

# ggplot2 themes

Hadley Wickham

November 9, 2009

A theme is made up of multiple *elements* which control the appearance of a single item on the plot, as listed in Table 1. There are three elements that have individual x and y settings: `axis.text`, `axis.title` and `strip.text`. Having a different setting for the horizontal and vertical elements allows you to control how text should appear in different orientations. The appearance of each element is controlled by an *element function*.

There are four basic types of built-in element functions: text, lines and segments, rectangles and blank. Each element function has a set of parameters that control the appearance as described below:

- `theme_text()` draws labels and headings. You can control the font family, face, colour, size, `hjust`, `vjust`, `angle` and `lineheight`.

The following code shows the effect of changing these parameters on the plot title. The results are shown in Figure 1. Changing the angle is probably more useful for tick labels. When changing the angle you will probably also need to change `hjust` to 0 or 1.

```
hgram <- qplot(rating, data = movies, binwidth = 1)
hgramt <- hgram +
  opts(title = "This is a histogram")
hgramt
hgramt + opts(plot.title = theme_text(size = 20))
hgramt + opts(plot.title = theme_text(size = 20, colour = "red"))
hgramt + opts(plot.title = theme_text(size = 20, hjust = 0))
hgramt + opts(plot.title = theme_text(size = 20, face = "bold"))
hgramt + opts(plot.title = theme_text(size = 20, angle = 180))
```

- `theme_line()` and `theme_segment()` draw lines and segments with the same options but in a slightly different way. Make sure you match the appropriate type or you will get strange grid errors. For these element functions you can control the colour, size and `linetype`. These options are illustrated with the code and the results are shown in Figure 2.

```
hgram + opts(panel.grid.major = theme_line(colour = "red"))
hgram + opts(panel.grid.major = theme_line(size = 2))
hgram + opts(panel.grid.major = theme_line(linetype = "dotted"))
hgram + opts(axis.line = theme_segment())
hgram + opts(axis.line = theme_segment(colour = "red"))
hgram + opts(axis.line = theme_segment(size = 0.5, linetype = "dashed"))
```

- `theme_rect()` draws rectangles, mostly used for backgrounds, you can control the fill colour and border colour, size and `linetype`. Examples shown in Figure 3 are created with the code below:

Theme element	Type	Description
<code>axis.line</code>	segment	line along axis
<code>axis.text.x</code>	text	x axis label
<code>axis.text.y</code>	text	y axis label
<code>axis.ticks</code>	segment	axis tick marks
<code>axis.title.x</code>	text	horizontal tick labels
<code>axis.title.y</code>	text	vertical tick labels
<code>legend.background</code>	rect	background of legend
<code>legend.key</code>	rect	background underneath legend keys
<code>legend.text</code>	text	legend labels
<code>legend.title</code>	text	legend name
<code>panel.background</code>	rect	background of panel
<code>panel.border</code>	rect	border around panel
<code>panel.grid.major</code>	line	major grid lines
<code>panel.grid.minor</code>	line	minor grid lines
<code>plot.background</code>	rect	background of the entire plot
<code>plot.title</code>	text	plot title
<code>strip.background</code>	rect	background of facet labels
<code>strip.text.x</code>	text	text for horizontal strips
<code>strip.text.y</code>	text	text for vertical strips

Table 1: Theme elements

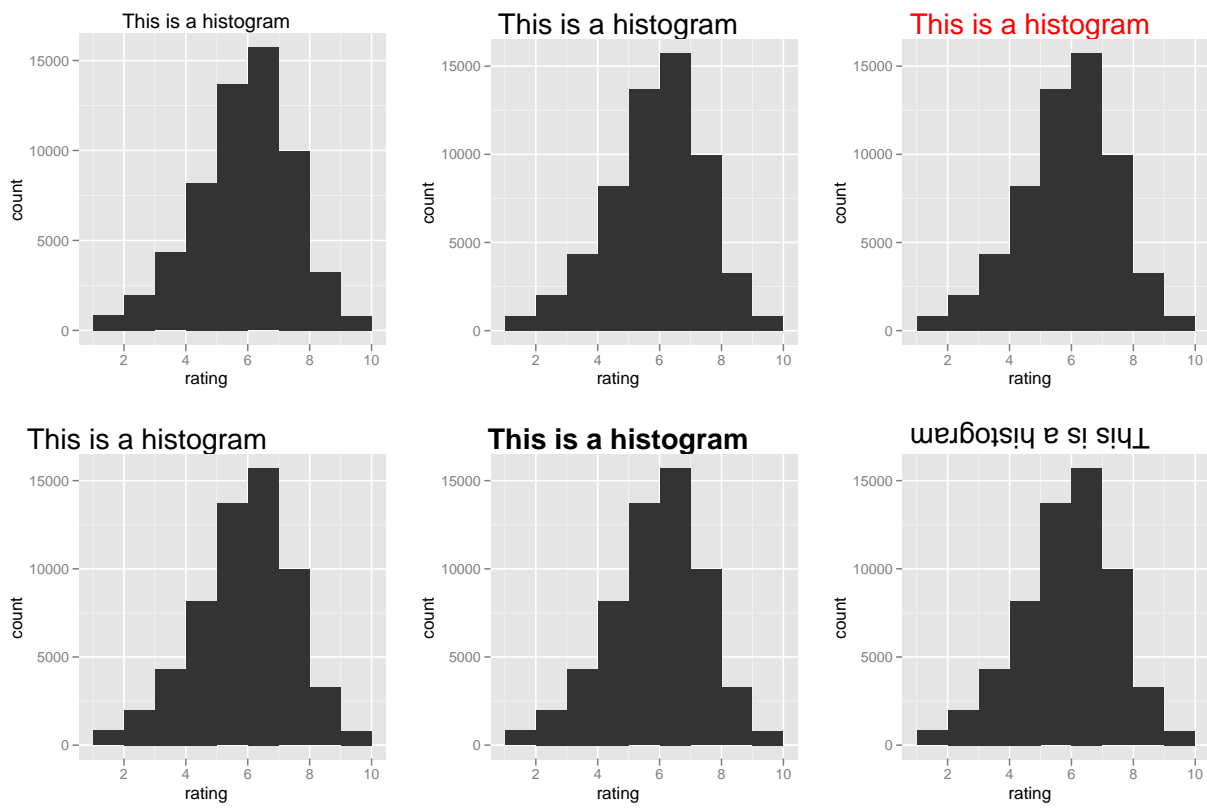


Figure 1: Changing the appearance of the plot title.

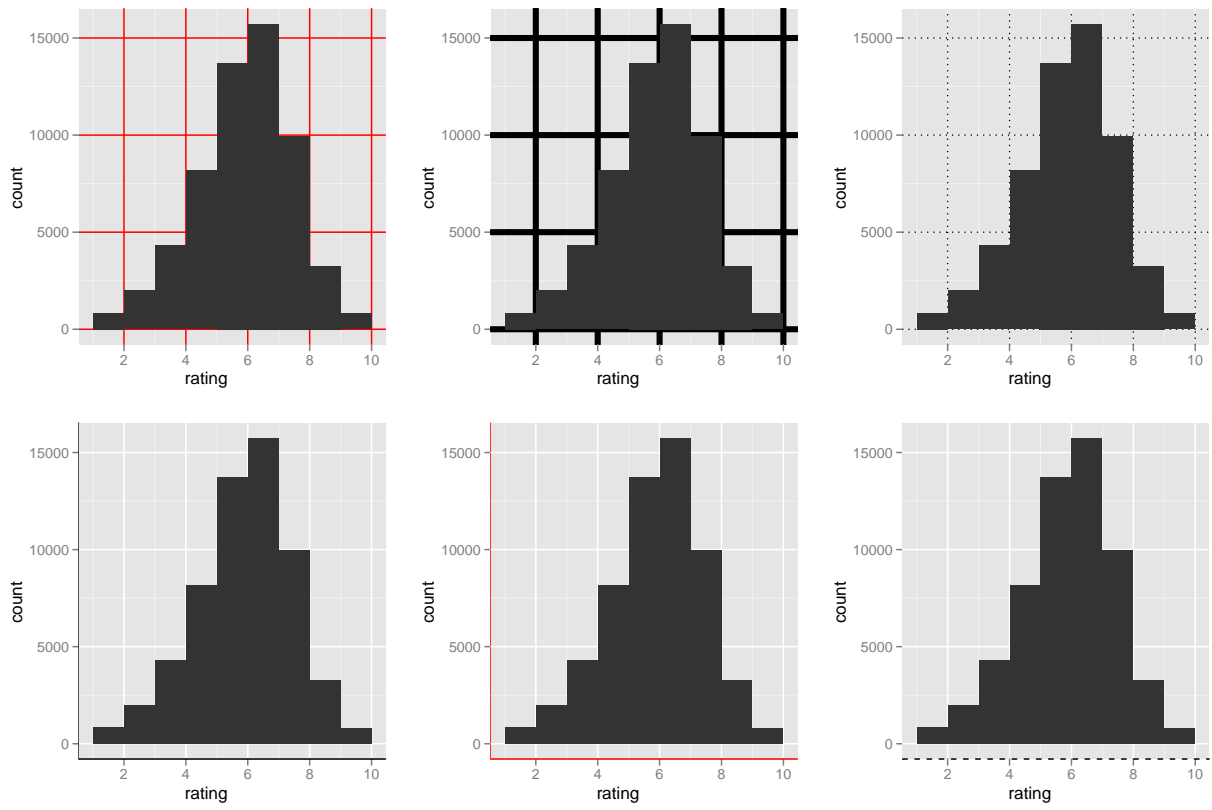


Figure 2: Changing the appearance of lines and segments in the plot.

```

hgram + opts(plot.background = theme_rect(fill = "grey80", colour = NA))
hgram + opts(plot.background = theme_rect(size = 2))
hgram + opts(plot.background = theme_rect(colour = "red"))
hgram + opts(panel.background = theme_rect())
hgram + opts(panel.background = theme_rect(colour = NA))
hgram + opts(panel.background = theme_rect(linetype = "dotted"))

```

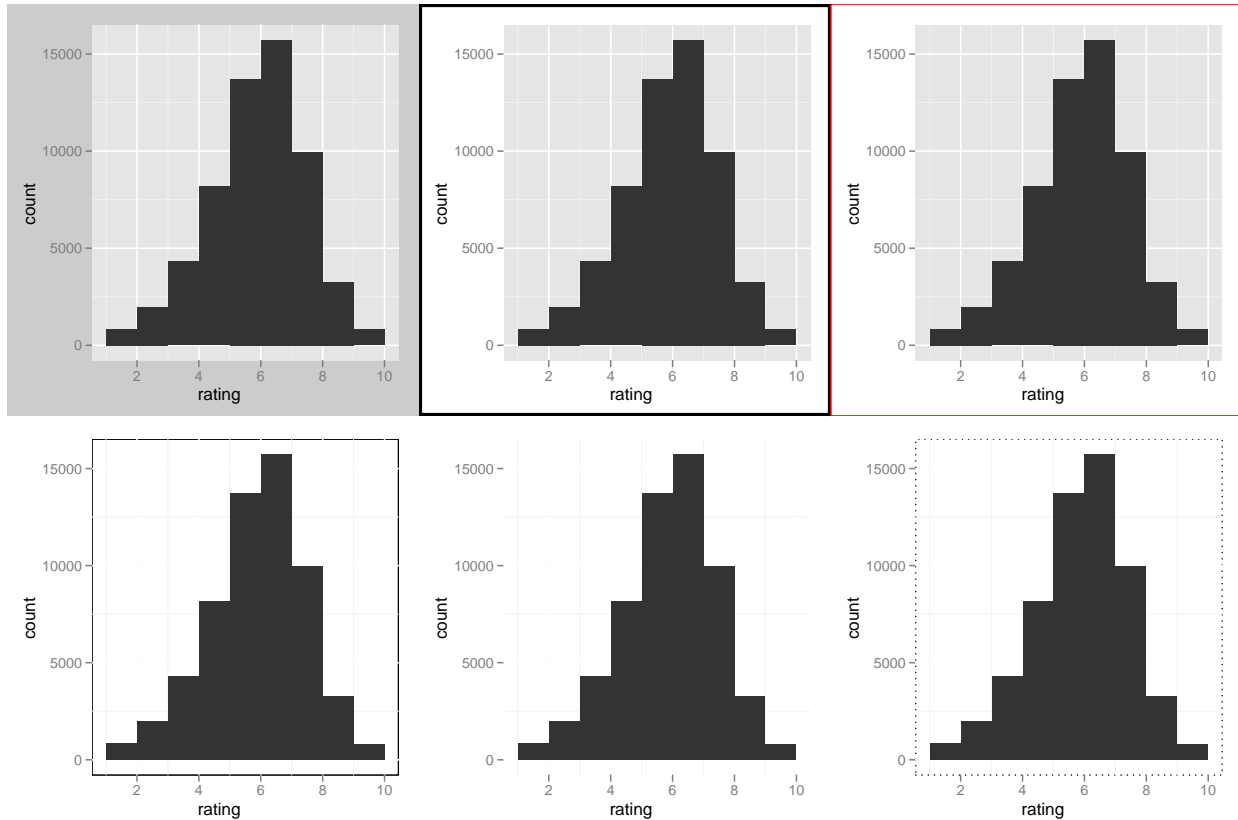


Figure 3: Changing the appearance of the plot and panel background

- `theme_blank()` draws nothing. Use this element type if you don't want anything drawn, and no space allocated for that element. The following example uses `theme_blank()` to progressively suppress the appearance of elements we're not interested in. The results are shown in Figure 4. Notice how the plot automatically reclaims the space previously used by these elements: if you don't want this to happen (perhaps because they need to line up with other plots on the page), use `colour = NA`, `fill = NA` as parameter to create invisible elements that still take up space.

```

hgram
last_plot() + opts(panel.grid.minor = theme_blank())
last_plot() + opts(panel.grid.major = theme_blank())
last_plot() + opts(panel.background = theme_blank())
last_plot() +
  opts(axis.title.x = theme_blank(), axis.title.y = theme_blank())
last_plot() + opts(axis.line = theme_segment())

```

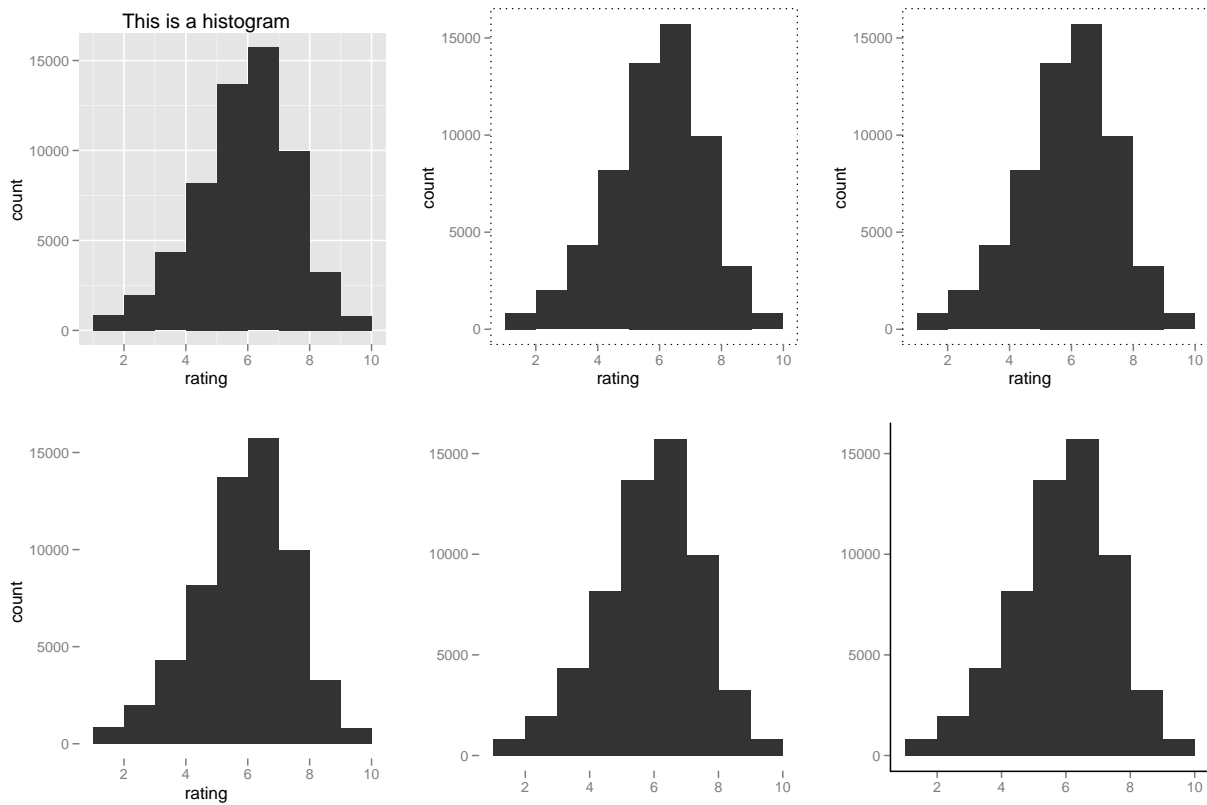


Figure 4: Progressively removing non-data elements from a plot with `theme_blank()`.

You can see the settings for the current theme with `theme_get()`. The output isn't included here because it takes up several pages. You can modify the elements locally for a single plot with `opts()` (as seen above), or globally for all future plots with `theme_update()`. Figure 5 shows the results of pulling together multiple theme settings with the following code.

```
old_theme <- theme_update(  
  plot.background = theme_rect(fill = "#3366FF"),  
  panel.background = theme_rect(fill = "#003DF5"),  
  axis.text.x = theme_text(colour = "#CCFF33"),  
  axis.text.y = theme_text(colour = "#CCFF33", hjust = 1),  
  axis.title.x = theme_text(colour = "#CCFF33", face = "bold"),  
  axis.title.y = theme_text(colour = "#CCFF33", face = "bold", angle = 90)  
)  
qplot(cut, data = diamonds, geom="bar")  
qplot(cty, hwy, data = mpg)  
theme_set(old_theme)
```

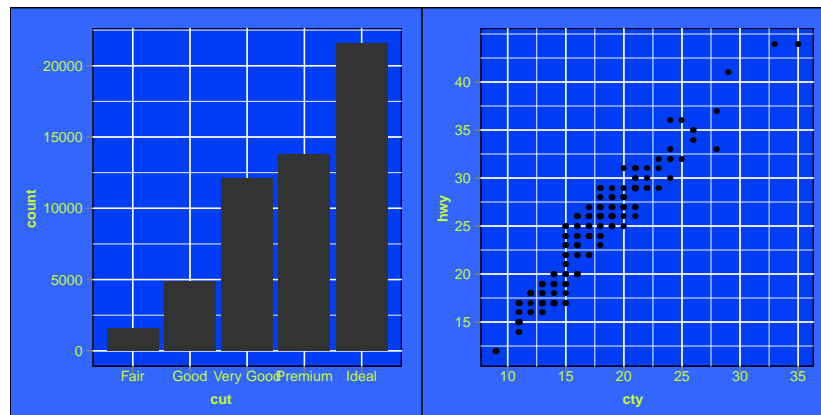


Figure 5: A bar chart and scatterplot created after a new visually consistent (if ugly!) theme has been applied.

There is some duplication in this example because we have to specify the x and y elements separately. This is a necessary evil so that you can have total control over the appearance of the elements. If you are writing your own theme, you would probably want to write a function to minimise this repetition.