Chapter 5

Toolbox

5.1 Introduction

The layered structure of `ggplot2` encourages you to design and construct graphics in a structured manner. You have learned what a layer is and how to add one to your graphic, but not what geoms and statistics are available to help you build revealing plots. This chapter lists some of the many geoms and stats included in `ggplot2`, broken down by their purpose. This chapter will provide a good overview of the available options, but it does not describe each geom and stat in detail. For more information about individual geoms, along with many more examples illustrating their use, see the online and electronic documentation. You may also want to consult the documentation to learn more about the datasets used in this chapter.

This chapter is broken up into the following sections, each of which deals with a particular graphical challenge. This is not an exhaustive or exclusive categorisation, and there are many other possible ways to break up graphics into different categories. Each geom can be used for many different purposes, especially if you are creative. However, this breakdown should cover many common tasks and help you learn about some of the possibilities.

- Basic plot types, § 5.3, to produce common, "named" graphics like scatterplots and line charts
- Displaying distributions, § 5.4, continuous and discrete, 1d and 2d, joint and conditional
- Dealing with overplotting in scatterplots, § 5.5, a challenge with large datasets
- Surface plots, § 5.6, display 3d surfaces in 2d.
- Statistical summaries, § 5.9, display informative data summaries
- Drawing maps, § 5.7
- Revealing uncertainty and error, § 5.8, with various 1d and 2d intervals
- Annotating a plot, § 5.10, to label, describe and explain with supplemental information
- Weighted data, § 5.11

H. Wickham, *ggplot2*, Use R, DOI 10.1007/978-0-387-98141-3_5,

The examples in this section use a mixture of `ggplot()` and `qplot()` calls, reflecting real-life use. If you need a reminder on how to translate between the two, see Appendix A.2. The examples do not go into much depth, but hopefully if you flick through this chapter, you'll be able to see a plot that looks like the one you're trying to create.

5.2 Overall layering strategy

It is useful to think about the purpose of each layer before it is added. In general, there are three purposes for a layer:

- To display the **data**. We plot the raw data for many reasons, relying on our skills at pattern detection to spot gross structure, local structure, and outliers. This layer appears on virtually every graphic. In the earliest stages of data exploration, it is often the only layer.
- To display a statistical **summary** of the data. As we develop and explore models of the data, it is useful to display model predictions in the context of the data. We learn from the data summaries and we evaluate the model. Showing the data helps us improve the model, and showing the model helps reveal subtleties of the data that we might otherwise miss. Summaries are usually drawn on top of the data.
  If you review the examples in the preceding chapter, you'll see many examples of plots of data with an added layer displaying a statistical summary.
- To add additional **metadata**, context and annotations. A metadata layer displays background context or annotations that help to give meaning to the raw data. Metadata can be useful in the background and foreground.
  A map is often used as a background layer with spatial data. Background metadata should be rendered so that it doesn't interfere with your perception of the data, so is usually displayed underneath the data and formatted so that it is minimally perceptible. That is, if you concentrate on it, you can see it with ease, but it doesn't jump out at you when you are casually browsing the plot.
  Other metadata is used to highlight important features of the data. If you have added explanatory labels to a couple of inflection points or outliers, then you want to render them so that they pop out at the viewer. In that case, you want this to be the very last layer drawn.

5.3 Basic plot types

These geoms are the fundamental building blocks of `ggplot2`. They are useful in their own right, but also to construct more complex geoms. Most of these

geoms are associated with a named plot: when that geom is used by itself in a plot, that plot has a special name.

Each of these geoms is two dimensional and requires both `x` and `y` aesthetics. All understand `colour` and `size` aesthetics, and the filled geoms (bar, tile and polygon) also understand `fill`. The point geom uses `shape` and line and path geoms understand `linetype`. The geoms are used for displaying data, summaries computed elsewhere, and metadata.

- `geom_area()` draws an **area plot**, which is a line plot filled to the y-axis (filled lines). Multiple groups will be stacked on top of each other.
- `geom_bar(stat = "identity")()` makes a **barchart**. We need `stat = "identity"` because the default stat automatically counts values (so is essentially a 1d geom, see § 5.4). The identity stat leaves the data unchanged.

  By default, multiple bars in the same location will be stacked on top of one another.
- `geom_line()` makes a **line plot**. The `group` aesthetic determines which observations are connected; see Section 4.5.3 for more details. `geom_path` is similar to a `geom_line`, but lines are connected in the order they appear in the data, not from left to right.
- `geom_point()` produces a **scatterplot**.
- `geom_polygon()` draws polygons, which are filled paths. Each vertex of the polygon requires a separate row in the data. It is often useful to merge a data frame of polygon coordinates with the data just prior to plotting. Section 5.7 illustrates this concept in more detail for map data.
- `geom_text()` adds labels at the specified points. This is the only geom in this group that requires another aesthetic: `label`. It also has optional aesthetics `hjust` and `vjust` that control the horizontal and vertical position of the text; and `angle` which controls the rotation of the text. See Appendix B for more details.
- `geom_tile()` makes a image plot or level plot. The tiles form a regular tessellation of the plane and typically have the `fill` aesthetic mapped to another variable.

Each of these geoms is illustrated in Figure 5.1, created with the code below.

```
df <- data.frame(
  x = c(3, 1, 5),
  y = c(2, 4, 6),
  label = c("a","b","c")
)
p <- ggplot(df, aes(x, y, label = label)) +
  xlab(NULL) + ylab(NULL)
p + geom_point() + opts(title = "geom_point")
p + geom_bar(stat="identity") +
  opts(title = "geom_bar(stat=\"identity\")")
```

```
p + geom_line() + opts(title = "geom_line")
p + geom_area() + opts(title = "geom_area")
p + geom_path() + opts(title = "geom_path")
p + geom_text() + opts(title = "geom_text")
p + geom_tile() + opts(title = "geom_tile")
p + geom_polygon() + opts(title = "geom_polygon")
```

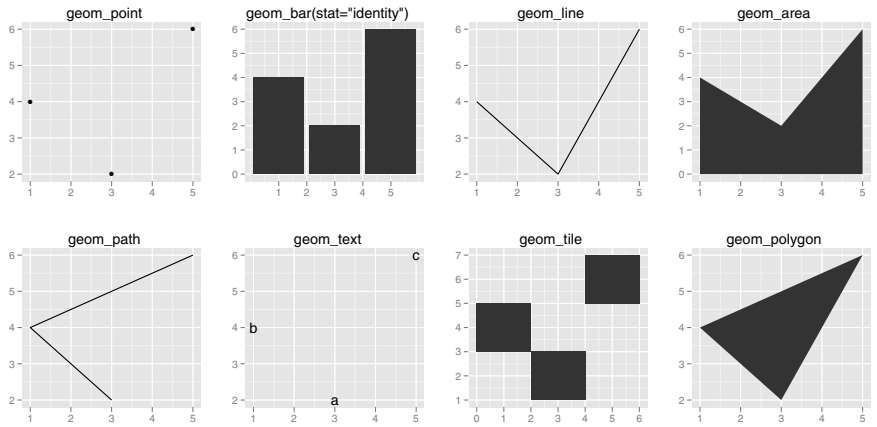


Fig. 5.1: The basic geoms applied to the same data. Many give rise to to named plots (from top left to bottom right): scatterplot, bar chart, line chart, area chart, path plot, labelled scatterplot, image/level plot and polygon plot. Observe the different axis ranges for the bar, area and tile plots: these geoms take up space outside the range of the data, and so push the axes out.

5.4 Displaying distributions

There are a number of geoms that can be used to display distributions, depending on the dimensionality of the distribution, whether it is continuous or discrete, and whether you are interested in conditional or joint distribution.

For 1d continuous distributions the most important geom is the histogram. Figure 5.2 uses the histogram to display the distribution of diamond `depth`. It is important to experiment with bin placement to find a revealing view. You can change the `binwidth`, or specify the exact location of the `breaks`.

If you want to compare the distribution between groups, you have a few options: create small multiples of the histogram, `facets = . ~ var`; use a frequency polygon, `geom = "freqpoly"`; or create a conditional density plot,

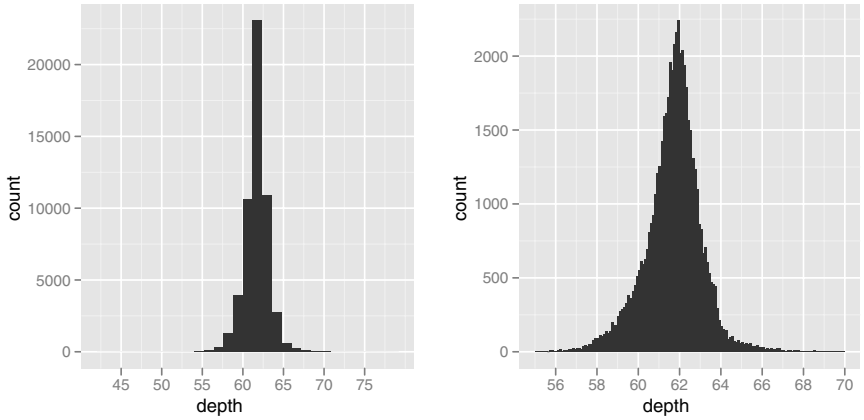


Fig. 5.2: (Left) Never rely on the default parameters to get a revealing view of the distribution. (Right) Zooming in on the x axis, `xlim = c(55, 70)`, and selecting a smaller bin width, `binwidth = 0.1`, reveals far more detail. We can see that the distribution is slightly skew-right. Don't forget to include information about important parameters (like bin width) in the caption.

`position = "fill"`. These options are illustrated in Figure 5.3, created with the code below.

```
depth_dist <- ggplot(diamonds, aes(depth)) + xlim(58, 68)
depth_dist +
  geom_histogram(aes(y = ..density..), binwidth = 0.1) +
  facet_grid(cut ~ .)
depth_dist + geom_histogram(aes(fill = cut), binwidth = 0.1,
  position = "fill")
depth_dist + geom_freqpoly(aes(y = ..density.., colour = cut),
  binwidth = 0.1)
```

Both the histogram and frequency polygon geom use `stat_bin`. This statistic produces two output variables `count` and `density`. The count is the default as it is most interpretable. The density is basically the count divided by the total count, and is useful when you want to compare the shape of the distributions, not the overall size. You will often prefer this when comparing the distribution of subsets that have different sizes.

Many of the distribution-related geoms come in geom/stat pairs. Most of these geoms are aliases: a basic geom is combined with a stat to produce the desired plot. The boxplot may appear to be an exception to this rule, but behind the scenes `geom_boxplot` uses a combination of the basic bars, lines and points.

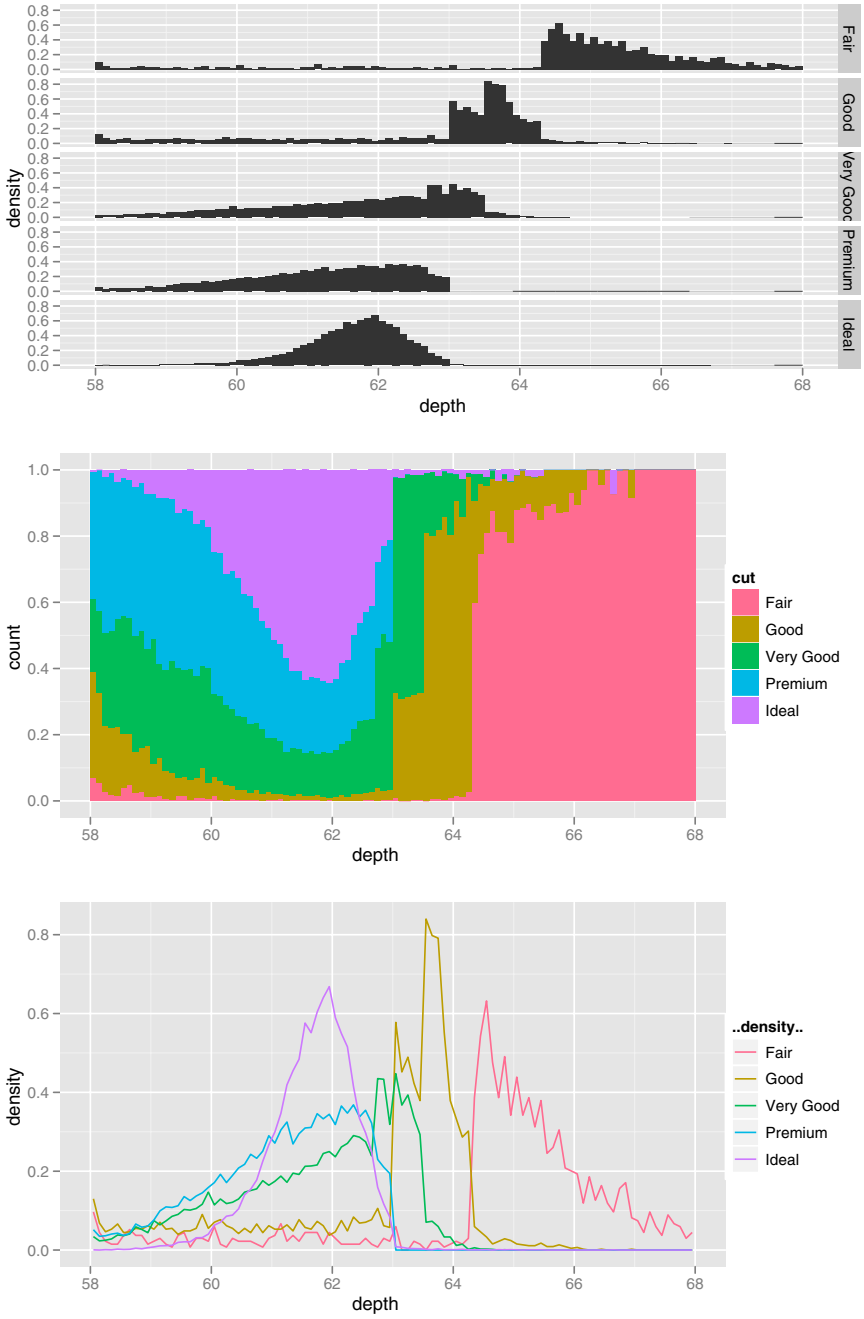


Fig. 5.3: Three views of the distribution of depth and cut. From top to bottom: faceted histogram, a conditional density plot, and frequency polygons. All show an interesting pattern: as quality increases, the distribution shifts to the left and becomes more symmetric.

- `geom_boxplot` = `stat_boxplot` + `geom_boxplot`: box-and-whisker plot, for a continuous variable conditioned by a categorical variable. This is a useful display when the categorical variable has many distinct values. When there are few values, the techniques described above give a better view of the shape of the distribution. This technique can also be used for continuous variables, if they are first finely binned. Figure 5.4 shows boxplots conditioned on both categorical and continuous variables.

```
qplot(cut, depth, data=diamonds, geom="boxplot")
qplot(carat, depth, data=diamonds, geom="boxplot",
  group = round_any(carat, 0.1, floor), xlim = c(0, 3))
```

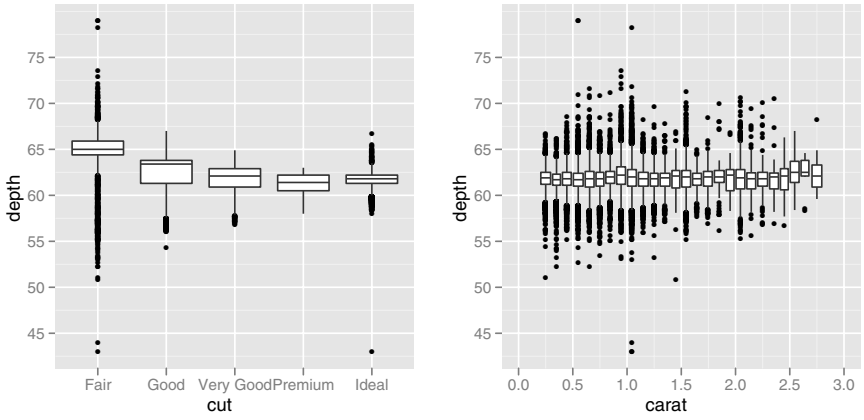


Fig. 5.4: The boxplot geom can be use to see the distribution of a continuous variable conditional on a discrete varable like cut (left), or continuous variable like carat (right). For continuous variables, the group aesthetic must be set to get multiple boxplots. Here `group = round_any(carat, 0.1, floor)` is used to get a boxplot for each 0.1 carat bin.

- `geom_jitter` = `position_jitter` + `geom_point`: a crude way of looking at discrete distributions by adding random noise to the discrete values so that they don't overplot. An example is shown in Figure 5.5 created with the code below.

```
qplot(class, cty, data=mpg, geom="jitter")
qplot(class, drv, data=mpg, geom="jitter")
```

- `geom_density` = `stat_density` + `geom_area`: a smoothed version of the frequency polygon based on kernel smoothers. Also described in Section 2.5.3. Use a density plot when you know that the underlying density is smooth, continuous and unbounded. You can use the `adjust` parameter to

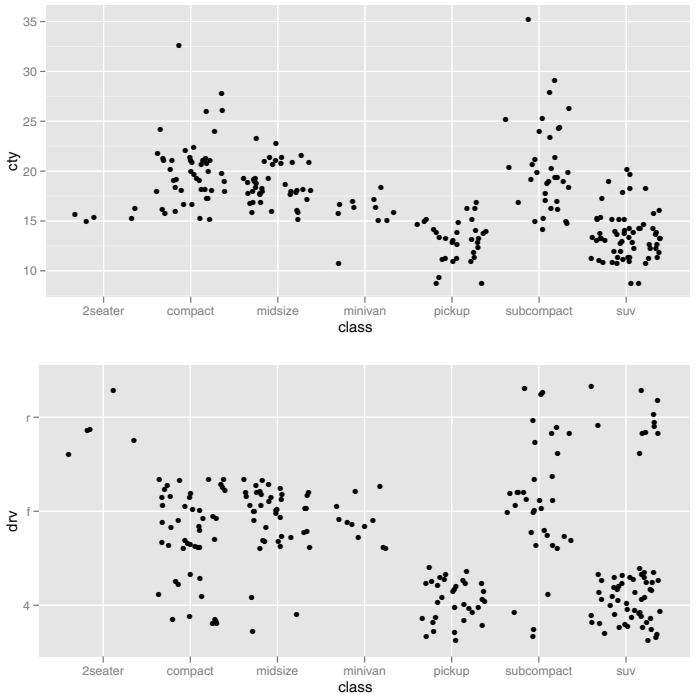


Fig. 5.5: The jitter geom can be used to give a crude visualisation of 2d distributions with a discrete component. Generally this works better for smaller datasets. Car class vs. continuous variable city mpg (top) and discrete variable drive train (bottom).

make the density more or less smooth. An example is shown in Figure 5.6 created with the code below.

```
qplot(depth, data=diamonds, geom="density", xlim = c(54, 70))
qplot(depth, data=diamonds, geom="density", xlim = c(54, 70),
  fill = cut, alpha = I(0.2))
```

Visualising a joint 2d continuous distribution is described in the next section.

5.5 Dealing with overplotting

The scatterplot is a very important tool for assessing the relationship between two continuous variables. However, when the data is large, often points will be plotted on top of each other, obscuring the true relationship. In extreme cases, you will only be able to see the extent of the data, and any conclusions

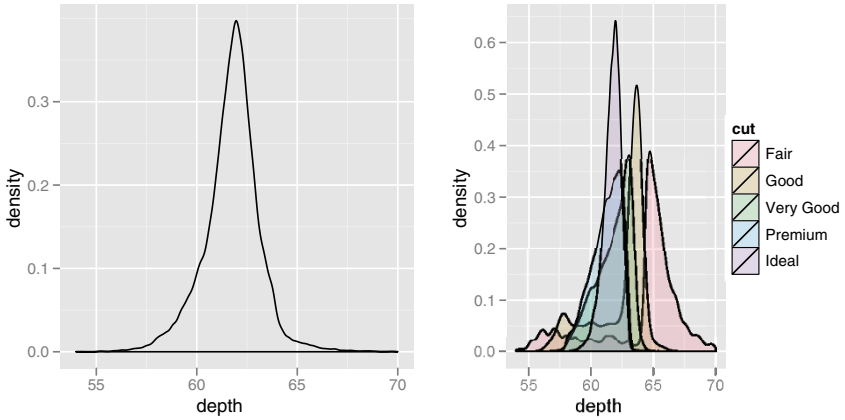


Fig. 5.6: The density plot is a smoothed version of the histogram. It has desirable theoretical properties, but is more difficult to relate back to the data. A density plot of depth (left), coloured by cut (right).

drawn from the graphic will be suspect. This problem is called overplotting and there are a number of ways to deal with it:

- Small amounts of overplotting can sometimes be alleviated by making the points smaller, or using hollow glyphs, as shown in Figure 5.7. The data is 2000 points sampled from two independent normal distributions, and the code to produce the graphic is shown below.

```
df <- data.frame(x = rnorm(2000), y = rnorm(2000))
norm <- ggplot(df, aes(x, y))
norm + geom_point()
norm + geom_point(shape = 1)
norm + geom_point(shape = ".") # Pixel sized
```

- For larger datasets with more overplotting, you can use alpha blending (transparency) to make the points transparent. If you specify alpha as a ratio, the denominator gives the number of points that must be overplotted to give a solid colour. In R, the lowest amount of transparency you can use is 1/256, so it will not be effective for heavy overplotting. Figure 5.8 demonstrates some of these options with the following code.

```
norm + geom_point(colour = alpha("black", 1/3))
norm + geom_point(colour = alpha("black", 1/5))
norm + geom_point(colour = alpha("black", 1/10))
```

- If there is some discreteness in the data, you can randomly jitter the points to alleviate some overlaps. This is particularly useful in conjunction

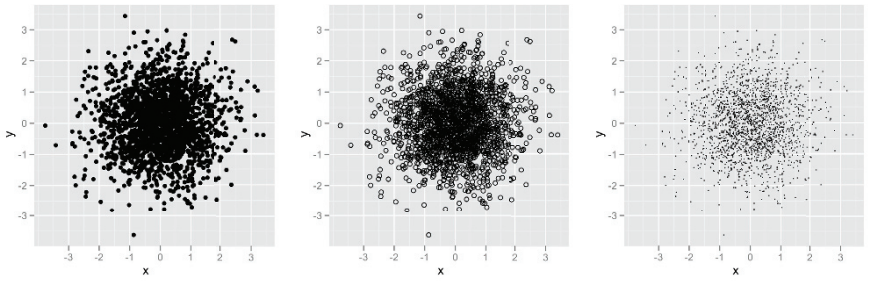


Fig. 5.7: Modifying the glyph used can help with mild to moderate overplotting. From left to right: the default shape, `shape = 1` (hollow points), and `shape= "."` (pixel points).

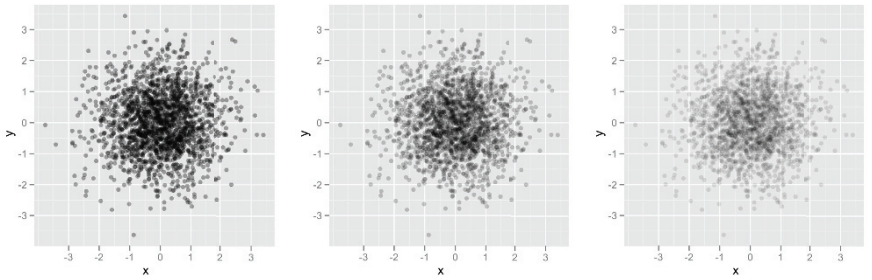


Fig. 5.8: Using alpha blending to alleviate overplotting in sample data from a bivariate normal. Alpha values from left to right: 1/3, 1/5, 1/10.

with transparency. By default, the amount of jitter added is 40% of the resolution of the data, which leaves a small gap between adjacent regions. In Figure 5.9, table is recorded to the nearest integers, so we set a jitter width of half of that. The complete code is shown below.

```
td <- ggplot(diamonds, aes(table, depth)) +
  xlim(50, 70) + ylim(50, 70)
td + geom_point()
td + geom_jitter()
jit <- position_jitter(width = 0.5)
td + geom_jitter(position = jit)
td + geom_jitter(position = jit, colour = alpha("black", 1/10))
td + geom_jitter(position = jit, colour = alpha("black", 1/50))
td + geom_jitter(position = jit, colour = alpha("black", 1/200))
```

Alternatively, we can think of overplotting as a 2d density estimation problem, which gives rise to two more approaches:

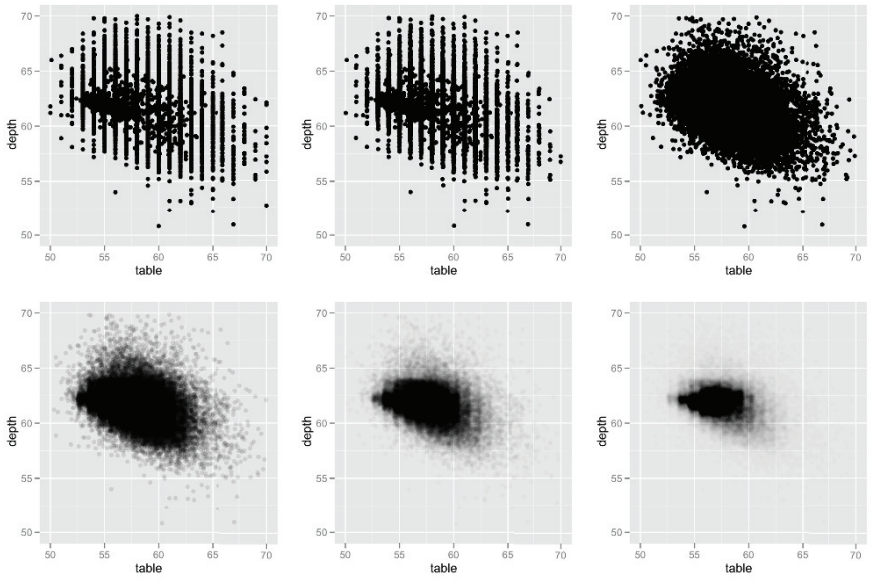


Fig. 5.9: A plot of table vs. depth from the diamonds data, showing the use of jitter and alpha blending to alleviate overplotting in discrete data. From left to right: geom point, geom jitter with default jitter, geom jitter with horizontal jitter of 0.5 (half the gap between bands), alpha of 1/10,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

- Bin the points and count the number in each bin, then visualise that count in some way (the 2d generalisation of the histogram). Breaking the plot into many small squares can produce distracting visual artefacts. Carr et al. (1987) suggests using hexagons instead, and this is implemented with `geom_hexagon`, using the capabilities of the `hexbin` package (Carr et al., 2008). Figure 5.10 compares square and hexagonal bins, using parameters `bins` and `binwidth` to control the number and size of the bins. The complete code is shown below.

```
d <- ggplot(diamonds, aes(carat, price)) + xlim(1,3) +
  opts(legend.position = "none")
d + stat_bin2d()
d + stat_bin2d(bins = 10)
d + stat_bin2d(binwidth=c(0.02, 200))
d + stat_binhex()
d + stat_binhex(bins = 10)
d + stat_binhex(binwidth=c(0.02, 200))
```

- Estimate the 2d density with `stat_density2d`, and overlay contours from this distribution on the scatterplot, or display the density by itself as

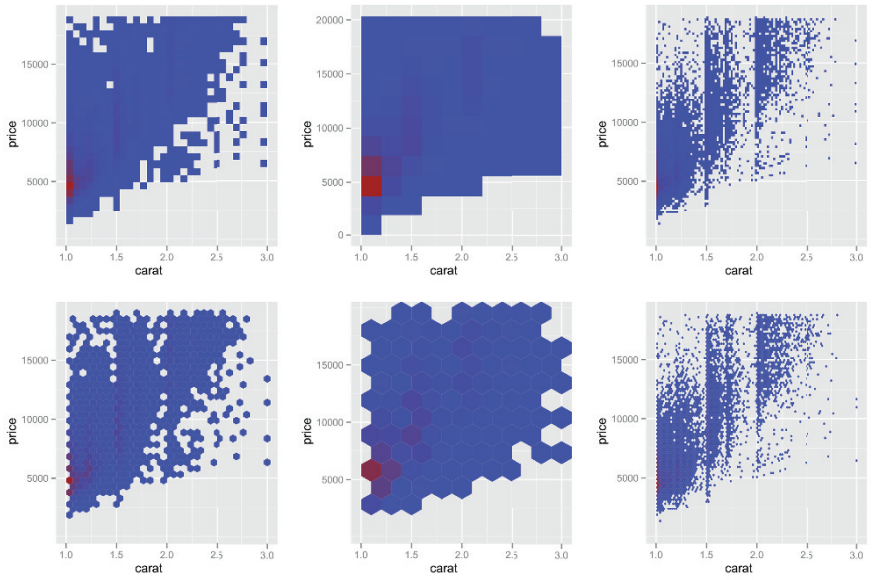


Fig. 5.10: Binning with, top row, square bins, and bottom row, hexagonal bins. Left column uses default parameters, middle column `bins = 10`, and right column `binwidth = c(0.02, 200)`. Legends have been omitted to save space.

coloured tiles, or points with size proportional to density. Figure 5.11 shows a few of these options with the code below.

```
d <- ggplot(diamonds, aes(carat, price)) + xlim(1,3) +
  opts(legend.position = "none")
d + geom_point() + geom_density2d()
d + stat_density2d(geom = "point", aes(size = ..density..),
 contour = F) + scale_area(to = c(0.2, 1.5))
d + stat_density2d(geom = "tile", aes(fill = ..density..),
  contour = F)
last_plot() + scale_fill_gradient(limits = c(1e-5,8e-4))
```

- If you are interested in the conditional distribution of y given x, then the techniques of Section 2.5.3 will also be useful.

Another approach to dealing with overplotting is to add data summaries to help guide the eye to the true shape of the pattern within the data. For example, you could add a smooth line showing the centre of the data with `geom_smooth`. Section 5.9 has more ideas.

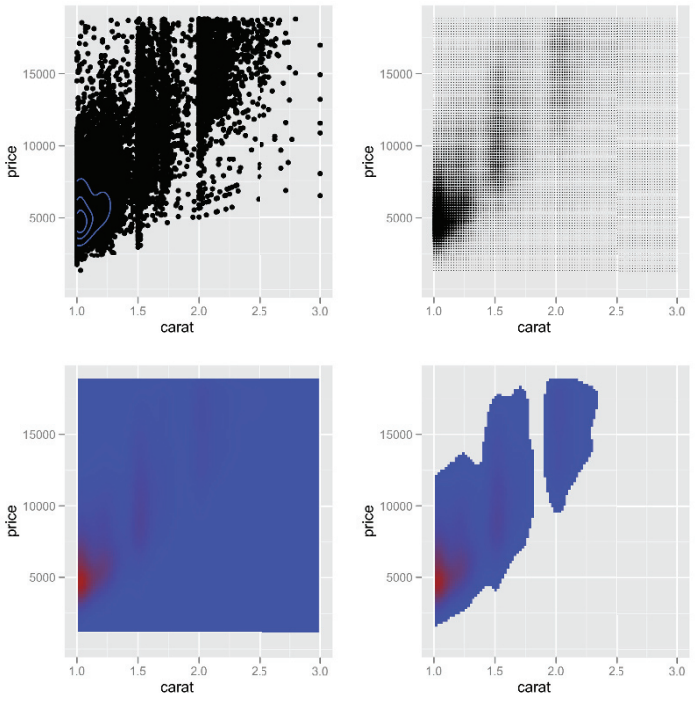


Fig. 5.11: Using density estimation to model and visualise point densities. (Top) Image displays of the density; (bottom) point and contour based displays.

5.6 Surface plots

`ggplot2` currently does not support true 3d surfaces. However, it does support the common tools for representing 3d surfaces in 2d: contours, coloured tiles and bubble plots. These were used to illustrated the 2d density surfaces in the previous section. You may also want to look at RGL, `http://rgl.neoscientists.org/about.shtml`, for interactive 3d plots, including true 3d surfaces.

5.7 Drawing maps

`ggplot2` provides some tools to make it easy to combine maps from the `maps` package with other `ggplot2` graphics. Table 5.1 lists the available maps, which are unfortunately rather US centric. There are two basic reasons you might want to use map data: to add reference outlines to a plot of spatial data, or to construct a choropleth map by filling regions with colour.

Adding map border is performed by the `borders()` function. The first two arguments select the `map` and `region` within the map to display. The

Country	Map name
France	france
Italy	italy
New Zealand	nz
USA at county level	county
USA at state level	state
USA borders	usa
Entire world	world

Table 5.1: Maps available in the maps package

remaining arguments control the appearance of the borders: their `colour` and `size`. If you'd prefer filled polygons instead of just borders, you can set the `fill` colour. The following code uses `borders()` to display the spatial data shown in Figure 5.12.

```
library(maps)
data(us.cities)
big_cities <- subset(us.cities, pop > 500000)
qplot(long, lat, data = big_cities) + borders("state", size = 0.5)

tx_cities <- subset(us.cities, country.etc == "TX")
ggplot(tx_cities, aes(long, lat)) +
  borders("county", "texas", colour = "grey70") +
  geom_point(colour = alpha("black", 0.5))
```

Choropleth maps are a little trickier and a lot less automated because it is challenging to match the identifiers in your data to the identifiers in the map data. The following example shows how to use `map_data()` to convert a map into a data frame, which can then be `merge()`d with your data to produce a choropleth map. The results are shown in Figure 5.13. The details for your data will probably be different, but the key is to have a column in your data and a column in the map data that can be matched.

```
library(maps)
states <- map_data("state")
arrests <- USArrests
names(arrests) <- tolower(names(arrests))
arrests$region <- tolower(rownames(USArrests))

choro <- merge(states, arrests, by = "region")
# Reorder the rows because order matters when drawing polygons
# and merge destroys the original ordering
choro <- choro[order(choro$order), ]
qplot(long, lat, data = choro, group = group,
```

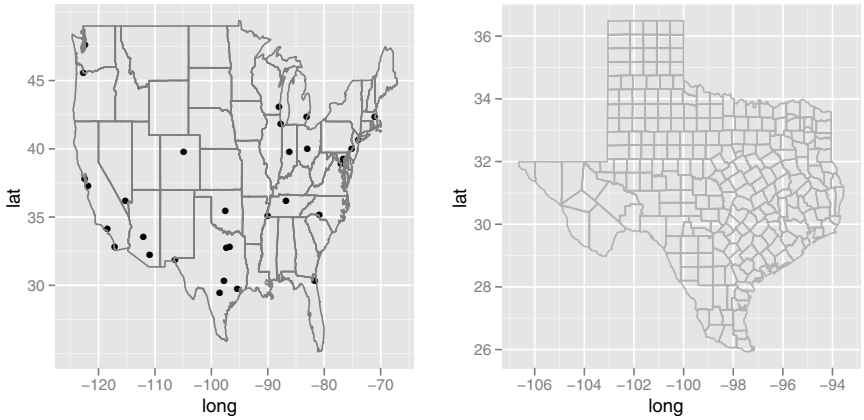


Fig. 5.12: Example using the borders function. (Left) All cities with population (as of January 2006) of greater than half a million, (right) cities in Texas.

```
  fill = assault, geom = "polygon")
qplot(long, lat, data = choro, group = group,
  fill = assault / murder, geom = "polygon")
```

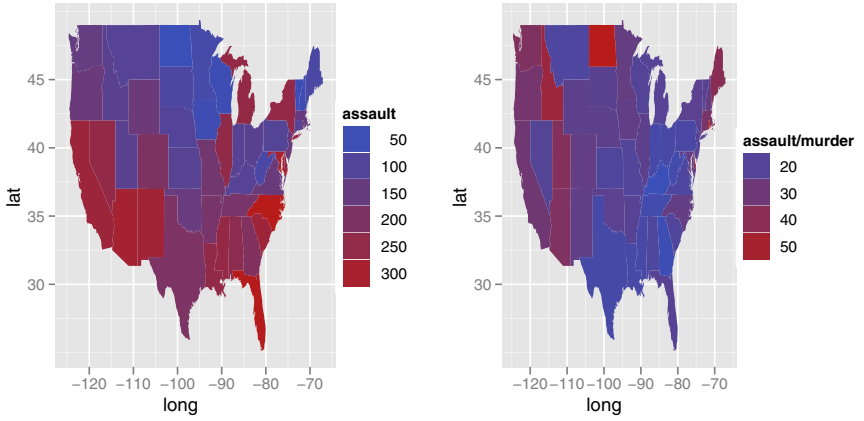
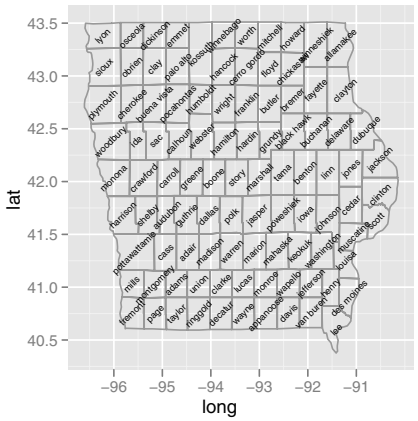


Fig. 5.13: Two choropleth maps showing number of assaults (left) and the ratio of assaults to murders (right).

The `map_data()` function is also useful if you'd like to process the map data in some way. In the following example we compute the (approximate) centre of each county in Iowa and then use those centres to label the map.

```
> ia <- map_data("county", "iowa")
> mid_range <- function(x) mean(range(x, na.rm = TRUE))
> centres <- ddply(ia, .(subregion),
+   colwise(mid_range, .(lat, long)))
> ggplot(ia, aes(long, lat)) +
+   geom_polygon(aes(group = group),
+     fill = NA, colour = "grey60") +
+   geom_text(aes(label = subregion), data = centres,
+     size = 2, angle = 45)
```



5.8 Revealing uncertainty

If you have information about the uncertainty present in your data, whether it be from a model or from distributional assumptions, it is often important to display it. There are four basic families of geoms that can be used for this job, depending on whether the x values are discrete or continuous, and whether or not you want to display the middle of the interval, or just the extent. These geoms are listed in Table 5.2. These geoms assume that you are interested in the distribution of y conditional on x and use the aesthetics `ymin` and `ymax` to determine the range of the y values. If you want the opposite, see `coord_flip`, Section 7.3.3.

X variable	Range	Range plus centre
Continuous	`geom_ribbon`	`geom_smooth(stat="identity")`
Discrete	`geom_errorbar`	`geom_crossbar`
	`geom_linerange`	`geom_pointrange`

Table 5.2: Geoms that display intervals, useful for visualising uncertainty.

Because there are so many different ways to calculate standard errors, the calculation is up to you. For very simple cases, `ggplot2` provides some tools in the form of summary functions described in Section 5.9, otherwise you will have to do it yourself. The `effects` package (Fox, 2008) is particularly useful for extracting these values from linear models. The following example fits a two-way model with interaction, and shows how to extract and visualise marginal and conditional effects. Figure 5.15 focusses on the categorical variable colour, and Figure 5.16 focusses on the continuous variable carat.

```
> d <- subset(diamonds, carat < 2.5 &
+   rbinom(nrow(diamonds), 1, 0.2) == 1)
> d$lcarat <- log10(d$carat)
> d$lprice <- log10(d$price)
>
> # Remove overall linear trend
> detrend <- lm(lprice ~ lcarat, data = d)
> d$lprice2 <- resid(detrend)
>
> mod <- lm(lprice2 ~ lcarat * color, data = d)
>
> library(effects)
> effectdf <- function(...) {
+   suppressWarnings(as.data.frame(effect(...)))
+ }
> color <- effectdf("color", mod)
> both1 <- effectdf("lcarat:color", mod)
>
> carat <- effectdf("lcarat", mod, default.levels = 50)
> both2 <- effectdf("lcarat:color", mod, default.levels = 3)
```

Note, when captioning such figures, you need to carefully describe the nature of the confidence intervals, and whether or not it is meaningful to look at the overlap. That is, are the standard errors for the means or for the differences between means? The packages `multcomp` and `multcompView` are useful calculating and displaying these errors while correctly adjusting for multiple comparisons.

5.9 Statistical summaries

It's often useful to be able to summarise the y values for each unique x value. In `ggplot2`, this role is played by `stat_summary()`, which provides a flexible way of summarising the conditional distribution of y with the aesthetics `ymin`, `y` and `ymax`. Figure 5.17 shows some of the variety of summaries that can be achieved with this tool.

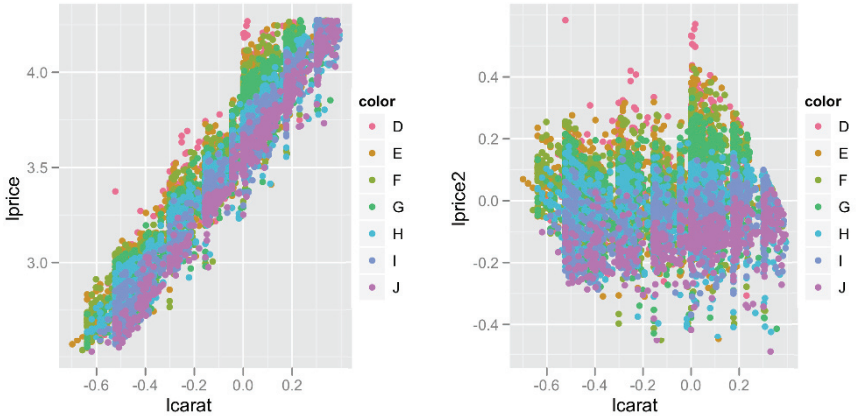


Fig. 5.14: Data transformed to remove most obvious effects. (Left) Both x and y axes are log10 transformed to remove non-linearity. (Right) The major linear trend is removed.

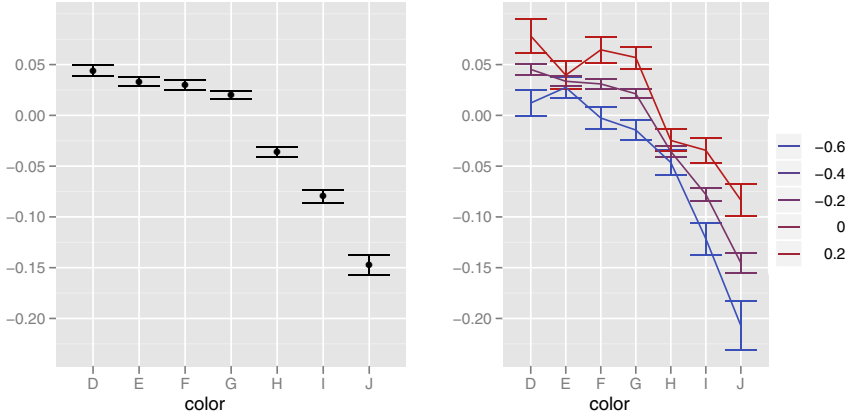


Fig. 5.15: Displaying uncertainty in model estimates for colour. (Left) Marginal effect of colour. (Right) conditional effects of colour for different levels of carat. Error bars show 95% pointwise confidence intervals.

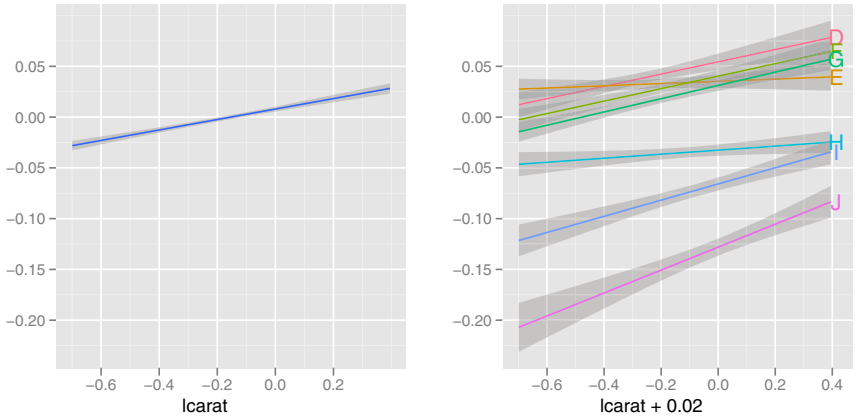


Fig. 5.16: Displaying uncertainty in model estimates for carat. (Left) marginal effect of carat. (Right) conditional effects of carat for different levels of colour. Bands show 95% point-wise confidence intervals.

When using `stat_summary()` you can either supply these the summary functions individually or altogether. These alternatives are described below.

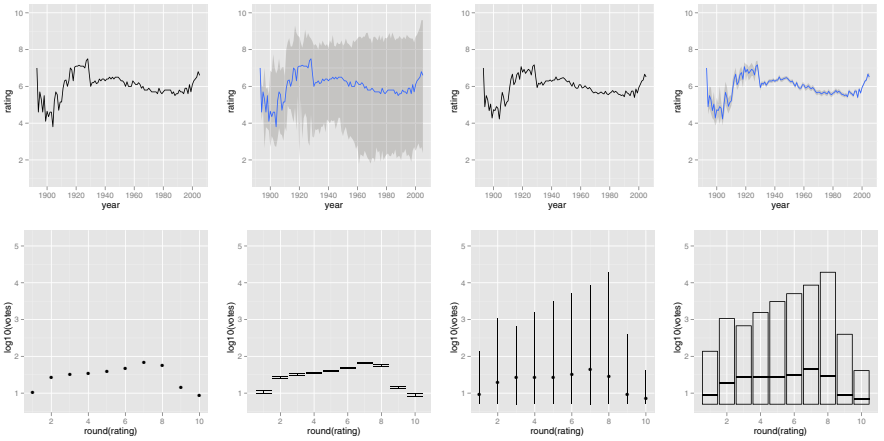
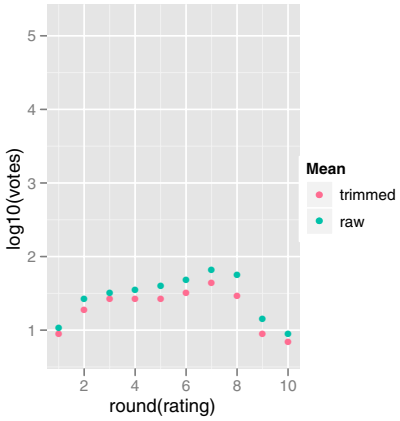


Fig. 5.17: Examples of `stat_summary` in use. (Top) Continuous x with, from left to right, median and line, `median_hilow()` and smooth, mean and line, and `mean_cl_boot()` and smooth. (Bottom) Discrete x with, from left to right, `mean()` and point, `mean_cl_normal()` and error bar, `median_hilow()` and point range, and `median_hilow()` and crossbar. Note that `ggplot2` displays the full range of the data, not just the range of the summary statistics.

5.9.1 Individual summary functions

The arguments `fun.y`, `fun.ymin` and `fun.ymax` accept simple numeric summary functions. You can use any summary function that takes a vector of numbers and returns a single numeric value: `mean()`, `median()`, `min()`, `max()`.

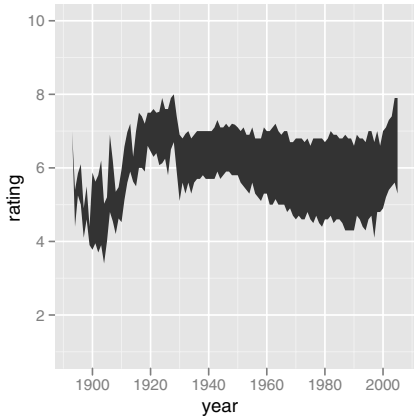
```
> midm <- function(x) mean(x, trim = 0.5)
> m2 +
+   stat_summary(aes(colour = "trimmed"), fun.y = midm,
+     geom = "point") +
+   stat_summary(aes(colour = "raw"), fun.y = mean,
+     geom = "point") +
+   scale_colour_hue("Mean")
```



5.9.2 Single summary function

`fun.data` can be used with more complex summary functions such as one of the summary functions from the `Hmisc` package (Harrell, 2008) described in Table 5.3. You can also write your own summary function. This summary function should return a named vector as output, as shown in the following example.

```
> iqr <- function(x, ...) {
+   qs <- quantile(as.numeric(x), c(0.25, 0.75), na.rm = T)
+   names(qs) <- c("ymin", "ymax")
+   qs
+ }
> m + stat_summary(fun.data = "iqr", geom="ribbon")
```



Function	Hmisc original	Middle	Range
`mean_cl_normal()`	`smean.cl.boot()`	Mean	Standard error from normal approximation
`mean_cl_boot()`	`smean.cl.boot()`	Mean	Standard error from bootstrap
`mean_sdl()`	`smean.sdl()`	Mean	Multiple of standard deviation
`median_hilow()`	`smedian.hilow()`	Median	Outer quantiles with equal tail areas

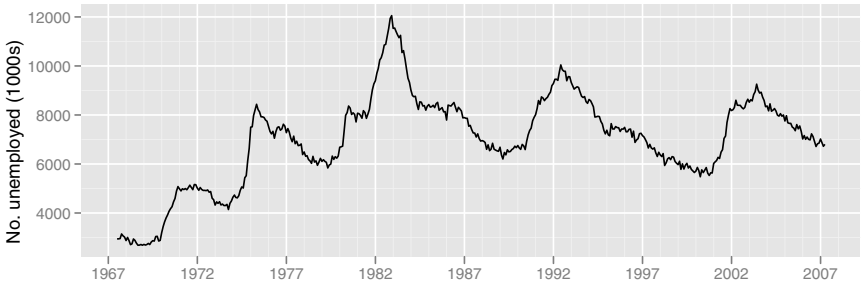
Table 5.3: Summary functions from the `Hmisc` package that have special wrappers to make them easy to use with `stat_summary()`.

5.10 Annotating a plot

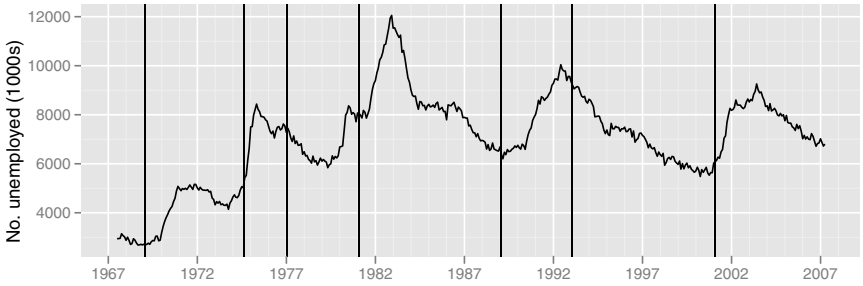
When annotating your plot with additional labels, the important thing to remember is that these annotations are just extra data. There are two basic ways to add annotations: one at a time, or many at once.

Adding one at a time works best for small numbers of annotations with varying aesthetics. You just set all the values to give the desired properties. If you have multiple annotations with similar properties, it may make sense to put them all in a data frame and add them at once. The example below demonstrates both approaches by adding information about presidents to economic data.

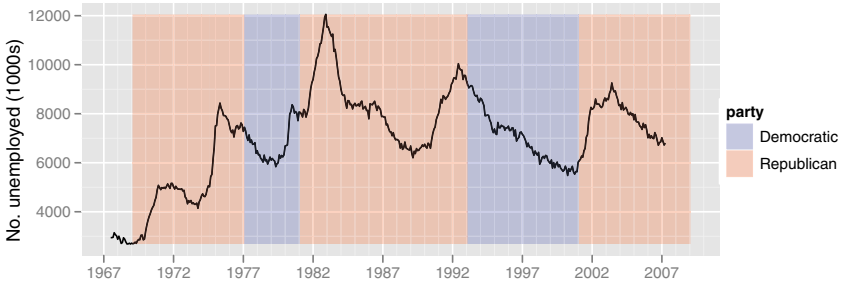
```
> (unemp <- qplot(date, unemploy, data=economics, geom="line",
+   xlab = "", ylab = "No. unemployed (1000s)"))
```



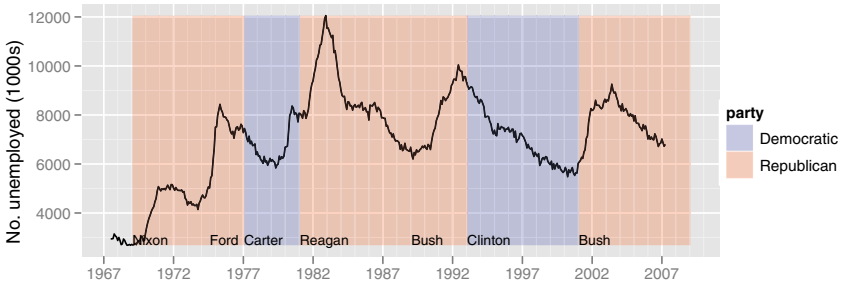

```
>
> presidential <- presidential[-(1:3), ]
>
> yrng <- range(economics$unemploy)
> xrng <- range(economics$date)
> unemp + geom_vline(aes(xintercept = start), data = presidential)
```



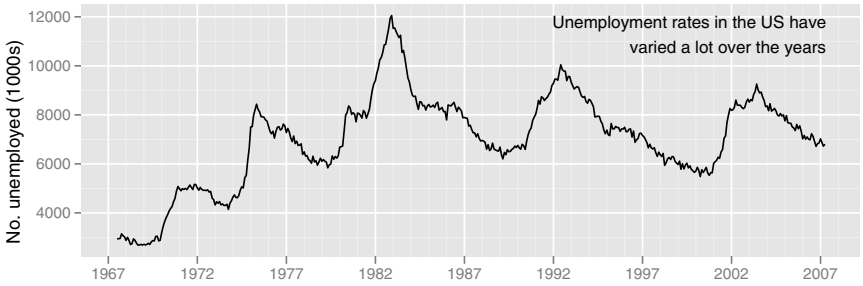

```
> unemp + geom_rect(aes(NULL, NULL, xmin = start, xmax = end,
+   fill = party), ymin = yrng[1], ymax = yrng[2],
+   data = presidential) + scale_fill_manual(values =
+   alpha(c("blue", "red"), 0.2))
```



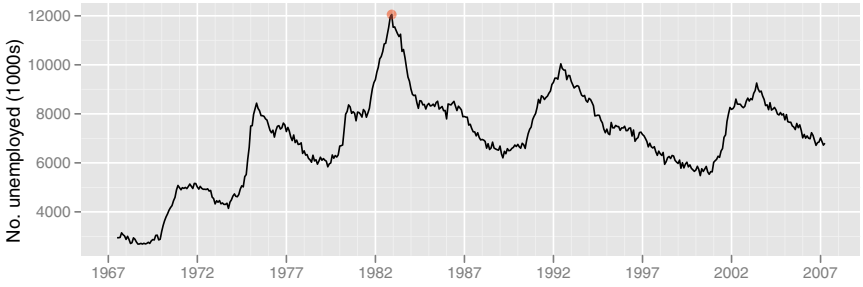

```
> last_plot() + geom_text(aes(x = start, y = yrng[1], label = name),
+   data = presidential, size = 3, hjust = 0, vjust = 0)
```




```
> caption <- paste(strwrap("Unemployment rates in the US have
+   varied a lot over the years", 40), collapse="\n")
> unemp + geom_text(aes(x, y, label = caption),
+   data = data.frame(x = xrng[2], y = yrng[2]),
+   hjust = 1, vjust = 1, size = 4)
```




```
>
> highest <- subset(economics, unemploy == max(unemploy))
> unemp + geom_point(data = highest,
+   size = 3, colour = alpha("red", 0.5))
```



- `geom_text` for adding text descriptions or labelling points. Most plots will not benefit from adding text to every single observation on the plot.

However, pulling out just a few observations (using subset) can be very useful. Typically you will want to label outliers or other important points.

- `geom_vline`, `geom_hline`: add vertical or horizontal lines to a plot.
- `geom_abline`: add lines with arbitrary slope and intercept to a plot.
- `geom_rect` for highlighting interesting rectangular regions of the plot. `geom_rect` has aesthetics `xmin`, `xmax`, `ymin` and `ymax`.
- `geom_line`, `geom_path` and `geom_segment` for adding lines. All these geoms have an `arrow` parameter, which allows you to place an arrowhead on the line. You create arrowheads with the `arrow()` function, which has arguments `angle`, `length`, `ends` and `type`.

5.11 Weighted data

When you have aggregated data where each row in the dataset represents multiple observations, you need some way to take into account the weighting variable. We will use some data collected on Midwest states in the 2000 US census. The data consists mainly of percentages (e.g., percent white, percent below poverty line, percent with college degree) and some information for each county (area, total population, population density).

There are a few different things we might want to weight by:

- nothing, to look at numbers of counties
- total population, to work with absolute numbers
- area, to investigate geographic effects

The choice of a weighting variable profoundly affects what we are looking at in the plot and the conclusions that we will draw. There are two aesthetic attributes that can be used to adjust for weights. Firstly, for simple geoms like lines and points, you can make the size of the grob proportional to the number of points, using the `size` aesthetic, as with the following code, whose results are shown in Figure 5.18.

```
qplot(percwhite, percbelowpoverty, data = midwest)
qplot(percwhite, percbelowpoverty, data = midwest,
  size = poptotal / 1e6) + scale_area("Population\n(millions)",
  breaks = c(0.5, 1, 2, 4))
qplot(percwhite, percbelowpoverty, data = midwest, size = area) +
  scale_area()
```

For more complicated grobs which involve some statistical transformation, we specify weights with the `weight` aesthetic. These weights will be passed on to the statistical summary function. Weights are supported for every case where it makes sense: smoothers, quantile regressions, boxplots, histograms, and density plots. You can't see this weighting variable directly, and it doesn't produce a legend, but it will change the results of the statistical summary.

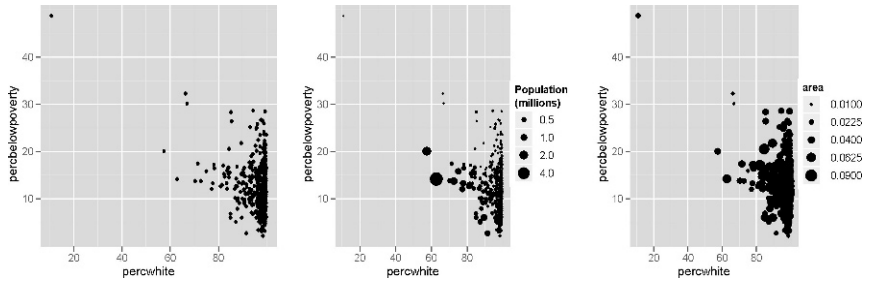


Fig. 5.18: Using size to display weights. No weighting (left), weighting by population (centre) and by area (right).

Figure 5.19 shows how weighting by population density affects the relationship between percent white and percent below the poverty line.

```
lm_smooth <- geom_smooth(method = lm, size = 1)
qplot(percwhite, percbelowpoverty, data = midwest) + lm_smooth
qplot(percwhite, percbelowpoverty, data = midwest,
  weight = popdensity, size = popdensity) + lm_smooth
```

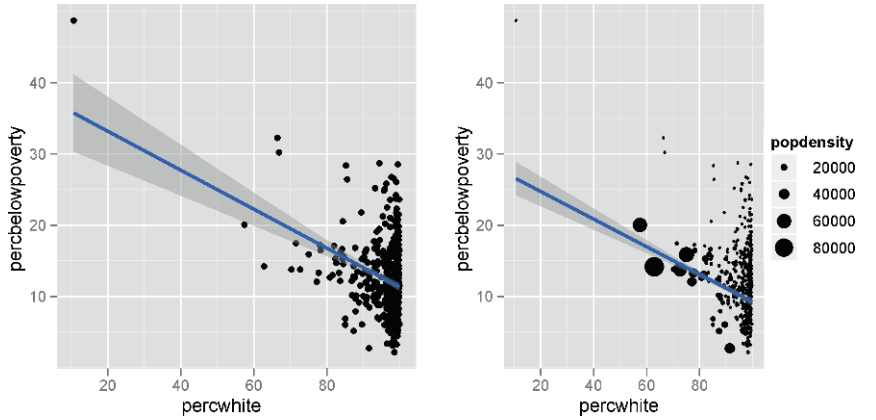


Fig. 5.19: An unweighted line of best fit (left) and weighted by population size (right).

When we weight a histogram or density plot by total population, we change from looking at the distribution of the number of counties, to the distribution of the number of people. Figure 5.20 shows the difference this makes for a histogram of the percentage below the poverty line.

```
qplot(percbelowpoverty, data = midwest, binwidth = 1)
```

```
qplot(percbelowpoverty, data = midwest, weight = poptotal,
  binwidth = 1) + ylab("population")
```

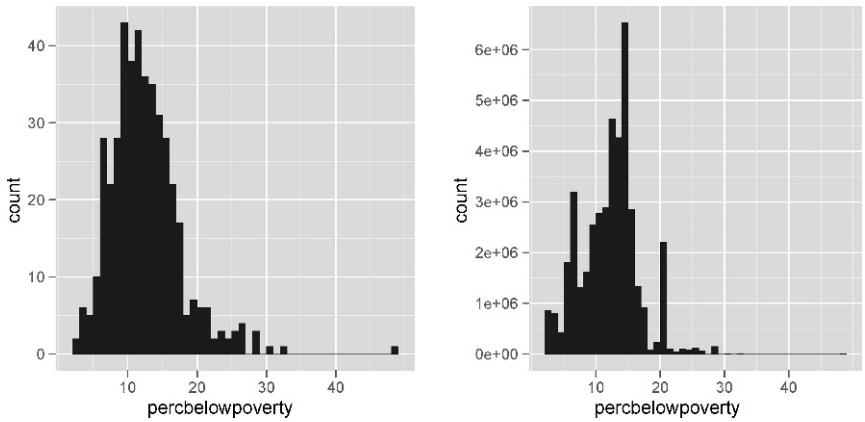


Fig. 5.20: The difference between an unweighted (left) and weighted (right) histogram. The unweighted histogram shows number of counties, while the weighted histogram shows population. The weighting considerably changes the interpretation!