DR JAMIE OWEN

CODATA GRAPHICS

jumping rivers

JUMPINGRIVERS.COM

Contents

“IF I CAN’T PICTURE IT, I CAN’T UNDERSTAND IT.”

ALBERT EINSTEIN.

“THE GREATEST VALUE OF A PICTURE IS WHEN IT FORCES US TO NOTICE WHAT WE NEVER EXPECTED TO SEE.”

JOHN TUKEY.

1 Background

Installing packages

Installing packages in R is straightforward. To install a package from the command line we use the `install.packages` command, i.e.

```
install.packages("ggplot2")
library("ggplot2")
```

Types of R graphics

Base graphics

Base graphics were written by Ross Ihaka based on his experience of implementing the S graphics driver. If you have created a histogram, scatter plot or boxplot, you've probably used base graphics. Base graphics are generally fast, but have limited scope. For example, you can only draw on top of the plot and cannot edit or alter existing graphics. For example, if you combine the `plot` and `points` commands, you have to work out the x- and y- limits before adding the points.

Grid graphics

Grid graphics were developed by Paul Murrell[1]. Grid grobs (graphical objects) can be represented independently of the plot and modified later. The viewports system makes it easier to construct complex plots. Grid doesn't provide tools for graphics, it provides primitives for creating plots. Lattice and ggplot2 graphics use grid.

[1] P Murrell. *R Graphics*. CRC Press, 2 edition, 2011

Lattice graphics

The lattice package uses grid graphics to implement the trellis graphics system[2]. It produces nicer plots than base graphics and legends are automatically generated. I initially started using lattice before `ggplot2`. However, I found it a bit confusing and so switched to `ggplot2`.

[2] D Sarkar. *Lattice: Multivariate Data Visualization with R (Use R!)*. Springer, 1st edition, 2008

manufacturer	model	displ	year	cyl	trans	cty	hwy	class
volkswagen	passat	2.0	2008	4	auto(s6)	19	28	midsize
volkswagen	passat	2.0	2008	4	manual(m6)	21	29	midsize
volkswagen	passat	2.8	1999	6	auto(l5)	16	26	midsize
volkswagen	passat	2.8	1999	6	manual(m5)	18	26	midsize
volkswagen	passat	3.6	2008	6	auto(s6)	17	26	midsize

Table 1.1: The last five cars in the `mpg` dataset. The variables `cty` and `hwy` record miles per gallon for city and highway driving respectively. The variable `displ` is the engine displacement in litres.

ggplot2 graphics

`ggplot2` started in 2005[3] and follows the "Grammar of Graphics"[4] Like `lattice`, `ggplot2` uses grid to draw graphics, which means you can exercise low-level control over the plot appearance.

[3] H Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York, 2009. ISBN 978-0-387-98140-6

[4] We'll come on to that later.

Data sets

Throughout the course, we will use a few different datasets.

Fuel economy data

This dataset includes car make, model, class, engine size and fuel economy for a selection of US cars in 1999 and 2008. It is included with the `ggplot2` package[5] and is loaded using the `data` function:

[5] The data originally comes from the EPA fuel economy website, `http://fueleconomy.gov`

```
data(mpg, package="ggplot2")
```

Table 1.1 gives the last five cars in this data set.

The tips data set

A single waiter recorded information about each tip he received over a few months while working in a particular restaurant. He collected data on several variables

- tip($),
- bill($),
- gender of the bill payer,
- whether there were smokers in the party,
- day of the week[6]
- time of day,
- party size.

[6] The waiter only worked Thursday, Friday, Saturday and Sundays.

There were a total of 244 tips. The first few rows of this data set are shown in table 1.2.

The data comes with the `reshape2` package and is loaded using the `data` function:

total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.50	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4

Table 1.2: The first five rows of the `tips` data set. There are 244 rows in this data set.

```
data(tips, package="reshape2")
```

Movie data set

The internet movie database[7] is a website devoted to collecting movie data supplied by studios and fans. It claims to be the biggest movie database on the web and is run by amazon. More information about IMDB can be found online at

`http://imdb.com/help/show_leaf?about`

including information about the data collection process

`http://imdb.com/help/show_leaf?infosource`

[7] `http://imdb.com/`

IMDB makes their raw data available at `http://uk.imdb.com/interfaces/`.

Example rows are given in table 1.1. This data set contains information on over 50,000 movies. We will use this dataset to illustrate the concepts covered in this class.

The dataset contains the following fields:

This is the full version of the data set used in the Introduction to R course.

- Title. Title of the movie.
- Year. Year of release.
- Budget. Total budget in US dollars. If the budget isn't known, then it is stored as '-1'.
- Length. Length in minutes.
- Rating. Average IMDB user rating.
- Votes. Number of IMDB users who rated this movie.
- r1. The percentage (to the nearest 10%) of users who rated this movie a 1.
- r2 – r10: Similar to r1.
- mpaa. The MPAA rating - PG, PG-13, R, NC-17.
- Action, Animation, Comedy, Drama, Documentary, Romance, Short. Binary variables representing if movie was classified as belonging to that genre. A movie can belong to more one genre. See for example the film *Ablaze* in table 1.3.

This data set is part of the `ggplot2movies` package:

				Voting statistics					Movie genre							
Title	Year	Length	Budget	Rating	Votes	r1	...	r10	mpaa	Action	Animation	Comedy	Drama	Documentary	Romance	Short
A.k.a. Cassius	1970	85	-1	5.7	43	4.5	...	14.5	PG	0	0	0	0	1	0	0
AKA	2002	123	-1	6.0	335	24.5	...	14.5	R	0	0	0	1	0	0	0
Alien Vs. Pred	2004	102	45000000	5.4	14651	4.5	...	4.5	PG-13	1	0	0	0	0	0	0
Abandon	2002	99	25000000	4.7	2364	4.5	...	4.5	PG-13	0	0	0	1	0	0	0
Abendland	1999	146	-1	5.0	46	14.5	...	24.5	R	0	0	0	0	0	0	0
Aberration	1997	93	-1	4.8	149	14.5	...	4.5	R	0	0	0	0	0	0	0
Abilene	1999	104	-1	4.9	42	0.0	...	24.5	PG	0	0	0	1	0	0	0
Ablaze	2001	97	-1	3.6	98	24.5	...	14.5	R	1	0	0	1	0	0	0
Abominable Dr	1971	94	-1	6.7	1547	4.5	...	14.5	PG-13	0	0	0	0	0	0	0
About Adam	2000	105	-1	6.4	1303	4.5	...	4.5	R	0	0	1	0	0	1	0

Table 1.3: Sample rows of the movie data set. **Credit:** This data set was initially constructed by Hadley Wickham at http://had.co.nz/.

```r
data(movies, package="ggplot2movies")
```

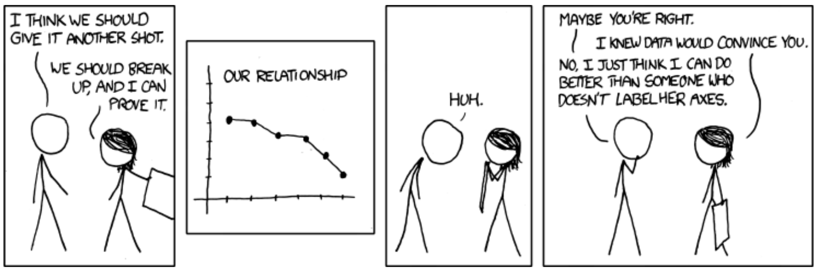
I THINK WE SHOULD GIVE IT ANOTHER SHOT.
WE SHOULD BREAK UP, AND I CAN PROVE IT.
OUR RELATIONSHIP
HUH.
MAYBE YOU'RE RIGHT.
I KNEW DATA WOULD CONVINCE YOU.
NO, I JUST THINK I CAN DO BETTER THAN SOMEONE WHO DOESN'T LABEL HER AXES.

Figure 1.1: http://xkcd.com/833/

2
ggplot2 overview

ggplot2 is a bit different from other graphics packages. It roughly follows the *philosophy* of Wilkinson, 1999.[1] Essentially, we think about plots as layers. By thinking of graphics in terms of layers it is easier for the user to iteratively add new components and for a developer to add new functionality.

[1] L Wilkinson. *The Grammar of Graphics*. Springer, 1st edition, 1999

A basic plot using base graphics

A reasonable first attempt at analysing the mpg data set would be to produce a scatter plot of (for example), engine displacement against city miles per gallon. To use base graphics, we would first construct a basic scatter plot of the data where the cylinder size is 4:[2]

[2] We've cheated here and pretended that we know the x- and y- limits.

```
plot(mpg[mpg$cyl==4,]$displ, mpg[mpg$cyl==4,]$cty,
     xlim=c(1, 8), ylim=c(5, 35))
```

Next we add in the other cars corresponding to different cylinder sizes:

```
points(mpg[mpg$cyl==5,]$displ, mpg[mpg$cyl==5,]$cty,
       col=2)
points(mpg[mpg$cyl==6,]$displ, mpg[mpg$cyl==6,]$cty,
       col=3)
points(mpg[mpg$cyl==8,]$displ, mpg[mpg$cyl==8,]$cty,
       col=4)
```

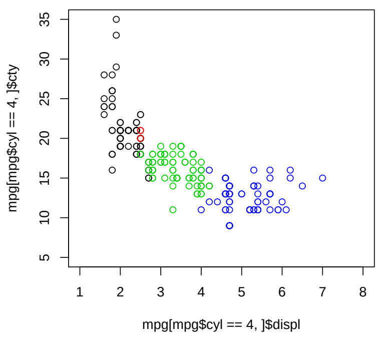

Figure 2.1: A scatter plot of engine displacement vs average city miles per gallon. The coloured points correspond to different cylinder sizes. The plot was constructed using base graphics.

This would produce figure 2.1. A few points to note:

- We have to manually set the scales in the plot command using xlim and ylim.
- We haven't created a legend. We would need to use the legend function.
- The default axis labels are terrible - mpg[mpg$cyl==4,]$displ
- If we wanted to look at highway miles per gallon, this is a bit of a pain.

Plot Name	Geom	Base graphic
Barchart	bar	`barplot`
Box-and-whisker	boxplot	`boxplot`
Histogram	histogram	`hist`
Line plot	line	`plot` and `lines`
Scatter plot	point	`plot` and `points`

Table 2.1: Basic `geom`'s and their corresponding standard plot names.

Let's now consider the equivalent `ggplot2` graphic - figure 2.2. After loading the necessary library, the plot is generated using the following code:

```
g = ggplot(data=mpg, aes(x=displ, y=cty))
g + geom_point(aes(colour=factor(cyl)))
```

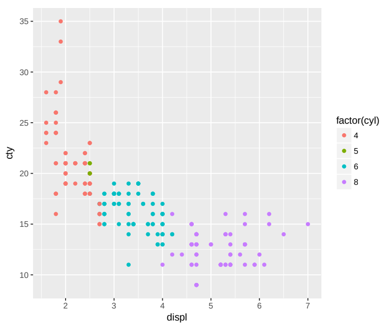

Figure 2.2: As figure 2.1, but created using `ggplot2`.

The `ggplot2` code is fundamentally different from the `base` code. The `ggplot` function sets the default data set, and attributes called **aesthetics**. The aesthetics are properties that are perceived on the graphic. A particular aesthetic can be mapped to a variable or set to a constant value. In figure 2.2, the variable `displ` is mapped to the x-axis and `cty` variable is mapped to the y-axis.

The other function, `geom_point` adds a layer to the plot. The `x` and `y` variables are inherited (in this case) from the first function, `ggplot`, and the colour aesthetic is set to the `cyl` variable. Other possible aesthetics are, for example, size, shape and transparency. In figure 2.2 these additional aesthetics are left at their default value.

This approach is very powerful and enables us to easily create complex graphics. For example, we could create a plot where the size of the points depends on an additional factor:

```
p = g + geom_point(aes(size=factor(cyl)))
```

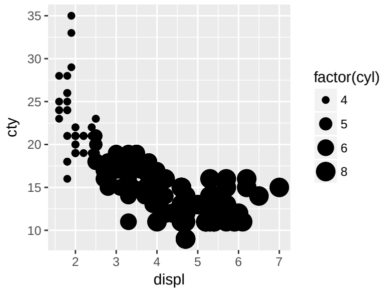

Figure 2.3: As figure 2.2, but where the size aesthetic depends on cylinder size.

which gives figure 2.3 or we could create a line chart

```
p = g + geom_line(
  aes(colour=factor(cyl), size = factor(cyl)))
```

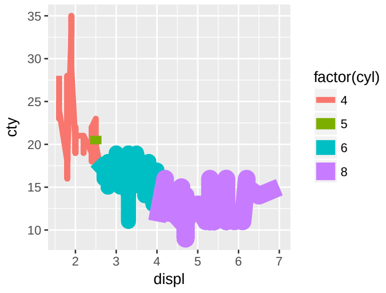

Figure 2.4: As figure 2.2, but using `geom_line`.

to get figure 2.4. Of course, figures 2.3 and 2.4 aren't particular good plots, they just illustrate the general idea.

Points, bars and lines are all examples of **geom**'s or geometric objects. Typically, if we use a single `geom`, we get a standard plot. Table 2.1 summarises some standard geoms and their equivalent base graphic counter part.

However using the idea of a graphical grammar, we can construct more complicated functions. For example, this code

```
p = g + geom_point(aes(colour=factor(cyl))) +
  stat_smooth(aes(colour=factor(cyl)))
```

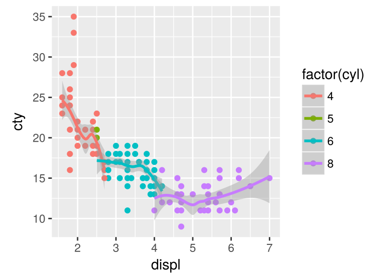

Figure 2.5: As figure 2.2, but with loess regression lines.

produces figure 2.5, which doesn't really have a simple name.

In each ggplot2 command, we are adding (multiple) layers. A single layer comprises of four elements:

- an aesthetic and data mapping;
- a statistical transformation (**stat**);
- a geometric object (**geom**);
- and a position adjustment, i.e. how should objects that overlap be handled.

When we use the command

```
g + geom_point(aes(colour=factor(cyl)))
```

this is actually a shortcut for the command:

```
g + layer(
     data = mpg,#inherited
     mapping = aes(color=factor(cyl)),#x,y are inherited
     stat = "identity",
     geom = "point",
     position = "identity",
     params = list(na.rm=FALSE)
)
```

In practice, we **never** use the layer function. Instead, we use

- geom_* which creates a layer with an emphasis on the geom;
- stat_* which create a layer with an emphasis on the stat;
- qplot which creates a ggplot and a layer.

qplot is short for quick plot. I don't cover qplot in this course. If you find yourself using ggplot2 a lot, then it is worth the time investment.

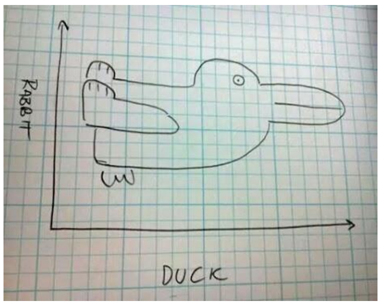

Figure 2.6: Remember: always label your axes.

3
Plot building

The basic plot object

To create an initial ggplot object, we use the `ggplot()` function. This function has two arguments:

- **`data`** and
- an aesthetic **`mapping`**.

These arguments set up the defaults for the various layers that are added to the plot and can be empty. For each plot layer, these arguments can be overwritten. The `data` argument is straightforward - it is a data frame[1]. The `mapping` argument creates default aesthetic attributes. For example

[1] `ggplot2` is very strict regarding the `data` argument. It doesn't accept matrices or vectors. The underlying philosophy is that `ggplot2` takes care of plotting, rather than messaging it into other forms. If you want to do some data manipulation, then use other tools.

```
g = ggplot(data=mpg,
      mapping=aes(x=displ, y=cty, colour=factor(cyl)))
```

or equivalently,

```
g = ggplot(mpg, aes(displ, cty, colour=factor(cyl)))
```

The above commands don't actually produce anything to be displayed, we need to add layers for that to happen.

The geom_ functions

The `geom_` functions are used to perform the actual rendering in a plot. For example, we have already seen that a line geom will create a line plot and a point geom creates a scatter plot. Each geom has a list of aesthetics that it expects.[2] However, some geoms have unique elements. The error-bar geom requires arguments `ymax` and `ymin`. Table 3.1 gives some standard geoms.[3]

[2] For example, `x`, `y`, `colour` and `size`.

[3] For a full list, see table 4.2 of the ggplot2 book or online at `http://had.co.nz/ggplot2/`.

Example: combining geoms

Let's look at the tips data set - see §1.3.2 for a description. We begin by creating a base ggplot object

Name	Description
`abline`	Line, specified by slope and intercept
`boxplot`	Box and whiskers plot
`density`	Kernel density plot
`density_2d`	Contours from a 2s density estimate
`histogram`	Histograms
`jitter`	Individual points are jittered to avoid overlap
`smooth`	Add a smoothed condition mean
`step`	Connect observations by stairs

Table 3.1: A few standard `geom_` functions in `ggplot2`.

```
g = ggplot(tips, aes(x=size, y=tip))
```

Remember, the above piece of code doesn't do anything. Now we'll create a boxplot using the boxplot geom:

```
(g1 = g + geom_boxplot())
```

This produces figure 3.1. Notice that the default axis labels are the column headings of the associated data frame. Figure 3.1 is a boxplot of all the tips data, a more useful plot would be to have individual boxplots conditional on table size

```
g2 = g + geom_boxplot(aes(group=size))
```

Notice that we have included a group aesthetic to the boxplot `geom`. Many `geom`'s have this aesthetic. For example, if we used `geom_line`, then we would have individual lines for each size - this doesn't make much sense in this scenario.

We are not restricted to a single geom - we can add multiple geoms. When data sets are reasonably small, it is useful to display the data on top of the boxplots:

```
g3 = g2  + geom_dotplot(aes(group=size),
              binaxis="y", stackdir="center",
              binwidth=0.05, stackratio=0.5)
```

This generates figure 3.3. The dotplot geom produces a sort of histogram. Notice that we can start picking off some patterns, such as people tend to tip "standard" amounts.

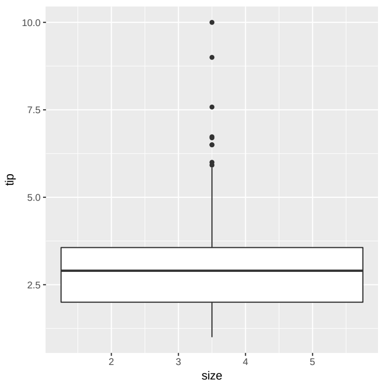

Figure 3.1: A boxplot of tips earned by the waiter.

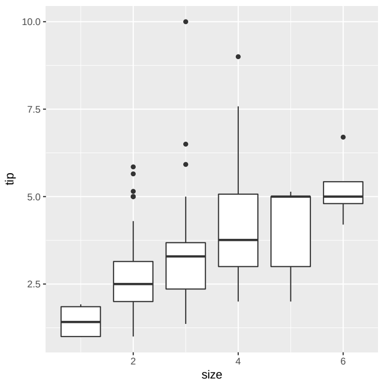

Figure 3.2: A boxplots of tips, conditional on table size.

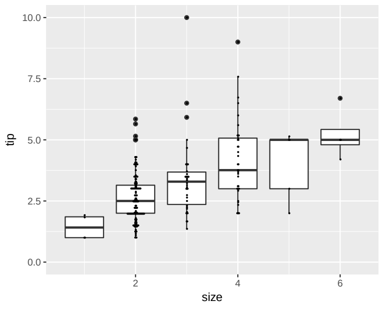

Figure 3.3: As figure 3.2, but including the data points.

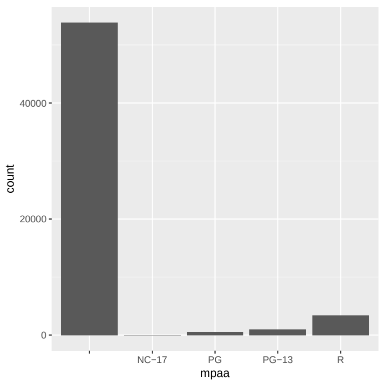

Figure 3.4: A bar chart of the MPAA rating.

Standard plots

There are a few standard geom's that are particular useful:

- geom_boxplot: produces a boxplot - see figure 3.1.
- geom_point: a scatter plot - see figure 3.3.
- geom_bar: produces a standard barplot that counts the x values. For example, to generate a bar plot in figure 3.4 of the MPAA ratings in the movie data set, we use the following code:

```
h = ggplot(movies, aes(x=mpaa)) + geom_bar()
```

- geom_line: a line plot - see practical 3.
- geom_text: adds labels to specified points. This has an additional (required) aesthetic: label. Other useful aesthetics, such as hjust and vjust control the horizontal and vertical position. The angle aesthetic controls the text angle.
- geom_raster: Similar to levelplot or image. For example,

```
data(raster_example, package="nclRggplot2")
g_rast = ggplot(raster_example, aes(x, y)) +
  geom_raster(aes(fill=z))
```

  generates figure 3.5. If the squares are unequal, then use the (slower) geom_tile function.

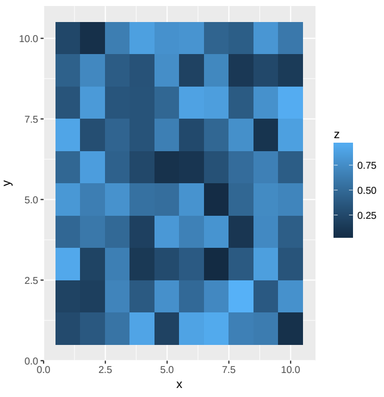

Figure 3.5: A heatmap of some example data using geom_raster. New to version 0.9.

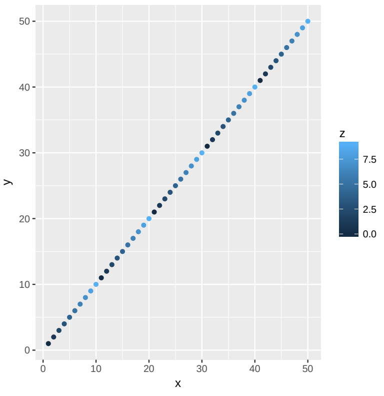

Figure 3.6: Illustration of the continuous colour aesthetic.

Aesthetics

The key to successfully using aesthetics is remembering that the aes() function maps data to an aesthetic. If the parameter is not data or is constant, then don't put it in an aesthetic. Only parameters that are inside of an aes() will appear in the legend. To illustrate these ideas, we'll generate a simple scatter-plot

```
d = data.frame(x=1:50, y = 1:50, z = 0:9)
g_aes = ggplot(d, aes(x = x, y = y))
g_aes + geom_point(aes(colour = z))
```

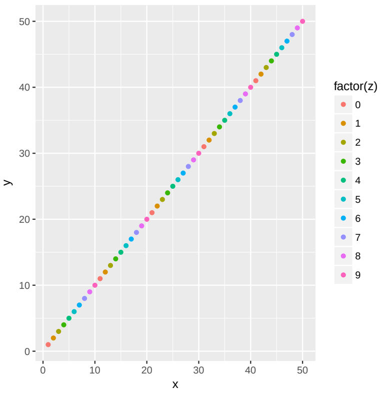

Figure 3.7: Illustration of the discrete colour aesthetic.

which gives figure 3.6. Here the z variable has been mapped to the colour aesthetic. Since this parameter is continuous, ggplot2 uses a continuous colour palette. Alternatively, if we make z a factor or a character, ggplot2 uses a different colour palette

```
g_aes + geom_point(aes(colour=factor(z)))
```

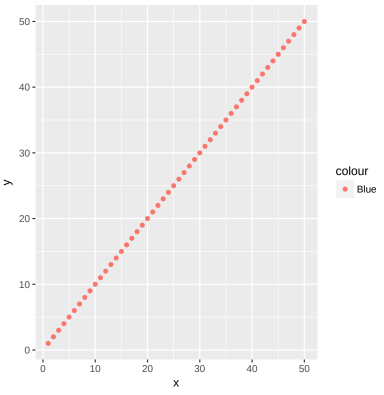

Figure 3.8: Illustration of a constant colour aesthetic.

to get figure 3.7. If we set the aesthetic to a constant value (figure 3.8)

Aesthetic	Description
`linetype`	Similar to `lty` in base graphics
`colour`	Similar to `col` in base graphics
`size`	Similar to `size` in base graphics
`fill`	See figure 3.5.
`shape`	Glyph choice
`alpha`	Control the transparency

Table 3.2: Standard aesthetics. Individual `geom`'s may have other aesthetics. For example, `geom_text` uses `label` and `geom_boxplot` has, amongst other things, `upper`.

```
g_aes + geom_point(aes(colour="Blue"))
```

the resulting plot is unlikely to be what we intended. The value '`Blue`' is just treated as a standard factor. Instead, you probably wanted

```
g_aes + geom_point(colour="Blue")
```

Another important point, is that when you specify mappings inside `ggplot(aes())`, these mappings are inherited by every subsequent layer. This is fine for `x` and `y`, but can cause trouble for other aesthetics. For example, using the `colour` aesthetic is fine for `geom_line`, but may not be suitable for `geom_text`.

There are few standard aesthetics that appear in most, but not all, `geom`'s and `stat`'s (see table 3.2). Individual `geom`'s can have additional optional and required aesthetics. See their help file for further information.

The `stat_` functions

The `stat_` functions focus on transforming data. For example, in figure 2.5 we use a loess[4] smoother function (conditional on the number of cylinders) to plot the overall data trend. Remember, all geoms have stats and, vice visa, all stats have geoms.

[4] A loess smoother is a non-parametric method for smoothing data. It is called local regression because value at point x is weighted toward the data nearest to x.

A stat takes a dataset as input and returns a dataset as an output. For example, the boxplot stat[5] takes in a data set and produces the following variables:

[5] Used by both `geom_boxplot` and `stat_boxplot`.

- lower
- upper
- middle
- ymin: bottom (vertical minimum)
- ymax: top (vertical maximum).

Typically, these statistics are used by the boxplot geom. Equally, they could be used by the error bar geom.

A widely used stat, is *identity*. This stat does not alter the underlying data and is used by a number of geoms, such as `geom_point` and `geom_line`.

Name	Description	Comment
`bin`	Bin data	histogram
`boxplot`	Calculates the components of box-and-whisker plots	See `geom_boxplot`
`contour`	Contours of 3d data	
`density`	1d density estimation	
`density_2d`	2d density estimation	
`function`	Superimpose a function	
`identity`	Leave the data untouched	Used in most geoms
`qq`	Calculation for q-q plots	
`quantile`	Continuous quantiles	
`smooth`	Add a smoother	
`spoke`	Convert angle and radius to `xend` and `yend`	
`step`	Create stair steps	See `geom_step`
`sum`	Sum unique values	
`summary`	Summarises y values at every unique x	
`unique`	Remove duplicates	

Table 3.3: Standard `stat_` functions.

Example: combining stats

Perhaps the easiest `stat` to consider is the `stat_summary` function. This function summarises `y` values at every unique `x` value. This is quite handy, for example, when adding single points that summarise the data or adding error bars.

A simple plot to create, is the mean tip amount based on table size, figure 3.9:

```
g4 = g + stat_summary(geom="point", fun.y= mean)
```

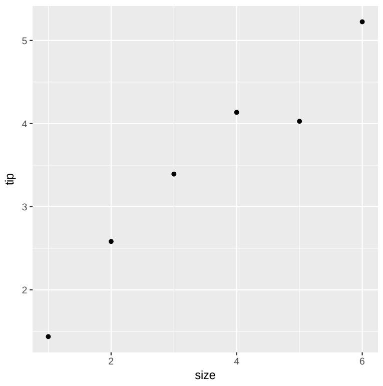

Figure 3.9: Average tip amount conditional on table size.

In the above piece of code we calculate the mean tip size for each unique `x` value, that is, for different table sizes. These x-y values are passed to the `point` geom. We can use any function for `fun.y` provided it takes in a vector and returns a single point. For example, we could calculate the ratio of the mean and median, as in figure 3.10:

```
g5 = g + stat_summary(geom="point",
  fun.y= function(i) mean(i)/median(i))
```

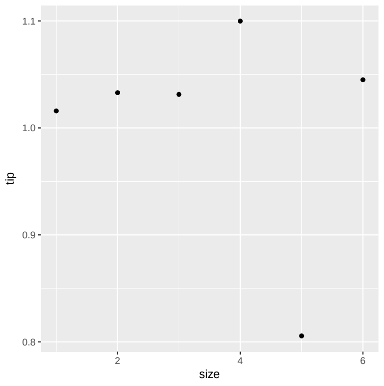

Figure 3.10: The ratio of the mean to median tip amount conditional on table size.

As with the `geom` example, we can combine multiple `stats`:

```
g6 =  g +
  stat_summary(fun.ymin = function(i) quantile(i, 0.25),
               fun.ymax = function(i) quantile(i, 0.75),
               colour="blue", geom="errorbar",
               width=0.2) +
  stat_smooth(aes(colour=smoker, lty=smoker),
               se=FALSE, method="lm")
```

Using the `stat_summary` function, we have created error bars that span the inter quantile range. The `stat_smooth` function plots the regression lines, conditional on whether someone on the table smokes - figure 3.11.

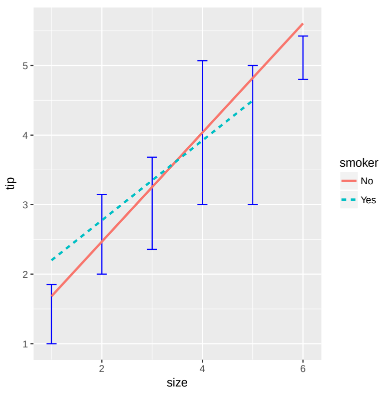

Figure 3.11: The IQR of the tip amount displayed using error bars. The `stat_smooth` function is used to add OLS regression lines, conditional on whether anyone in the party smoked.

4
Facets

Introduction

Faceting is a mechanism for automatically laying out multiple plots on a page. The data is split into subsets, with each subset plotted onto a different panel. `ggplot2` has two types of faceting:

- `facet_grid`: produces a 2d panel of plots where variables define rows and columns.
- `facet_wrap`: produces a 1d ribbon of panels which can be wrapped into 2d.

Facet grid

The function `facet_grid` lays out the plots in a 2d grid. The faceting formula specifies the variables that appear in the columns and rows. Suppose we are interested in movie length. A first plot we could generate is a basic histogram:

```
g = ggplot(movies, aes(x=length)) +  xlim(0, 200) +
  geom_histogram(aes(y=..density..), binwidth=3)
```

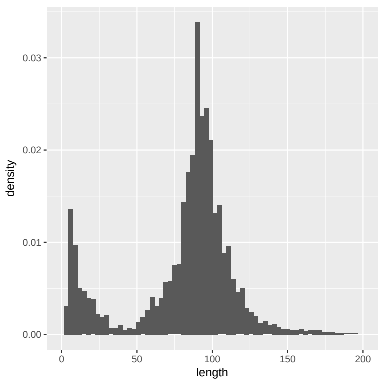

Figure 4.1: A histogram of movie length.

This produces figure 4.1. Notice that we have altered the x-axis since there are a couple of outlying films and adjusted the binwidth in the histogram. We have also used density as the y-axis scale. This just means that the area under the histogram sums to one. The data is clearly bimodal. Some movies are fairly short, whilst others have an average length of around one hundred minutes.

We will now use faceting to explore the data further.

- y ~ .: a single column with multiple rows. This can be handy for double column journals. For example, to create histograms conditional on whether they are comedy films, we use:

  ```
  g + facet_grid(Comedy ~ .)
  ```

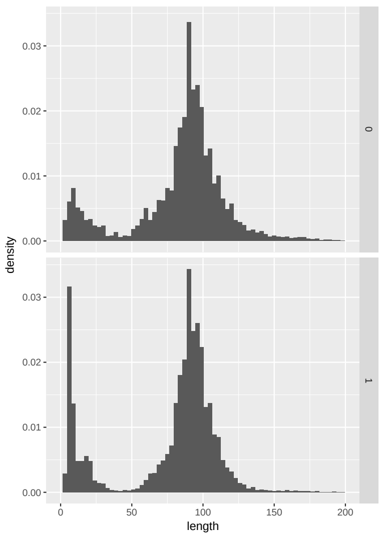

Figure 4.2: Movie length conditional on whether it is a comedy.

This gives figure 4.2.

- `. ~ x`: a single row with multiple columns. Very useful in wide screen monitors. In this piece of code, we create histograms conditional on whether the movie was animated:

  ```
  g + facet_grid(. ~ Animation)
  ```

  From figure 4.3, it's clear that the majority of short films are animations. For illustration purposes, we have used the `geom_density` function in figure 4.3.

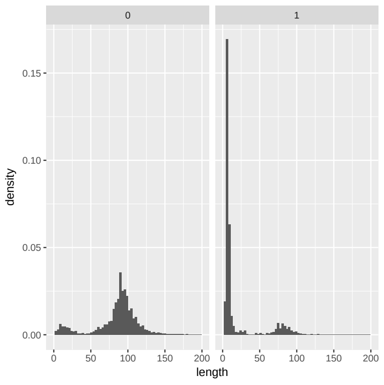

Figure 4.3: Histograms of movie length conditional on animation.

- `y ~ x`: multiple rows and columns. Typically the variable with the greatest number of factors is used for the columns. We can also add marginal plots when using `facet_grid`. By default, `margin=FALSE`.

  ```
  g + facet_grid(Comedy ~ Animation)
  ```

  Figure 4.4 splits movie length by comedy and animation. The panel labels aren't that helpful - they are either `0` or `1`. By default `ggplot2` uses the values set in the data frame. Typically I use more descriptive names in my data frame so the default is more appropriate.

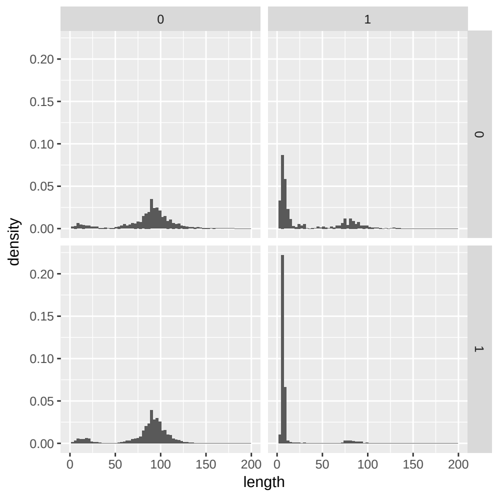

Figure 4.4: Movie length conditional on animation and action status.

Controlling facet scales

For both `facet_grid` and `facet_wrap` we can allow the scale to be the same in all panels (fixed) or vary between panels. This is controlled by the `scales` parameter in the `facet_*` function:

- `scales = 'fixed'`: x and y scales are fixed across all panels (default).
- `scales = 'free'`: x and y scales vary across all panels.
- `scales = 'free_x'`: the x scale is free.
- `scales = 'free_y`: the y scale is free.

We will experiment with these in the practical session.

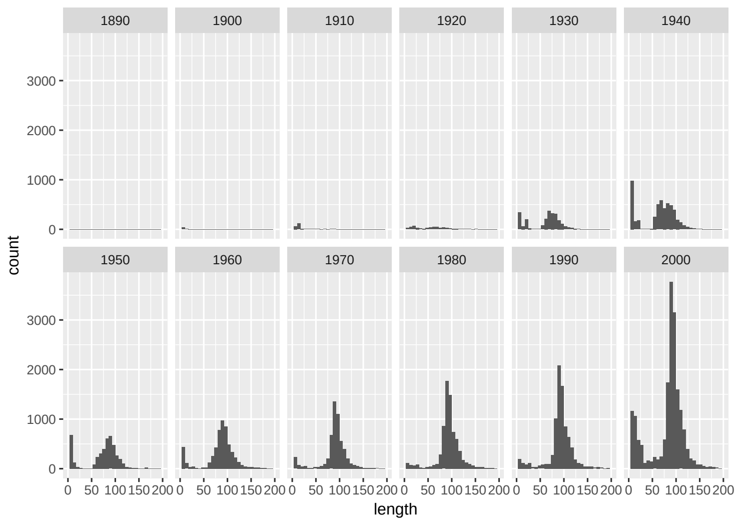

Figure 4.5: Movie length conditional on the decade the movie was created.

Facet wrap

The `facet_wrap` function creates a 1d ribbon of plots. This can be quite handy when trying to save space. To illustrate, let's examine movie length by decade. First, we a create new variable for the movie decade:[1]

[1] The function `round_any` is part of the `plyr` package.

```
movies$decade = round_any(movies$year, 10, floor)
```

Then to generate the ribbon of histograms histograms, we use the `facet_wrap` function:

```
ggplot(movies, aes(x=length)) + geom_histogram() +
  facet_wrap( ~ decade, ncol=6) + xlim(0, 200)
```

to figure 4.5. As before, we truncate the x-axis. Since we have counts on the y-axis, we notice that the number of movies made has increased through time. Also, shorter movies were popular in the 1950's and 1960's.

5
Scales

Axis scales

When we create complex plots involving multiple layers, `ggplot2` uses an iterative process to calculate the correct scales. For example, if in figure 3.11 we only plotted the regression lines, `ggplot2` would reduce the y-axis scale. We can specify set scales using the `xlim` and `ylim` functions. However, if we use these functions, any data that falls outside of the plotting region isn't plotted **and** isn't used in statistical transformations. For example, when calculating the `binwidth` in histograms. If you want to zoom into a plot region, then use `coord_cartesian(xlim = c(.., ..))` instead.

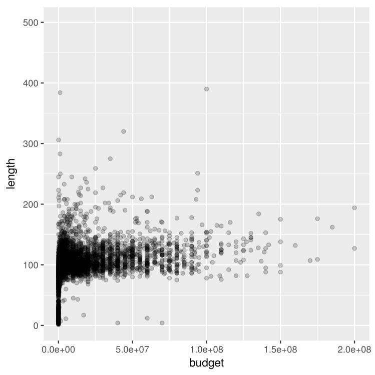

Figure 5.1: Scatter plot of movie budget against length.

At times, we may want to transform the data. A standard example is the log transformation. Suppose we wanted to create a scatter plot of length against budget. We remove any movies that have a zero budget or length. Then we use the following commands

```
data(movies, package="ggplot2movies")
h = ggplot(subset(movies, length>0 & budget>0),
  aes(y=length)) + ylim(0, 500)
h1 = h +  geom_point(aes(budget), alpha=0.2)
```

to get figure 5.1. Notice that we have changed the alpha transparency value to help with over plotting.

To plot the log budgets, there are two possibilities. First, we could transform the scale

```
h2 = h + geom_point(aes(log10(budget)), alpha=0.2)
```

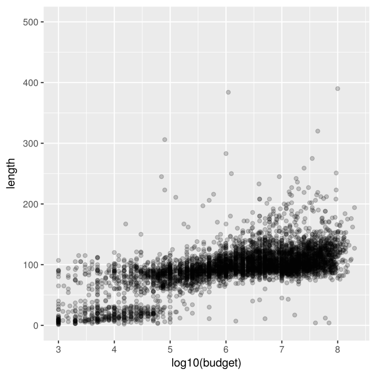

Figure 5.2: Scatter plot of movie log10(budget) against length.

to get figure 5.2. Note that `ylim(0, 500)` is shorthand for `scale_y_continuous(limits=c(0, 500))`. Alternatively, we can transform the data:

```
h3 = h1 + scale_x_log10()
##Or equivalently
h1 + scale_x_continuous(trans="log10")
```

to get figure 5.3. Figures 5.2 and 5.3 are identical, but in figure 5.3 we are still using the original scale. To generate figure 5.3 we used `scale_x_log10()` this is a convenience function of the `scale_x_continuous(trans="log10")` function. Some standard scale transformations are given in table 5.1. As an aside, the `scale` functions are fundamentally different from `geom`'s, since they don't add a layer to the plot.

The `scale_*` functions can also adjust the tick marks and labels. For example,

```
h4 = h3 +
  scale_y_continuous(breaks=seq(0, 500, 100),
                     limits=c(0, 500),
                     minor_breaks = seq(0, 500, 25),
                     labels=c(0, "", "", "", "", 500),
                     name="Movie Length")
```

gives figure 5.4. If you just want to change the x-axis limits or name, then you can use the convenience functions `xlim` and `xlab`. There are similar functions for the y-axis.

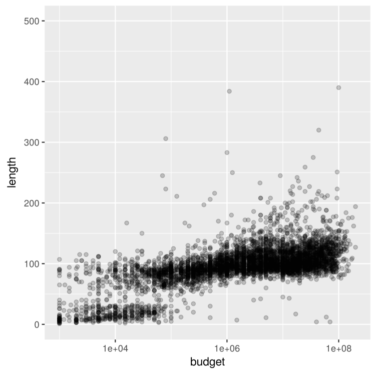

Figure 5.3: Scatter plot of movie budget against length, with the budget data transformed.

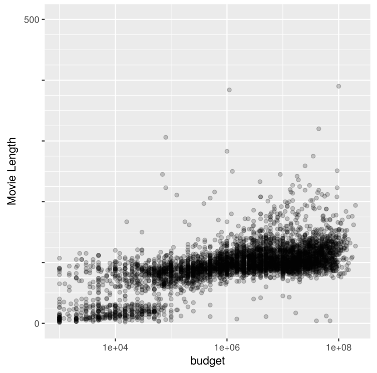

Figure 5.4: Scatter plot of movie budget against length. Using `scale_y_continuous` gives us more control of tick marks and grid lines.

Function	Description
`*_continuous(...)`	Main scale function.
`*_log10(...)`	$\log_{10}$ transformation.
`*_reverse(...)`	Reverse the axis.
`*_sqrt(...)`	The square root transformation.
`*_datetime(...)`	Precise control over dates and times.
`*_discrete(...)`	Not usually needed - see §6.3 of Wickham, 2009.

Table 5.1: Standard scales in `ggplot2`. In the above, replace `*` with either `scale_x` or `scale_y`. Common arguments are `breaks`, `labels`, `na.value`, `trans` and `limits`. See the help files for further details.

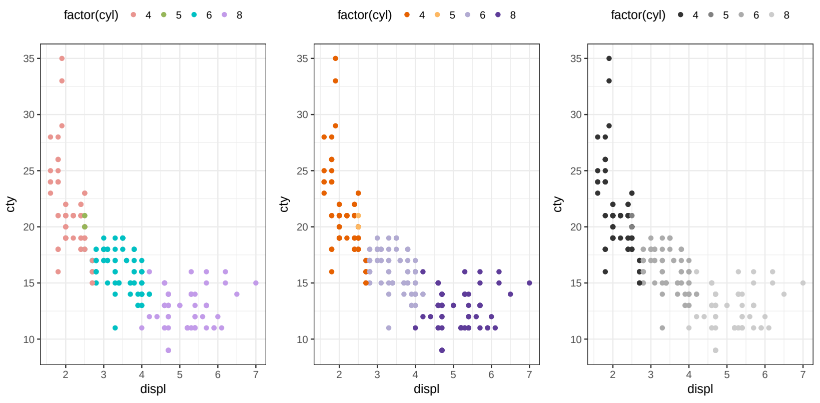

Figure 5.5: Scatter plots of the `mpg` data set showing different colour schemes. The theme has been changed to `theme_bw()`.

Colour and fill scales

For discrete data, there are two methods for choosing colour schemes. One that chooses colours in an automated way and another from hand-picked sets. The default is `scale_colour_hue()`, which picks evenly spaced hues around the hcl colour scheme.

Discrete colours

As a test, we will use the scatter plot from chapter 2:

```
g = ggplot(data=mpg, aes(x=displ, y=cty)) +
  geom_point(aes(colour=factor(cyl)))
```

We can alter the hue and intensity of the colours (figure 5.5a):[1]

```
g + scale_colour_hue(l=70, c=60)
```

or use predefined colour palettes from colour brewer (figure 5.5b):[2]

```
g + scale_colour_brewer(palette="PuOr", type="div")
```

or specify our own colour schemes:

```
g  + scale_colour_manual(
   values=c("4"="red", "5"="blue",
     "6"="green", "8"="black"))
```

For black and white, you can always use:

```
g + scale_colour_grey()
```

to get figure 5.5c.

Continuous Colour

When we have continuous parameters, we use a gradient of colour, instead of discrete values. There are three types of continuous colour gradients[3]:

[1] If you want to change the `fill` aesthetic, use `scale_fill_*`.

[2] There are three possible types: `seq` (sequential), `div` (diverging) and `qual` (qualitative). See `http://colorbrewer2.org/` for other palettes.

[3] The * can be either `fill` or `colour`.

- `scale_*_gradient`: a two colour gradient, with arguments `low` and `high` to control the end points.
- `scale_*_gradient2`: a three colour gradient. As above, with additional arguments: `mid` (for the colour) and `midpoint`. The midpoint defaults to 0, but can be set to any value.
- `scale_*_gradientn`: an n-colour gradient. This requires a vector of colours, which default to being evenly spaced.

See the associated help pages for examples.

Multiple plots

When we want to create a figure in base graphics that contains multiple plots, we use the `par` function. For example, to create a 2×2 plot, we would use

```
par(mfrow=c(2, 2))
```

In ggplot2, we can do something similar. Using the `gridExtra` package, we have

```
library("gridExtra")
grid.arrange(g1, g2, g3, g4, nrow=2)
```

where `g1`, `g2`, `g3` and `g4` are standard ggplot2 graph objects.

An alternative way of creating figure grids, is to use viewports. First, we load the `grid` package and create a convenience function

Using viewports gives you more flexibility, but is more complicated.

```
library("grid")
vplayout = function(x, y)
  viewport(layout.pos.row = x, layout.pos.col = y)
```

Next we create a new page, with a 2×2 layout

```
grid.newpage()
pushViewport(viewport(layout = grid.layout(2, 2)))
```

Finally, we add the individual graphics. The plot created using the `h` object, is placed on the first row and spans both columns:

```
print(g1, vp = vplayout(1, 1:2))
```

The others figures are placed on the second row (figure 5.6):

```
print(g2, vp = vplayout(2, 1))
print(g3, vp = vplayout(2, 2))
```

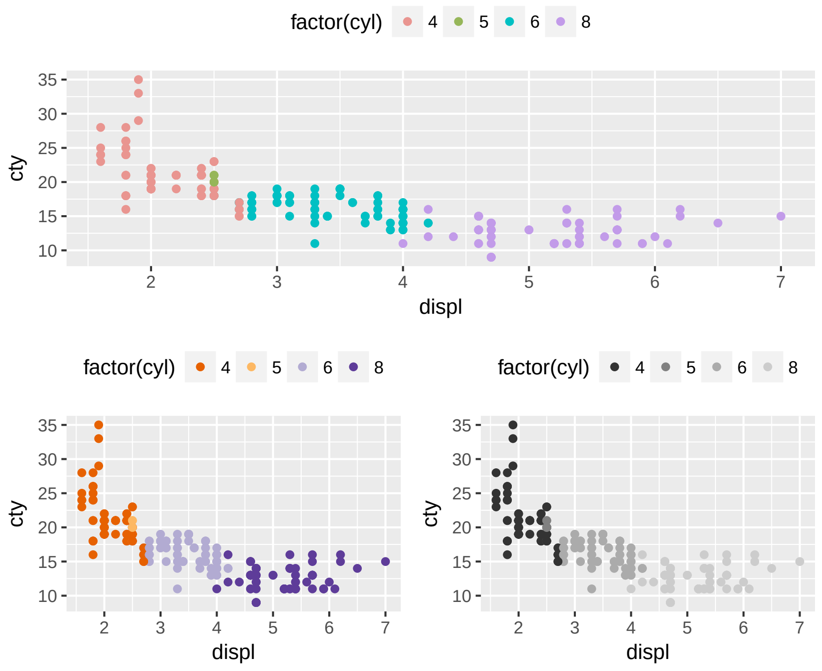

Figure 5.6: An example plot using the viewports. The top plot is spans two columns.

Other topics

There are a few topics that I have skipped, mainly due to space and time.

- themes: if you want to make consistent changes to all your plots - say reduce the font size, then you should use themes. One useful theme is `theme_bw()`. This can be set globally using `theme_set(theme_bw())` or using the standard notation: `+ theme_bw()`.
- coordinate systems: unlike transforming data or scales, transforming the coordinate system transforms the *appearance* of the geoms. For example, a rectangle becomes a doughnut; in a map projection, the shortest path will no longer be a straight line. See §7.3 of the ggplot2 book for further details.
- Multiple plots: this includes having sub-figures on top of larger figures or multiple plots on a single page. See §8.4 in the ggplot2 book.
- Legend manipulation: changing legend titles and positions.
- There is also a `geom_map` for plotting maps. However, I haven't really used this in earnest. There is also a `ggmap` package that might be worth looking at.

Appendix

Course R package

This course has an associated R package. Installing this package is straightforward. First install the `drat` package

```
install.packages("drat")
```

Then run the command[4]

[4] This adds a new repo URL to you list of repositories.

```
drat::addRepo("rcourses")
```

Then install the package as usual

```
install.packages("nclRggplot2")
```

To load the package, use

```
library("nclRggplot2")
```

Bibliography

P Murrell. *R Graphics*. CRC Press, 2 edition, 2011.

D Sarkar. *Lattice: Multivariate Data Visualization with R (Use R!)*. Springer, 1st edition, 2008.

H Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York, 2009. ISBN 978-0-387-98140-6.

L Wilkinson. *The Grammar of Graphics*. Springer, 1st edition, 1999.