Notes on ggplot2

Melissa Dell

February 2016

Outline

Graphics

- Communicating results in a concise and clear manner using figures is central for writing compelling papers and for designing effective presentations.
- R's ggplot2 package provides a powerful tool for visualizing and communicating results graphically
- Stata can be cumbersome if you want to move beyond its basic pre-canned graphics capabilities

The Grammar of Graphics

- The gg in ggplot2 stands for the grammar of graphics.
- This package was designed by the statistician Hadley Wickham, whose innovation was to postulate that just as written communication is constructed by combining small building blocks - nouns, pronouns, verbs, adjectives, adverbs, articles, etc - into complex sentences following the rules of grammar, figures can be thought of in the same way.

The Grammar of Graphics

- This is a powerful analogy that considerably simplifies complex figures by breaking them down into their component parts
- Once you understand the building blocks and how the grammar of graphics is used to combine them, you can easily create increasingly sophisticated plots.
- You are not limited to a set of pre-canned graphs but rather can create graphics that are finely tailored to your particular application
- It is actually straightforward to learn because there is a set of core principles and few special cases

A reference

- Unless otherwise noted, all figures, tables, and examples in these slides are drawn from Hadley Wickham's book: *Ggplot2: Elegant Graphics for Data Analysis*, available for free download from Hollis.
- These notes organize and summarize the most useful information in that book, which I highly recommend reading carefully.
- The book is long and organized in a way that makes it plausibly most useful to someone with some familiarity with the package already. This lecture is an intro for those with little or no knowledge of ggplot2 and should be seen as a companion to the book.
- Read the notes carefully first, then the book.

This Lecture

- I first provide a more theoretical overview of how ggplot2 functions
- Then I provide more details on how the actual implementation of the code works
- We will not likely get through the entire presentation, but it should be self-explanatory

Outline

The Grammar of Graphics

- ggplot2 works in a layered fashion: you start with a layer showing the raw data and then cad add annotations, statistical summaries, scales, etc.
- If you just have a vector or set of vectors and want to make a quick plot to visualize the data, you can use qplot (see ch. 2 of "ggplot2: Elegant Graphics for Data Analysis"), henceforth "the ggplot2 book"

A data layer

Layers create the objects we perceive on the plot and are composed of the above four parts:

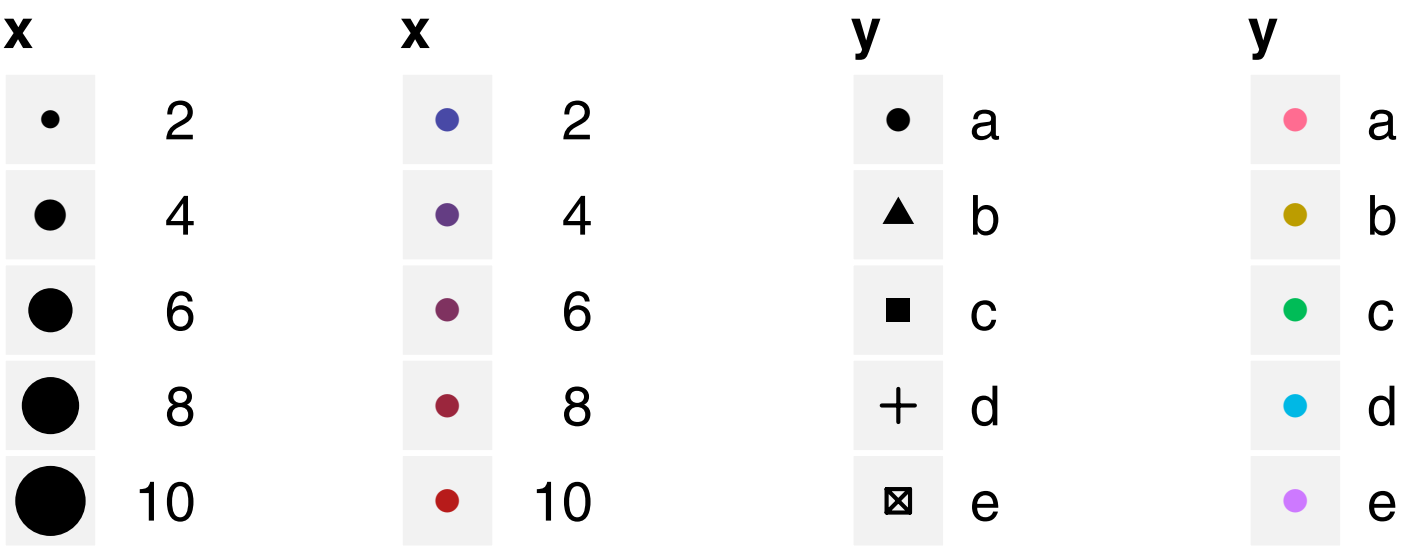
1. Data and aesthetic mapping (aes)
2. A statistical transformation (stat)
3. A geometric object (geom)
4. A position adjustment

GG Building Blocks: The data frame and aesthetic mapping

- The data frame consists of the raw data
- The aesthetics mappings: these relate variables in the data frame to visual representations
- The units in your data (i.e. dollars or years), have no meaning to the computer. They need to be converted into physical units - such as pixels, shapes, colors (represented by a hexadecimal string) - that can be displayed. For example, in a scatter plot each point has a size, color, and shape.
- The inverse of the scale is used to construct the legend and axes

GG Building Blocks: The data frame and aesthetic mapping

Here's some legend examples: legends are inverse aesthetic mappings from graphics to the raw data



GG Building Blocks: The geometry

Geoms are the geometric objects that form a plot. Examples include:

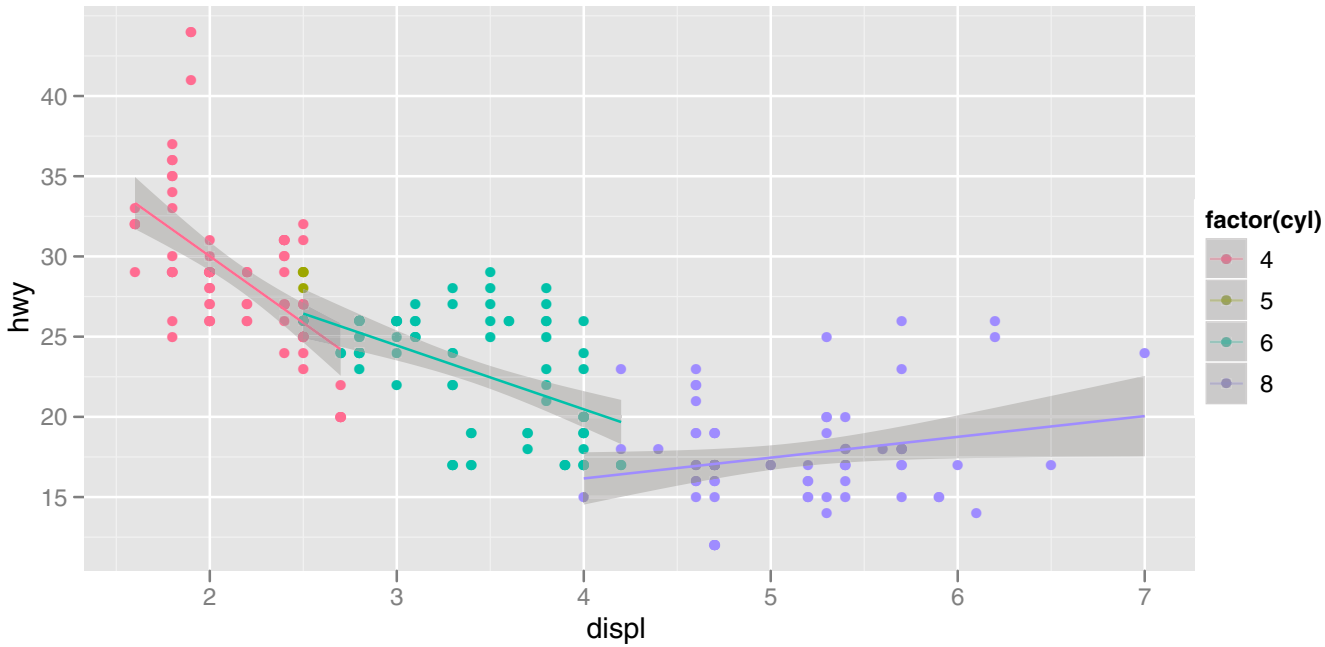
- points
- lines
- bars
- boxplots

GG Building Blocks: Statistical Transformations

Rather than plotting the data, statistical transformations (stats) plot a statistically transformed version of the raw data. For example:

- Regression lines
- Kernel regression
- Mean

GG Building Blocks: Statistical Transformations



GG Building Blocks: Position Adjustment

The position adjustment deals with overlapping graphs objects.

Adjustment	Description
dodge	Adjust position by dodging overlaps to the side
fill	Stack overlapping objects and standardise have equal height
identity	Don't adjust position
jitter	Jitter points to avoid overplotting
stack	Stack overlapping objects on top of one another

"Ggplot2: Elegant Graphics for Data Analysis" by Hadley Wickham.

GG Building Blocks: Position Adjustment

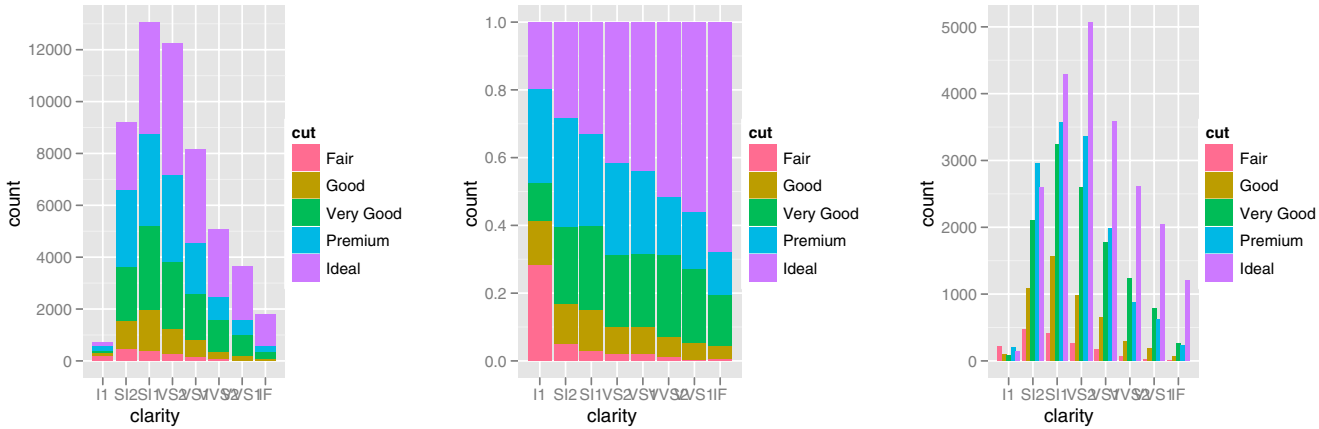


Fig. 4.8: Three position adjustments applied to a bar chart. From left to right, stacking, filling and dodging.

Outline

Objects that belong to the figure as a whole

There can be many layers in a single plot, but there are also gg building blocks that apply to the plot as a whole:

- A coordinate system
- The faceting specification
- Annotations

GG Building Blocks: Coordinate Systems

- A coordinate system (coord) maps the position of objects onto the plane of the plot.
- Typically the Cartesian coordinate system is used

GG Building Blocks: Faceting

This is a very useful feature of ggplot2. Faceting allows you to create a series of plots each showing a different subset of the whole dataset. The faceting specification indicates:

- which variable(s) should be used to split the data
- whether position scales should be free or constrained

Annotations

These are needed to complete the figure. They include

- the plot background
- the plot title

Outline

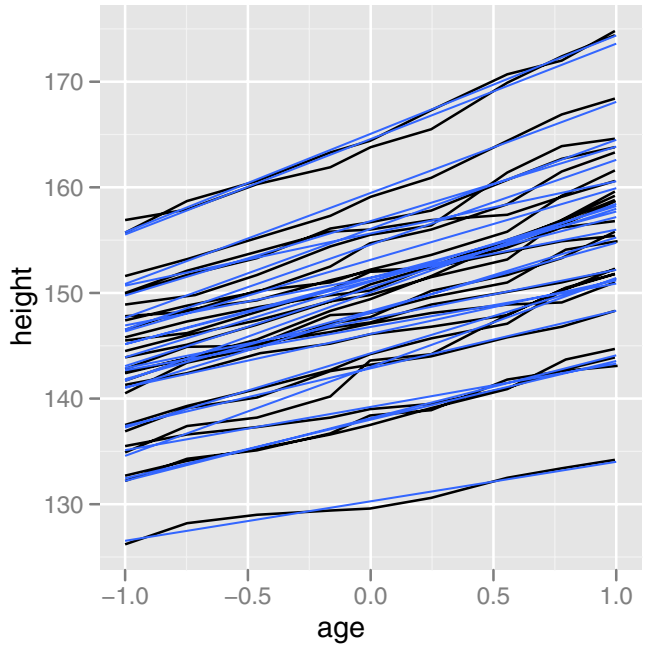
The overall plot

Overall, a plot consists of the following

- A default dataset and aesthetic mappings
- One or more layers, each composed of a geometric object, statistical transformation, position adjustment, and - optionally - a dataset and aesthetic mapping. If the latter is not specified, the default data and mapping will be used.
- A scale for each aesthetic mapping
- A coordinate system
- The faceting specification
- The annotations

Using multiple datasets and aesthetics across different geom layers

- This is incredibly useful when you want to plot multiple datasets on the same plot
- For example, you may wish to plot the raw data and predictions from an external model



Outline

More details about specifying the code

Your code and plots can be built up layer by layer. You need to start by creating the plot object itself. Taking the example from the ggplot2 book

```
p <- ggplot(diamonds,
   aes(x = carat, y = price, colour = cut))
```

- Diamonds is the default dataset and aes() gives the default aesthetic mapping. We cannot view this plot until we add a layer.
- The data must be in a data frame and cannot be a series of vectors.
- The data is stored in the plot object as a copy, not a reference.

Specifying aesthetics

The aesthetic (aes) function takes a list of aesthetic-variable pairs, such as:

```
aes(x = carat, y = price, colour = cut)
```

The first two arguments can be left without names, in which case your code would be:

```
aes(carat, price, colour = cut)
```

Layers

A minimal layer could simply contain a geom:

```
p <- ggplot(diamonds,
   aes(carat, price, colour = cut))
p <- p + layer(geom = ``point'')
```

This layer uses the plot defaults for the data and aesthetic mapping and it uses default values for the stat and position adjustment, two optional arguments of layer.

Layers

Here is a more complete layer:

```
p <- ggplot(diamonds, aes(x = carat))
p <- p + layer(
   geom = "bar",
   geom_params = list(fill = "steelblue"),
   stat = "bin",
   stat_params = list(binwidth = 2)
)
```

This is verbose and can be simplified by using ggplot2 shortcuts, which utilize the fact that every geom has a default stat and every stat a default geom (i.e. a bar graph implies a stat that bins the raw data by default)

ggplot shortcuts: stats and geoms

The shortcut syntax

```
geom_XXX(mapping, data, ..., geom, position) or
stat_XXX(mapping, data, ..., stat, position)
```

can be used to specify a layer

- mapping (optional): aesthetic mappings, specified using aes()
- data (optional): used if you need to override the default dataset
- ...: parameters such as the bin width of a histogram or the bandwidth of a lowess smoother. This sets this property to a fixed value (in contrast to aesthetics, which map to a variable in the dataset)
- geom or stat (optional): allows you to override the default stat for a geom or default geom for a stat
- position (optional): allows you to optionally choose a method for adjusting overlapping objects

An example

```
p <- ggplot(diamonds, aes(x = carat))
p <- p + layer(
   geom = "bar",
   geom_params = list(fill = "steelblue"),
   stat = "bin",
   stat_params = list(binwidth = 2)
)

versus

p <- p + geom_histogram(binwidth = 2,
    fill = "steelblue")
```

Combining geoms with non-default stats

- See p. 67 for a complete list of the default stats corresponding to each geom - i.e. the bar geom and histogram correspond to the bin stat (these are basically the same; histograms can have weights) and the point geom corresponds to the identity stat.
- Usually sticking with the defaults is preferable; people may not react well to seeing something too unusual.

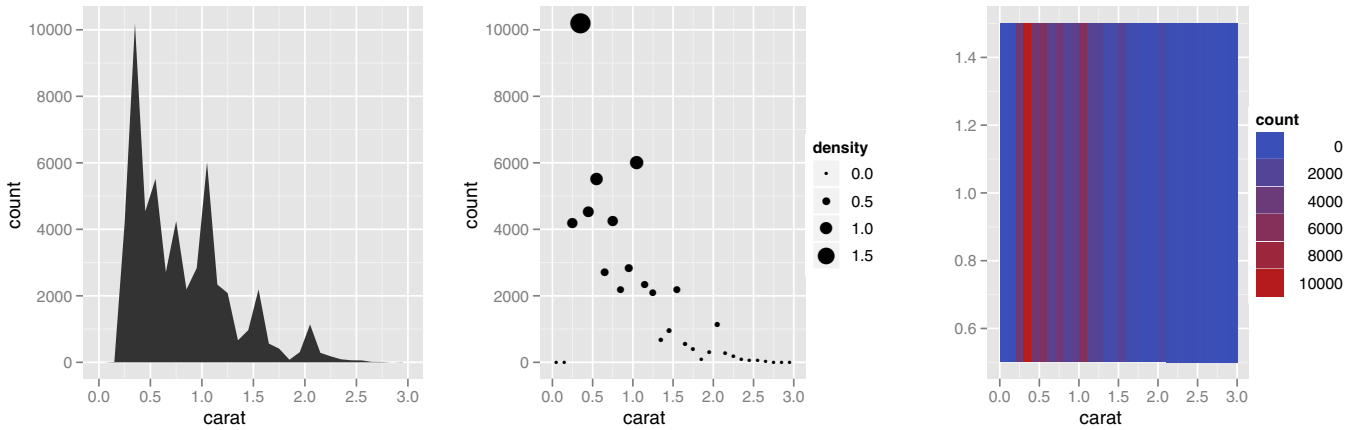


Fig. 4.10: Three variations on the histogram. (Left) A frequency polygon; (middle) a scatterplot with both size and height mapped to frequency; (right) a heatmap representing frequency with colour.

Geoms and aesthetics

- Each geom has a required set of aesthetics and a larger set of aesthetics it understands
  - For example, the point geom (scatterplot) requires x and y positions; it accepts color, size and shape aesthetics.
  - The bar geom requires height (ymax) and accepts width, border color, and fill color aesthetics.
- See p. 66 of the ggplot2 book for a complete listing for all geoms.

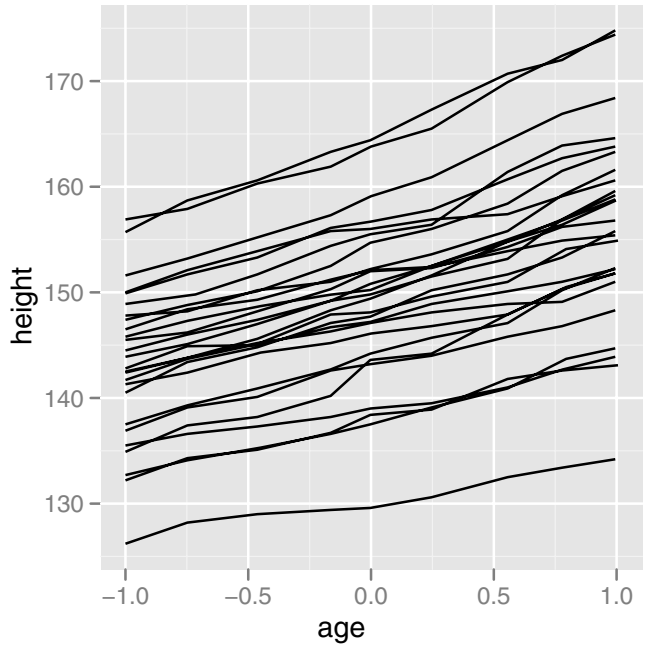
Using the same layer repeatedly

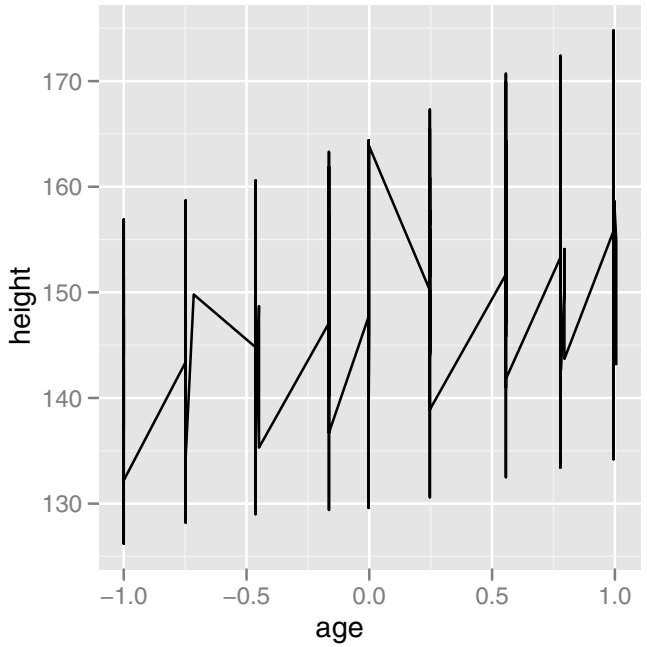
Layers are R objects that can be stored as variables and used repeatedly in different plots. For example:

```
fit <- geom_smooth(method = "lm", se = F,
   colour = alpha("steelblue", 0.5), size = 2)
```

Grouping

Sometimes you may need to group the data into separate plots within the same layer - for example, in a line plot.





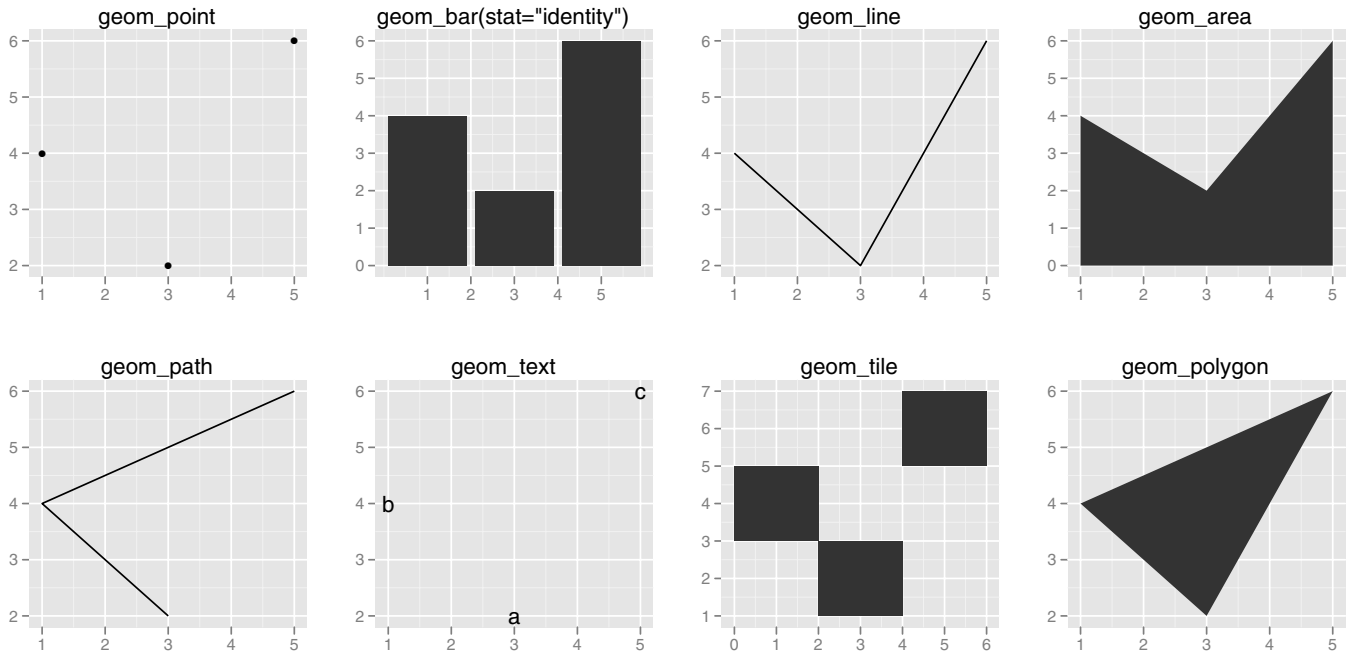

```
p <- ggplot(Oxboys, aes(age, height, group = Subject))
   + geom_line()
```

Examples of geoms

Ch. 5 of ggplot book provides example code snippets

- geom_area: A line plot filled to the x axis. Multiple groups are stacked.
- geom_bar: A bar chart.
- geom_line: A line chart. The group aesthetic determines which observations are connected, from left to right. geom_path is similar but connects in the order that the obs. appear in the data
- geom_point: Scatter plot
- geom_polygon: Draws filled paths. Each vertice is a row.
- geom_text: Adds labels to specified points.
- geom_tile: Tiles form a regular tessellation of the plane and typically have the fill aesthetic mapped to another variable.

Basic Geom Examples

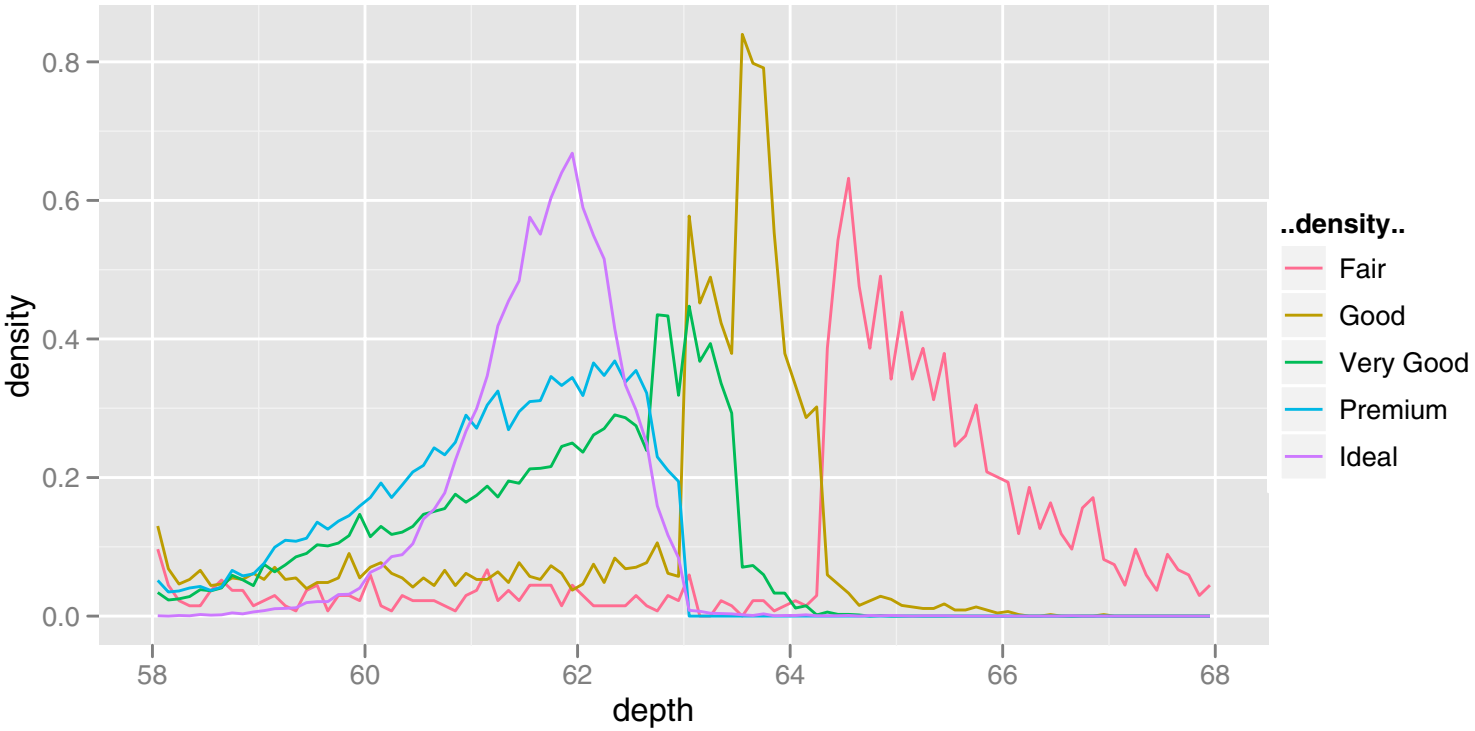


Geoms to display distributions

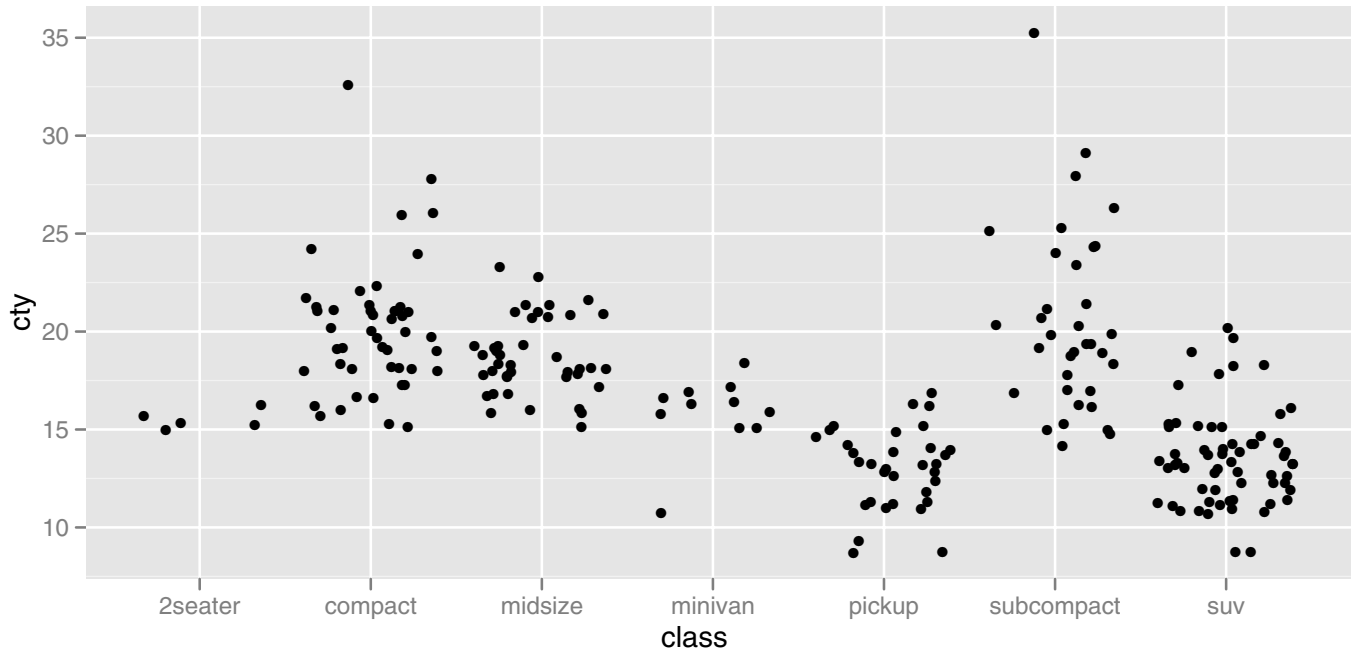
Again, see ch. 5 (p. 70) of the ggplot book

- geom_histogram: need to set the bin width parameter, or can set the exact bin endpoints
- geom_freqpoly: shows frequencies with connecting lines rather than bins
- geom_boxplot: a box and whiskers plot; shows a continuous variable conditioned by a categorical variable.
- geom_jitter: a way to look at discrete distributions by adding random noise; generally works best for small datasets
- geom_density: A frequency polygon that is smoothed using kernel smoothers

A frequency polygon



A jitter plot



Making dense plots more legible

If there are many data points, it can be difficult to visualize the raw data. If the data are discrete, you can use alpha blending transparency:

```
g=g+geom_point(colour=alpha("black", 1/alpha))
```

where α is the number of points that must be overlaid to produce a solid point. The max is $1/256$. Discrete data can be jittered.

Addressing Overplotting

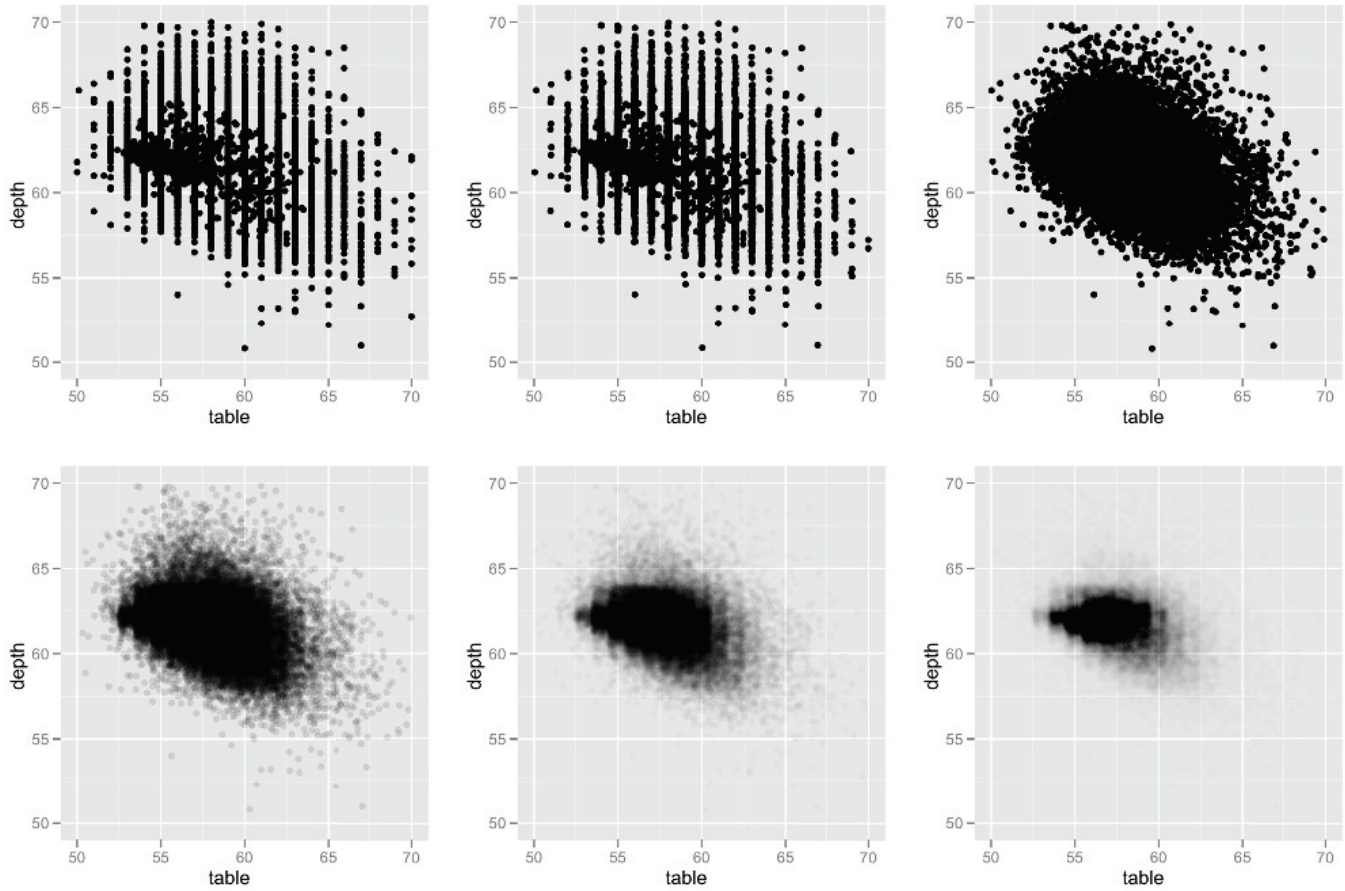
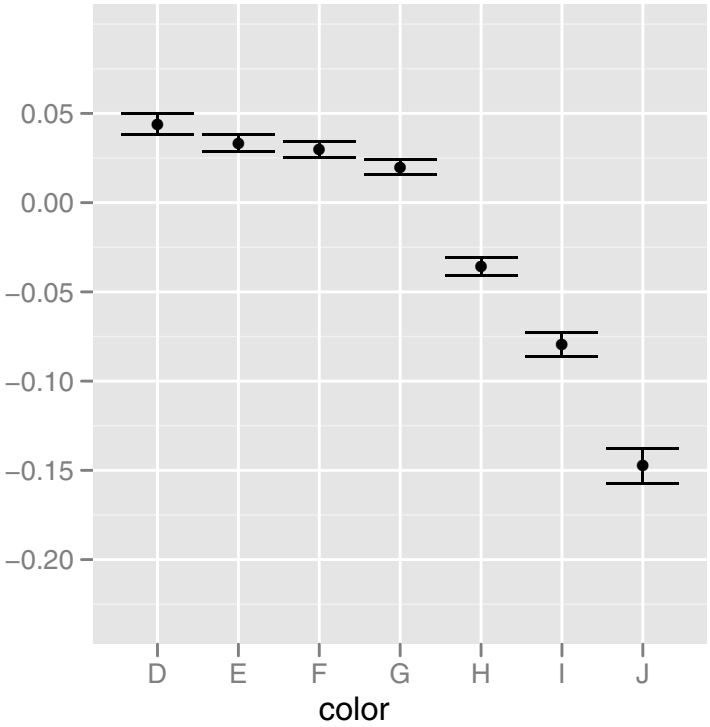


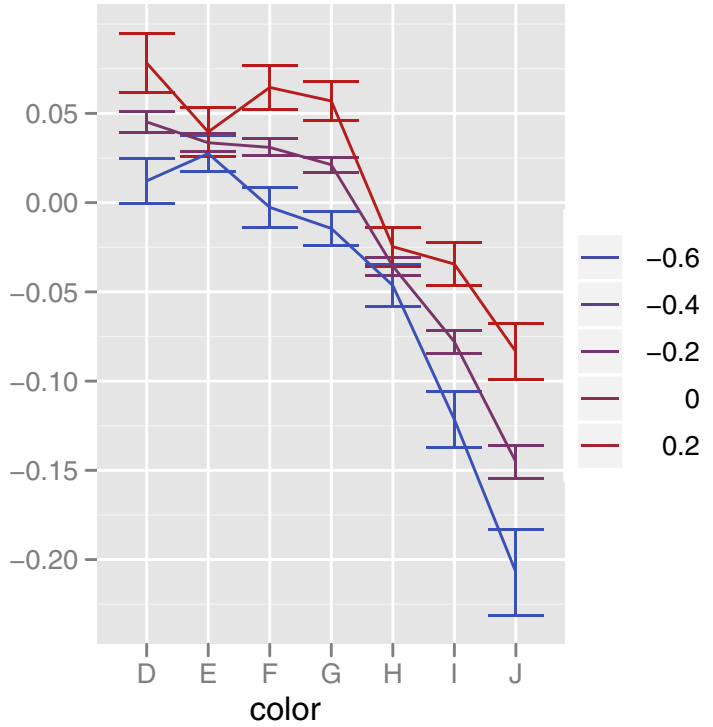
Fig. 5.9: A plot of table vs. depth from the diamonds data, showing the use of jitter and alpha blending to alleviate overplotting in discrete data. From left to right: geom point, geom jitter with default jitter, geom jitter with horizontal jitter of 0.5 (half the gap between bands), alpha of 1/10, alpha of 1/50, alpha of 1/200.

Plotting confidence intervals

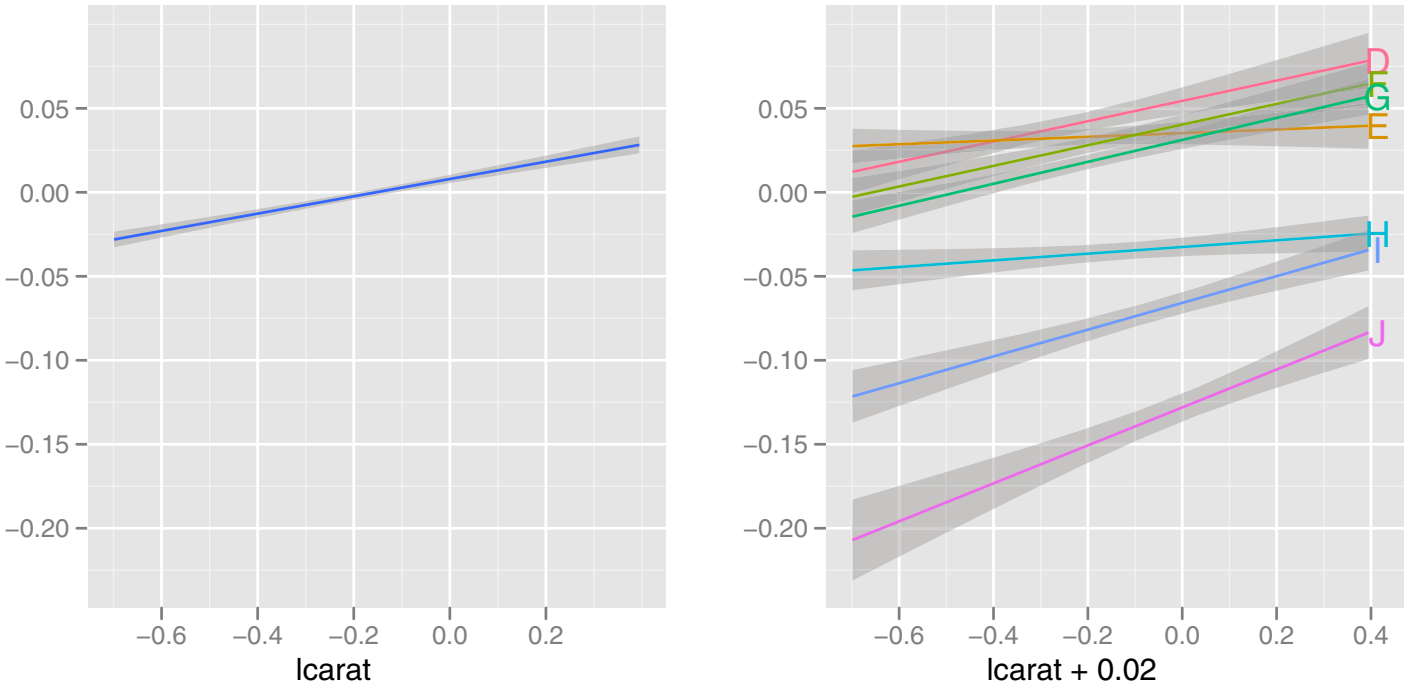
- For continuous data use geom_ribbon to show the range or geom_smooth to show the range and point estimate
- For discrete data you can use geom_errorbar or geom_linerange to show the confidence intervals, or analogously geom_crossbar or geom_pointrange to show the confidence intervals and point estimates.

Plotting confidence intervals





Plotting confidence intervals



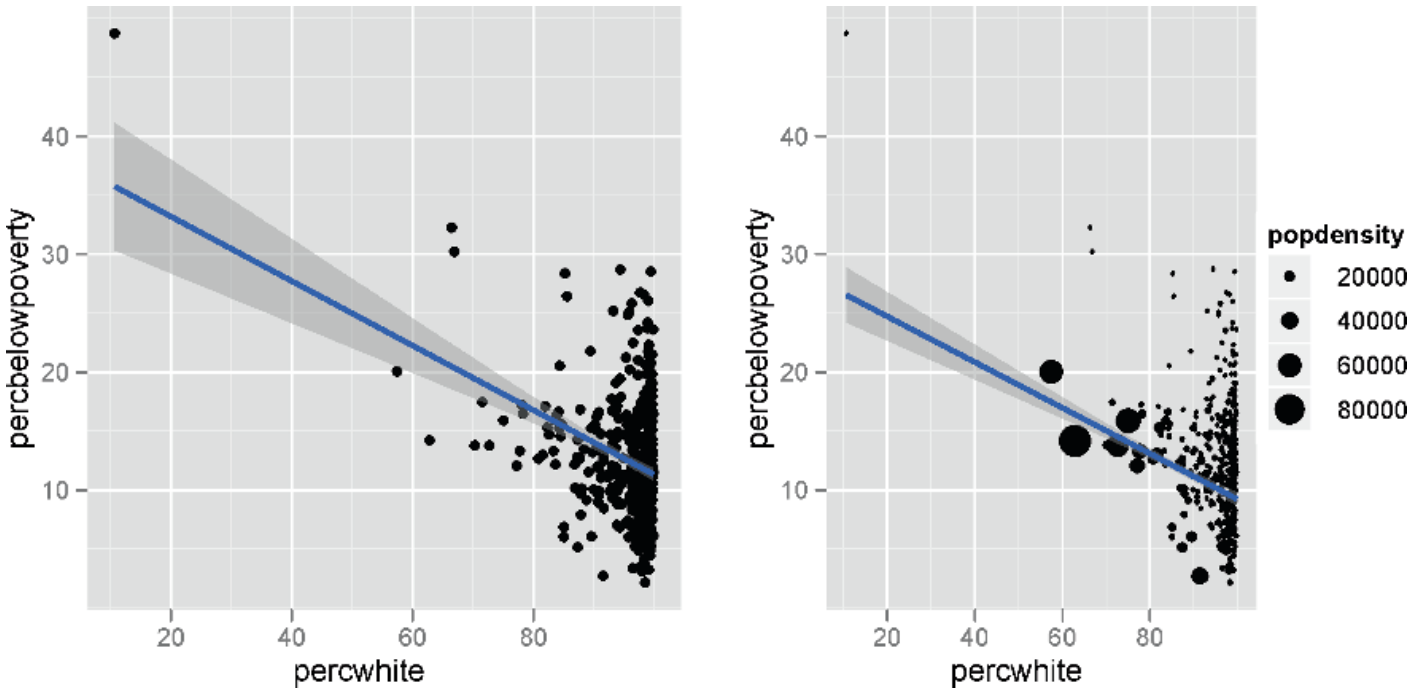
Annotations

- geom_text: put your annotations in a data frame and add using geom_text
- geom_vline and geom_hline: add horizontal and vertical lines
- geom_abline: adds line with arbitrary slope and intercept
- geom_rect: add a rectangle using the aesthetics xmax, xmin, ymax, ymin

Weights

- For simple data plots, the size of points, etc can be made proportional to a weight, such as population, using the size aesthetic
- For statistical transformations (i.e. geoms that don't use the identity stat function), there is a weight aesthetic

Weights


percbelowpoverty
40
30
20
10
20
40
60
80
percwhite
popdensity
20000
40000
60000
80000

Outline

Scales

- Scales are a function from the data space to the aesthetic space
- Scales turn data into graphics and determine the axes and legends, which are guides to understand how the data has been mapped to visual symbols
- A scale is required for every aesthetic used in the plot
- ggplot2 provides defaults, but understanding how scales work will give you more control over your figures' appearance

The Domain and Range

- For a discrete variable, the domain of the scale is a set of values that are stored as a factor, character, or logical vector
- For a continuous variable, the domain is an interval on the real line, stored as a numeric vector of length two: i.e. $[0, 5]$
- For discrete domains, the range is a vector of aesthetic values corresponding to each value of the domain
- For a continuous variable, it is a single dimensional path through a given space, such as the color space

ggplot's four scale groups

- Position scales: map continuous, discrete, and date-time variables to the plotting region and construct the axes
- Color scales: map continuous and discrete variables to colors
- Manual scales: map discrete variables to user-defined symbols, shapes, colors, etc
- Identity scales: used to plot variables that are already aesthetics - i.e. if you had a color value variable

Scales

- Each plot has a default scale type, but it may not give the desired result
- Scales follow the naming convention:
  - scale_aesthetic name_scale name

Examples:

```
scale_x_discrete()
scale_y_continuous()
scale_colour_hue()
scale_fill_brewer()
```

For example:

```
p=p+scale_x_discrete()+scale_y_continuous()
  +scale_colour_hue()
```

Common scale arguments: labels

- `name` sets the labels that appear on the axis or legend.
- Because labels are so commonly modified, there are three helper functions to make your code clearer: `xlab(), ylab(), and labs()`.

For example:

```
p=p+ scale_x_continuous()
  +xlab("exprop")+ylab("gdp")
```

versus

```
p=p+ scale_x_continuous()
  + labs(x="exprop", y="gdp", colour="continent")
```

Breaks and labels

- Breaks control where the tick marks appear on an axis and how a continuous variable is segmented in the legend
- They also control the ordering of discrete data
- Labels specify how the breakpoints are labeled. `labels` can only be used if `breaks` is specified

```
p=p+ scale_x_discrete(
  breaks=c(-69, -65, -60, "LD1a", 1, "1_b"),
  labels=c(-69, -65, -60,"LD", "1", "1")
  )
```

Limits

- Limits affect what appears in the plot: only data falling between the lower and upper limits will appear
- Like labels and scales, these are commonly used so ggplot2 provides helper functions to save typing: `xlim` and `ylim`
  - `xlim(10,20)`: continuous
  - `ylim("a", "b", "c")`: discrete
- Data outside the limits is neither plotted nor used in statistical transformations
- To just visually zoom in, you add `xlim` and `ylim` parameters to `coord_cartesian()`
- The `expand` argument controls the amount of space between the edge of the scale and the axis. For no extra space, use `expand=c(0,0)`

```
p + scale_x_continuous(limits = c(5.5, 6.5))
```

or

```
p + xlim(5.5, 6.5)
```

Date and Time Scales

- If your data are not formatted as dates or times, you'll need to convert them: `as.Date()`; `as.POSIXct()`
- There are three arguments that control the appearance and location of date or time axes: `major`, `minor`, and `format`
- Usually the defaults are ok; if you need to change them see p. 101 of the ggplot2 book

```
p=p+ scale_x_date(major = "10 years")

p=p + scale_x_date(
  limits = as.Date(c("2004-01-01", "2005-01-01")),
  format = "%Y-%m-%d"
)
```

Transformations

- The most commonly used scale transformation is log.
- There is a complete listing of scale transformations on p. 99 of the ggplot2 book.
- These also have helper functions, using the convention `scale_x_trans` or `scale_y_trans`
- Of course you could also transform the data, but it may be easier just to transform the scale

```
scale_x_log()
```

is the shortcut, versus

```
scale_x_continuous(trans = "log")
```

```
p=p+scale_x_log() + scale_y_log()
```

Color

There are three continuous color scales

- `scale_colour_gradient()` and `scale_fill_gradient()`: 2 color high-low graident
- `scale_colour_gradient2()` and `scale_fill_gradient2()`: 3 color gradient, with a midpoint. Useful for diverging color scales.
- `scale_colour_gradientn()` and `scale_fill_gradientn()`: custom gradient

And the following discrete color scales:

- `scale_colour_hue()` and `scale_fill_hue()`: picks even spaced hues around the color wheel
- `scale_colour_brewer()` and `scale_fill_brewer()`: custom color schemes: `http://colorbrewer.org`
- `scale_colour_manual()` and `scale_fill_manual()`: custom

```
p=p+ scale_fill_gradient(limits = c(0, 0.04),
low = "white", high = "black")

p=p+scale_fill_brewer(pal = "Pastel1")

p=p+ scale_colour_manual(
  values=c("#000000", "#FF0000", "#009E73", "#D55E00"))
```

Manual Scales

- Manual scales are discrete scales, such as `scale_shape_manual()`, `scale_linetype_manual()`, etc
- The important argument is `values`
- Will match the order of the levels of the discrete variable
- See Appendix B of the ggplot2 book for info about valid aesthetic values

```
data$Period <- factor(data$annotation,
levels = c("Pre-period", "Lame Duck 1", "Post Term 1", "Lame Duck 2")
)

p=p+ scale_colour_manual(
  values=c("#000000", "#FF0000", "#009E73", "#D55E00")
)
```

Legends

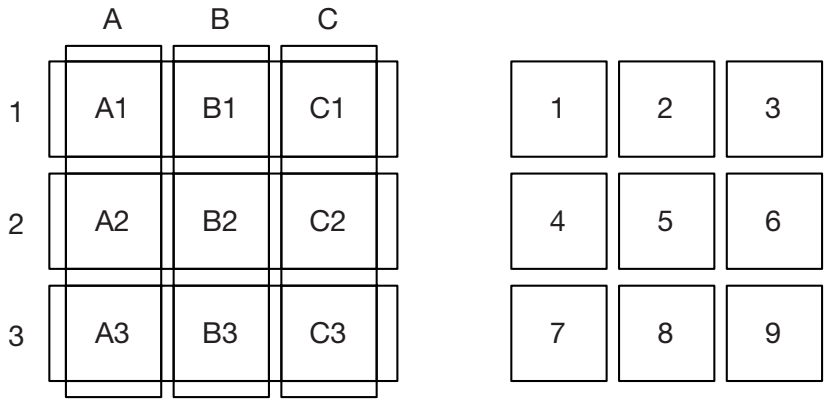
- In ggplot2, there's relatively little that you can do to control the legend
- `breaks` and `labels` are the main determinants of the legend's appearance
- The theme setting `legend.position` controls the location, typically the default is fine

Outline

Faceting

- Faceting generates subplots that each show a different subset of the data
- Facets are very useful for exploratory data analysis, allowing you to easily visualize whether patterns in different subsets differ
- Two types:
  - `facet_grid` (2d)
  - `facet_wrap` (1d)

Faceting


A
B
C
1
2
3
A1
B1
C1
A2
B2
C2
A3
B3
C3
1
2
3
4
5
6
7
8
9
facet_grid
facet_wrap

Faceting

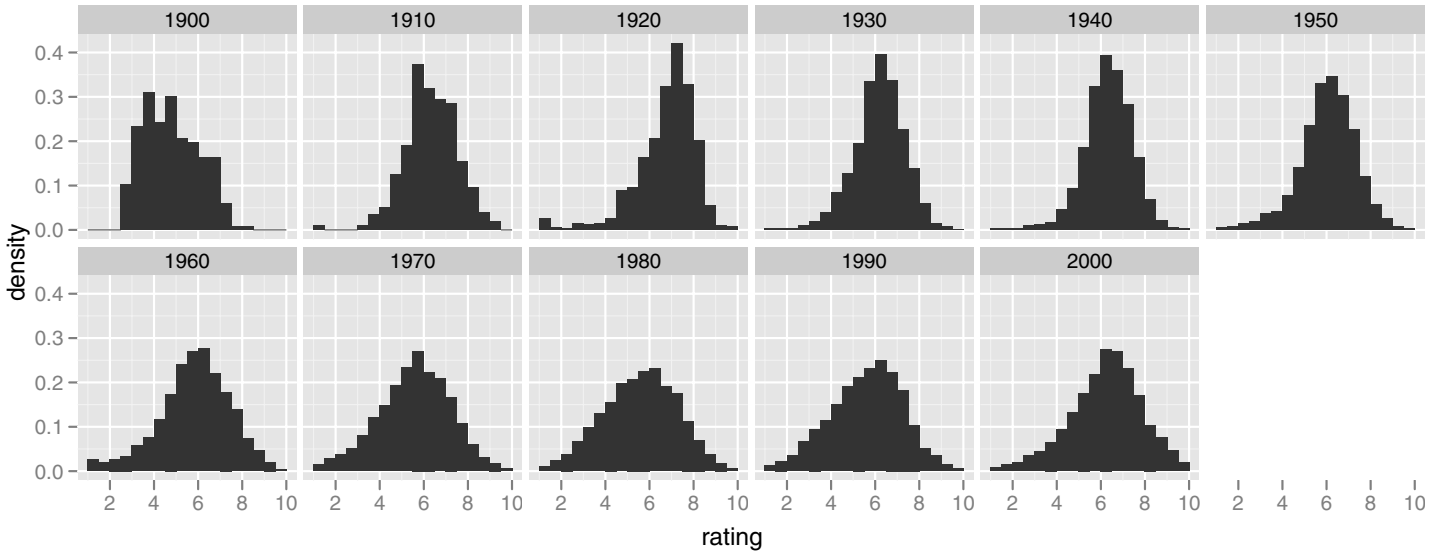
Faceting has two main arguments

- The number of columns (`facet_wrap`) or the variable(s) to facet grid by
  - `. ~ .`: None, the default, produces a single panel
  - `. ~ v1`: A single row with multiple columns. Most useful since computer screens have a greater width than height
  - `v2 ~ .`: A single column with multiple rows. Facilitates comparisons along the x axes, so particularly useful for comparing distributions
  - `v2 ~ v1`: Multiple rows and columns. Put the variable with more levels as the column, to maximize your screen's aspect ratio
- Whether the position scales should be global or local
  - `scales = "fixed"`: x and y scales fixed; default
  - `scales = "free"`: x and y scales vary
  - `scales = "free_x"`: x scale is free, y scale fixed.
  - `scales = "free_y"`: y scale is free, x scale fixed

Facet Wrap

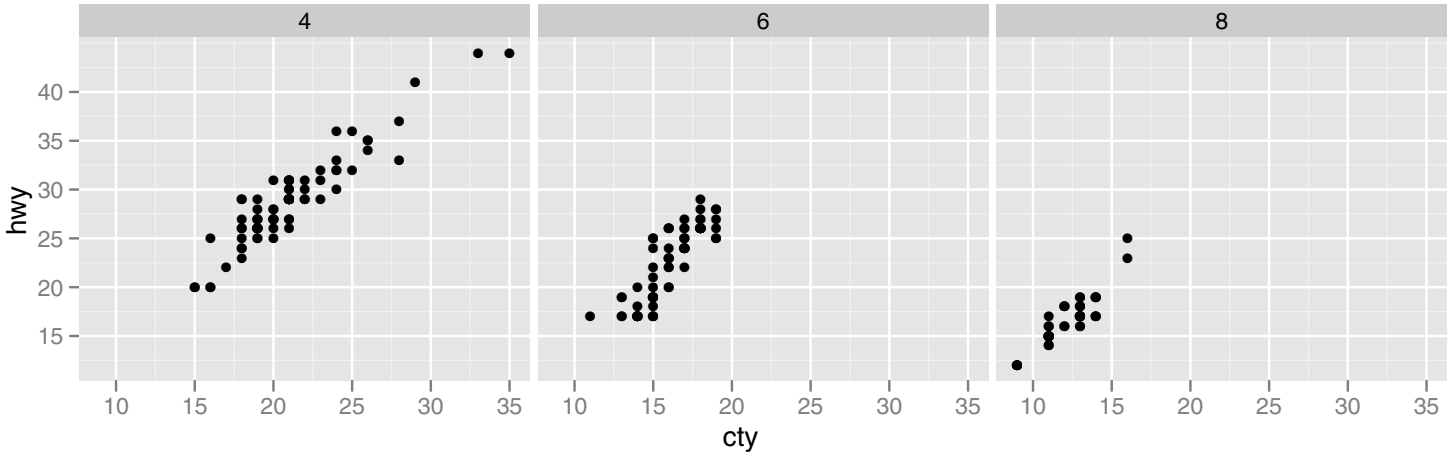
Need to specify the number of columns

```
p=p + facet_wrap(~ decade, ncol = 6)
```



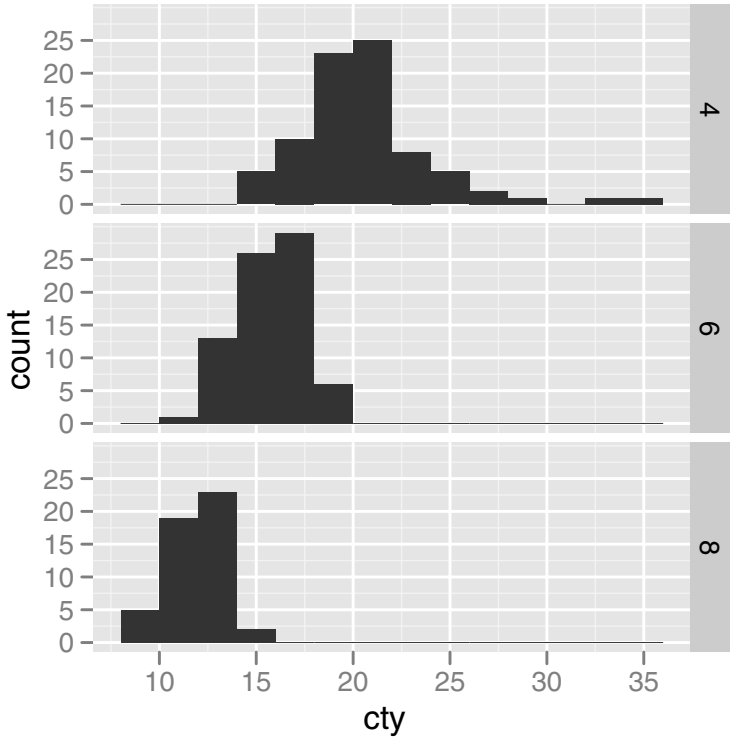
Example

```
facet_grid(. ~ v1)
```



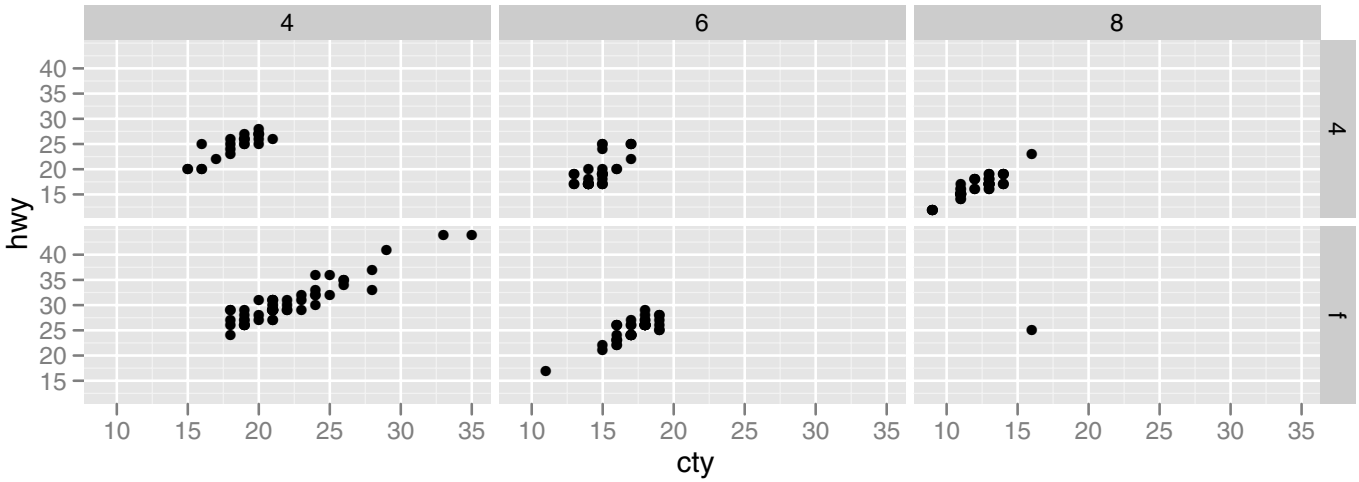
Example

```
facet_grid(v2 ~ .)
```



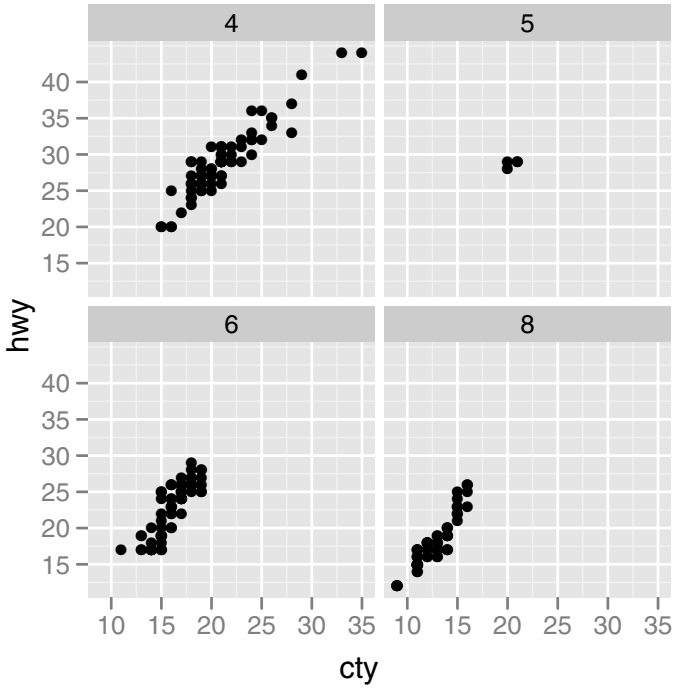
Example

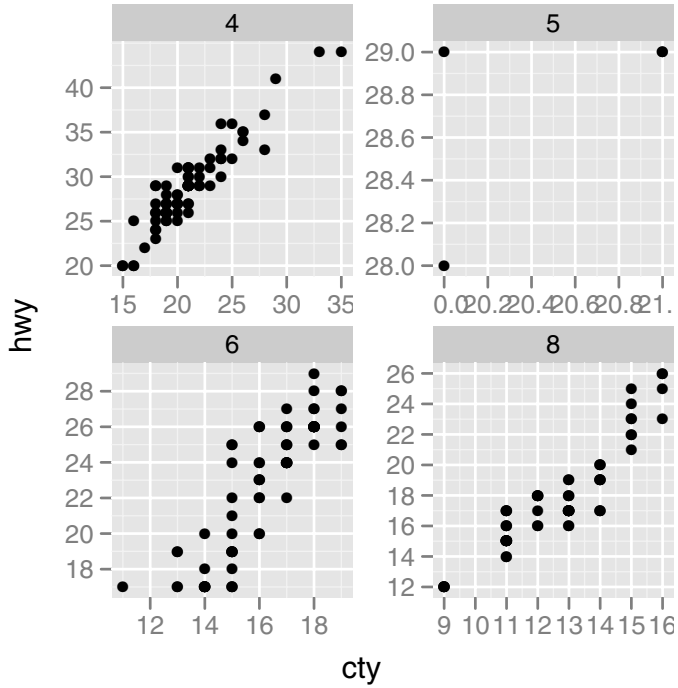
```
facet_grid(v2 ~ v1)
```



Scales

```
p + facet_wrap(~ cyl)
p + facet_wrap(~ cyl, scales = "free")
p + facet_grid(variable ~ ., scale = "free_y")
```





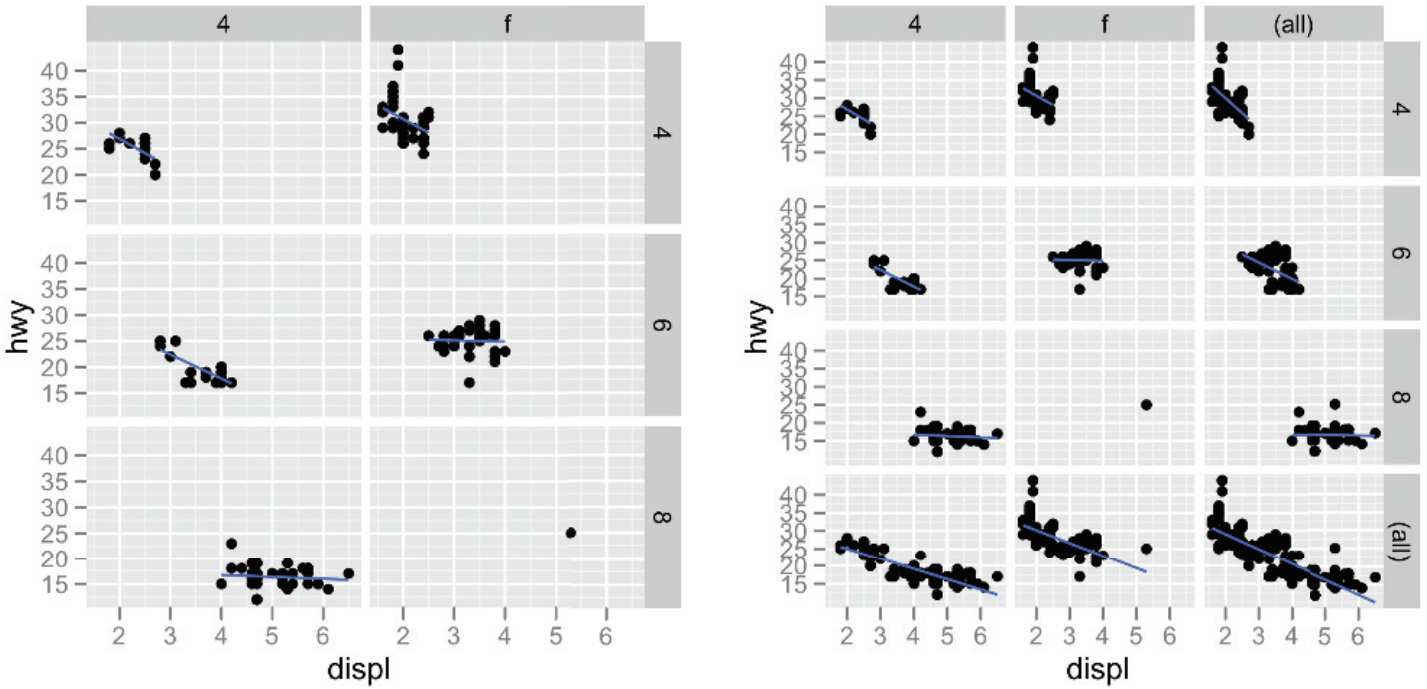
In `facet_grid`, all panels in a column must have the same x scale and all panels in a row must have the same y scale, because columns share an x axis and rows share a y axis

Margins

- Faceting a figure is analogous to creating a contingency table
- It can be useful to display marginal totals (across columns or rows)
- Use `margins= TRUE`

Margins

```
p=p + facet_grid(cyl ~ drv, margins = T)
```



Faceting v. Grouping

- Faceting is easier to read when there are lots of overlapping data
- Faceting allows you to easily split along two dimensions
- Grouping (i.e. by a color aesthetic) makes it easier to see small differences

Faceting Continuous Data

To facet continuous variables, create a discrete version of the variable using `cut_interval` (bins by length) or `cut_number` (bins with the same number of points)

Examples from the ggplot2 book:

```
mpg2$disp_ww <- cut_interval(mpg2$displ, length = 1)
mpg2$disp_wn <- cut_interval(mpg2$displ, n = 6)
mpg2$disp_nn <- cut_number(mpg2$displ, n = 6)
```

Coordinate Systems

- You will probably always use the Cartesian coordinate system (why would an economist need polar coordinates?)
- Recall from above: setting limits on the scale throws out observations outside the limits when calculating stats, whereas setting limits on the coordinate system simply zooms in on the data

More examples:

```
p + scale_x_continuous(limits = c(0, 1))
p + coord_cartesian(xlim = c(0, 1))
```

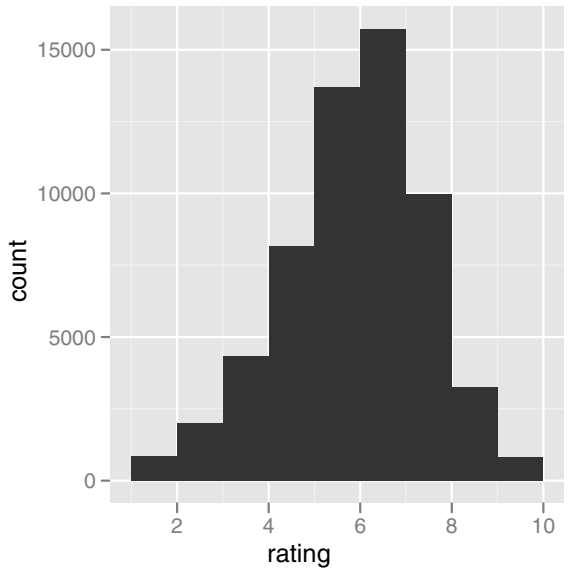
Outline

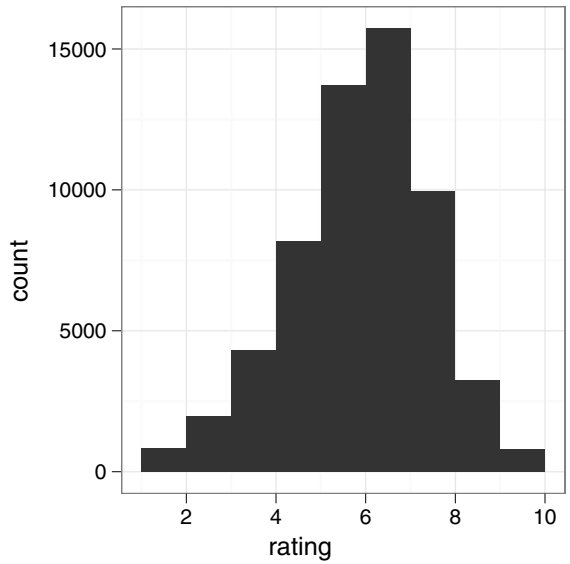
Themes

- Themes control the appearance of non-data elements in the plot
- Themes do not affect how the data is rendered by geoms, or how it is transformed by scales.
- They do affect the font and the color of tick marks, grid lines, and the background
- I usually add `+theme_bw()` to my figures for a clean, professional look
- See Ch. 8 of the ggplot2 book for more details

Themes

Default versus `+theme_bw()`





Outline

Making code more concise and readable

- The results of the last plot you run are stored in the R object `last_plot`
- `last_plot` is useful in interactive work, where you want to iteratively add layers or change scales

An example from the ggplot2 book:

```
qplot(x, y, data = diamonds, na.rm = TRUE)
last_plot() + xlim(3, 11) + ylim(3, 11)
last_plot() + xlim(4, 10) + ylim(4, 10)
last_plot() + xlim(4, 5) + ylim(4, 5)
last_plot() + xlim(4, 4.5) + ylim(4, 4.5)
```

Making code more concise and readable

Save code fragments you will use over and over as R objects; another example from the ggplot2 book:

```
gradient_rb <- scale_colour_gradient(low = "red", high = "blue")
qplot(cty, hwy, data = mpg, colour = displ) + gradient_rb
qplot(bodywt, brainwt, data = msleep, colour = awake, log="xy") +
gradient_rb
```

You can also save vectors of ggplot2 components.

```
xquiet <- scale_x_continuous("", breaks = NA)
yquiet <- scale_y_continuous("", breaks = NA)
quiet <- c(xquiet, yquiet)
qplot(mpg, wt, data = mtcars) + quiet
qplot(displ, cty, data = mpg) + quiet
```

Store your own custom geom function as an R object:

```
geom_lm <- function(formula = y ~ x) {
geom_smooth(formula = formula, se = FALSE, method = "lm")
}
qplot(mpg, wt, data = mtcars) + geom_lm()
library(splines)
qplot(mpg, wt, data = mtcars) + geom_lm(y ~ ns(x, 3))
```