layers
  - Plot the data
  - Overlay a summary
  - Metadata and annotation

# Example: data(mpg)

- Do cars ,vith big engines use more fuel than cars with small engines?
- What does the relationship between engine size and fuel efficiency look like? Is it positive? Negative? Linear? Nonlinear?

```
> head(mpg)
```

|   | manufacturer | model | displ | year | cyl | trans      | drv | cty | hwy | fl | class   |
|---|--------------|-------|-------|------|-----|------------|-----|-----|-----|----|---------|
| 1 | audi         | a4    | 1.8   | 1999 | 4   | auto(l5)   | f   | 18  | 29  | p  | compact |
| 2 | audi         | a4    | 1.8   | 1999 | 4   | manual(m5) | f   | 21  | 29  | p  | compact |
| 3 | audi         | a4    | 2.0   | 2008 | 4   | manual(m6) | f   | 20  | 31  | p  | compact |
| 4 | audi         | a4    | 2.0   | 2008 | 4   | auto(av)   | f   | 21  | 30  | p  | compact |
| 5 | audi         | a4    | 2.8   | 1999 | 6   | auto(l5)   | f   | 16  | 26  | p  | compact |
| 6 | audi         | a4    | 2.8   | 1999 | 6   | manual(m5) | f   | 18  | 26  | p  | compact |



# Building Up in Layers

```
> head(mpg)
```

|   | manufacturer | model | displ | year | cyl | trans      | drv | cty | hwy | fl | class   |
|---|--------------|-------|-------|------|-----|------------|-----|-----|-----|----|---------|
| 1 | audi         | a4    | 1.8   | 1999 | 4   | auto(l5)   | f   | 18  | 29  | p  | compact |
| 2 | audi         | a4    | 1.8   | 1999 | 4   | manual(m5) | f   | 21  | 29  | p  | compact |
| 3 | audi         | a4    | 2.0   | 2008 | 4   | manual(m6) | f   | 20  | 31  | p  | compact |
| 4 | audi         | a4    | 2.0   | 2008 | 4   | auto(av)   | f   | 21  | 30  | p  | compact |
| 5 | audi         | a4    | 2.8   | 1999 | 6   | auto(l5)   | f   | 16  | 26  | p  | compact |
| 6 | audi         | a4    | 2.8   | 1999 | 6   | manual(m5) | f   | 18  | 26  | p  | compact |

Data Frame

Aesthetics

Initial call to  
ggplot

```
g <- ggplot(mpg, aes(hwy, cyl))
```

Summary of  
ggplot object

```
> summary(g)
```

```
data: manufacturer, model, displ, year, cyl, trans, drv, cty, hwy, fl, class  
[234x11]  
mapping: x = ~hwy, y = ~cyl  
faceting: <ggproto object: Class FacetNull, Facet, gg>
```

# No plot yet without geom\_point()!

- With ggplot2, you begin a plot with the function `ggplot()`.
- ❖ `ggplot()` creates a coordinate system that you can add layers to.
- ❖ The first argument of `ggplot()` is the dataset to use in the graph. So `ggplot(data = mpg)` creates an empty graph but it's not very interesting so I'm not going to show it here.
- ❖ You complete your graph by adding one or more layers to `ggplot()`: function `geom_point()` adds a layer of points to your plot, which creates a scatterplot.

```
g <- ggplot(mpg, aes(hwy, cyl))  
> print(g)  
Error: No layers in plot
```

```
> p <- g + geom_point()  
> print(p)
```

Explicitly save and print  
ggplot object



```
> g + geom_point()
```

Auto-print plot object  
without saving



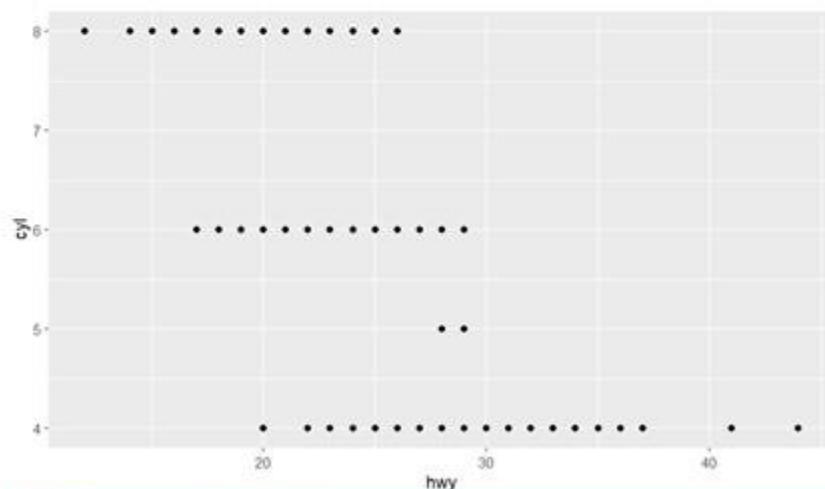
# First Plot with PointLayer

- To make a graph, replace the bracketed sections in the following code with a dataset, a geom function, or a collection of

```
ggplot(data = <DATA>)+  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

- Each geom function in ggplot2 takes a mapping argument.
- This defines how variables in your dataset are mapped to visual properties.
- The mapping argument is always paired with aes(), and the x and y arguments of aes() specify which variables to map to the x- and y-axes.
- ggplot2 looks for the mapped variable in the data

```
g <- ggplot(mpg, aes(hwy, cyl))  
g + geom_point()  
##Or, you can write:  
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = hwy, y = cyl))
```

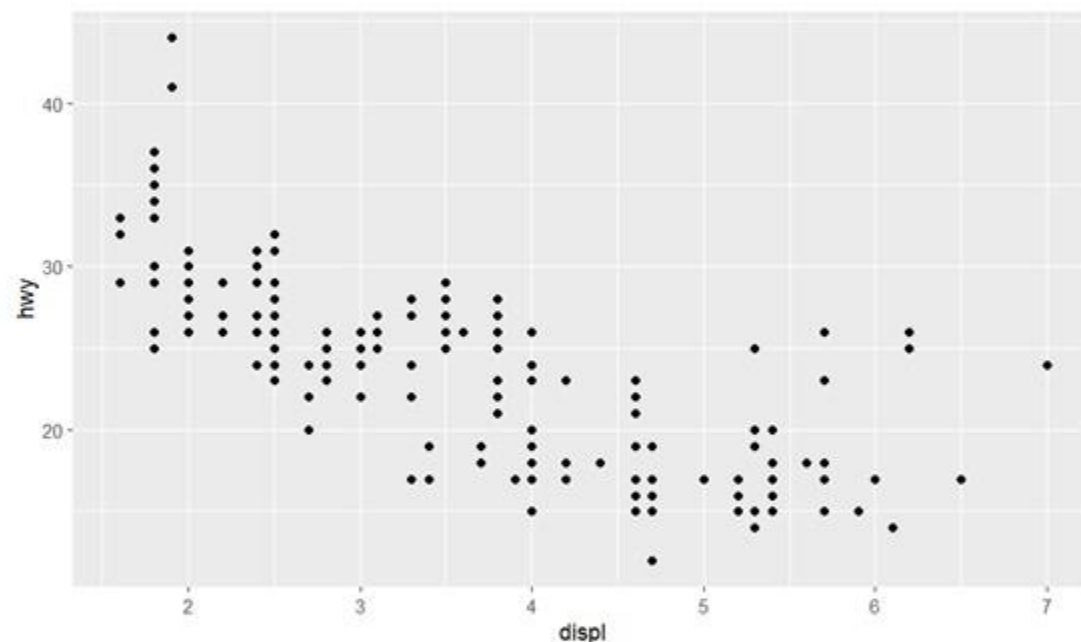


# geom

- A *geom* is the geometrical object that a plot uses to represent data.
- People often describe plots by the type of geom that the plot uses.
- **Example:**
  - bar charts use bar geoms,
  - line charts use line geoms,
  - boxplots use boxplot geoms, and so on.
  - Scatterplots break the trend; they use the point geom

# Creating ggplot

```
# To plot mpg, run this code to put displ on the x-axis and hwy on the y-axis  
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy))
```

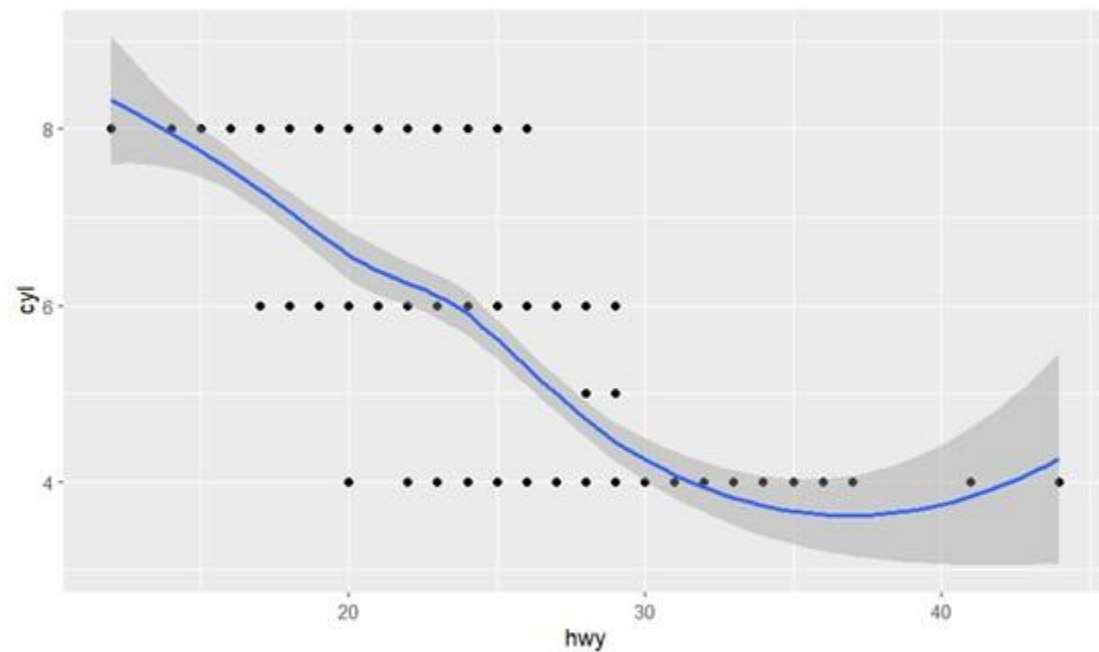


- **The plot shows a negative relationship between engine size (displ) and fuel efficiency (hwy).**

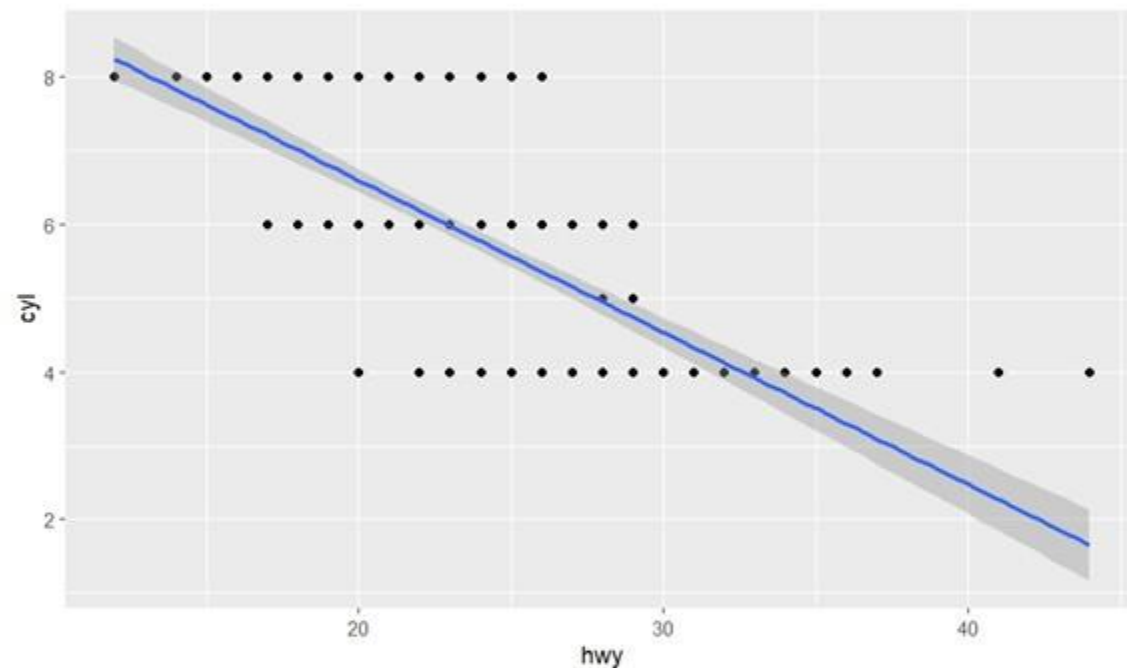
In other words, cars with big engines use more fuel.

# Adding more Layers:Smooth

- `g <- ggplot(mpg, aes(hwy, displ))`



`g + geom_point() + geom_smooth()`



`g + geom_point() + geom_smooth(method = "lm")`

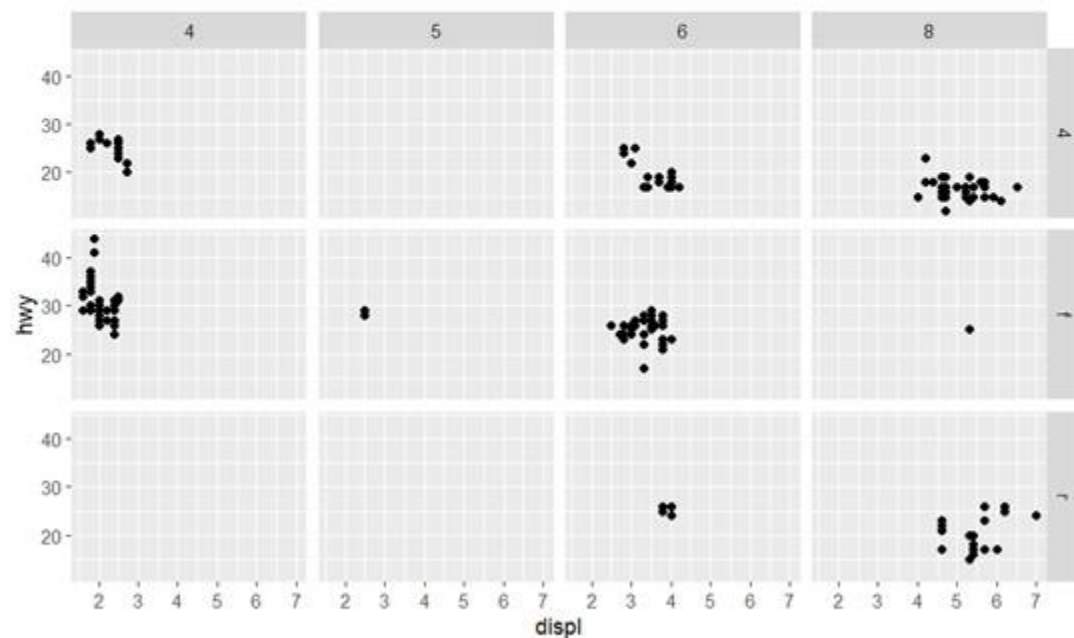
# Adding more Layers: Facets

- Split your plot into facets, subplots that each display one subset of the data.
- To facet your plot by a single variable, use `facet_wrap()`.

The first argument of `facet_wrap()` should be a formula, which you create with `~` followed by a variable name (here "formula" is the name of a data structure in R, not a synonym for "equation").

```
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy)) + facet_grid(drv ~ cyl)
```

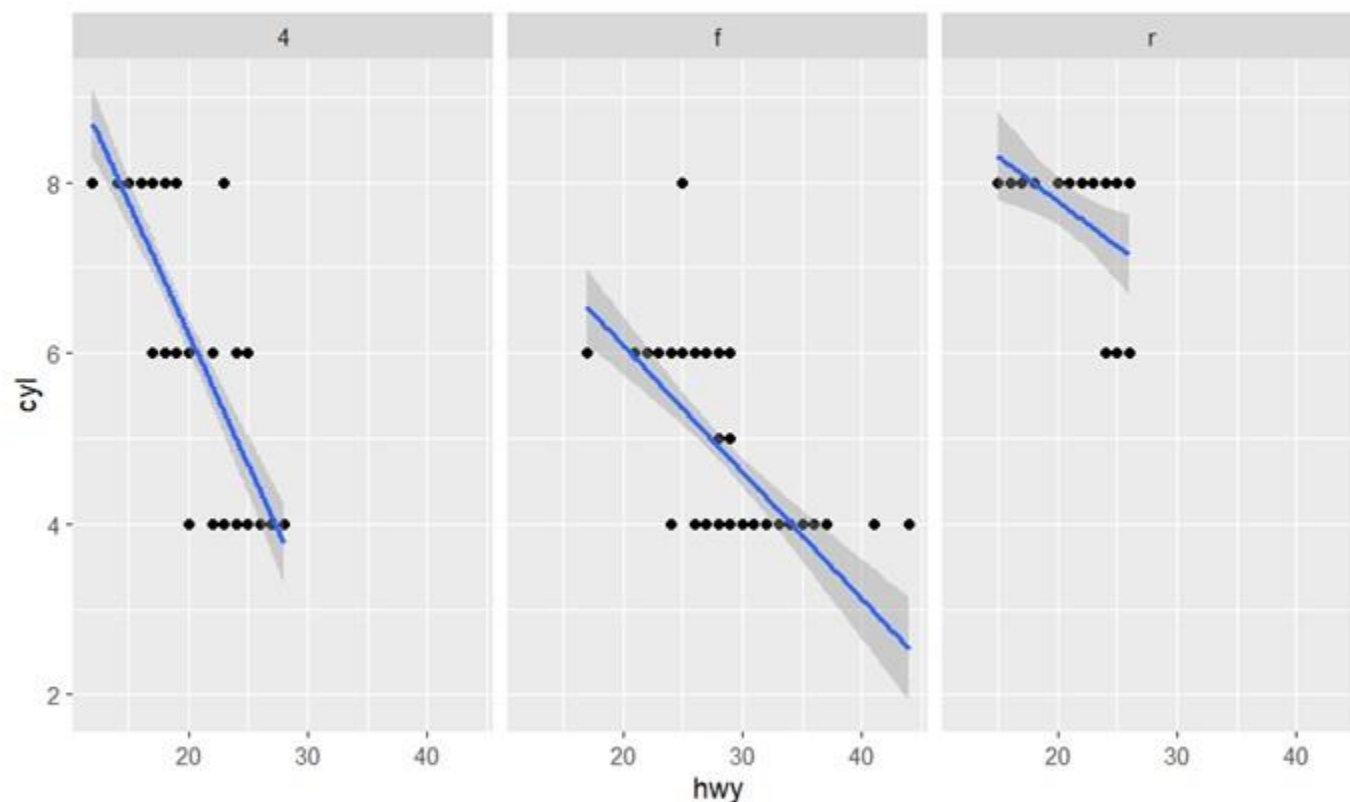
*#If you prefer to not facet in the rows or columns dimension, use a . in stead of a variable name, e.g., `+facet_grid(. ~ cyl)`.*



# Adding more Layers: Facets

To facet your plot on the combination of two variables, add `facet_grid()` to your plot call. The first argument of `facet_grid()` is also a formula.

This time the formula should contain two variable names separated by a `~`



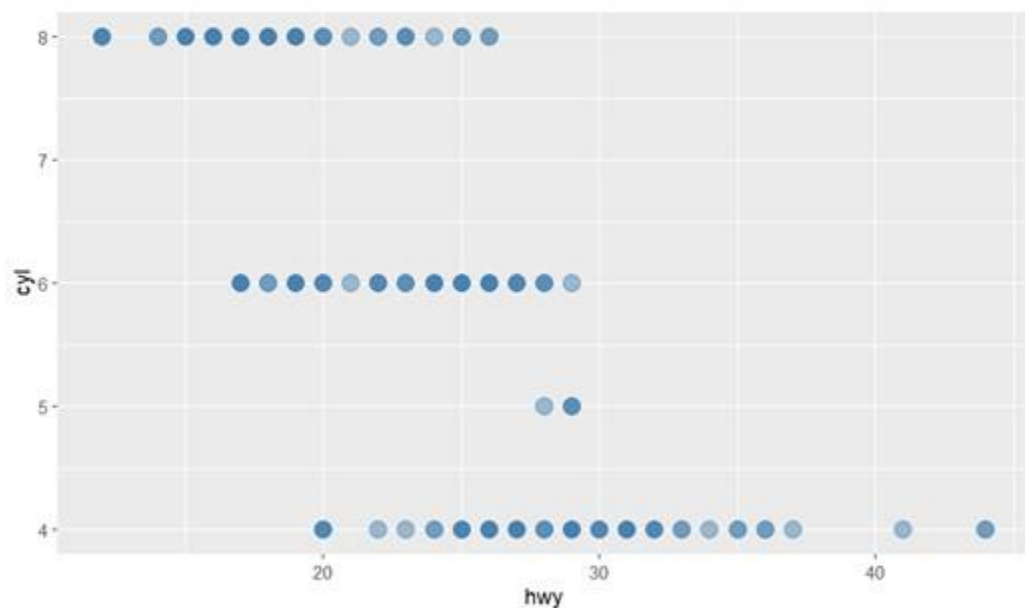
```
g + geom_point() + facet_grid(.~drv) + geom_smooth(method = "lm")
```

# Annotation

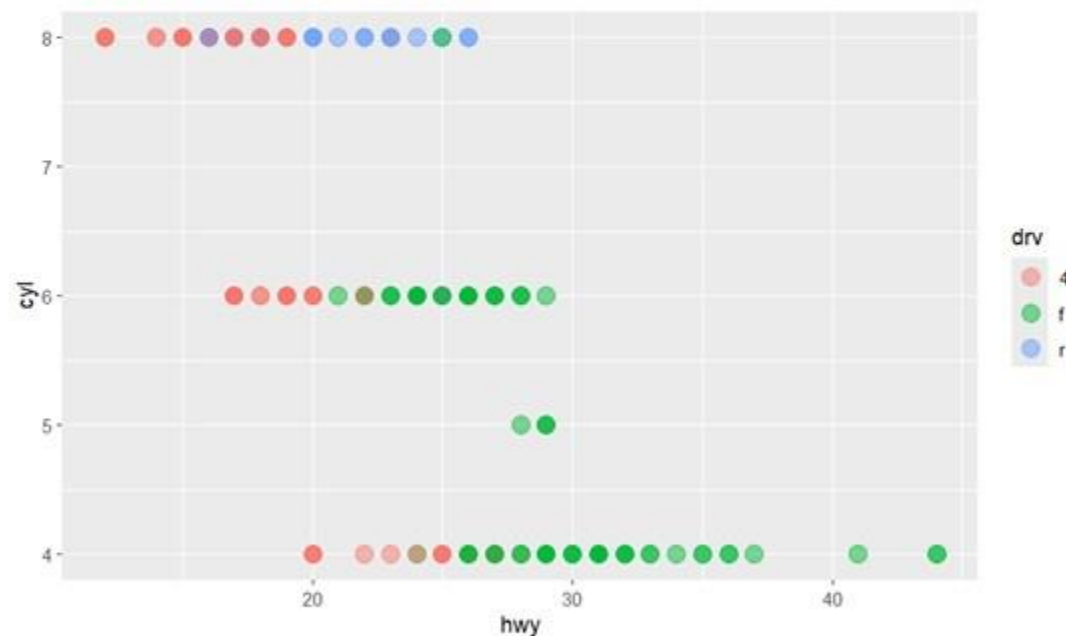
- Labels: `xlab()`, `ylab()`, `labs()`, `ggtitle()`
- Each of the "geom" functions has options to modify
- For things that only make sense globally, use `theme()`
  - Example: `theme(legend.position = "none")`
- Two standard appearance themes are included
  - `theme_gray()`: The default theme (gray background)
  - `theme_bw()`: More stark/plain

# Modifying Aesthetics

- You can add a third variable, like `drv`, to a two-dimensional scatterplot by mapping it to an *aesthetic*.
- An aesthetic is a visual property of the objects in your plot. Aesthetics include things like the size, the shape, or the color of your points.



```
g+ geom_point(color= "steelblue",size= 4, alpha= 1/2)
```

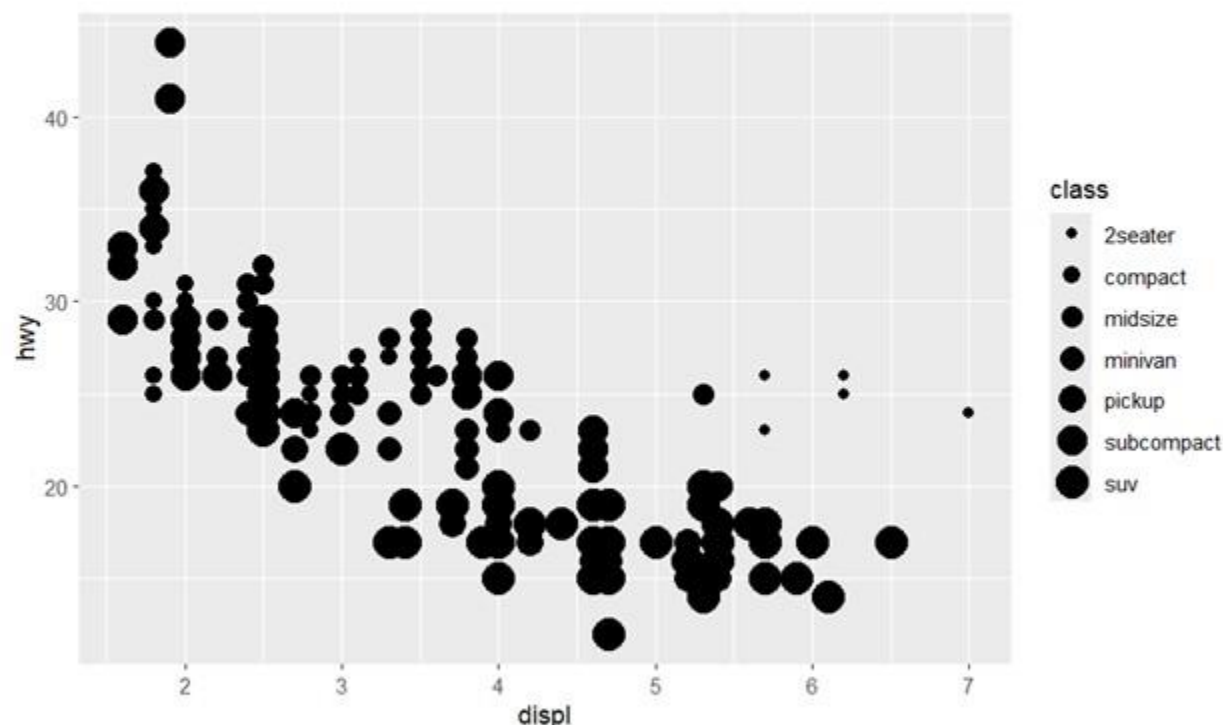


```
g+ geom_point(aes(color= drv),size= 4, alpha= 1/2)
```

# Modifying Aesthetics

- **Map "class" variable to the size aesthetic**

In this case, the exact size of each point would reveal its class affiliation. We get a warning here, because mapping an unordered variable (class) to an ordered aesthetic (size) is not a good idea:



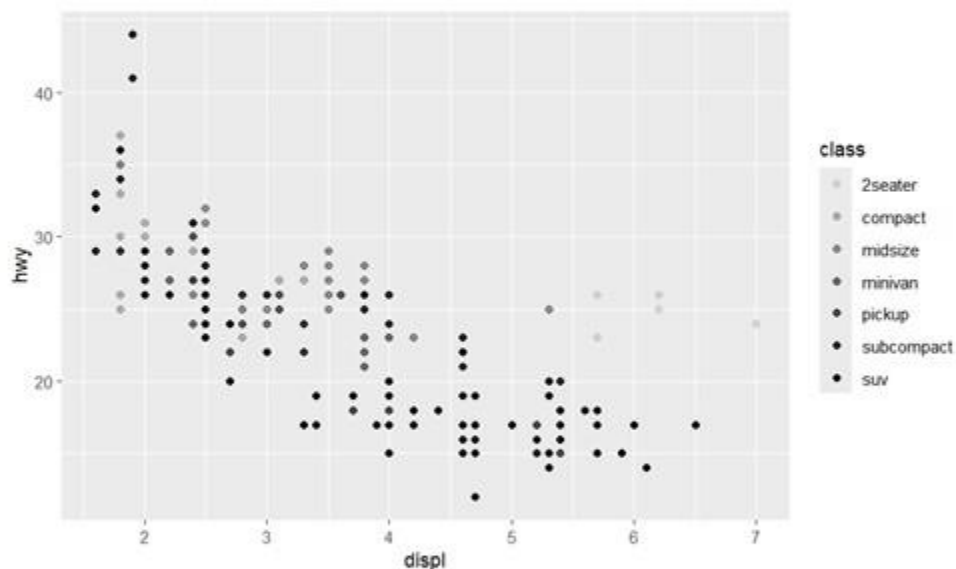
```
> ggplot(data = mpg) +  
+   geom_point(mapping = aes(x = displ, y = hwy, size=class))
```

Warning message:  
Using **size** for a discrete variable is not advised.

# Modifying Aesthetics

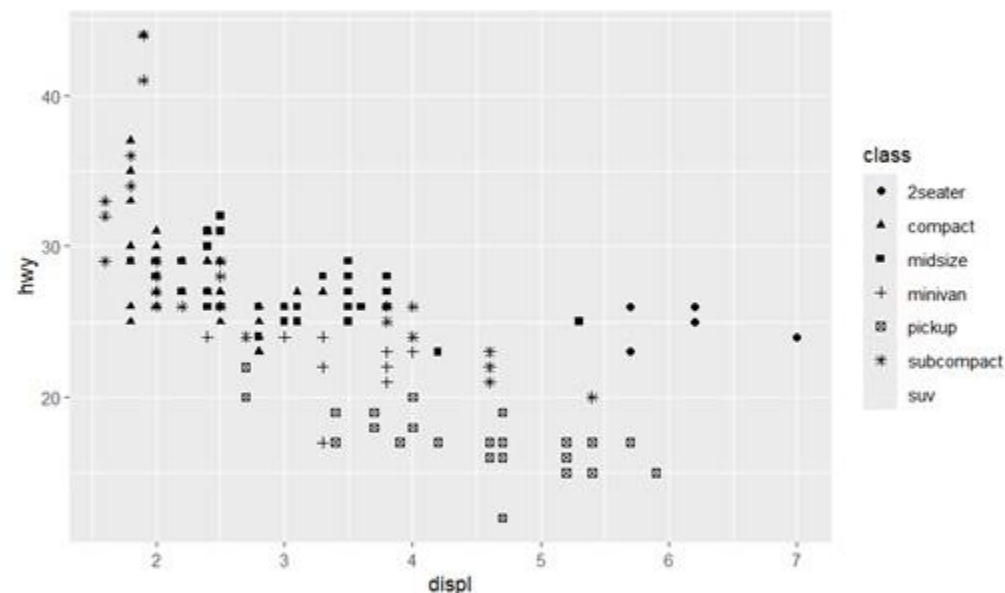
**Map class to the alpha aesthetic, which controls the transparency of the points, or the shape of the points:**

- In this case, the exact size of each point would reveal its class affiliation. We get a warning here because mapping an unordered variable (class) to an ordered aesthetic (size) is not a good idea:
- `shape = class`: What happened to the SUVs? ggplot2 will only use six shapes at a time. By default, additional groups will go unplotted when you use this aesthetic.



```
> ggplot(data = mpg) +  
+   geom_point(mapping = aes(x = displ, y = hwy, alpha=class))
```

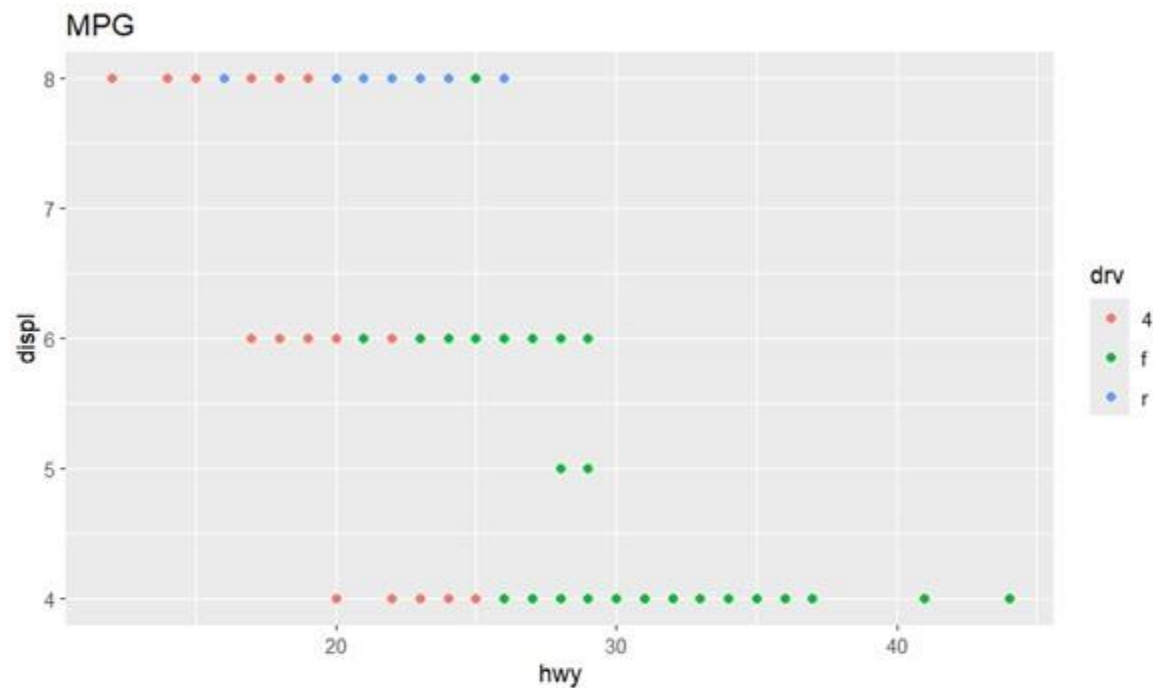
Warning message:  
Using alpha for a discrete variable is not advised.



```
> ggplot(data = mpg) +  
+   geom_point(mapping = aes(x = displ, y = hwy, shape=class))
```

Warning messages:  
1: The shape palette can deal with a maximum of 6 discrete values because more than 6 becomes difficult to discriminate  
i you have requested 7 values. Consider specifying shapes manually if you need that many have them.  
2: Removed 62 rows containing missing values or values outside the scale range ('geom\_point()').

# Modifying Labels

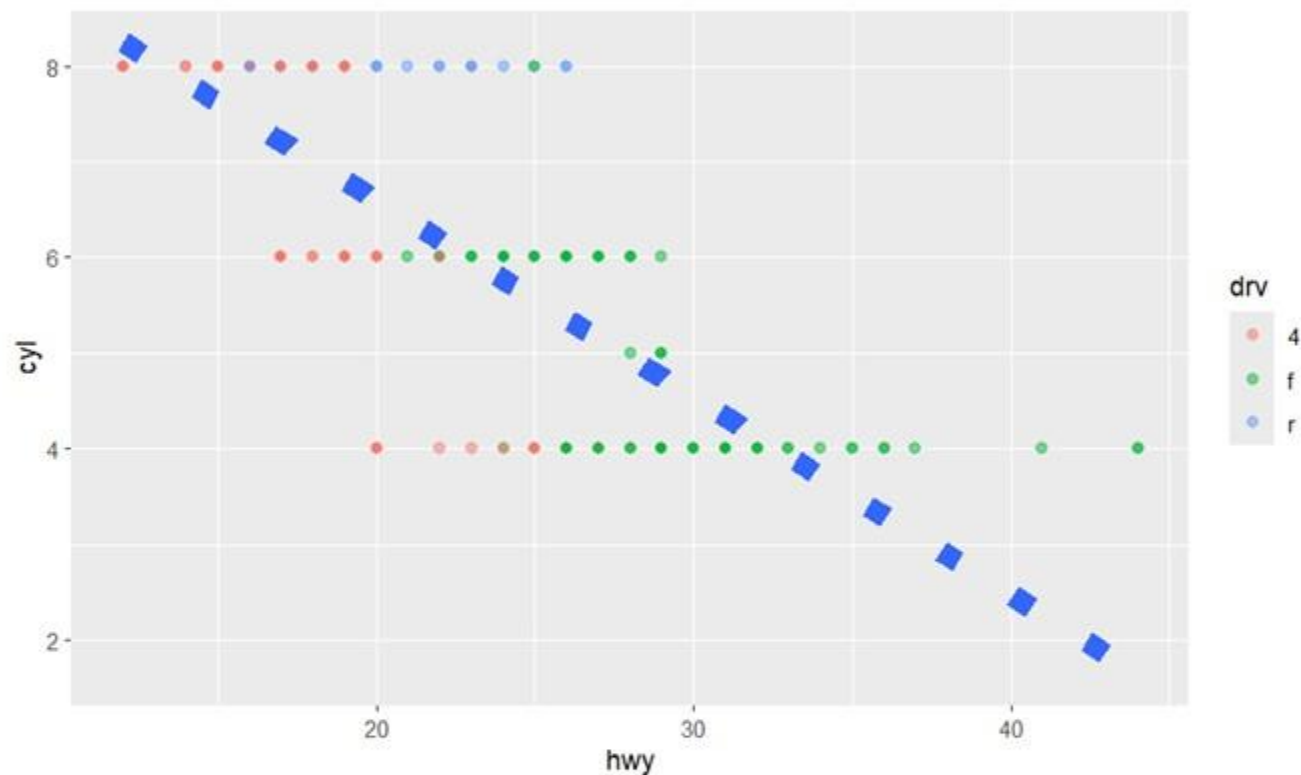


labs() function for  
modifying titles and  
x-, y-axis labels

```
> g + geom_point(aes(color = drv)) + labs(title = "
```

```
MPG") + labs(x = "hwy", y = "displ")
```

# Customizing the Smooth



```
g + geom_point(aes(color= drv), size = 2, alpha = 1/2) +  
geom_smooth(size = 4, linetype = 3, method = "lm", se = FALSE)
```

↑  
Modified  
smoother

# Ggplot2: Summary

- ggplot2 is very powerful and flexible if you learn the "grammar" and the various elements that can be tuned/modified.
- Many more types of plots can be made; explore and mess around with the package

**<https://w.vw.oreilly.com/library/view/r-for-data/9781491910382/ch01.html>**

# Plotting System: Summary

- Base: "artist's palette" model
- Lattice: Entire plot specified by one function; conditioning
- ggplot2: Mixes elements of Base and Lattice

# The Process of Making a Plot

When making a plot one must first make a few considerations (not necessarily in this order):

- Where will the plot be made? On the screen? In a file?
- How will the plot be used?

Is the plot for viewing temporarily on the screen?

Will it be presented in a web browser?

Will it eventually end up in a paper that might be printed?

Are you using it in a presentation?

- Is there a large amount of data going into the plot? Or is it just a few points?
- Do you need to be able to dynamically resize the graphic?

# The Process of Making a Plot

- **What graphics system will you use: base, lattice, or ggplot2?**

These generally cannot be mixed.

- **Base graphics are usually constructed piecemeal, with each aspect of the plot handled separately through a series of function calls;**

this is often conceptually simpler and allows plotting to mirror the thought process

- **Lattice graphics are usually created in a single function call, so all of the graphics parameters have to be specified at once;**

▶ specifying everything at once allows R to automatically calculate the necessary spacings and font sizes.

- **ggplot2 combines concepts from both base and lattice graphics but uses an independent implementation**
- **We focus on using the base plotting system to create graphics on the screen device.**

# What is a Graphics Device?

- **A graphics device is something where you can make a plot appear**  
When you make a plot in **R** it has to be "sent" to a specific graphics device
- **The most common place for a plot to be "sent" is :**
  - the screen device*
  - A window on your computer (screen device)*
  - A pdf file (file device)*
  - A png or jpeg file (file device)*

# What is a Graphics Device?

- **When making a plot, you need to consider how the plot will be used to determine what device the plot should be sent to.**

The list of devices is found in `?Devices`; there are also devices created by users on CRAN

- **For quick visualizations and exploratory analysis, usually you want to use the screen device**

Functions like `plot` in base, `xyplot` in lattice, or `qplot` in ggplot2 will by default send a plot to the screen device

On a given platform (Mac, Windows, Unix/Linux) there is only one screen device

- **For plots that may be printed out or be incorporated into a document (e.g. papers/reports, slide presentations), usually a *file device* is more appropriate**

There are many different file devices to choose from

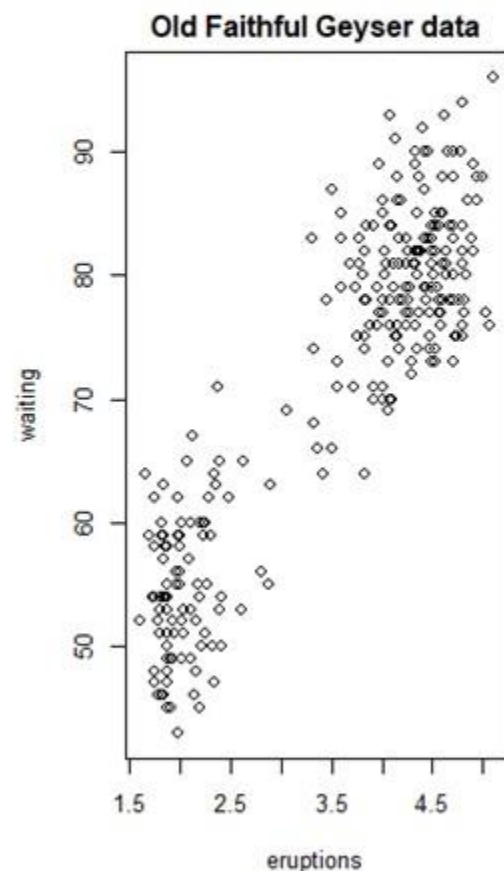
- **NOTE: Not all graphics devices are available on all platforms (i.e. you cannot launch the windows() on a Mac)**

# How Does a Plot Get Created?

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

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

```
library(datasets)
with(faithful, plot(eruptions, waiting)) ## Make plot appear on screen device
title(main = "Old Faithful Geyser data") ## Annotate with a title
```



# How Does a Plot Get Created?

- **The second approach to plotting is most commonly used for file devices:**
  1. Explicitly launch a graphics device
  2. Call a plotting function to make a plot (note: if you are using a file device, no plot will appear on the screen)
  3. Annotate plot if necessary
  4. Explicitly close graphics device with `dev.off()` (this is very important!)

```
pdf(file = "myplot.pdf") ## Open PDF device; create 'myplot.pdf' in my working directory
## Create plot and send to a file (no plot appears on screen)
with(faithful, plot(eruptions, waiting))
title(main = "Old Faithful Geyser data") ## Annotate plot; still nothing on screen
dev.off() ## Close the PDF file device
## Now you can view the file myplot.pdf on your computer under the working directory
```

# Multiple Open Graphics Devices

- It is possible to open multiple graphics devices (screen, file, or both), for example when viewing multiple plots at once
- Plotting can only occur on one graphics device at a time
- The currently active graphics device can be found by calling `dev.cur()`
- Every open graphics device is assigned an integer.
- You can change the active graphics device with `dev.set(<integer>)` where `<integer>` is the number associated with the graphics device you want to switch to

# Copying Plots

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

dev.copy: copy a plot from one device to another

dev.copy2pdf: specifically copy a plot to a PDF file

```
library(datasets)with(faithful, plot(eruptions,waiting)) ## Create plot on screen device
title(main = "Old Faithful Geyser data") ## Add a main title
dev.copy(png, file = "geyserplot.png") ## Copy my plot to a PNG file
dev.off() ## Don't forget to close the PNG device!
```

# Graphic Device: Summary

- Plots must be created on a graphics device
- The default graphics device is almost always the screen device, which is most useful for exploratory analysis
- File devices are useful for creating plots that can be included in other documents or sent to other people

# Questions