

# Data Science with R

## Exploring Data with GGPlot2

Graham.Williams@togaware.com

22nd September 2013

Visit <http://onepager.togaware.com/> for more OnePageR's.

The `ggplot2` (Wickham and Chang, 2013) package implements a grammar of graphics. The basic principle is to build the graphics through a series of layers, and commands. The idea is to build up the plot from the dataset and the aesthetics (often the x-axis and y-axis) of the plot. We then add geometric elements, statistical operations, scales, facets, coordinates, and options.

In this module we explore many of the kinds of plots demonstrating some of the options available for visualizing our data.

The required packages for this module include:

```
library(ggplot2)
library(scales)
library(rattle)
library(gridExtra)
library(wq)
library(xkcd)
library(extrafont)
```

As we work through this module, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the `?` command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the `help=` option of `library()`:

```
library(help=rattle)
```

This module is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

Copyright © 2013 Graham J Williams. You can freely copy, distribute, transmit, adapt, or make commercial use of this module, as long as the attribution is retained and derivative work is provided under the same license.



# 1 Preparing the Dataset

We use the relatively large **weatherAUS** dataset from **rattle** ([Williams, 2013](#)) to illustrate the capabilities of **ggplot2**.

```
library(rattle)
dsname <- "weatherAUS"
ds      <- get(dsname)
```

The dataset is summarised below.

```
dim(ds)
## [1] 75136    24

names(ds)
## [1] "Date"          "Location"       "MinTemp"        "MaxTemp"
## [5] "Rainfall"      "Evaporation"    "Sunshine"       "WindGustDir"
## [9] "WindGustSpeed" "WindDir9am"     "WindDir3pm"     "WindSpeed9am"
## [13] "WindSpeed3pm"  "Humidity9am"    "Humidity3pm"    "Pressure9am"
....

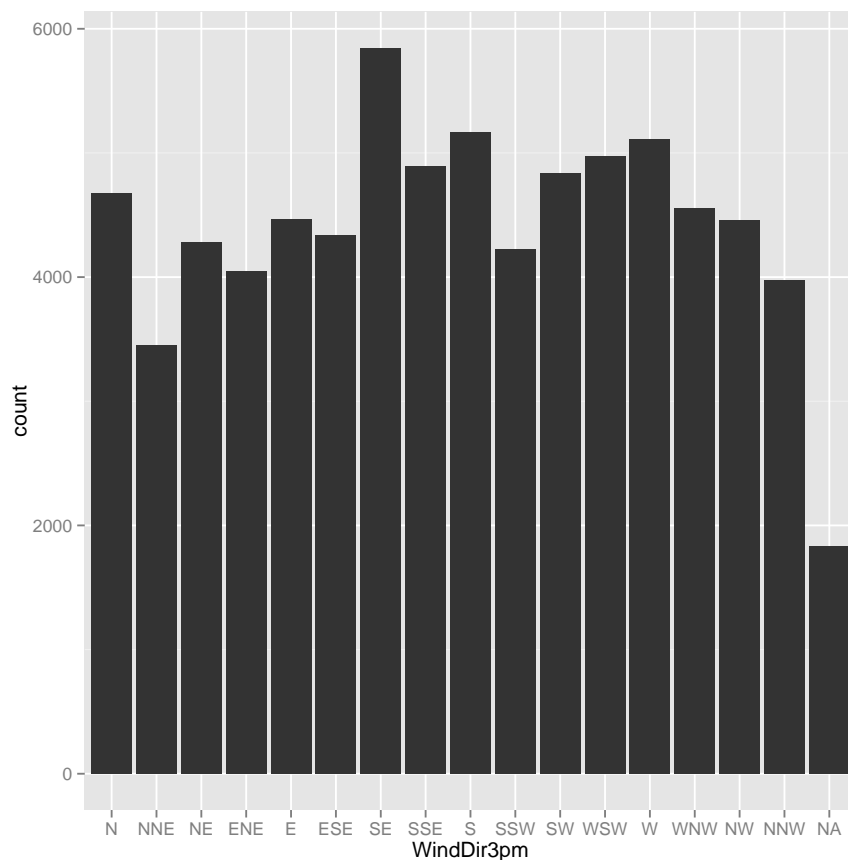
head(ds)
##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-01  Albury    13.4    22.9      0.6          NA         NA
## 2 2008-12-02  Albury     7.4    25.1      0.0          NA         NA
## 3 2008-12-03  Albury    12.9    25.7      0.0          NA         NA
....

tail(ds)
##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 75131 2013-07-25  Darwin    17.0    30.3         0           9.0      11.2
## 75132 2013-07-26  Darwin    17.3    30.6         0          10.2      11.2
## 75133 2013-07-27  Darwin    17.6    31.0         0           7.4      10.4
....

str(ds)
## 'data.frame': 75136 obs. of  24 variables:
##  $ Date       : Date, format: "2008-12-01" "2008-12-02" ...
##  $ Location    : Factor w/ 46 levels "Adelaide","Albany",...: 3 3 3 3 3 ...
##  $ MinTemp     : num  13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...
....

summary(ds)
##           Date           Location           MinTemp           MaxTemp
##  Min.       :2007-11-01   Canberra: 2010   Min.       : -8.5   Min.       : -3.7
##  1st Qu.: 2009-12-24   Sydney  : 1918   1st Qu.:  7.3   1st Qu.: 17.6
##  Median : 2011-02-06   Adelaide: 1767   Median : 11.7   Median : 22.0
....
```

## 2 Histogram: Displaying Frequencies



```
p <- ggplot(data=ds, aes(x=WindDir3pm))  
p <- p + geom_bar()  
p
```

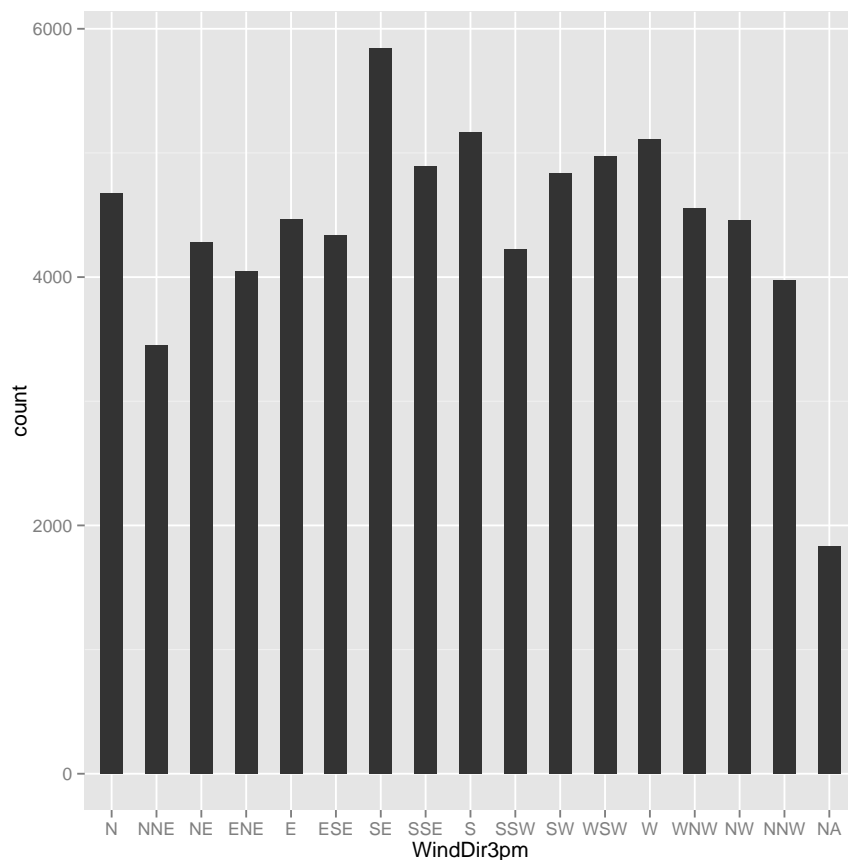
A [histogram](#) displays the frequency of observations using bars. Such plots are easy to display using `ggplot2`, as we see in the above code used to generate the plot. Here the `data=` is identified as `ds` and the `x=` aesthetic is `WindDir3pm`. Using these parameters we then add a bar geometric to build a bar plot for us.

The resulting plot shows the frequency of the levels of the categorical variable `WindDir3pm` from the dataset.

You will have noticed that we placed the plot at the top of the page so that as we turn over to the next page in this module we get a bit of an animation that highlights what changes.

In reviewing the above plot we might note that it looks rather dark and drab, so we try to turn it into an appealing graphic that draws us in to wanting to look at it and understand the story it is telling.

### 3 Histogram: Narrow Bars

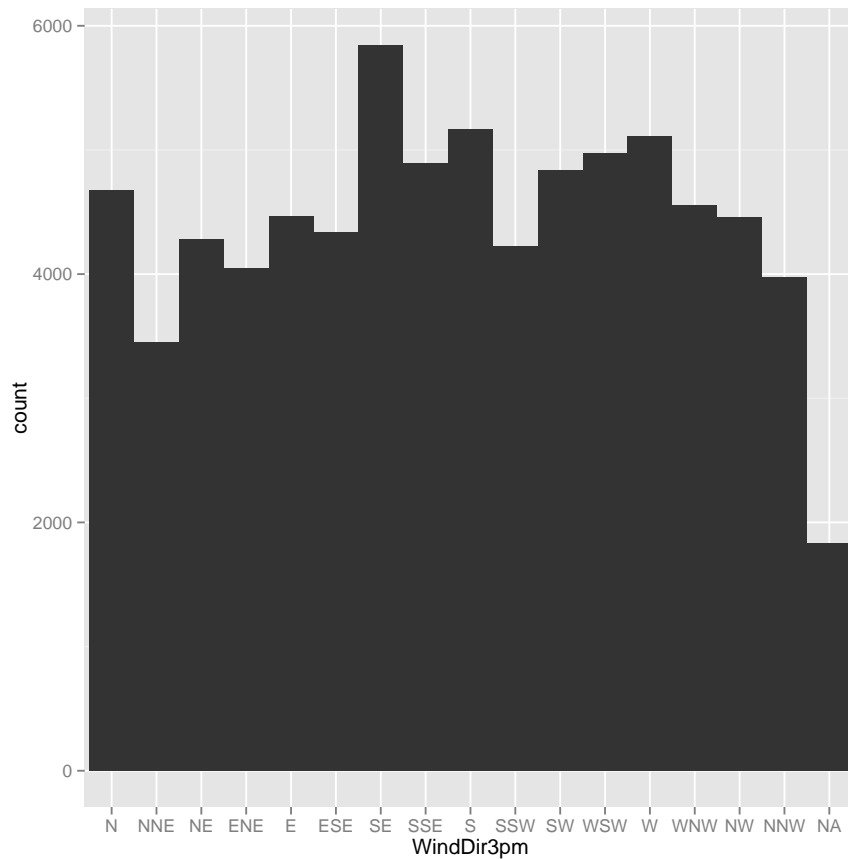


```
p <- ggplot(ds, aes(WindDir3pm))  
p <- p + geom_bar(width=0.5)  
p
```

There are many options available to change the appearance of the histogram to make it look like almost anything we could want. In the following pages we will illustrate a few simpler modifications.

This first example simply makes the bars narrower using the `width=` option. Here we make them half width. Perhaps that helps to make the plot look less dark!

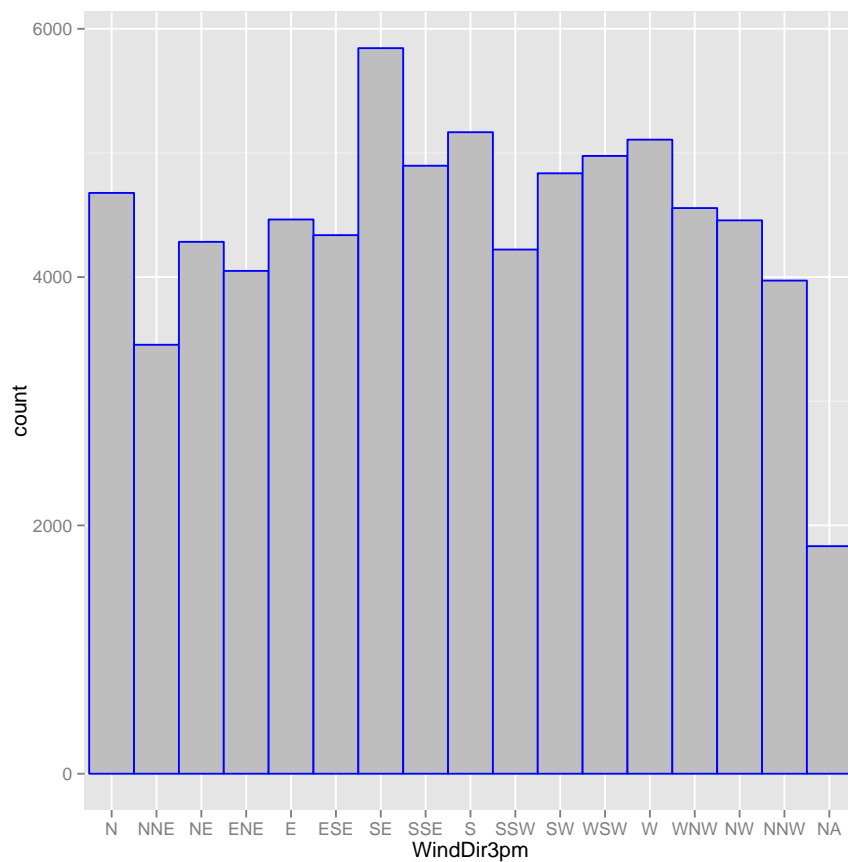
## 4 Histogram: Full Width Bars



```
p <- ggplot(ds, aes(WindDir3pm))  
p <- p + geom_bar(width=1)  
p
```

Going the other direction, the bars can be made to touch by specifying a full width with `width=1`.

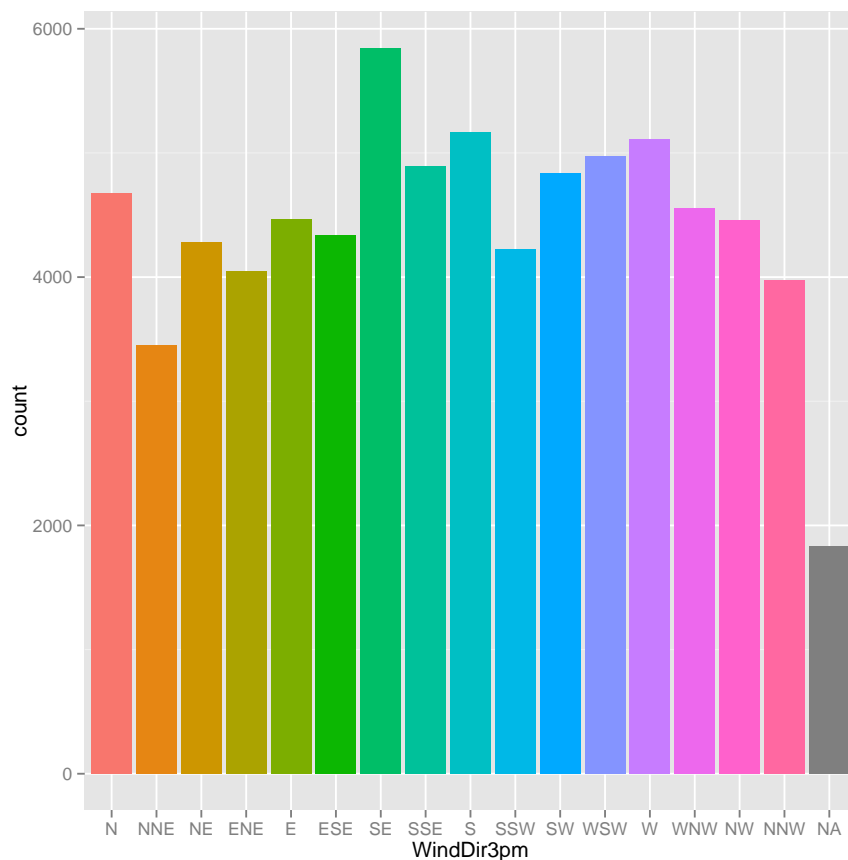
## 5 Histogram: Full Width Bars with Borders



```
p <- ggplot(ds, aes(WindDir3pm))  
p <- p + geom_bar(width=1, colour="blue", fill="grey")  
p
```

We can change the appearance by adding a blue border to the bars, using the `colour=` option. By itself that would look a bit ugly, so we also fill the bars with a grey rather than a black fill. We can play with different colours to achieve a pleasing and personalised result.

## 6 Histogram: Coloured Histogram Without a Legend

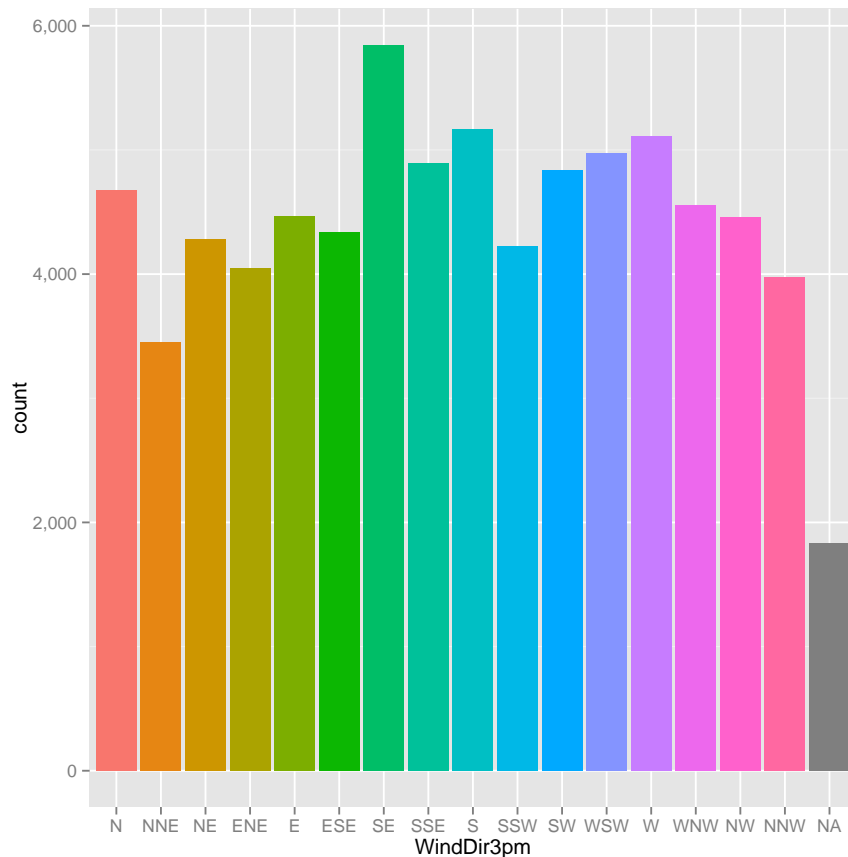


```
p <- ggplot(ds, aes(WindDir3pm, fill=WindDir3pm))
p <- p + geom_bar()
p <- p + theme(legend.position="none")
p
```

Now we really add a flamboyant streak to our plot by adding quite a spread of colour. To do so we simply specify a `fill=` aesthetic to be controlled by the values of the variable `WindDir3pm` which of course is the variable being plotted on the x-axis. A good set of colours is chosen by default.

We add a `theme()` to remove the legend that would be displayed by default, by indicating that the `legend.position=` is `none`.

## 7 Histogram: Comma Formatted Labels



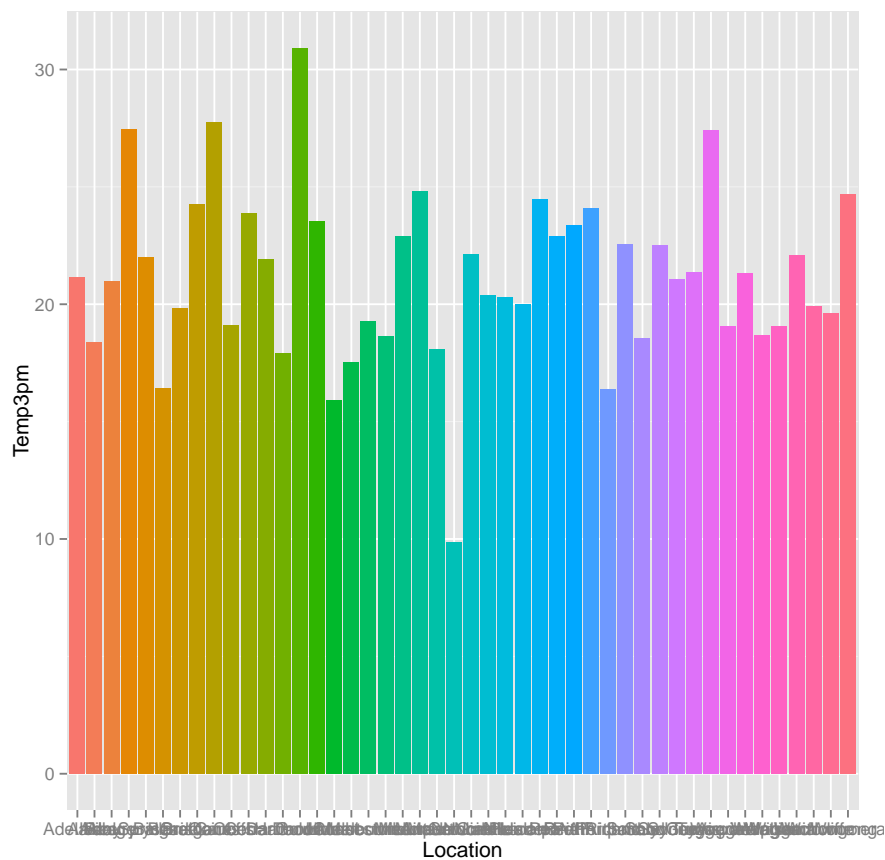
```
p <- ggplot(ds, aes(WindDir3pm, fill=WindDir3pm))
p <- p + geom_bar()
p <- p + scale_y_continuous(labels=comma)
p <- p + theme(legend.position="none")
p
```

Since `ggplot2` Version 0.9.0 the `scales` (Wickham, 2012) package has been introduced to handle many of the scale operations, in such a way as to support base and lattice graphics, as well as `ggplot2` graphics. Scale operations include position guides, as in the axes, and aesthetic guides, as in the legend.

Notice that the y-axis has numbers using commas to separate the thousands. This is always a good idea as it assists us in quickly determining the magnitude of the numbers we are looking at. As a matter of course, I recommend we always use commas in plots (and tables). We do this through `scale_y_continuous()` and indicating `labels=` to include a `comma`.



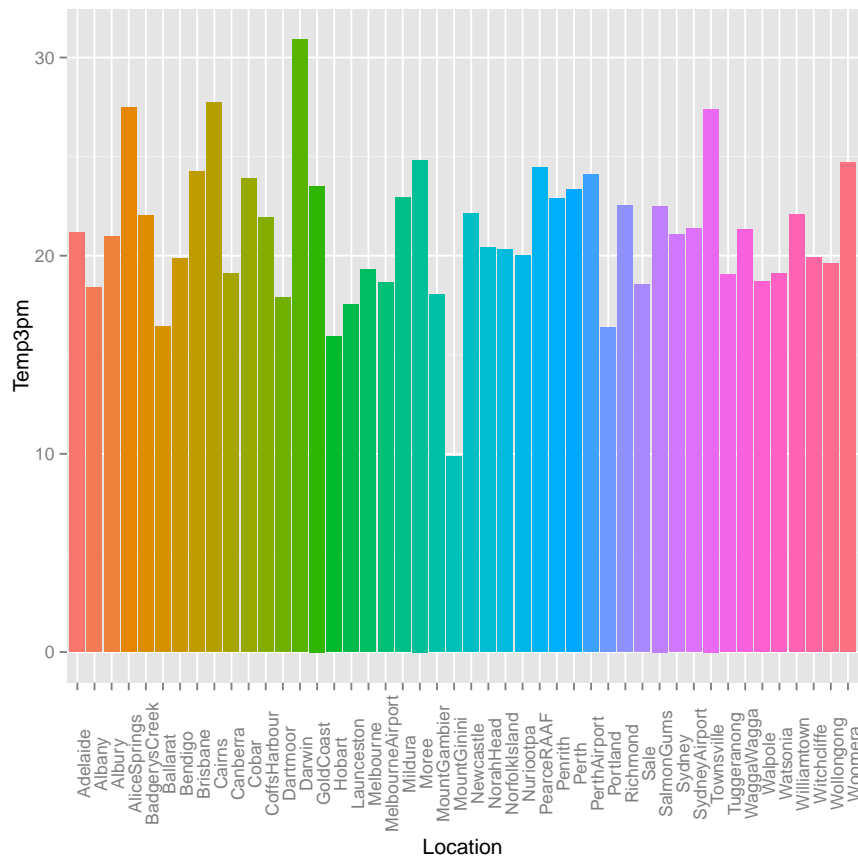
## 8 Histogram: Too Many Bars



```
p <- ggplot(data=ds, aes(x=Location, y=Temp3pm, fill=Location))
p <- p + stat_summary(fun.y="mean", geom="bar")
p <- p + theme(legend.position="none")
p
```

Here we see another interesting plot, showing the mean temperature at 3pm for each location in the dataset. However, we notice that the `Location` labels overlap and are quite a mess. Something has to be done about that.

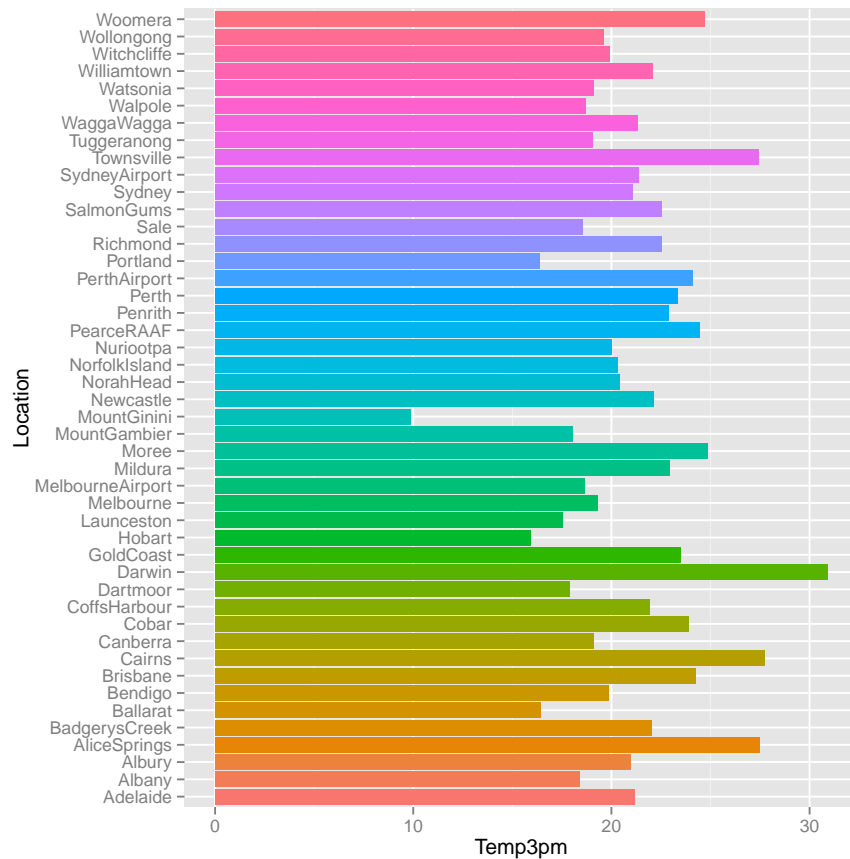
## 9 Histogram: Rotated Labels



```
p <- ggplot(ds, aes(Location, Temp3pm, fill=Location))
p <- p + stat_summary(fun.y="mean", geom="bar")
p <- p + theme(legend.position="none",
               axis.text.x=element_text(angle=90))
p
```

The obvious solution is to rotate the labels. We achieve this through modifying the `theme()`, setting the `axis.text.x=` to be rotated 90 degrees.

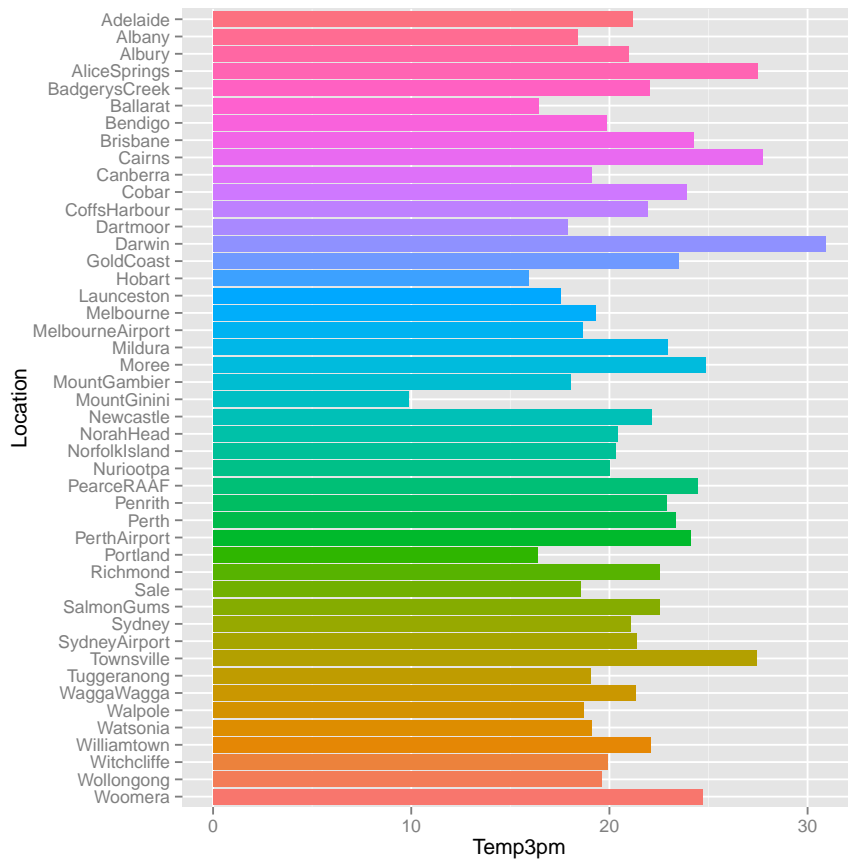
## 10 Histogram: Horizontal Histogram



```
p <- ggplot(ds, aes(Location, Temp3pm, fill=Location))
p <- p + stat_summary(fun.y="mean", geom="bar")
p <- p + theme(legend.position="none")
p <- p + coord_flip()
p
```

Alternatively perhaps it would be better to flip the coordinates and produce a horizontal histogram:

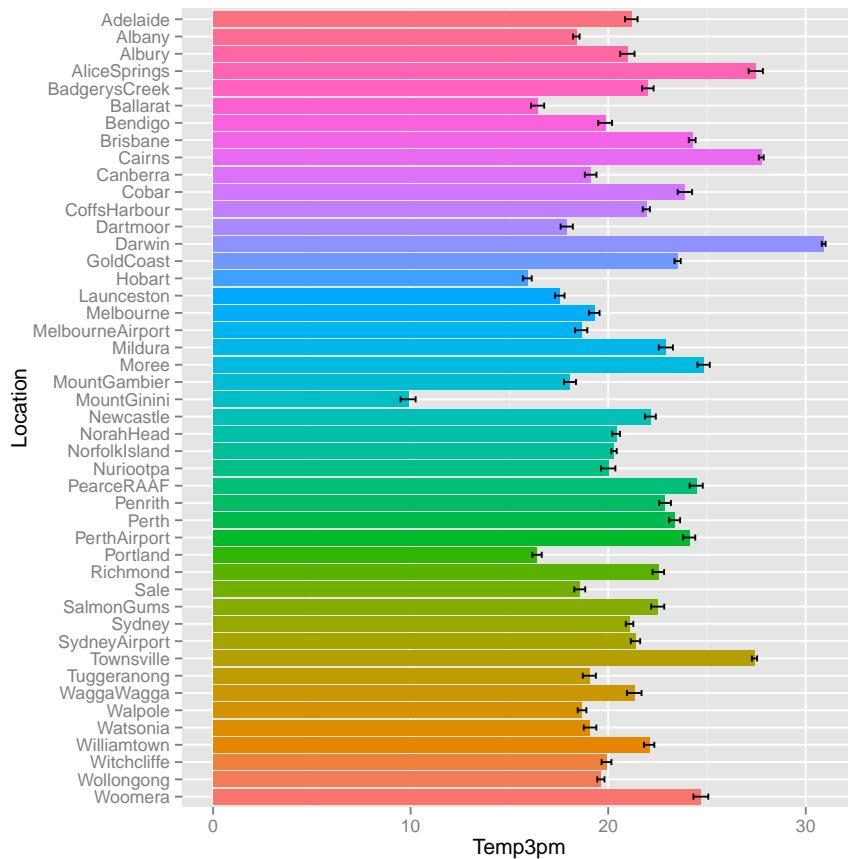
## 11 Histogram: Reorder the Levels



```
ds <- within(ds, Location <- factor(Location, levels=rev(levels(Location))))
p <- ggplot(ds, aes(Location, Temp3pm, fill=Location))
p <- p + stat_summary(fun.y="mean", geom="bar")
p <- p + theme(legend.position="none")
p <- p + coord_flip()
p
```

We also want to have the labels in alphabetic order which makes the plot more accessible. This requires we reverse the order of the levels in the original dataset.

## 12 Histogram: Plot the Mean with CI

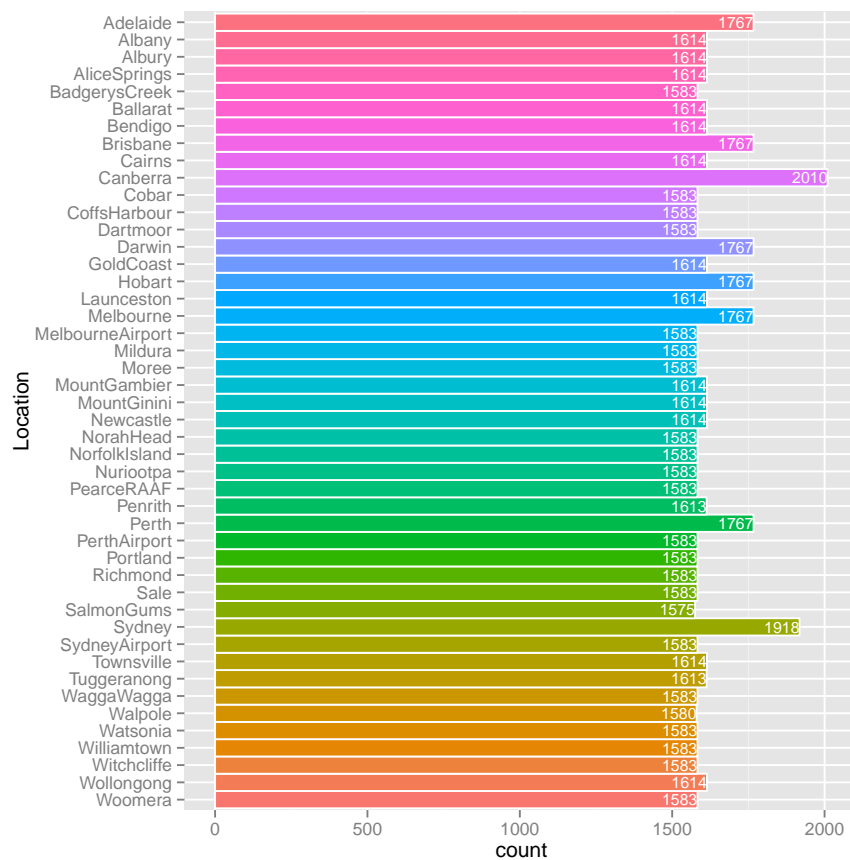


```
p <- ggplot(ds, aes(Location, Temp3pm, fill=Location))
p <- p + stat_summary(fun.y="mean", geom="bar")
p <- p + stat_summary(fun.data="mean_cl_normal",
                      geom="errorbar",
                      conf.int=0.95,
                      width=0.35)

p <- p + coord_flip()
p <- p + theme(legend.position="none")
p
```

Here we add a confidence interval around the mean.

## 13 Histogram: Text Annotations

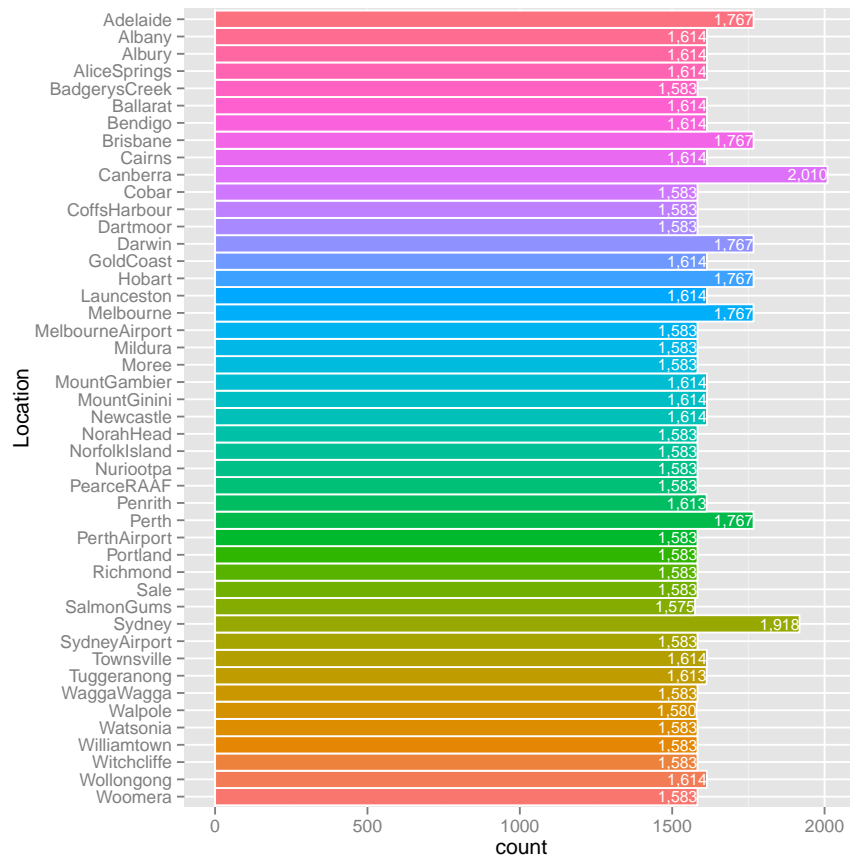


```
p <- ggplot(ds, aes(Location, fill=Location))
p <- p + geom_bar(width=1, colour="white")
p <- p + theme(legend.position="none")
p <- p + coord_flip()
p <- p + geom_text(stat="bin",
  color="white",
  hjust=1.0,
  size=3,
  aes(y=..count.., label=..count..))
p
```

It would be informative to also show the actual numeric values on the plot. This plot shows the counts.

**Exercise:** Instead of plotting the counts, plot the mean Temp3pm, and include the textual value.

## 14 Histogram: Text Annotations with Commas

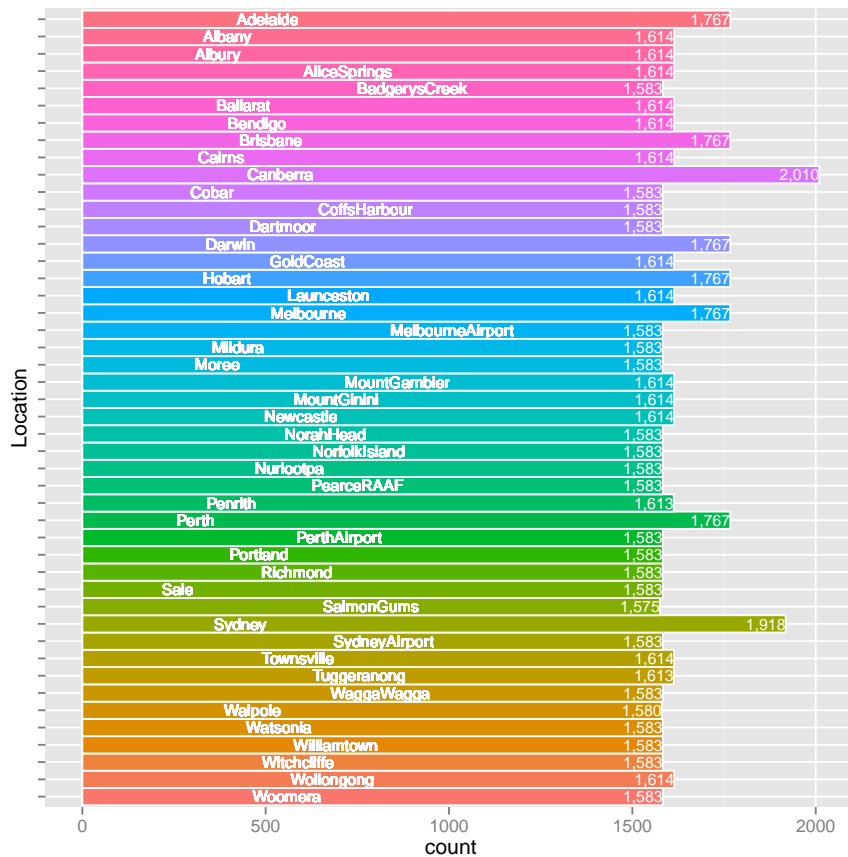


```
p <- ggplot(ds, aes(Location, fill=Location))
p <- p + geom_bar(width=1, colour="white")
p <- p + theme(legend.position="none")
p <- p + coord_flip()
p <- p + geom_text(stat="bin", color="white", hjust=1.0, size=3,
  aes(y=..count.., label=scales::comma(..count..)))
p
```

A small variation is to add commas to the numeric annotations to separate the thousands (Will Beasley 121230 email).

Exercise: Do this without having to use scales::, perhaps using a format.

## 15 Histogram: Multiple Text Annotations



```
p <- ggplot(ds, aes(Location, fill=Location))
p <- p + geom_bar(width=1, colour="white")
p <- p + theme(legend.position="none")
p <- p + coord_flip()
p <- p + geom_text(stat="bin", color="white", hjust=1.0, size=3,
  aes(y=..count.., label=scales::comma(..count..)))
p <- p + geom_text(stat="identity", colour="white", hjust=-2.5, size=3,
  aes(y=0, label=Location))
p <- p + theme(axis.text.y=element_blank())
p
```

We can add Location as a label in the bar rather than on the axis.

**Exercise:** Render the names better. Line the names with the right hjust value. Do this with the means instead of the counts.



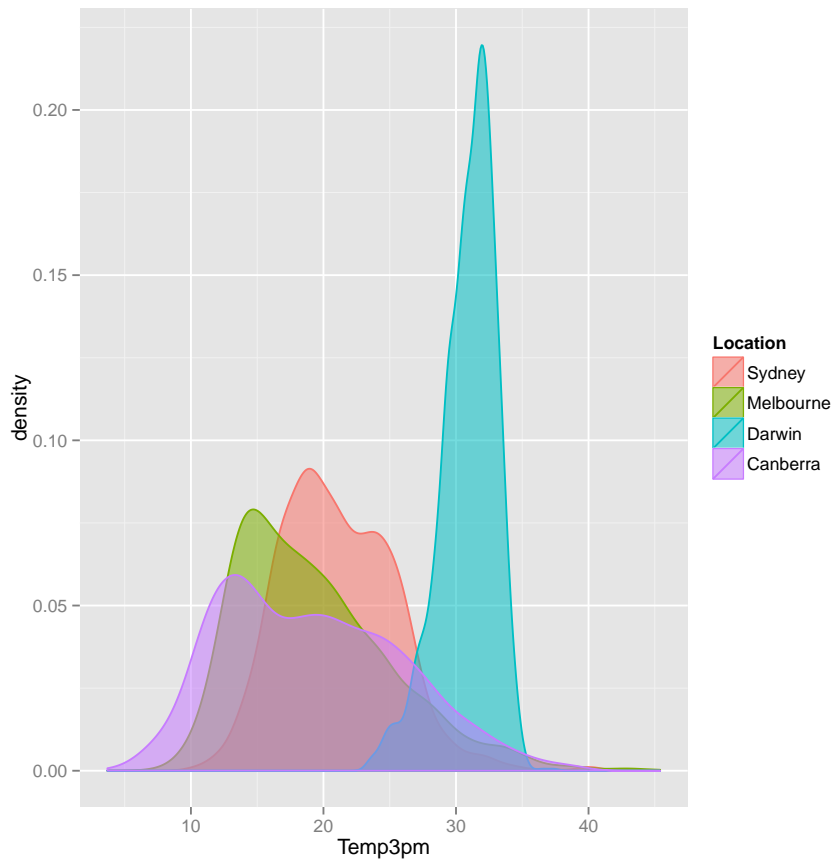
## 16 Histogram: Stacked Bar Chart

Exercise: Generate a stacked bar chart histogram.

## 17 Distributions

Exercise: Generate the basic plot for a distribution.

## 18 Distributions: Transparent Categorical Density Plot



```
cities <- c("Canberra", "Darwin", "Melbourne", "Sydney")
p <- ggplot(subset(ds, Location %in% cities & !is.na(Temp3pm)),
  aes(Temp3pm, colour = Location, fill = Location))
p <- p + geom_density(alpha = 0.55)
p
```

## 19 Distributions: Box Plot

Exercise: Generate a box plot.

## 20 Distributions: Violin Plot

A violin plot is another interesting way to present a distribution, using a shape that resembles a violin.

Exercise: Generate a violin plot.

## 21 Scatter Plot

A [scatter plot](#) displays points scattered over a plot. Locations are specified as x and y for a two dimensional plot.

Exercise: Generate a scatter plot.

## 22 Scatter Plot: Comparing Individual Changes Over Time

```
load(file.path("data", "tolerance.RData"))
dim(tolerance)

## [1] 80  6

names(tolerance)

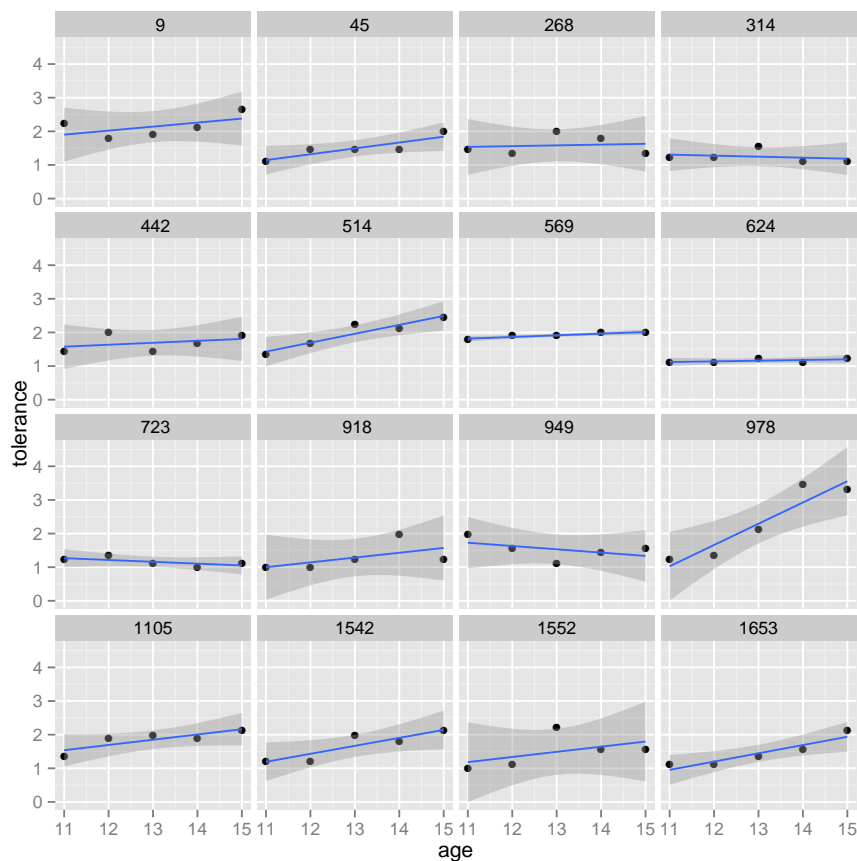
## [1] "id"      "age"      "tolerance" "male"      "exposure" "time"

length(unique(tolerance$id))

## [1] 16
```

Here we use facet to separately plot each entity by their id. The code for the plot comes from <http://heuristically.wordpress.com/2012/03/14/plotting-individual-growth-charts/>.

```
p <- ggplot(tolerance, aes(age, tolerance))
p <- p + geom_point()
p <- p + geom_smooth(method=lm)
p <- p + facet_wrap(~id)
p
```



## 23 Using grid.arrange(): Multiple Plots Code

Here we illustrate the ability to layout multiple plots in a regular grid using `grid.arrange()` from `gridExtra` (Auguie, 2012). We illustrate this with the **weatherAUS** dataset from `rattle`. We generate a number of informative plots, using the `plyr` (?) package to aggregate the cumulative rainfall. The `scales` (Wickham, 2012) package is used to provide labels with commas.

```
library(rattle)
library(plyr)
library(ggplot2)
library(scales)

cities <- c("Adelaide", "Canberra", "Darwin", "Hobart")
ds <- subset(weatherAUS, Location %in% cities & Date >= "2009-01-01")
ds <- within(ds, Rainfall[is.na(Rainfall)] <- 0)
ds <- ddply(ds, .(Location), transform, cumRainfall=cumsum(Rainfall))

p <- ggplot(ds, aes(x=Date, y=cumRainfall, colour=Location))
p1 <- p + geom_line()
p1 <- p1 + ylab("millimetres")
p1 <- p1 + scale_y_continuous(labels=comma)
p1 <- p1 + ggtitle("Cumulative Rainfall")

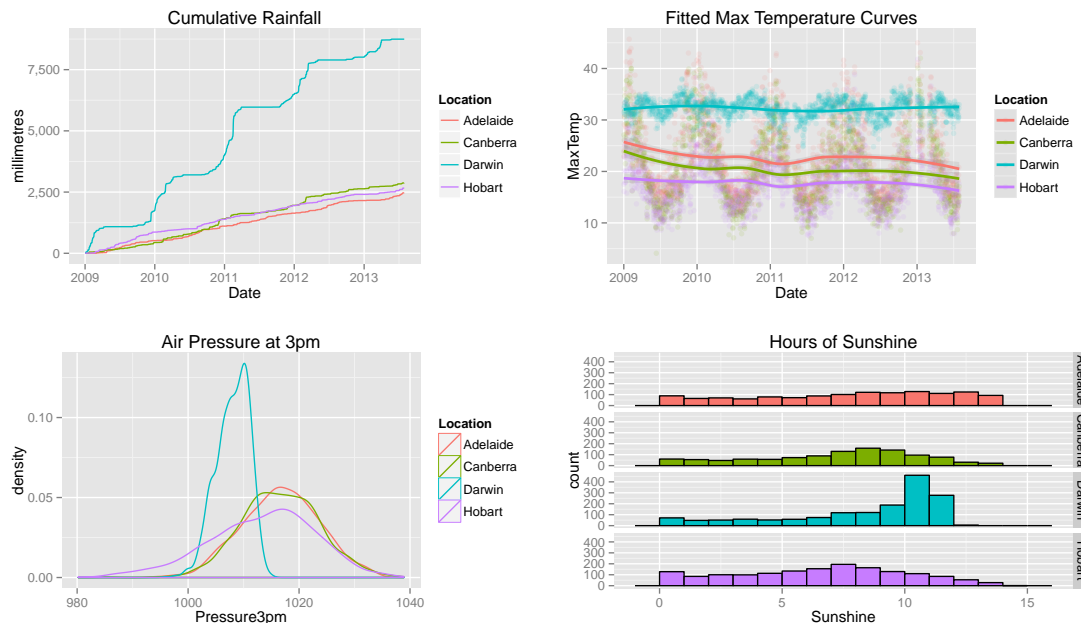
p2 <- ggplot(ds, aes(x=Date, y=MaxTemp, colour=Location))
p2 <- p2 + geom_point(alpha=.1)
p2 <- p2 + geom_smooth(method="loess", alpha=.2, size=1)
p2 <- p2 + ggtitle("Fitted Max Temperature Curves")

p3 <- ggplot(ds, aes(x=Pressure3pm, colour=Location))
p3 <- p3 + geom_density()
p3 <- p3 + ggtitle("Air Pressure at 3pm")

p4 <- ggplot(ds, aes(x=Sunshine, fill=Location))
p4 <- p4 + facet_grid(Location ~ .)
p4 <- p4 + geom_histogram(colour="black", binwidth=1)
p4 <- p4 + ggtitle("Hours of Sunshine")
p4 <- p4 + theme(legend.position="none")
```



## 24 Using grid.arrange(): Multiple Plots



```
library(gridExtra)
grid.arrange(p1, p2, p3, p4)
```

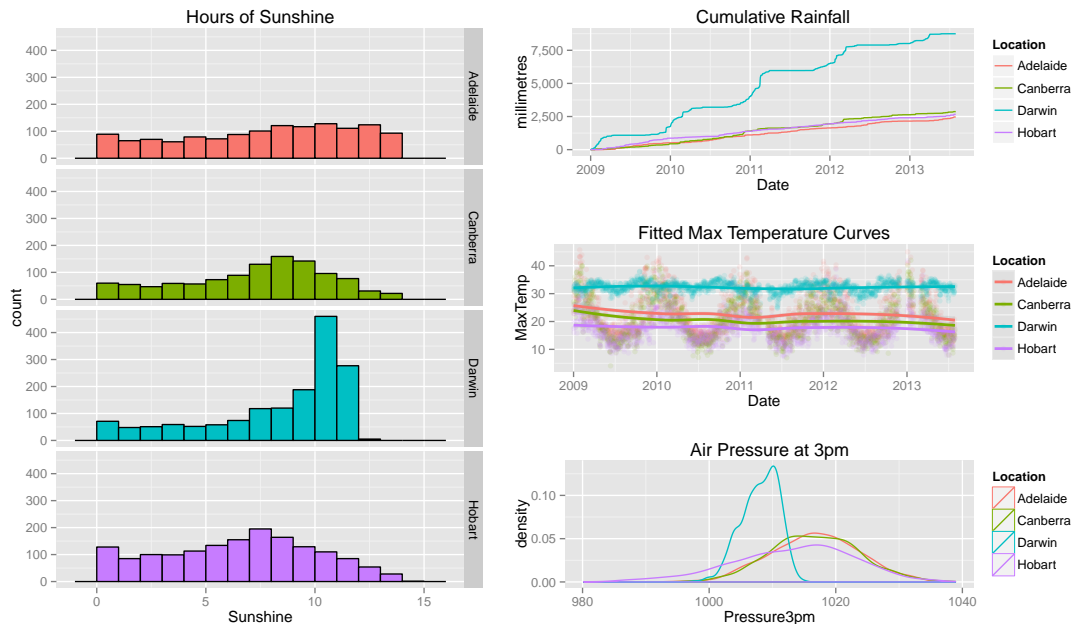
The actual plots are arranged by `grid.arrange()`.

A collection of plots such as this can be quite informative and effective in displaying the information efficiently. We can see that Darwin is quite a stick out. It is located in the tropics, whereas the remaining cities are in the southern regions of Australia.

## 25 Using grid.arrange(): Arranging Plots

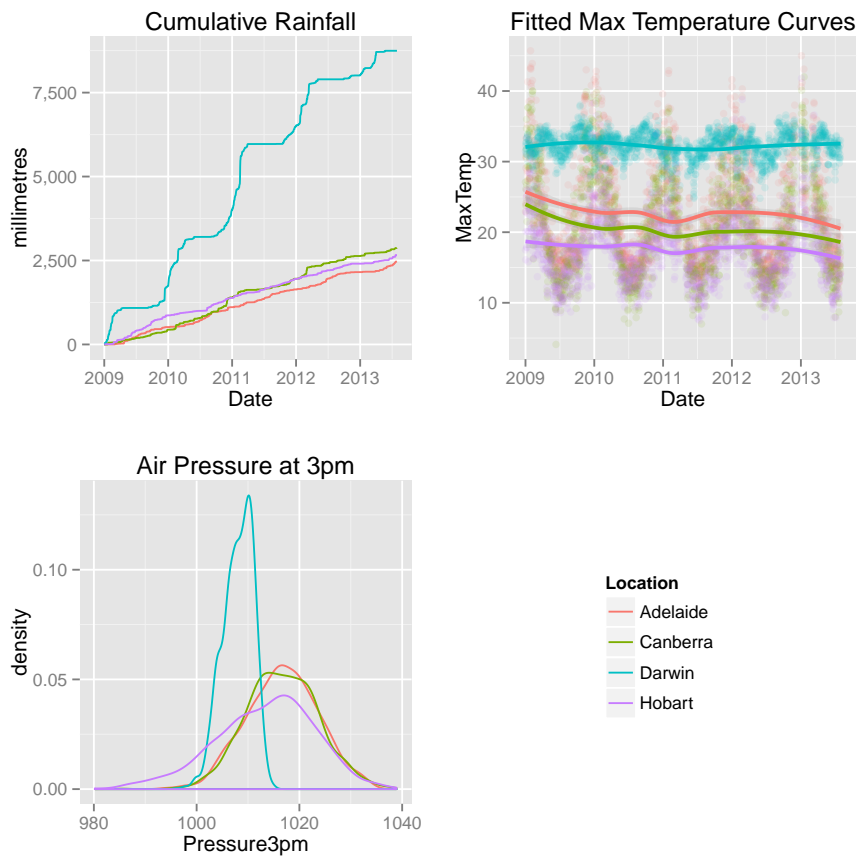
We are able to arrange the plots in quite a flexible manner.

```
grid.arrange(p4, arrangeGrob(p1, p2, p3, ncol=1), ncol=2, widths=c(1,1.2))
```



## 26 Using grid.arrange(): Sharing a Legend

```
plegend<-function(p)
{
  tmp <- ggplot_gtable(ggplot_build(p))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
  legend <- tmp$grobs[[leg]]
  return(legend)
}
legend <- plegend(p1)
grid.arrange(p1 + theme(legend.position="none"),
             p2 + theme(legend.position="none"),
             p3 + theme(legend.position="none"),
             legend)
```



## 27 Using grid.arrange(): 2D Histogram Code

The following code is based on an example from [Michael Kuhn](#). In this code block we generate the `ggplot2` objects that we will then arrange and print on the next page.

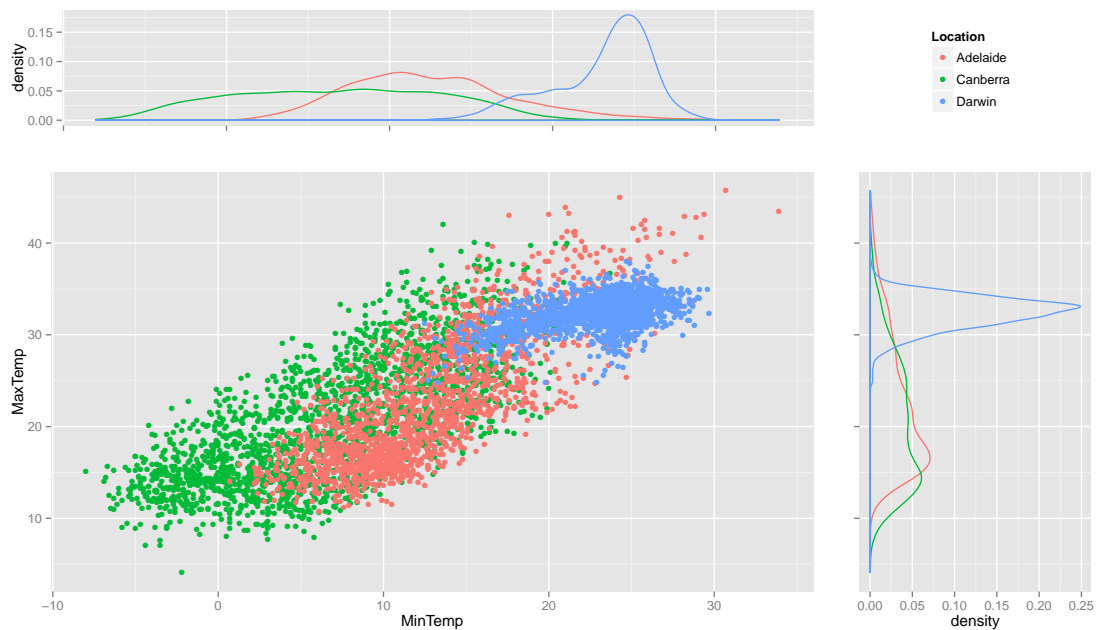
The data we use comes from `rattle`. From the full `weatherAUS` dataset we select a `subset()` covering just three cities. A basic scatter plot is built displaying the minimum and maximum daily temperatures. Two density plots are then generated, one for each of the variables in the scatter plot. The fourth object is the legend for the scatter plot.

```
library(rattle)
library(ggplot2)
ds <- subset(weatherAUS, Location %in% c("Canberra", "Adelaide", "Darwin"))
ds$Location <- ordered(ds$Location)
p <- ggplot(ds, aes(MinTemp, MaxTemp, colour=Location))
p <- p + geom_point()
p5 <- p + theme(legend.position="none")
p6 <- ggplot(ds, aes(x=MinTemp, group=Location, colour=Location))
p6 <- p6 + stat_density(fill=NA, position="dodge")
p6 <- p6 + theme(legend.position="none",
                 axis.title.x=element_blank(),
                 axis.text.x=element_blank())
p7 <- ggplot(ds, aes(x=MaxTemp, group=Location, colour=Location))
p7 <- p7 + stat_density(fill=NA, position="dodge")
p7 <- p7 + coord_flip()
p7 <- p7 + theme(legend.position="none",
                 axis.title.y=element_blank(),
                 axis.text.y=element_blank())
legend <- plegend(p)
```

## 28 Using grid.arrange(): 2D Histogram Plot

Having generated a number of graphical objects, we arrange them using `grid.arrange()` from `gridExtra`.

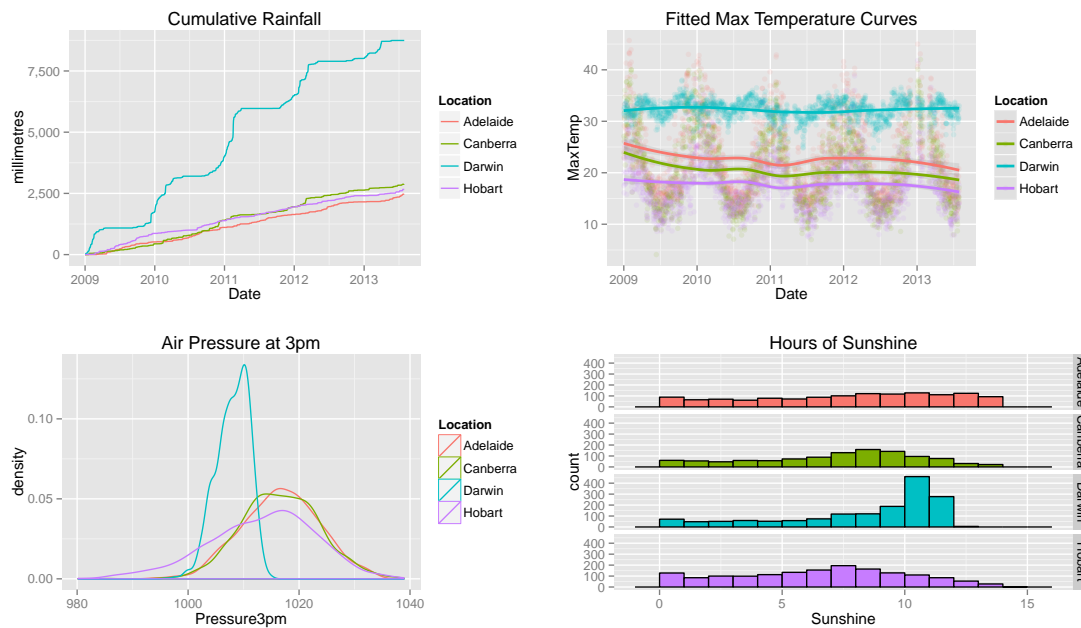
```
library(gridExtra)
grid.arrange(arrangeGrob(p6, legend, p5, p7,
  widths= unit.c(unit(0.75, "npc"), unit(0.25, "npc")),
  heights=unit.c(unit(0.25, "npc"), unit(0.75, "npc")),
  nrow=2))
```



## 29 Using `layout()`: Multiple Plots

Here we illustrate the ability to layout multiple plots in a regular grid using `layout()` from `wq` (Jassby and Cloern, 2013).

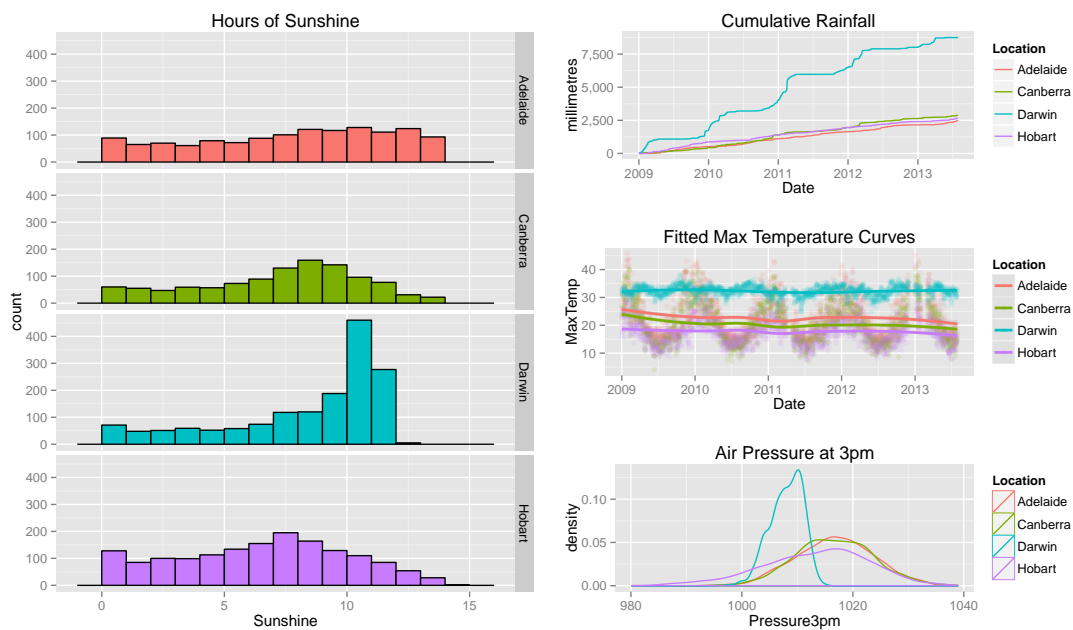
```
library(wq)
layout(list(p1, 1, 1), list(p2, 1, 2), list(p3, 2, 1), list(p4, 2, 2))
```



## 30 Using `layout()`: Arranging Plots

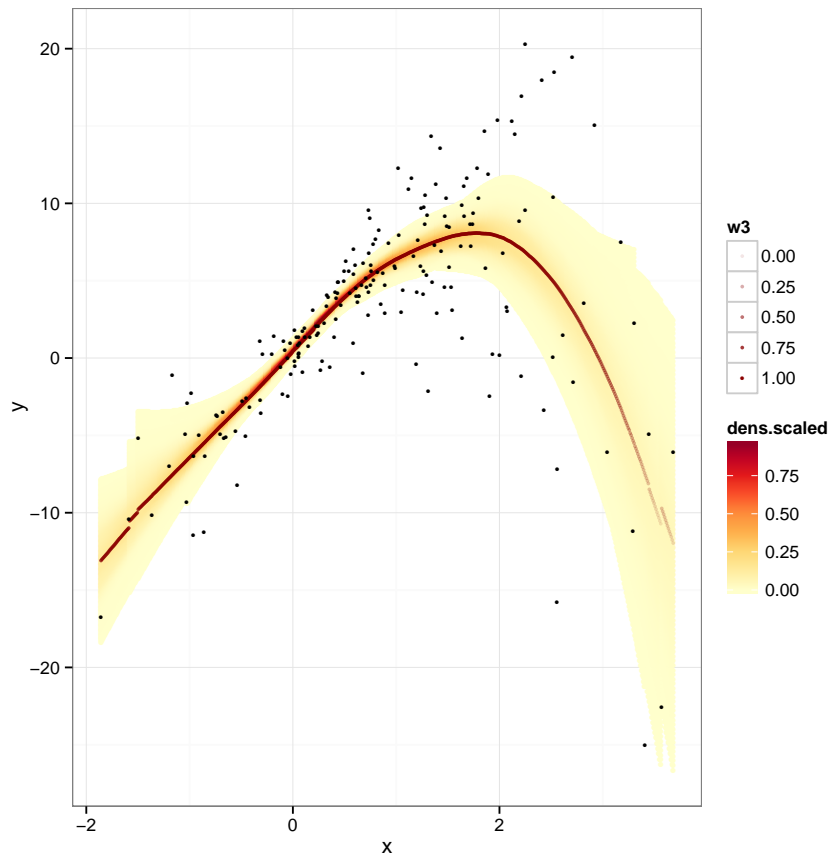
Here we illustrate the ability to layout multiple plots in a grid of differing sizes using `layout()` from `wq`.

```
library(wq)
layout(list(p4, 1:3, 1), list(p1, 1, 2), list(p2, 2, 2), list(p3, 3, 2))
```



## 31 Visually Weighted Regression

From Nirebread [www.nicebread.de](http://www.nicebread.de) (and posted on Bloggers on R) by Felix Schoenbrodt 30 August 2012 addressing Solomon Hsiang's proposal of an appealing method for visually displaying the uncertainty in regressions and using shading in response to Gelman's note that traditional statistical summaries such as 95% intervals give too much weight to the edges.





## 32 F1: Exploring the Dataset

We can now explore a particular dataset using `ggplot2` graphics to get an understanding of the story behind the data. The data and the original plots (some are now modified) are from [Tony Hirst's](#) blog.

```
(load("data/f1.RData"))
```

```
## [1] "f1"
```

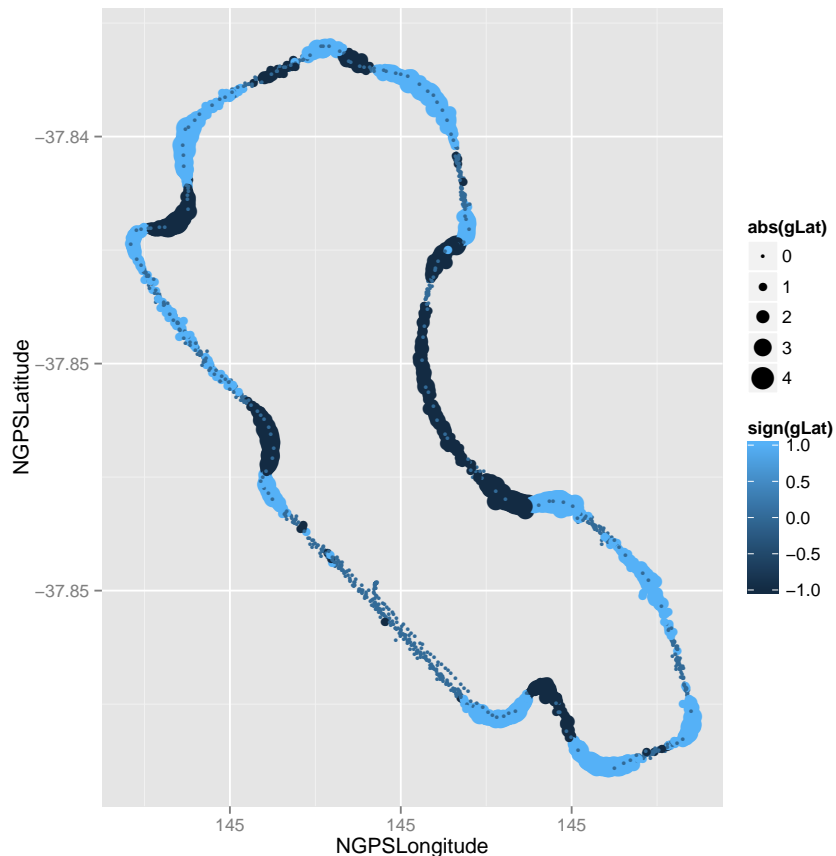
```
head(f1)
```

```
##           file timestamp NGPSLatitude NGPSLongitude NGear nEngine
## 1 1269758114  17:35:10      -37.85           145      4   13422
## 2 1269758115  17:35:11      -37.85           145      3   13383
## 3 1269758116  17:35:12      -37.85           145      3   14145
## .....
```

### 33 F1: Simple Map

We can draw the particular F1 circuit using the longitude and latitude as the x and y coordinates, and using the sign of the latitudinal g-force on the driver. We believe that a positive value of `gLat` indicates force to the left and a negative value indicates a force to the right.

```
p <- ggplot(f1, aes(NGPSLongitude, NGPSLatitude))
p <- p + geom_point(aes(col=sign(gLat), size=abs(gLat)))
p
```



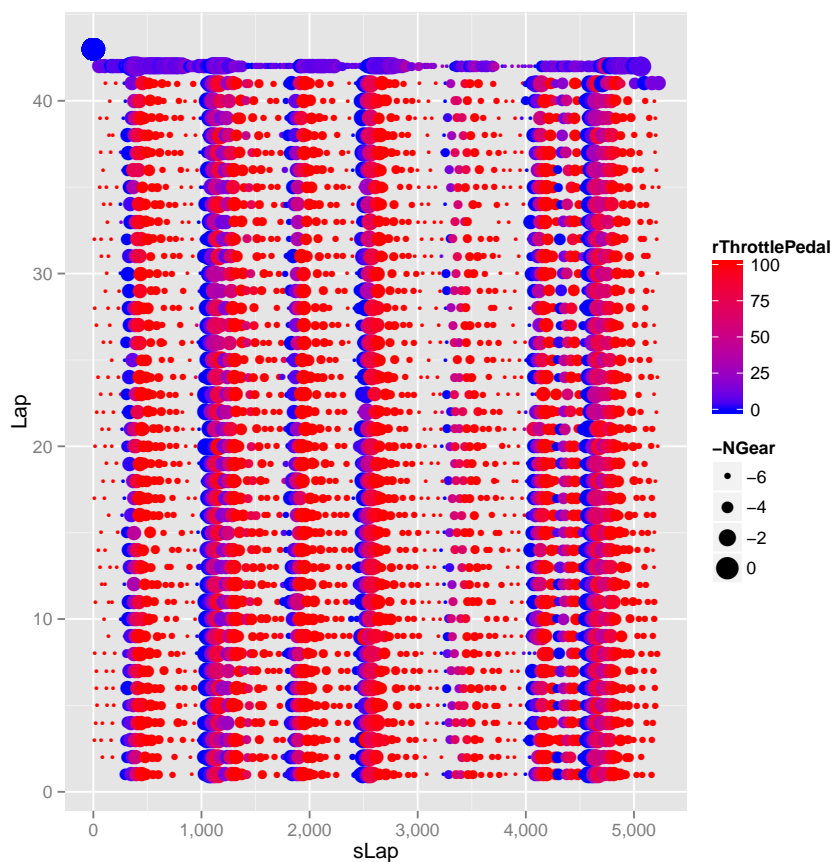
Based on Tony Hirst's Blog Post, March 2012

We should be able to see from the plot the forces on the driver on the left and right hand corners, and see how tight the corner is based on the size of the dots.

## 34 F1: Driver Behaviour

We can explore the driver's behaviour in using low gear and throttle. The distance around the track is plotted on the x-axis and the lap number on y axis. The node size is inversely proportional to gear number (low gear, large point size) and the colour is the relative amount of throttle pedal depression.

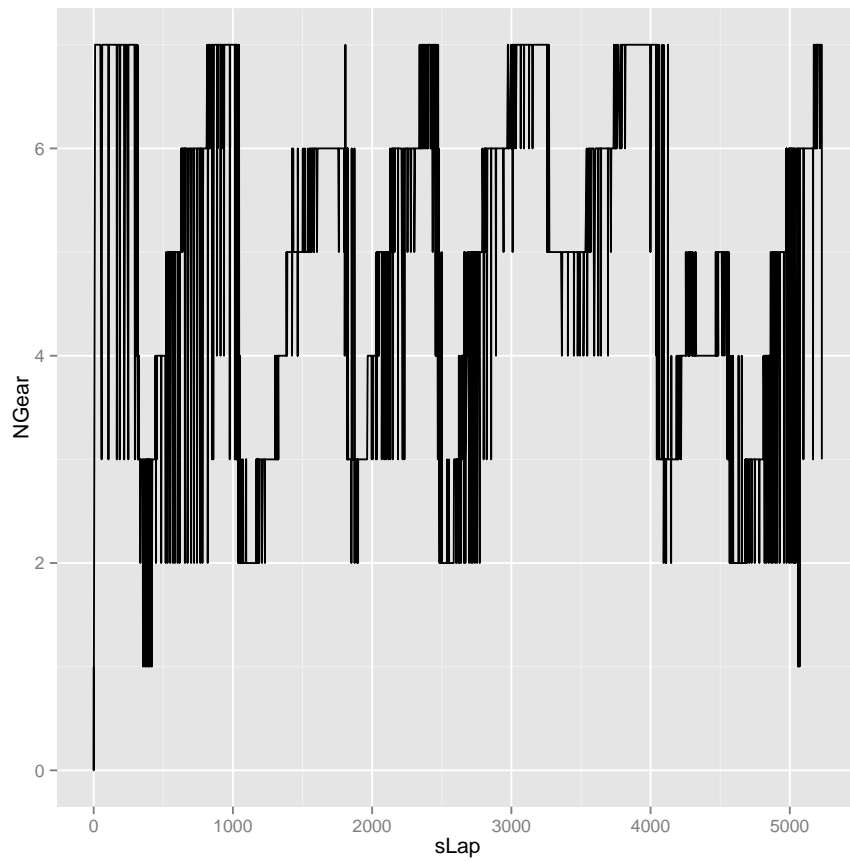
```
library(scales)
p <- ggplot(f1, aes(sLap, Lap))
p <- p + geom_point(aes(col=rThrottlePedal, size=-NGear))
p <- p + scale_colour_gradient(low="blue", high="red")
p <- p + scale_x_continuous(labels=comma)
p
```



Based on Tony Hirst's Blog Post, March 2012

## 35 F1: Gear Usage Around the Track

```
p <- ggplot(f1, aes(sLap, NGear))  
p <- p + geom_line()  
p
```

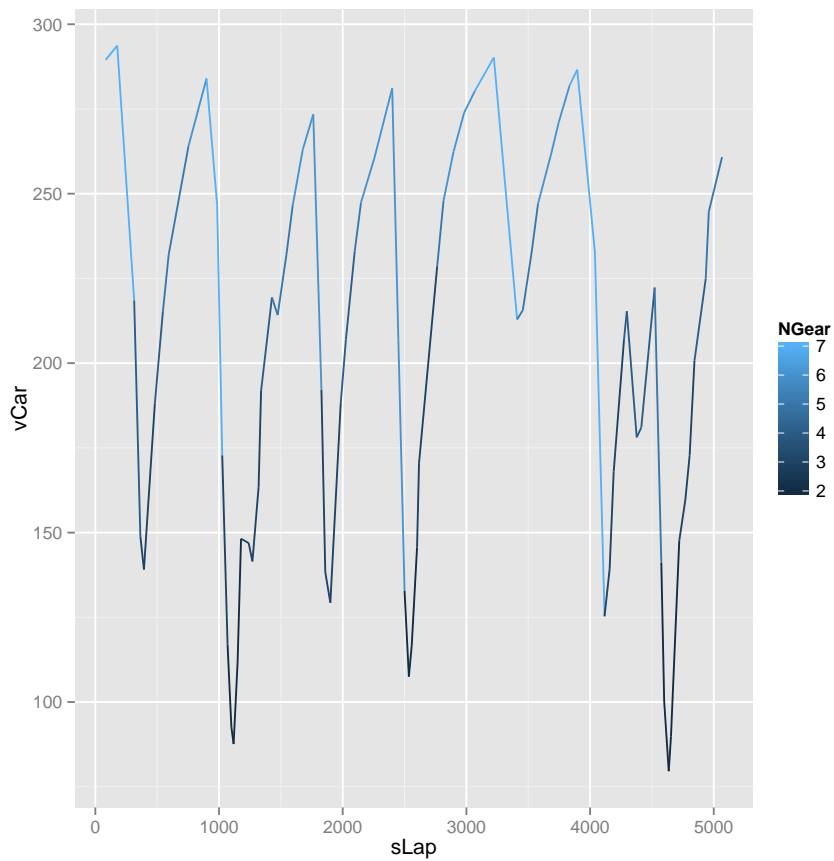


Based on [Tony Hirst's Blog Post](#), March 2012

## 36 F1: Trace a Single Lap

We can trace a single lap to display the speed (y-axis) coloured by gear as the vehicle travels around the circuit:

```
ggplot(subset(f1, Lap==2), aes(sLap, vCar)) +  
  geom_line(aes(colour=NGear))
```

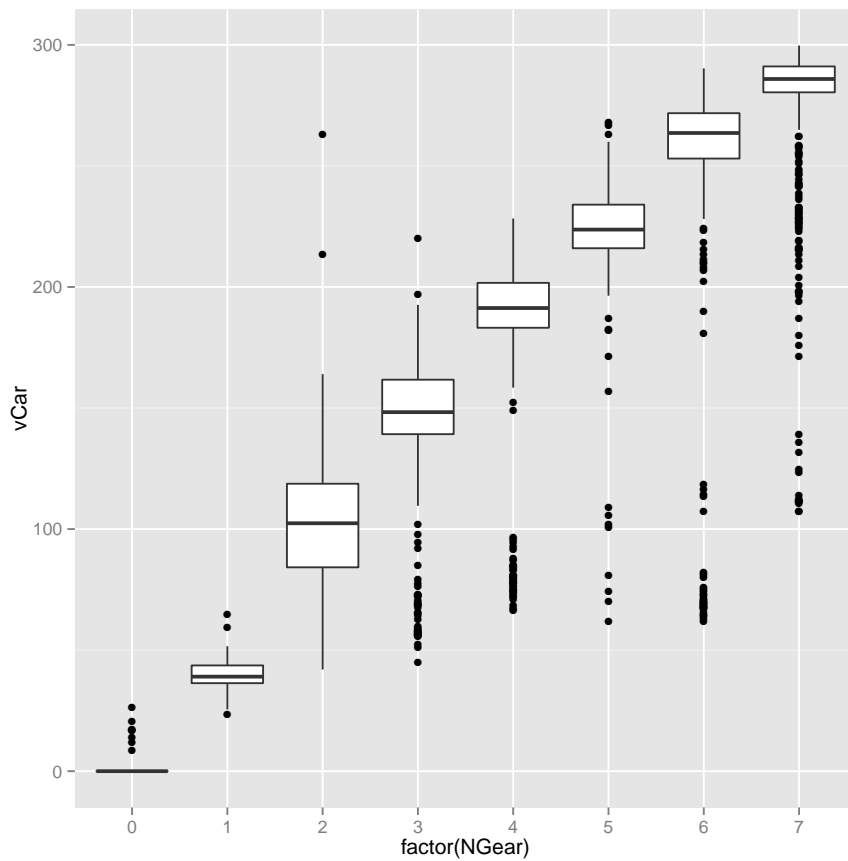


Based on [Tony Hirst's Blog Post](#), March 2012

## 37 F1: Box Plot of Speed by Gear

Statistical graphics provide important insights. The box plot here makes sense, in that higher gears correspond to higher speeds.

```
ggplot(f1, aes(factor(NGear), vCar)) +  
  geom_boxplot()
```

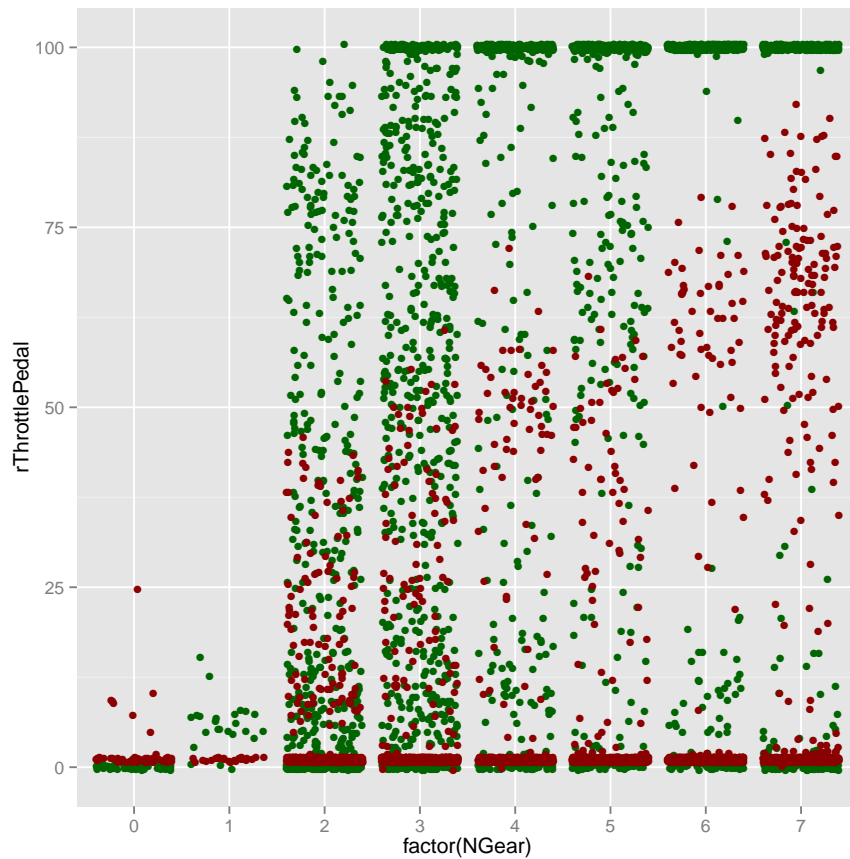


Based on [Tony Hirst's Blog Post](#), March 2012

## 38 F1: Footwork

How busy are the feet? We can summarise the brake (red) and throttle (green) depression based on gear.

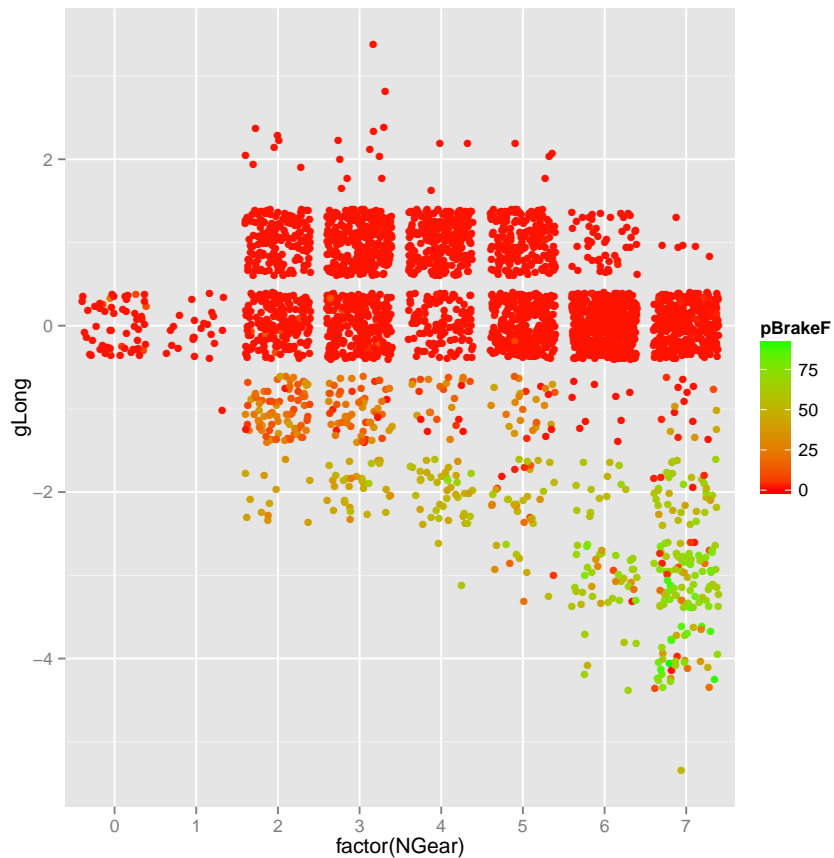
```
ggplot(f1, aes(factor(NGear))) +  
  geom_jitter(aes(y=rThrottlePedal), colour='darkgreen') +  
  geom_jitter(aes(y=pBrakeF), colour='darkred')
```



Based on [Tony Hirst's Blog Post](#), March 2012

## 39 F1: Forces on the Driver

```
ggplot(f1, aes(factor(NGear), gLong)) +  
  geom_jitter(aes(col=pBrakeF)) +  
  scale_colour_gradient(low='red', high='green')
```

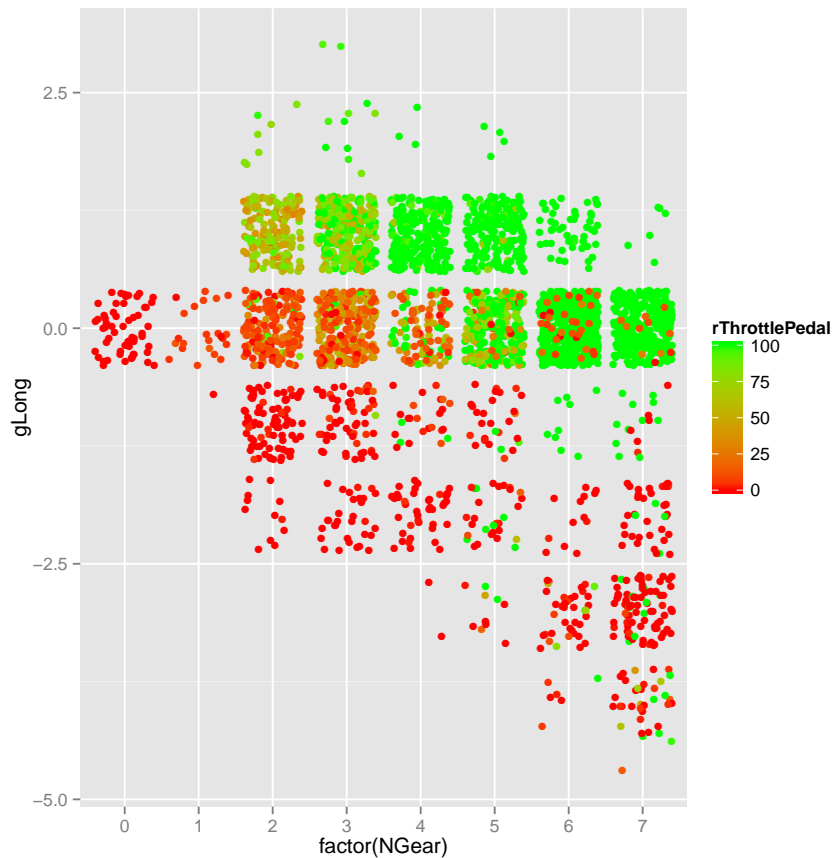


Based on [Tony Hirst's Blog Post](#), March 2012



## 40 F1: More Forces

```
ggplot(f1, aes(factor(NGear), gLong)) +  
  geom_jitter(aes(col=rThrottlePedal)) +  
  scale_colour_gradient(low='red', high="green")
```

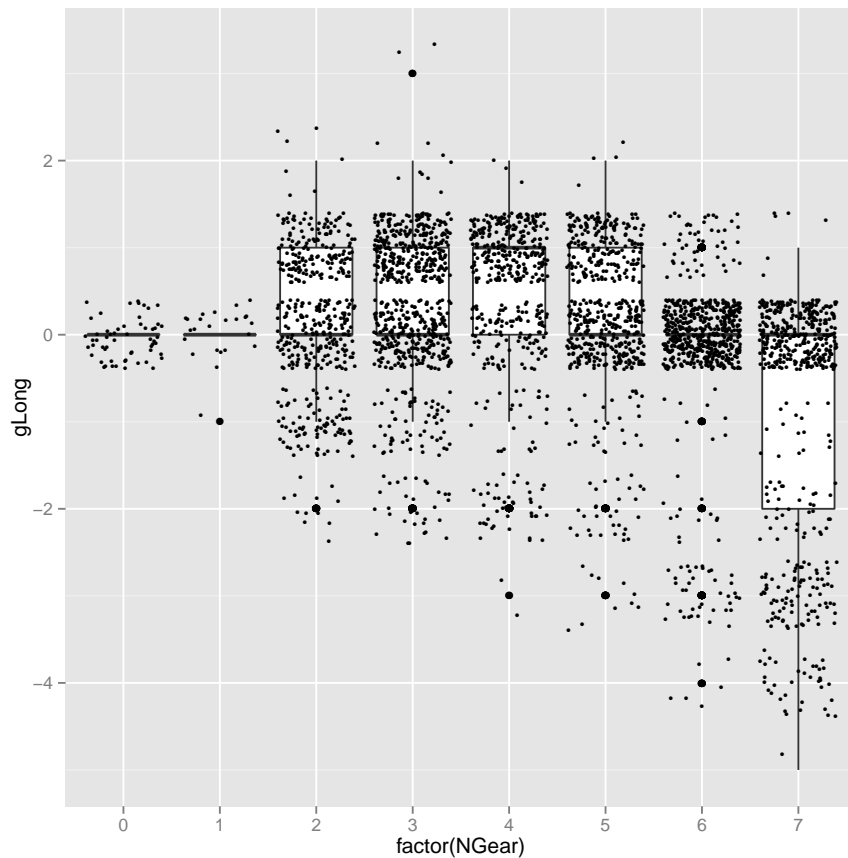


Based on [Tony Hirst's Blog Post](#), March 2012

## 41 F1: Box Plot of Forces

We can use a box plot to investigate the longitudinal g-force's relationship with acceleration or braking by gear. Note that a random jitter is used to scatter points around their actual integer values.

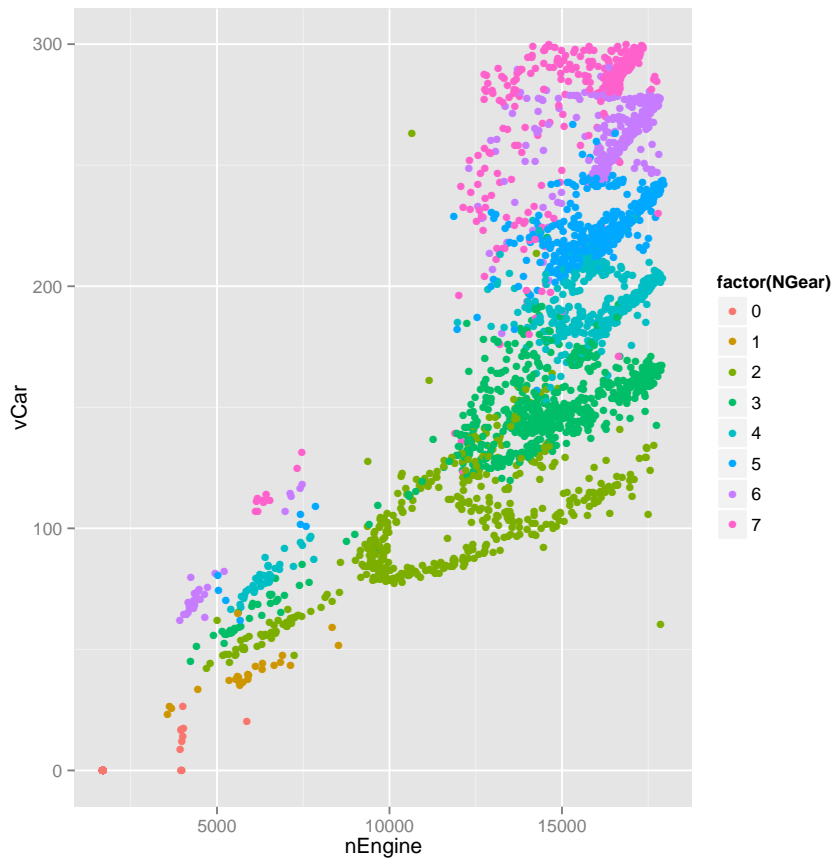
```
ggplot(f1, aes(factor(NGear), gLong)) +  
  geom_boxplot() +  
  geom_jitter(size=1)
```



Based on [Tony Hirst's Blog Post](#), March 2012

## 42 F1: RPM and Speed in Relation to Gear

```
ggplot(f1, aes(nEngine, vCar)) +  
  geom_point(aes(col=factor(NGear)))
```



Based on [Tony Hirst's Blog Post](#), March 2012

## 43 Plotting a Table

```
grid.table(iris[1:10,])
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

Exercise: Get the above to display on the page.

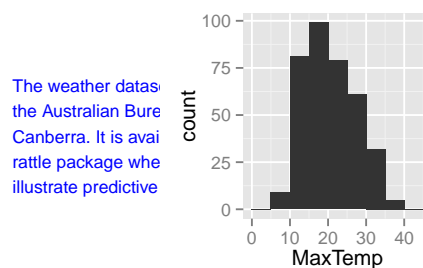
## 44 Plotting a Table and Text

Positioning of the textGrob (t2) remains a problem. Perhaps need to get the width of the t2 textGrob and pass it to arrangeGrob().

```
t1 <- tableGrob(head(weather[1:7], 10))
t2 <- textGrob("The weather dataset is collected from
the Australian Bureau of Meterology,
Canberra. It is available from the
rattle package where it is used to
illustrate predictive data mining.",
              just="left", gp=gpar(col="blue", fontsize=10))
t3 <- ggplot(weather, aes(MaxTemp))
t3 <- t3 + geom_histogram(binwidth=5)
grid.arrange(t1, t2, arrangeGrob(t2, t3, ncol=3), ncol=1)
```

1	2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3
2	2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7
3	2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3
4	2007-11-04	Canberra	13.3	15.5	39.8	7.2	9.1
5	2007-11-05	Canberra	7.6	16.1	2.8	5.6	10.6
6	2007-11-06	Canberra	6.2	16.9	0.0	5.8	8.2
7	2007-11-07	Canberra	6.1	18.2	0.2	4.2	8.4
8	2007-11-08	Canberra	8.3	17.0	0.0	5.6	4.6
9	2007-11-09	Canberra	8.8	19.5	0.0	4.0	4.1

The weather dataset is collected from  
the Australian Bureau of Meterology,  
Canberra. It is available from the  
rattle package where it is used to  
illustrate predictive data mining.



## 45 Interactive Plot Building

The Plot Builder provided as part of the Deducer package, which is a plugin for the JGR console, can be used to build ggplot2 plots interactively. It is a very powerful tool.

```
library(Deducer)
deducer(cmd="Plot builder")
```

After starting it up, select some element to draw, and you will be offered a choice of currently available data frames. You can interactively explore various options, enabling and disabling them as you develop the plot you desire. Once completed click the Run button and a R plot will be displayed, together with the ggplot commands printed on the console, and the Plot Builder exits.

## 46 Having Some Fun: xkcd

For a bit of fun we can generate plots in the style of the popular online comic strip *xkcd* using *xkcd* (Manzanera, 2013). The examples here come from the package vignette which should be referred to for more details.

On my Ubuntu system I had to install the required fonts. The following steps can do that.

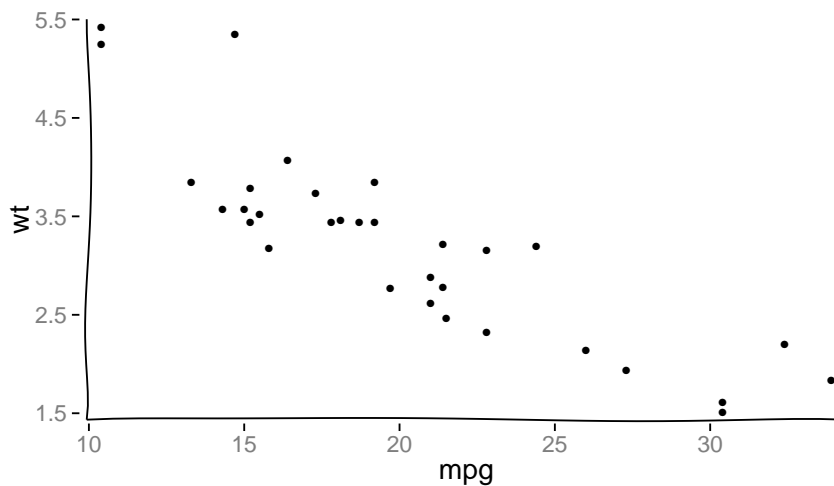
```
library(extrafont)
download.file("http://simonsoftware.se/other/xkcd.ttf", dest="xkcd.ttf")
system("mkdir ~/.fonts")
system("mv xkcd.ttf ~/.fonts")
font_import()
loadfonts()
```

The installation can be uninstalled with:

```
remove.packages(c("extrafonts", "extrafontdb"))
```

We can then generate a roughly yet neatly drawn plot, as if it might have been drawn by the hand of the author of the comic strip.

```
library(xkcd)
xrange <- range(mtcars$mpg)
yrange <- range(mtcars$wt)
p <- ggplot(mtcars, aes(mpg, wt))
p <- p + geom_point()
p <- p + xkcdaxis(xrange, yrange)
p
```



## 47 Having Some More Fun: xkcd Bar Chart

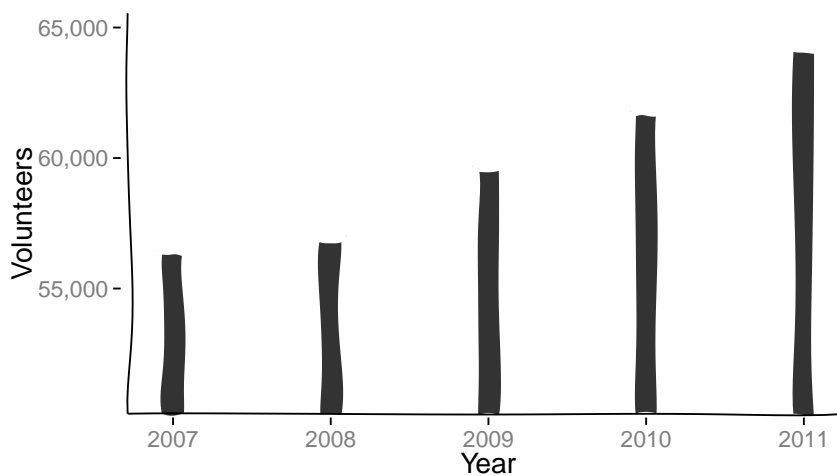
Another example, this time a bar chart.

```
library(xkcd)
library(scales)
volunteers <- data.frame(year=c(2007:2011),
                          number=c(56470, 56998, 59686, 61783, 64251))

ds <- volunteers
ds$xmin <- ds$year - 0.1
ds$xmax <- ds$year + 0.1
ds$ymin <- 50000
ds$ymax <- ds$number
xrange <- range(min(ds$xmin) - 0.1, max(ds$xmax) + 0.1)
yrange <- range(min(ds$ymin) + 500, max(ds$ymax) + 1000)
mapping <- aes(xmin=xmin, ymin=ymin, xmax=xmax, ymax=ymax)
p <- ggplot()
p <- p + xkcdrect(mapping, ds)
p <- p + xkcdaxis(xrange, yrange)

## Warning: Not xkcd fonts installed!

p <- p + xlab("Year")
p <- p + ylab("Volunteers")
p <- p + scale_y_continuous(labels=comma)
p
```





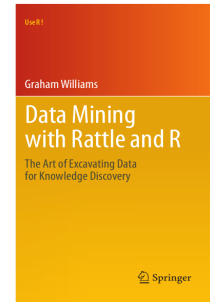
## 48 Further Reading

The [Rattle Book](#), published by Springer, provides a comprehensive introduction data mining and analytics using Rattle and R. It is available from [Amazon](#). Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from <http://datamining.togaware.com>, including the [Datamining Desktop Survival Guide](#).

This module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular, following the links on the website with a \* which indicate the generally more developed OnePageR modules.

Other resources include:

- The [R Cookbook](#) is a great resource explaining how to do many types of plots using ggplot2.



## 49 References

- Auguie B (2012). *gridExtra: functions in Grid graphics*. R package version 0.9.1, URL <http://CRAN.R-project.org/package=gridExtra>.
- Jassby AD, Cloern JE (2013). *wq: Exploring water quality monitoring data*. R package version 0.3-11, URL <http://CRAN.R-project.org/package=wq>.
- Manzanera ET (2013). *xkcd: Plotting ggplot2 graphics in a XKCD style*. R package version 0.0.1, URL <http://CRAN.R-project.org/package=xkcd>.
- R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <http://www.R-project.org/>.
- Wickham H (2012). *scales: Scale functions for graphics*. R package version 0.2.3, URL <http://CRAN.R-project.org/package=scales>.
- Wickham H, Chang W (2013). *ggplot2: An implementation of the Grammar of Graphics*. R package version 0.9.3.1, URL <http://CRAN.R-project.org/package=ggplot2>.
- Williams GJ (2009). “Rattle: A Data Mining GUI for R.” *The R Journal*, **1**(2), 45–55. URL [http://journal.r-project.org/archive/2009-2/RJournal\\_2009-2\\_Williams.pdf](http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf).
- Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer, New York. URL [http://www.amazon.com/gp/product/1441998896/ref=as\\_li\\_qf\\_sp\\_asin\\_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896](http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896).
- Williams GJ (2013). *rattle: Graphical user interface for data mining in R*. R package version 2.6.27, URL <http://rattle.togaware.com/>.

*This document, sourced from GGPlot2O.Rnw revision 219, was processed by KnitR version 1.4.9 of 2013-08-10 and took 51.6 seconds to process. It was generated by gjw on nyx running Ubuntu 13.04 with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2013-09-22 21:44:43.*

