Data Science with R
Exploring Data with GGPlot2

Graham.Williams@togaware.com

14th February 2014

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.

The required packages for this module include:

```
library(ggplot2)      # Grammar of graphics.
library(scales)       # Include commas in numbers.
library(rattle)       # Weather dataset.
library(randomForest) # Use na.roughfix() to deal with missing data.
library(gridExtra)    # Layout multiple plots.
library(wq)           # Regular grid layout.
library(xkcd)         # Some xkcd fun.
library(extrafont)    # Fonts for xkcd.
library(GGally)       # Parallel coordinates.
```

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 present 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 © 2013-2014 Graham J Williams. You can 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: Reviewing the Data

We use the relatively large **weatherAUS** dataset from rattle (Williams, 2014) to illustrate the capabilities of ggplot2.

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

The dataset is summarised below.

```
dim(ds)

## [1] 82169    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
## 82164 2014-01-25   Darwin    23.5    31.6     34.0         2.0      8.8
## 82165 2014-01-26   Darwin    27.2    31.2      0.0         6.4      4.5
## 82166 2014-01-27   Darwin    24.2    31.4      3.2         5.4      3.6
....

str(ds)

## 'data.frame': 82169 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 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: 2163   Min.   :-8.5   Min.   :-3.7
##  1st Qu.:2010-01-31   Sydney  : 2071   1st Qu.: 7.5   1st Qu.:17.8
##  Median :2011-05-23   Adelaide: 1920   Median :11.9   Median :22.3
....
```

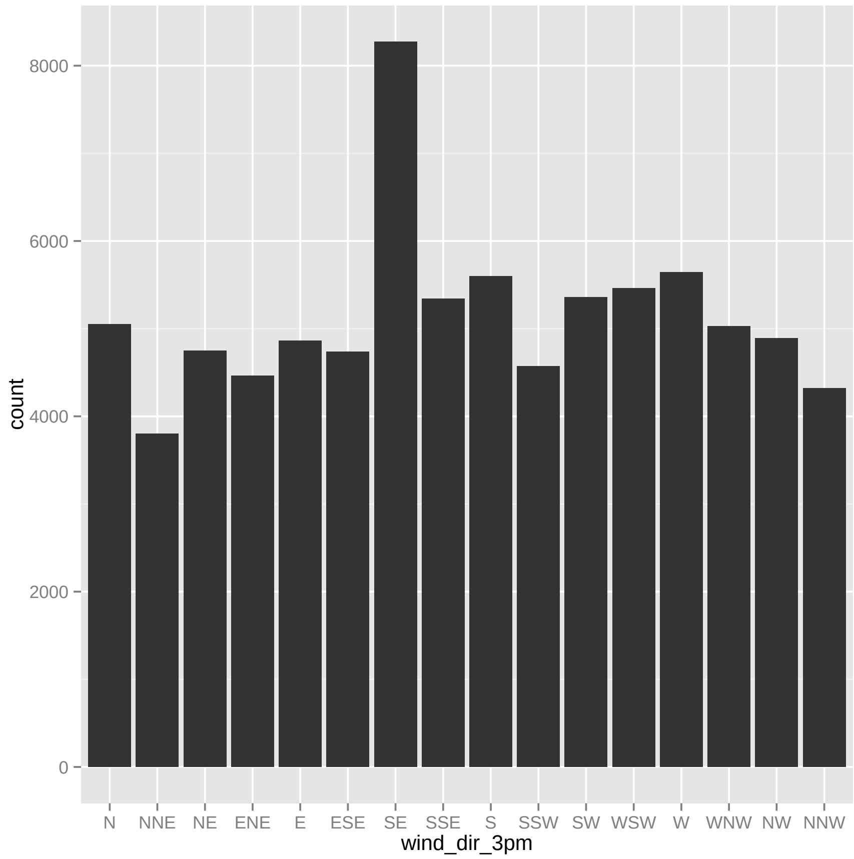
2 Preparing the Dataset: Collecting Information

```
names(ds)  <- normVarNames(names(ds)) # Optional lower case variable names.
vars       <- names(ds)
target     <- "rain_tomorrow"
id         <- c("date", "location")
ignore     <- id
inputs     <- setdiff(vars, target)
numi       <- which(sapply(ds[vars], is.numeric))
numi

##        min_temp        max_temp        rainfall     evaporation
##               3               4               5               6
##        sunshine wind_gust_speed   wind_speed_9am  wind_speed_3pm
##               7               9              12              13
....

numerics   <- names(numi)
numerics

##  [1] "min_temp"        "max_temp"        "rainfall"
##  [4] "evaporation"     "sunshine"        "wind_gust_speed"
##  [7] "wind_speed_9am"  "wind_speed_3pm"  "humidity_9am"
## [10] "humidity_3pm"    "pressure_9am"    "pressure_3pm"
....

cati       <- which(sapply(ds[vars], is.factor))
cati

##      location wind_gust_dir  wind_dir_9am  wind_dir_3pm    rain_today
##             2             8            10            11            22
## rain_tomorrow
##            24
....

categorics <- names(cati)
categorics

## [1] "location"      "wind_gust_dir" "wind_dir_9am"  "wind_dir_3pm"
## [5] "rain_today"    "rain_tomorrow"
```

We perform missing value imputation simply to avoid warnings from ggplot2, ignoring whether this is appropriate to do so from a data integrity point of view.

```
library(randomForest)
sum(is.na(ds))

## [1] 172050

ds[setdiff(vars, ignore)] <- na.roughfix(ds[setdiff(vars, ignore)])
sum(is.na(ds))

## [1] 0
```

3 Histogram: Displaying Frequencies




```
p <- ggplot(data=ds, aes(x=wind_dir_3pm))
p <- p + geom_bar()
p
```

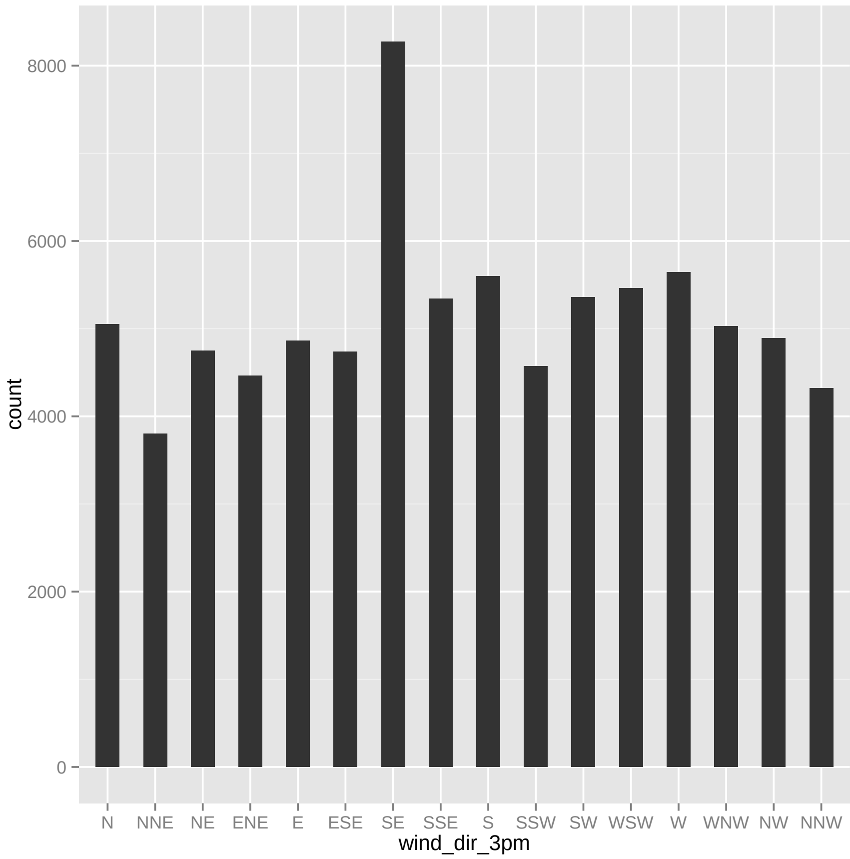
A histogram displays the frequency of observations using bars. Such plots are easy to display using ggplot2, as we see in the above code used to generate the plot. Here the `data=` is identified as **ds** and the `x=` aesthetic is `wind_dir_3pm`. Using these parameters we then add a bar geometric to build a bar plot for us.

The resulting plot shows the frequency of the levels of the categoric variable `wind_dir_3pm` 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.

4 Histogram: Narrow Bars

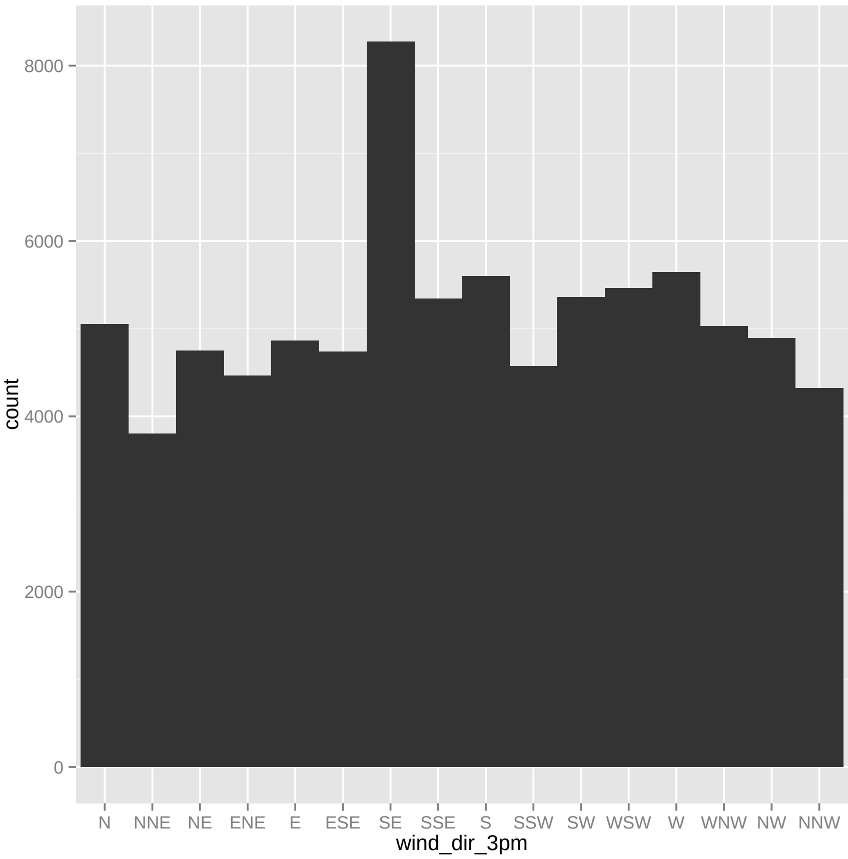



```
p <- ggplot(ds, aes(wind_dir_3pm))
p <- p + geom_bar(width=0.5)
p
```

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!

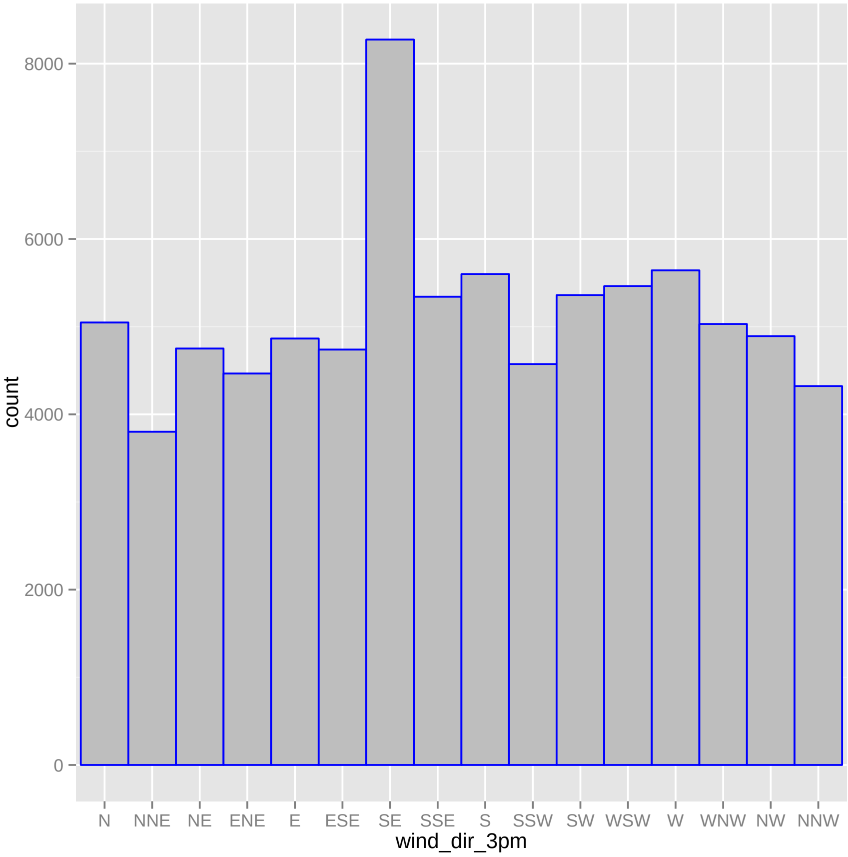
5 Histogram: Full Width Bars




```
p <- ggplot(ds, aes(wind_dir_3pm))
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`.

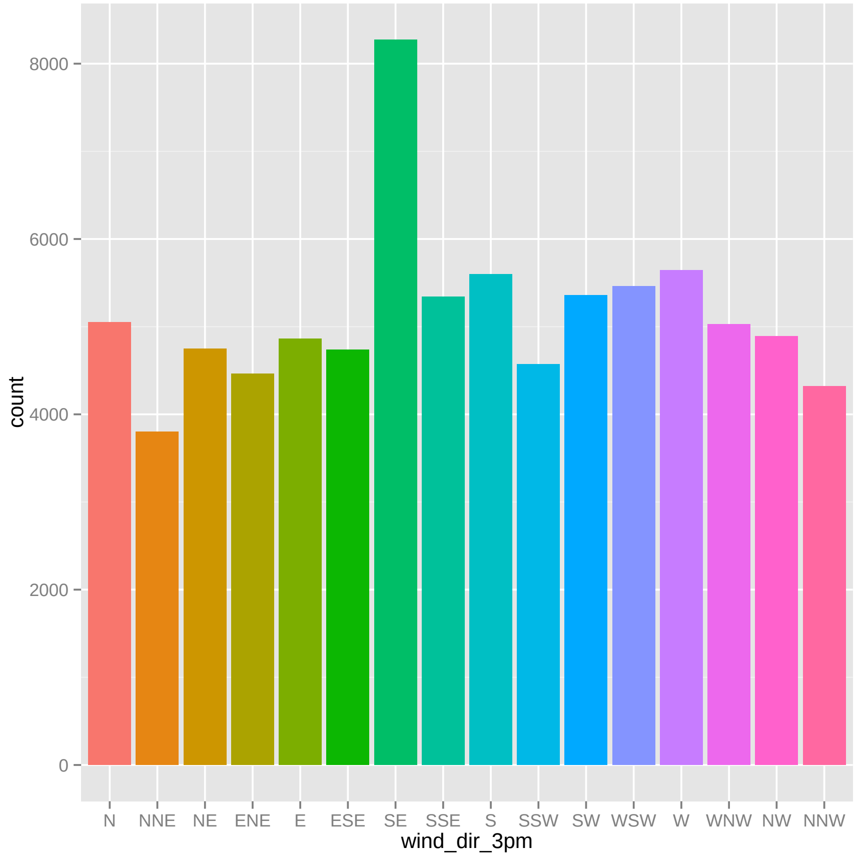
6 Histogram: Full Width Bars with Borders




```
p <- ggplot(ds, aes(wind_dir_3pm))
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.

7 Histogram: Coloured Histogram Without a Legend

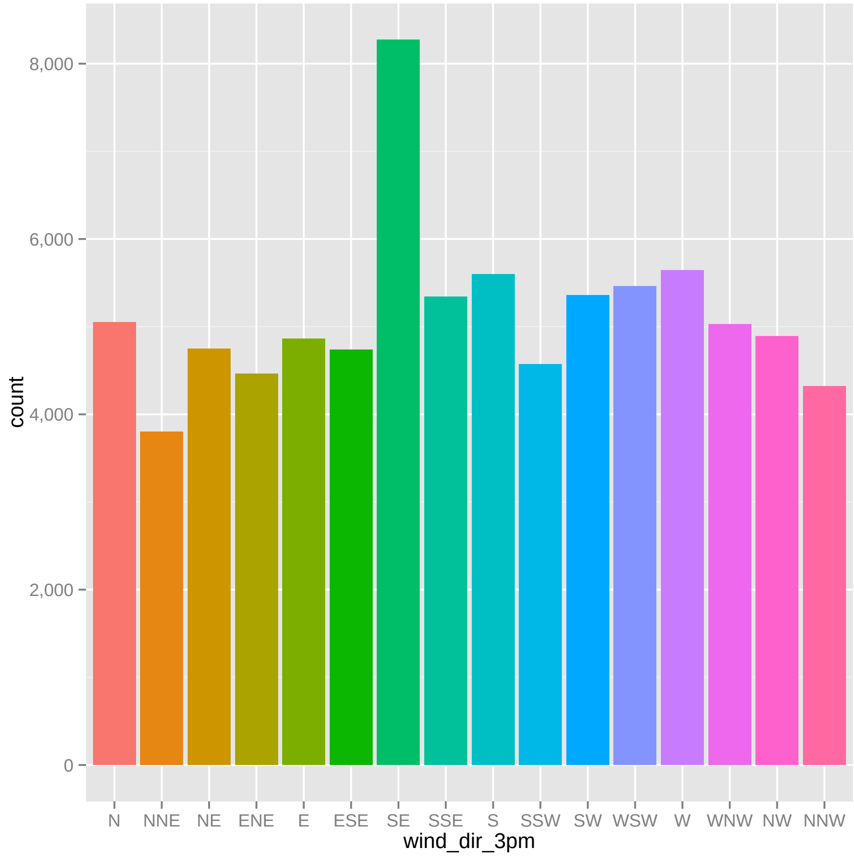



```
p <- ggplot(ds, aes(wind_dir_3pm, fill=wind_dir_3pm))
p <- p + geom_bar()
p <- p + theme(legend.position="none")
p
```

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 `wind_dir_3pm` 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`.

8 Histogram: Comma Formatted Labels

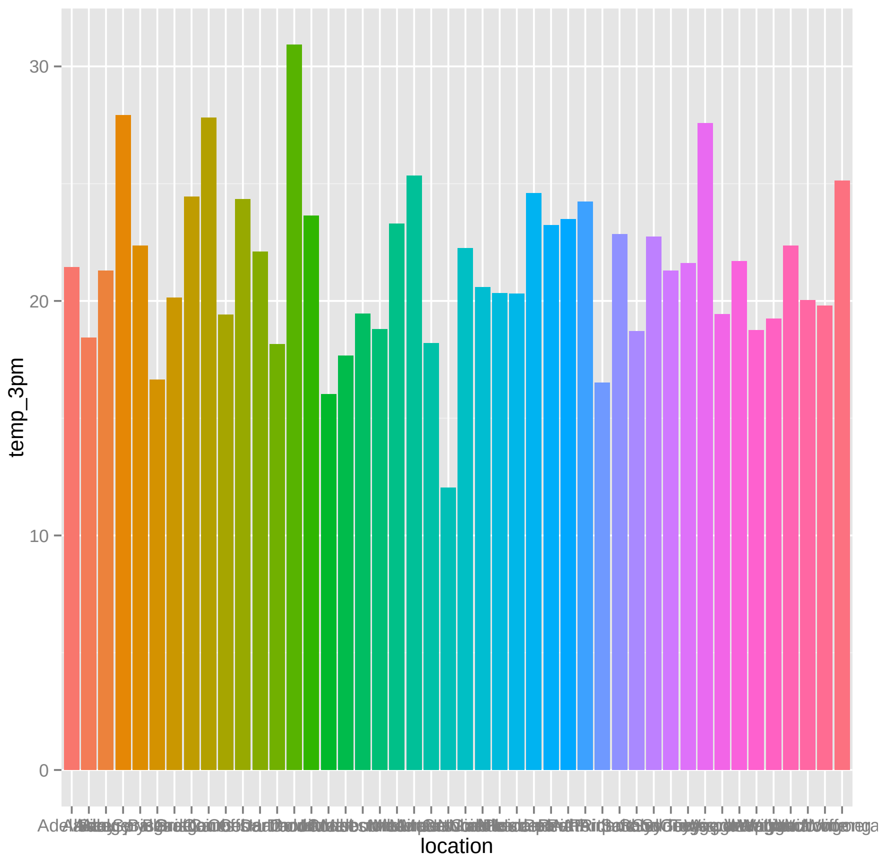



```
p <- ggplot(ds, aes(wind_dir_3pm, fill=wind_dir_3pm))
p <- p + geom_bar()
p <- p + scale_y_continuous(labels=comma)
p <- p + theme(legend.position="none")
p
```

Since ggplot2 Version 0.9.0 the scales (Wickham, 2012b) 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`.

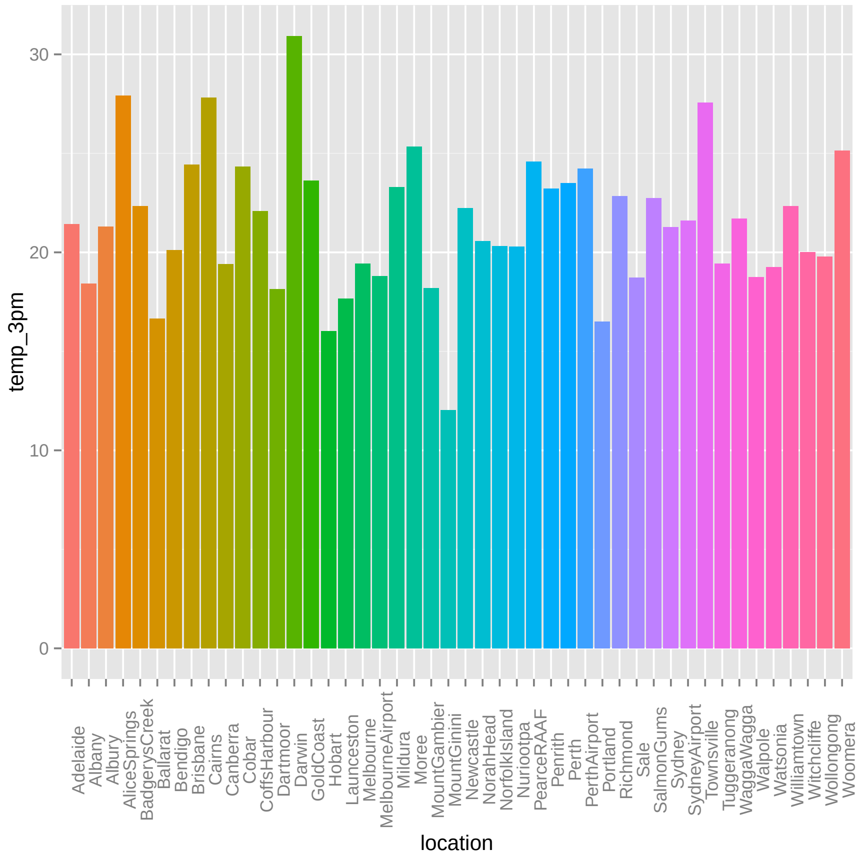
9 Histogram: Too Many Bars




```
p <- ggplot(data=ds, aes(x=location, y=temp_3pm, 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.

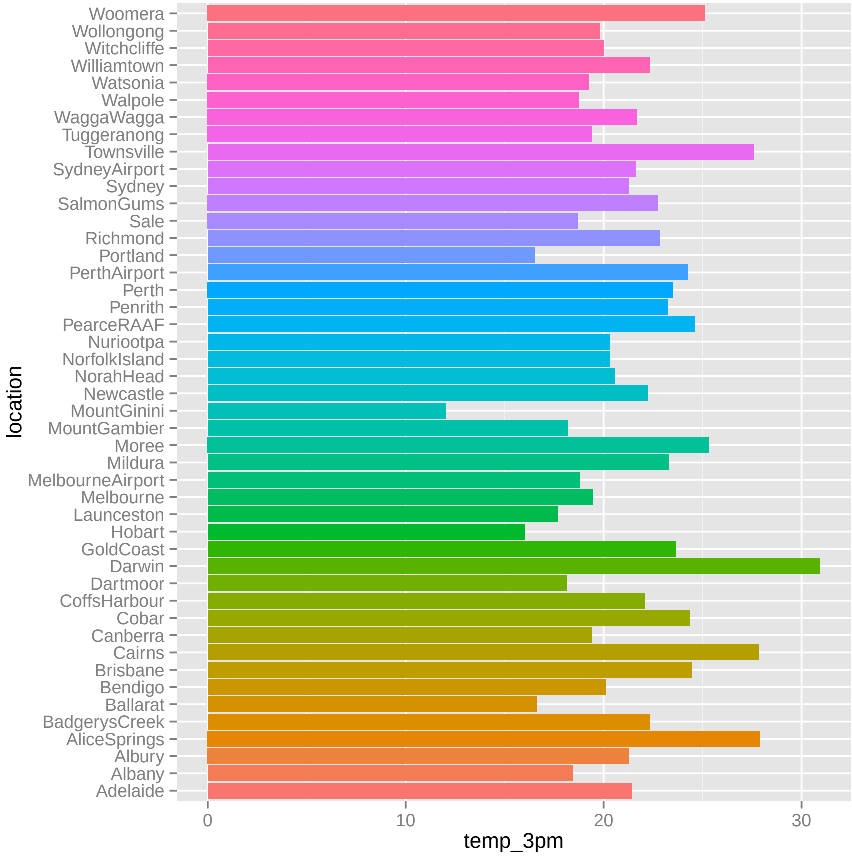
10 Histogram: Rotated Labels




```
p <- ggplot(ds, aes(location, temp_3pm, 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=` to be rotated 90 degrees.

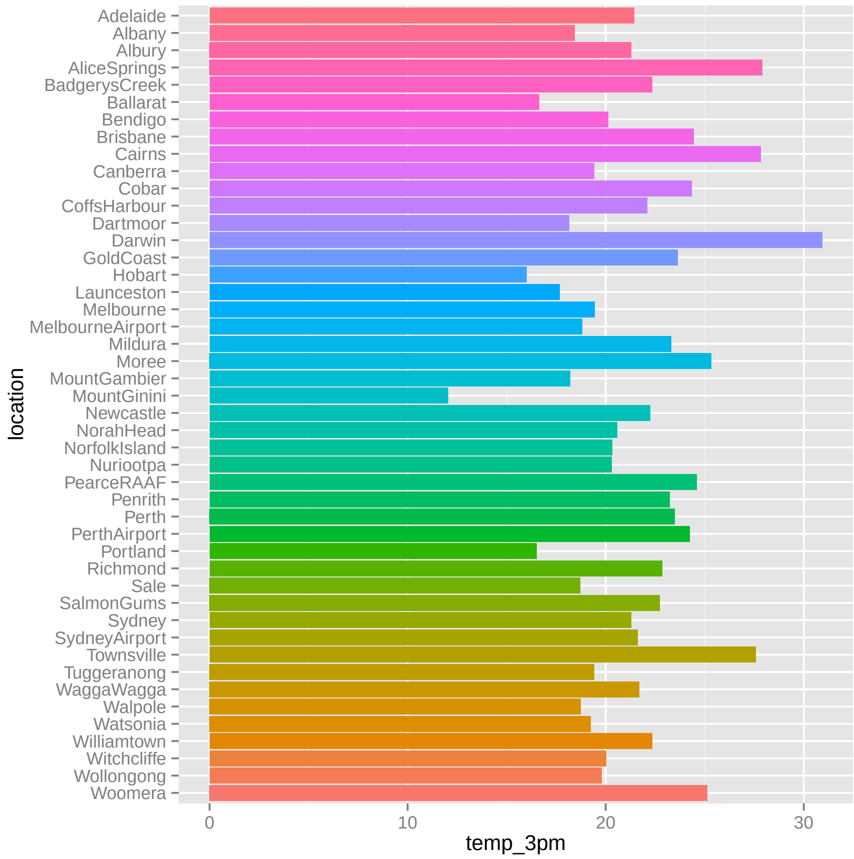
11 Histogram: Horizontal Histogram




```
p <- ggplot(ds, aes(location, temp_3pm, 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:

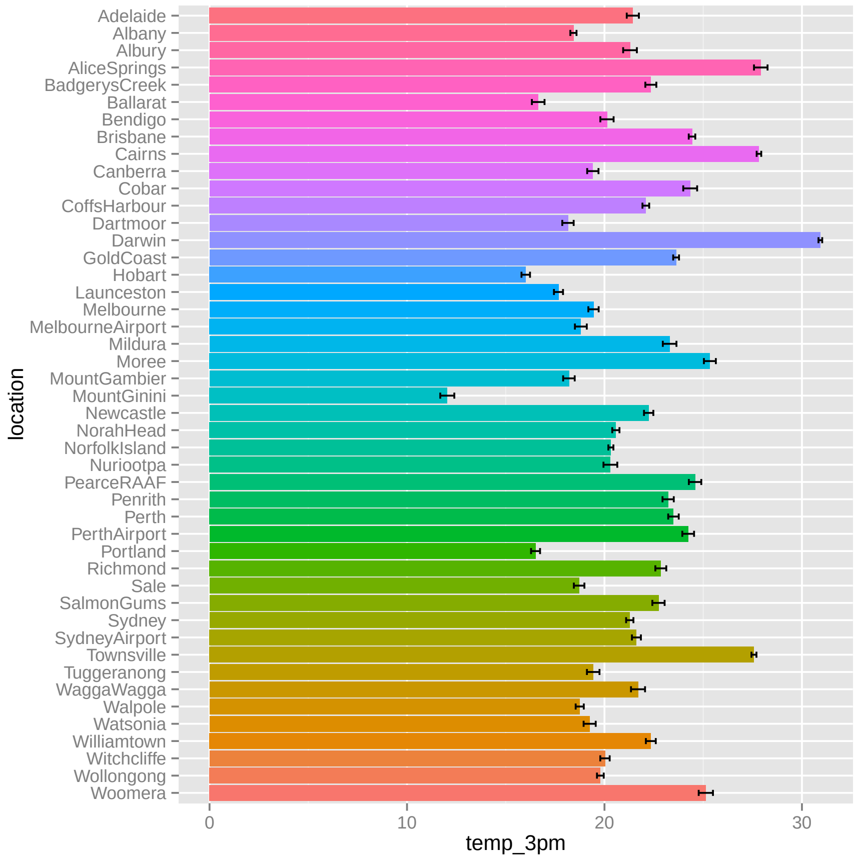
12 Histogram: Reorder the Levels




```
dslr <- within(ds, location <- factor(location, levels=rev(levels(location))))
p <- ggplot(dslr, aes(location, temp_3pm, 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. We do this and save the result into another dataset so as to revert to the original dataset when appropriate below.

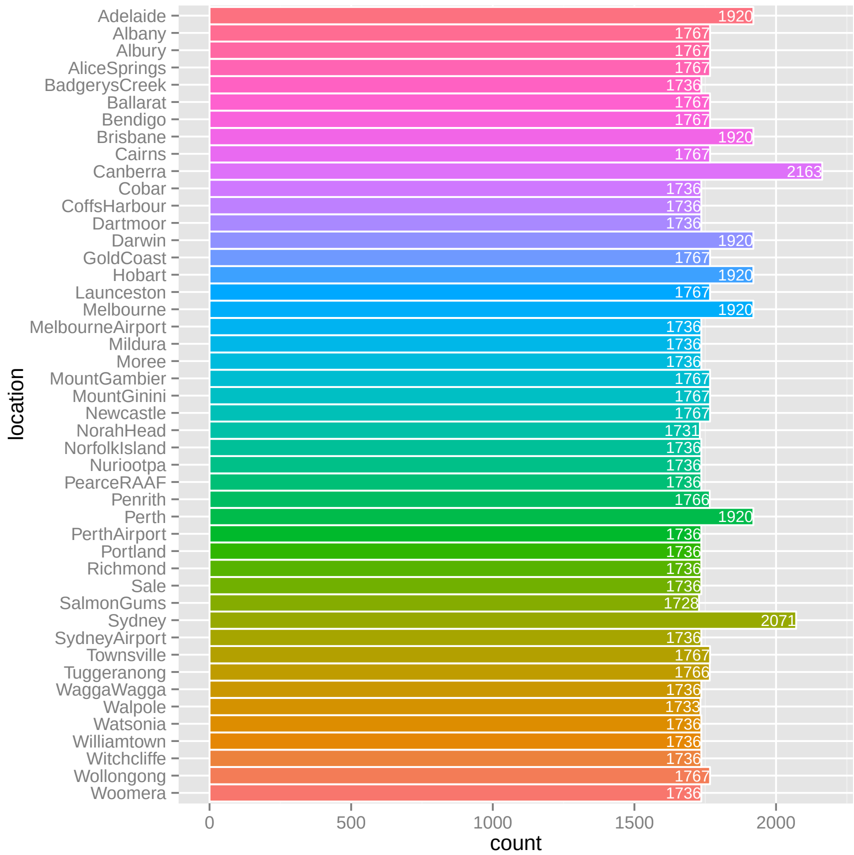
13 Histogram: Plot the Mean with CI




```
p <- ggplot(dslr, aes(location, temp_3pm, 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.

14 Histogram: Text Annotations

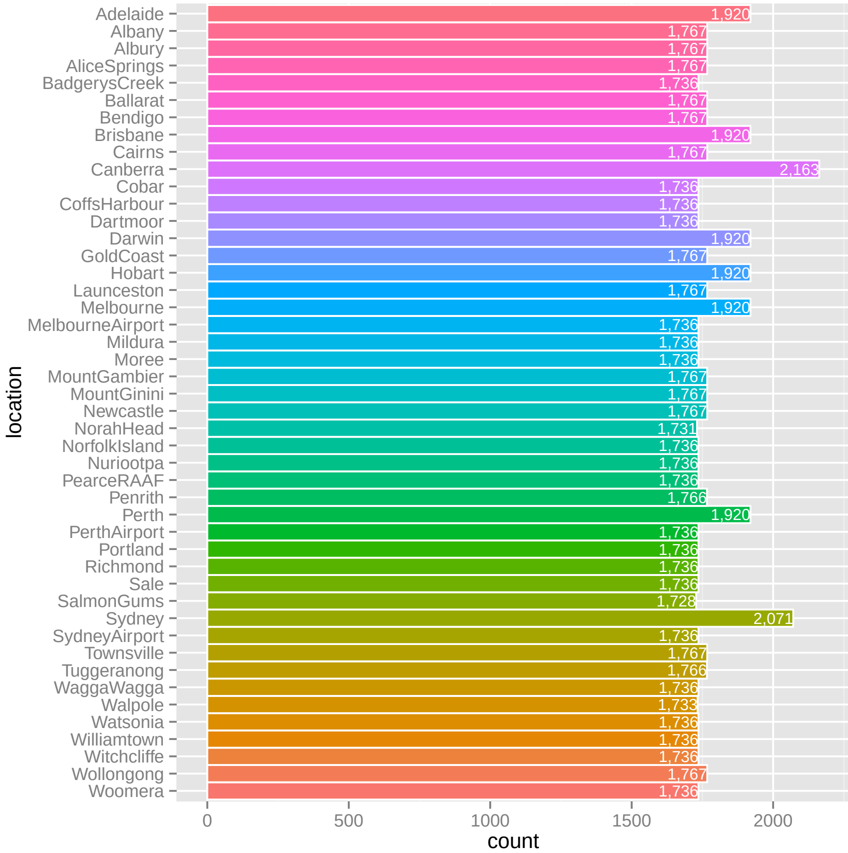



```
p <- ggplot(dslr, aes(location, fill=location))
p <- p + geom_bar(width=1, colour="white")
p <- p + theme(legend.position="none")
p <- p + coord_flip()
p <- p + geom_text(stat="bin",
                   color="white",
                   hjust=1.0,
                   size=3,
                   aes(y=..count.., 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 `temp_3pm`, and include the textual value.

15 Histogram: Text Annotations with Commas

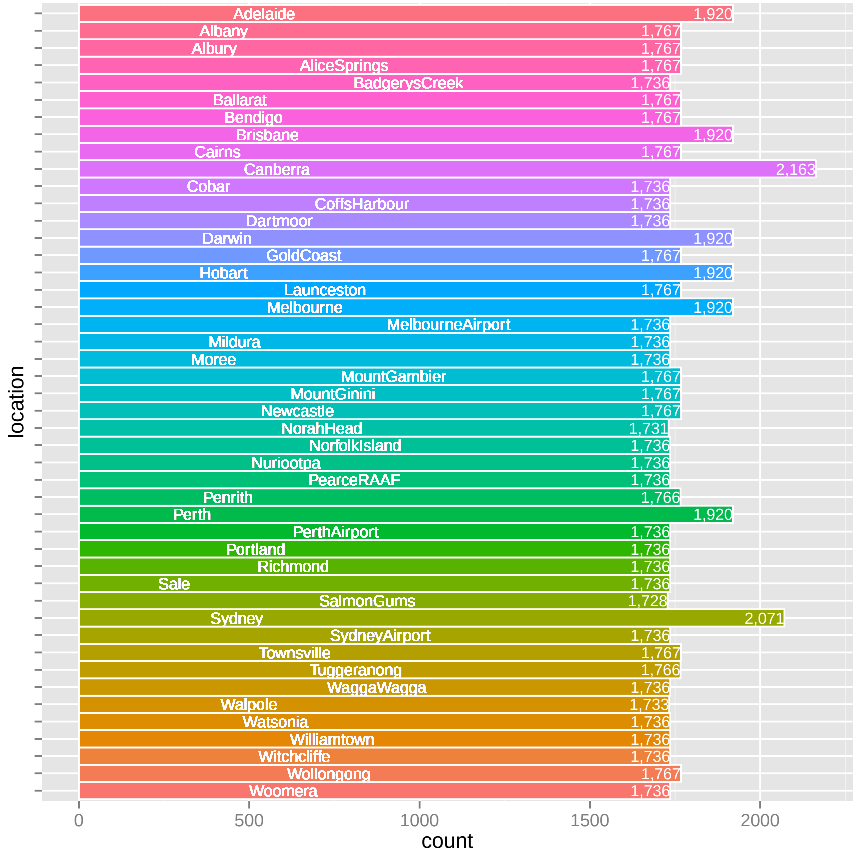



```
p <- ggplot(dslr, aes(location, fill=location))
p <- p + geom_bar(width=1, colour="white")
p <- p + theme(legend.position="none")
p <- p + coord_flip()
p <- p + geom_text(stat="bin", color="white", hjust=1.0, size=3,
                   aes(y=..count.., label=scales::comma(..count..)))
p
```

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.

16 Histogram: Multiple Text Annotations




```
p <- ggplot(dslr, 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.

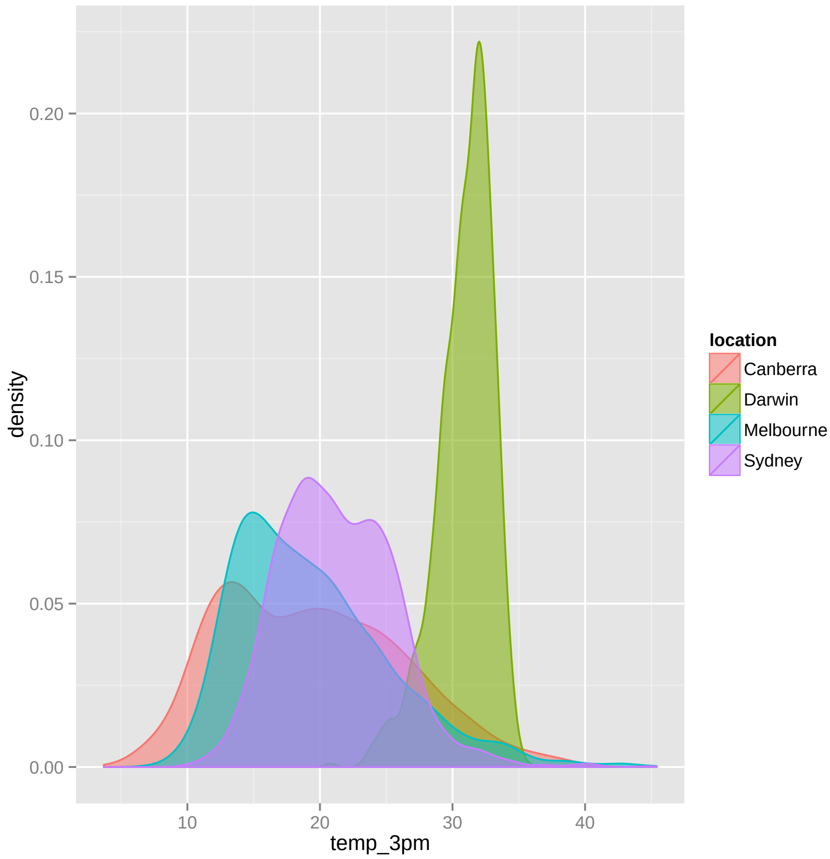
17 Histogram: Stacked Bar Chart

Exercise: Generate a stacked bar chart histogram.

18 Distributions

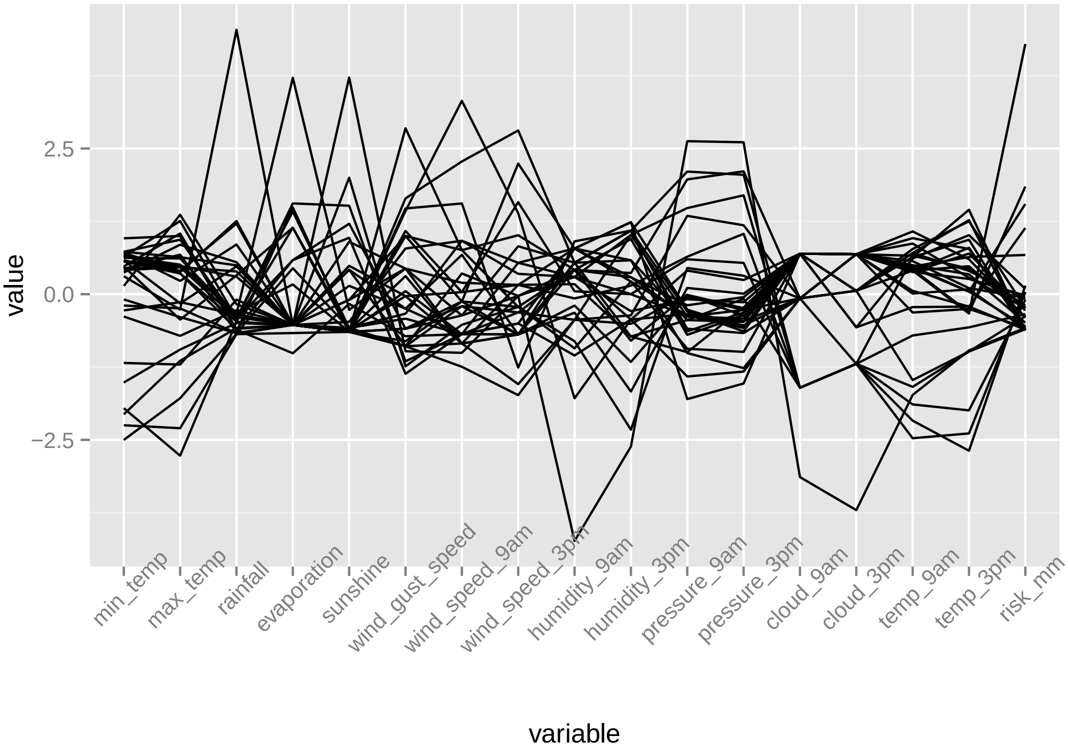
Exercise: Generate the basic plot for a distribution.

19 Distributions: Transparent Categoric Density Plot




```
cities <- c("Canberra", "Darwin", "Melbourne", "Sydney")
p <- ggplot(subset(ds, location %in% cities),
            aes(temp_3pm, colour = location, fill = location))
p <- p + geom_density(alpha = 0.55)
p
```

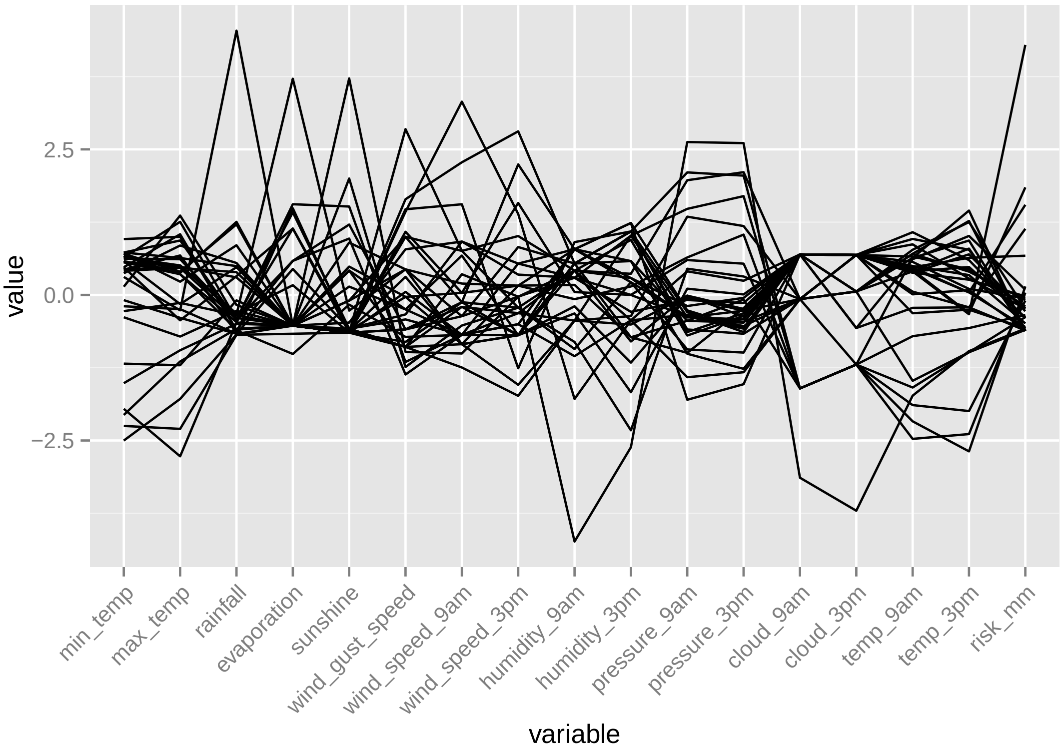
20 Parallel Coordinates Plot




```
library(GGally)
p <- ggparcoord(subset(ds, location %in% cities & rainfall>75), columns=numi)
p <- p + theme(axis.text.x=element_text(angle=45))
p
```

Exercise: Experiment with parallel coordinates to explore for any structure in the data.

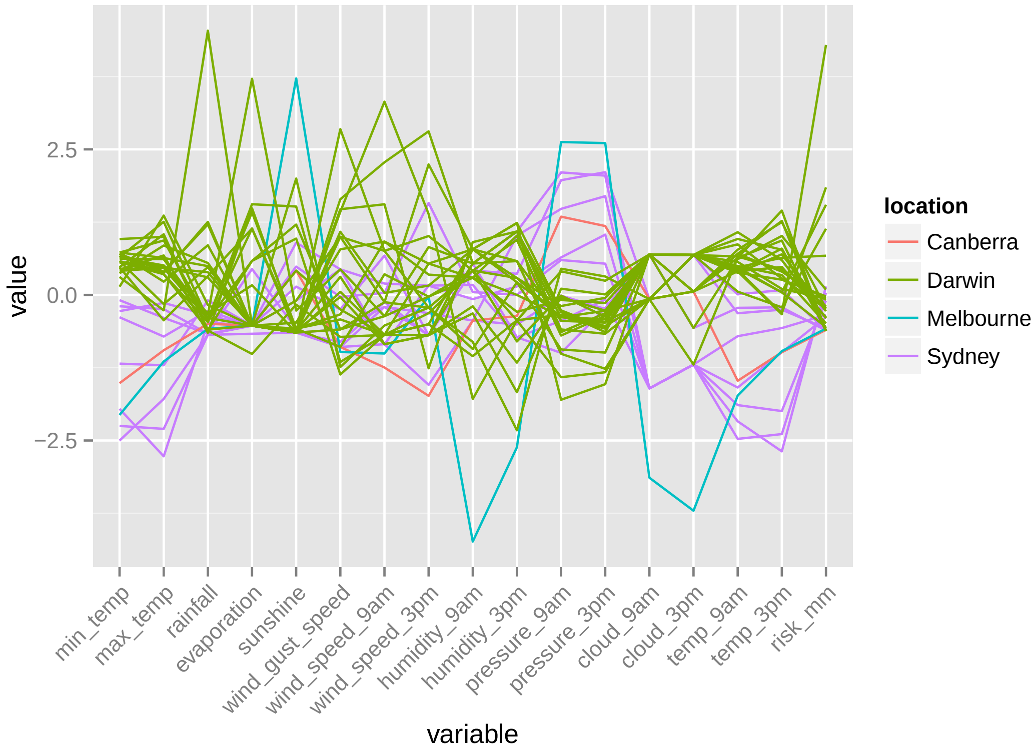
21 Parallel Coordinates Plot: Labels Aligned




```
p <- ggparcoord(subset(ds, location %in% cities & rainfall>75), columns=numi)
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p
```

We notice the labels are by default aligned by their centres. Rotating 45° causes the labels to sit over the plot region. We can ask the labels to be aligned at the top edge instead, using `hjust=1`.

22 Parallel Coordinates Plot: Colour by Location




```
p <- ggparcoord(subset(ds, location %in% cities & rainfall>75),
                columns=numi, group="location")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p
```

Here we add some colour. We can discern some structure relating to locations. We have limited the data to those days where more than 74mm of rain is recorded and clearly Darwin become prominent. Darwin has many days with at least this much rain, Sydney has a few days and Canberra and Melbourne only one day. We would confirm this with actual queries of the data. Appart from this the parallel coordinates in this instance is not showing much structure.

23 Distributions: Box Plot

Exercise: Generate a box plot.

24 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.

Copyright © 2013-2014 Graham.Williams@togaware.com

25 Scatter Plot

A scatter plot displays points scattered over a plot. Locations are specified as x and y for a two dimensional plot.

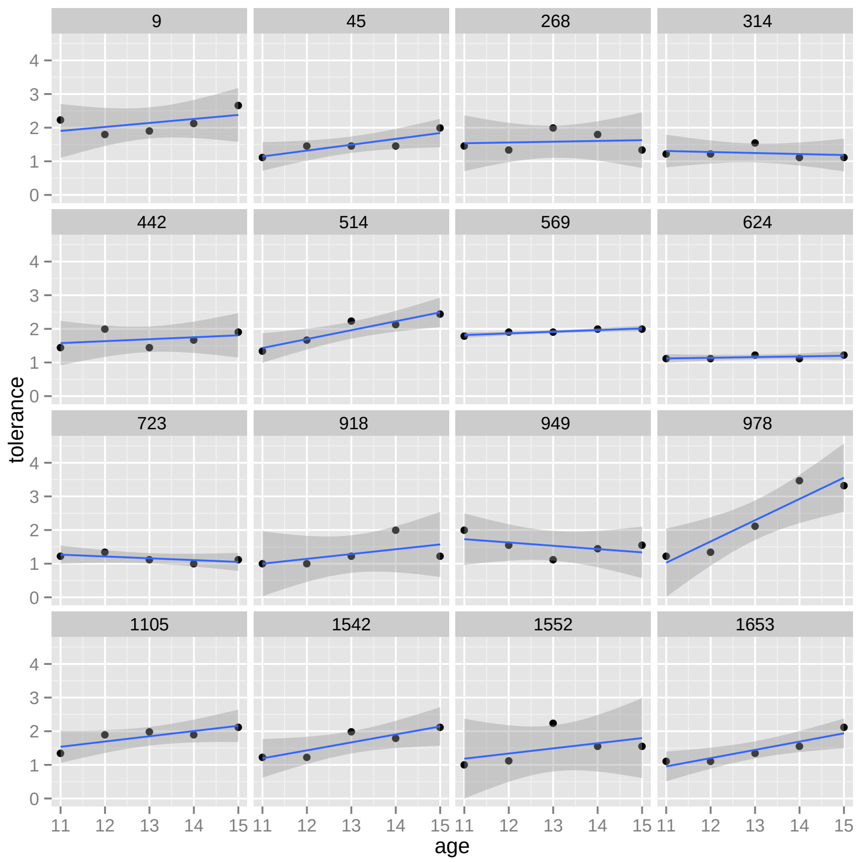
Exercise: Generate a scatter plot.

26 Scatter Plot: Comparing Individual Changes Over Time

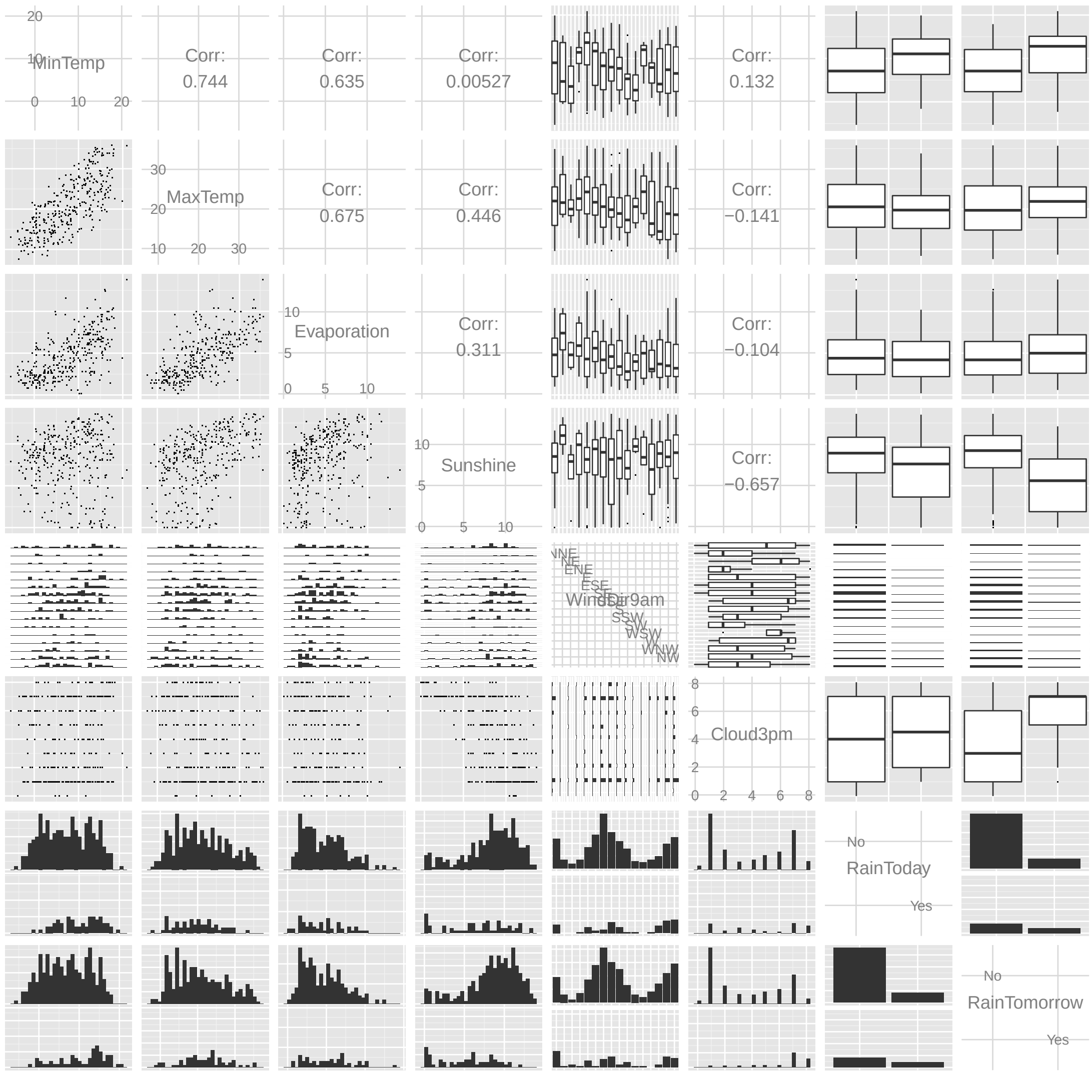
```
load(file.path("data", "tolerance.RData"))
dim(tolerance)

## [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
```



27 Scatter Plot: Using ggpairs()




```
wds <- na.omit(weather[c(3,4,6,7,10,19,22,24)])
ggpairs(wds,  params = c(shape = I("."), outlier.shape = I(".")))
```

This scatter plot uses `ggpairs()` from GGally (Schloerke *et al.*, 2014) to plot the **diamond** dataset from ggplot2 (Wickham and Chang, 2013).

28 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 (Wickham, 2012a) package to aggregate the cumulative rainfall. The scales (Wickham, 2012b) package is used to provide labels with commas.

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

```
cities <- c("Adelaide", "Canberra", "Darwin", "Hobart")

dss <- subset(ds, location %in% cities & date >= "2009-01-01")
dss <- ddply(dss, .(location), transform, cumRainfall=cumsum(rainfall))
```

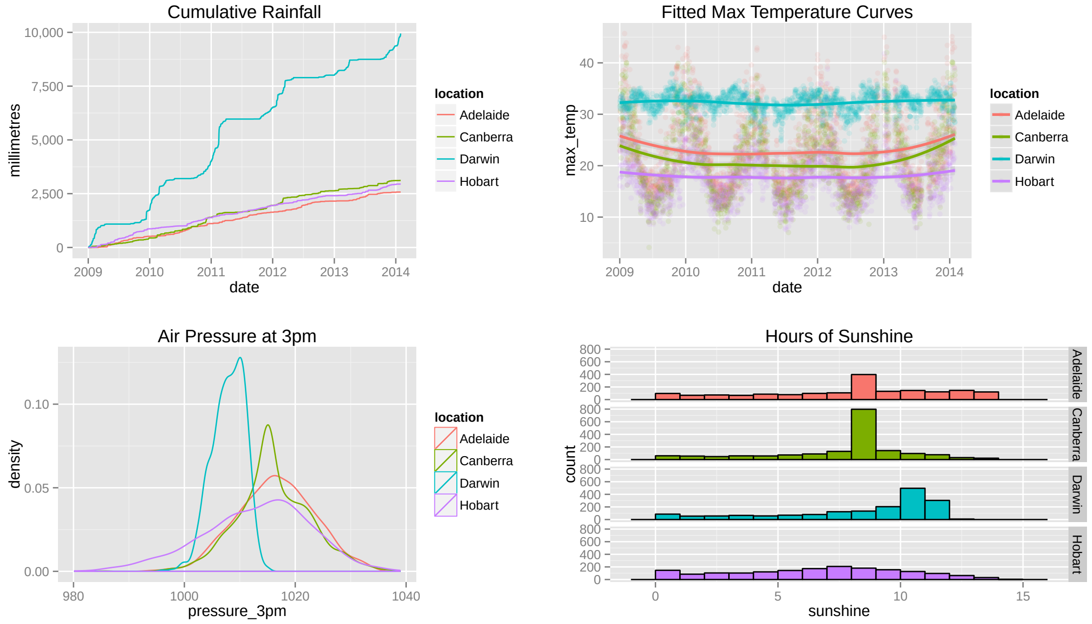
```
p  <- ggplot(dss, aes(x=date, y=cumRainfall, colour=location))
p1 <- p + geom_line()
p1 <- p1 + ylab("millimetres")
p1 <- p1 + scale_y_continuous(labels=comma)
p1 <- p1 + ggtitle("Cumulative Rainfall")
```

```
p2 <- ggplot(dss, aes(x=date, y=max_temp, 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(dss, aes(x=pressure_3pm, colour=location))
p3 <- p3 + geom_density()
p3 <- p3 + ggtitle("Air Pressure at 3pm")
```

```
p4 <- ggplot(dss, 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")
```

29 Using grid.arrange(): Multiple Plots




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

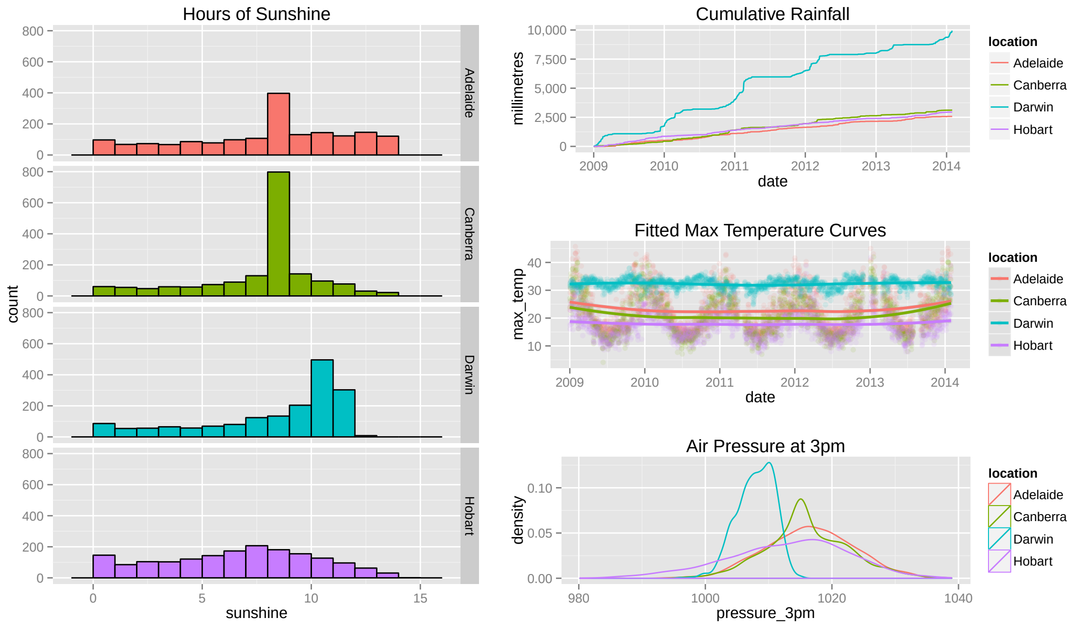
The actual plots are arranged by `grid.arrange()`.

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.

30 Using grid.arrange(): Arranging Plots

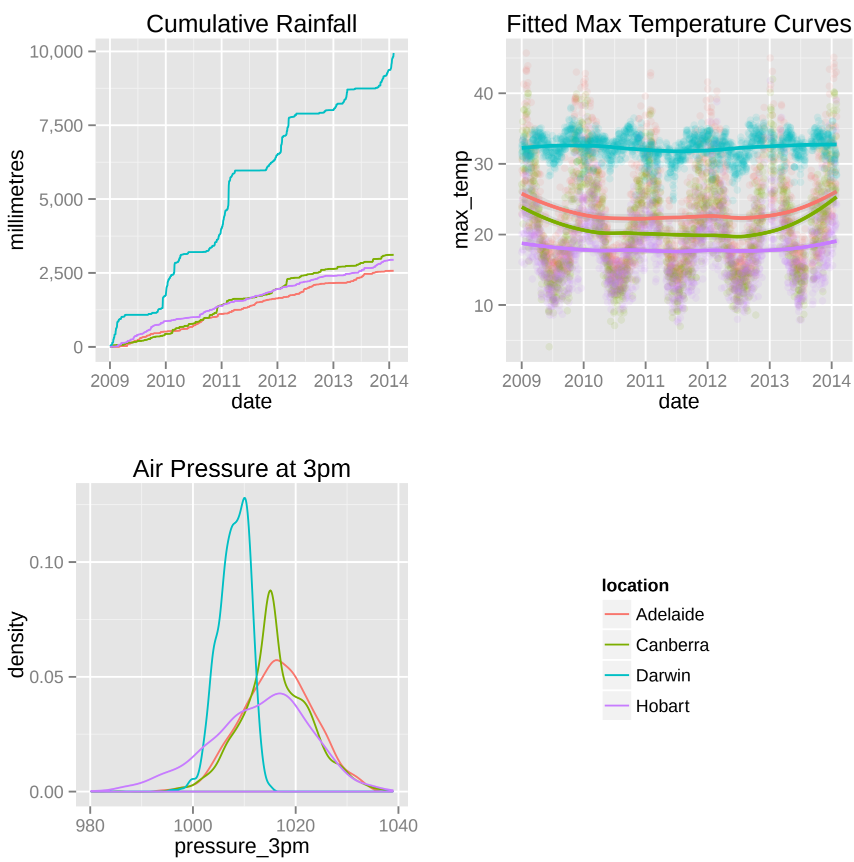
We are able to arrange the plots in quite a flexible manner.

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



31 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)
```



32 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.

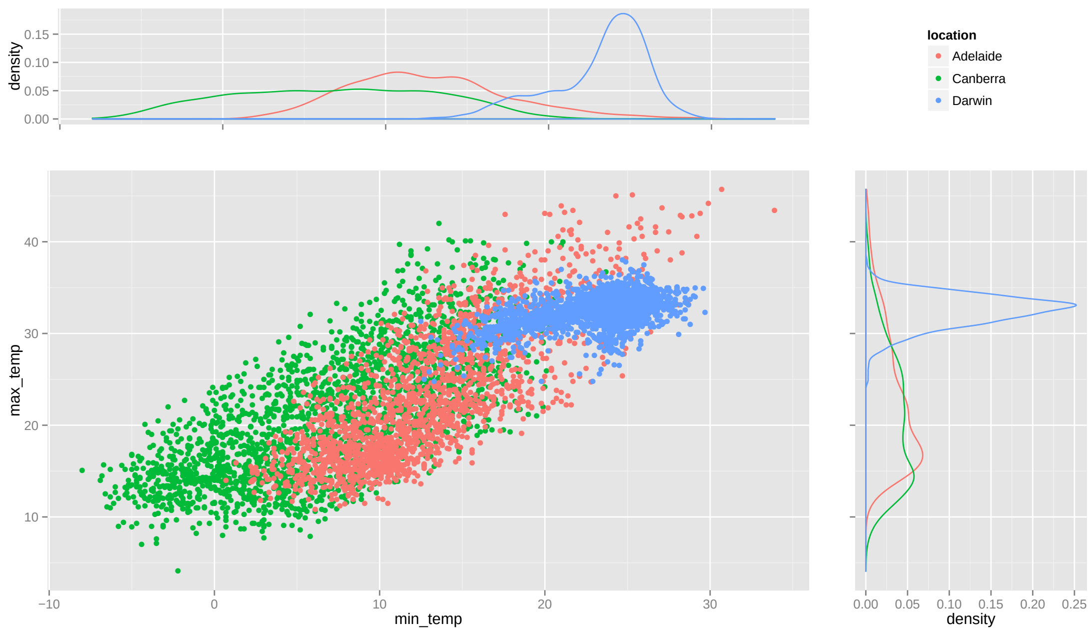
The data we use comes from rattle. From the full **weatherAUS** dataset we select a subset() covering just three cities. A basic scatter plot is built displaying the minimum and maximum daily temperatures. Two density plots are then generated, one for each of the variables in the scatter plot. The fourth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

```
library(rattle)
library(ggplot2)

dss <- subset(ds, location %in% c("Canberra", "Adelaide", "Darwin"))

dss$location <- ordered(dss$location)
p <- ggplot(dss, aes(min_temp, max_temp, colour=location))
p <- p + geom_point()

p5 <- p + theme(legend.position="none")

p6 <- ggplot(dss, aes(x=min_temp, 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(dss, aes(x=max_temp, 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)
```

33 Using grid.arrange(): 2D Histogram Plot

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

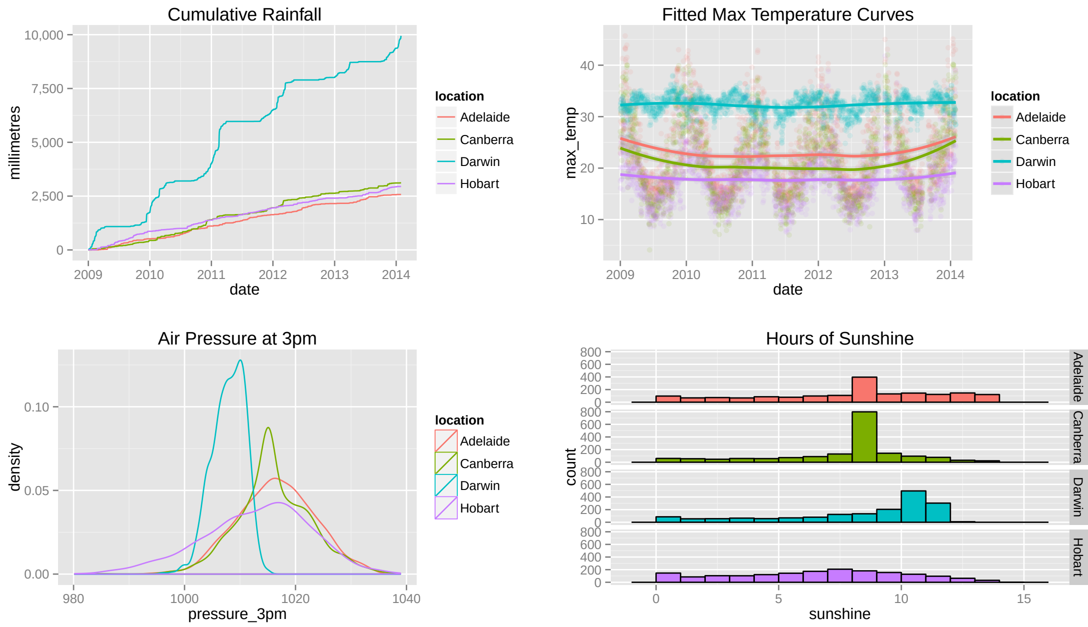
```
library(gridExtra)

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))
```



34 Using layOut(): Multiple Plots

Here we illustrate the ability to layout multiple plots in a regular grid using `layOut()` from `wq` (Jassby and Cloern, 2012).

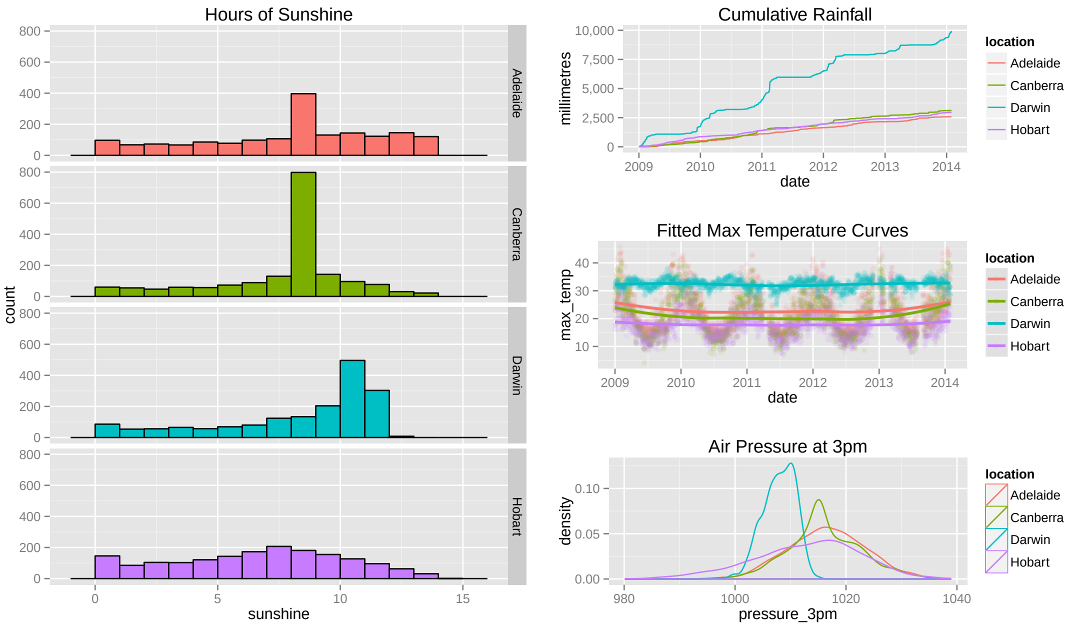
```
library(wq)
layOut(list(p1, 1, 1), list(p2, 1, 2), list(p3, 2, 1), list(p4, 2, 2))
```



35 Using layOut(): Arranging Plots

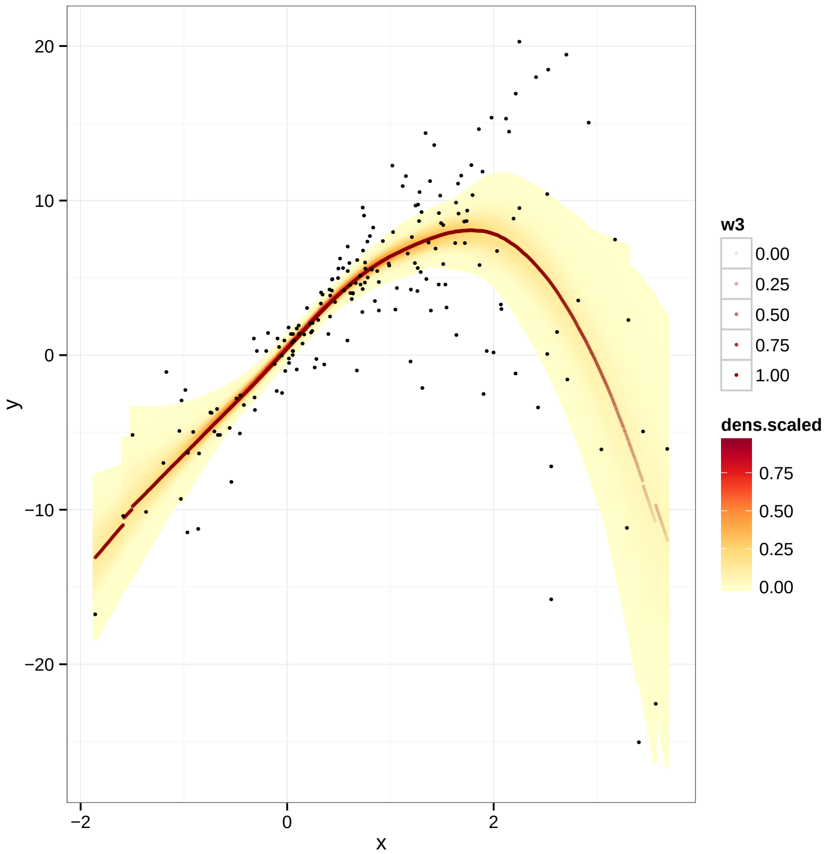
Here we illustrate the ability to layout multiple plots in a grid of differing sizes using `layOut()` from `wq`.

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



36 Visually Weighted Regression

From Nicrebread 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.



37 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.

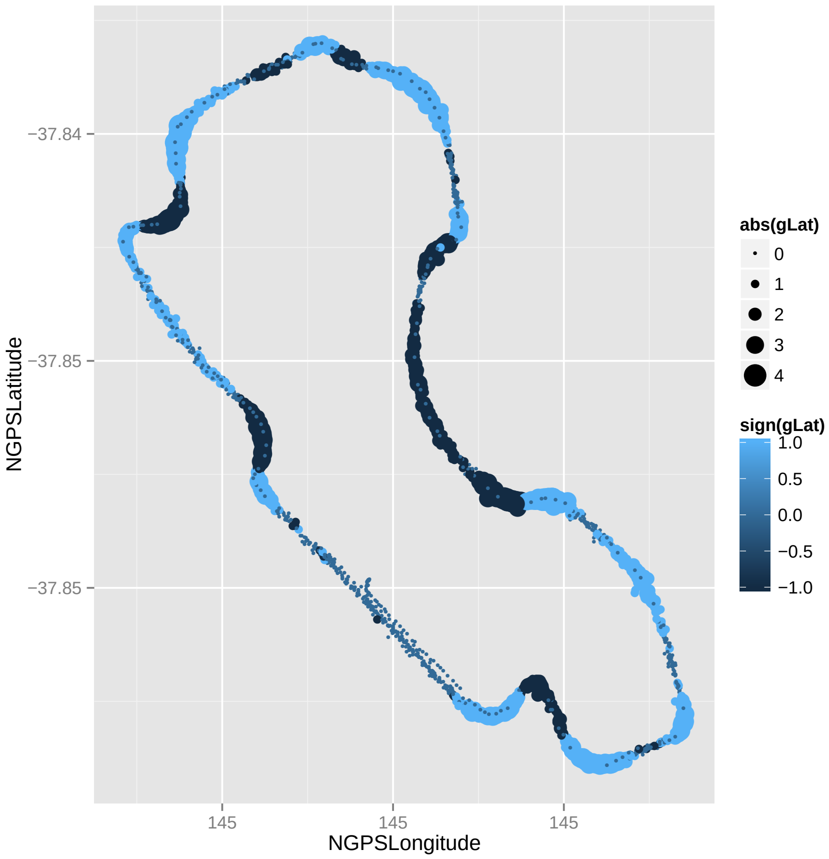
```
(load("data/f1.RData"))

## [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
....
```

38 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
```



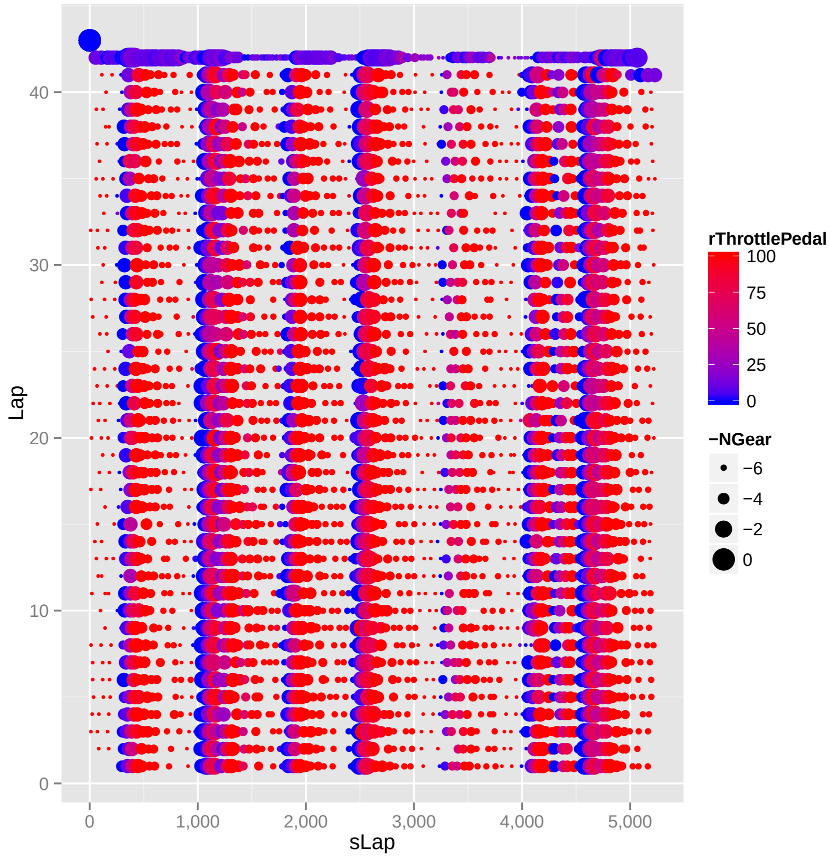
Based on Tony Hirst's Blog Post, March 2012

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.

39 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.

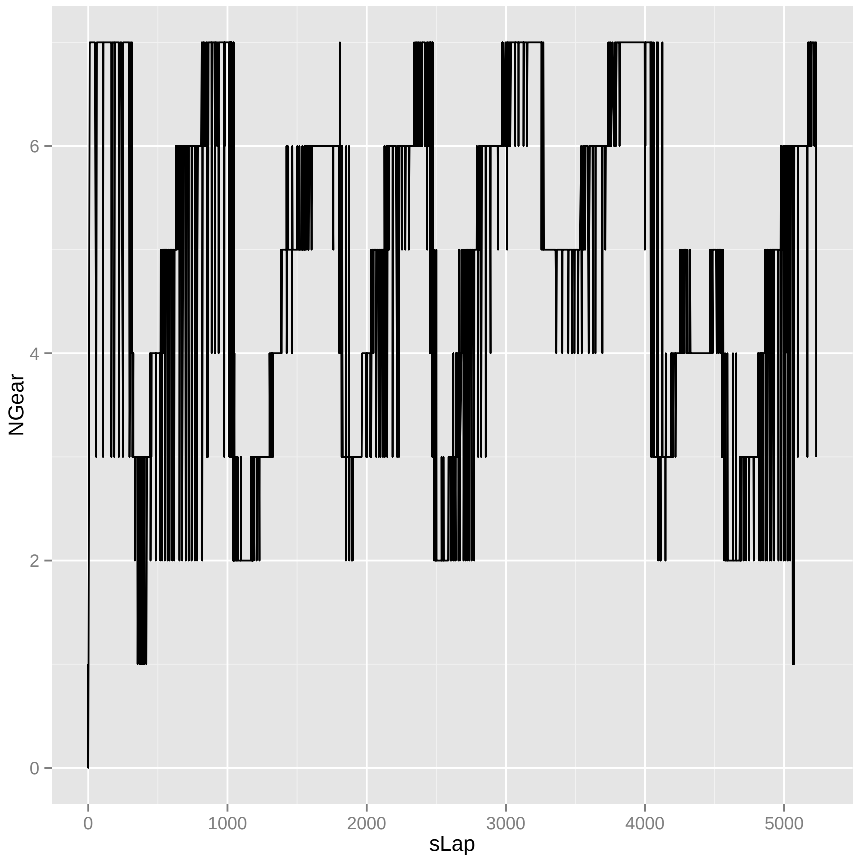
```
library(scales)
p <- ggplot(f1, aes(sLap, Lap))
p <- p + geom_point(aes(col=rThrottlePedal, size=-NGear))
p <- p + scale_colour_gradient(low="blue", high="red")
p <- p + scale_x_continuous(labels=comma)
p
```



Based on Tony Hirst's Blog Post, March 2012

40 F1: Gear Usage Around the Track

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

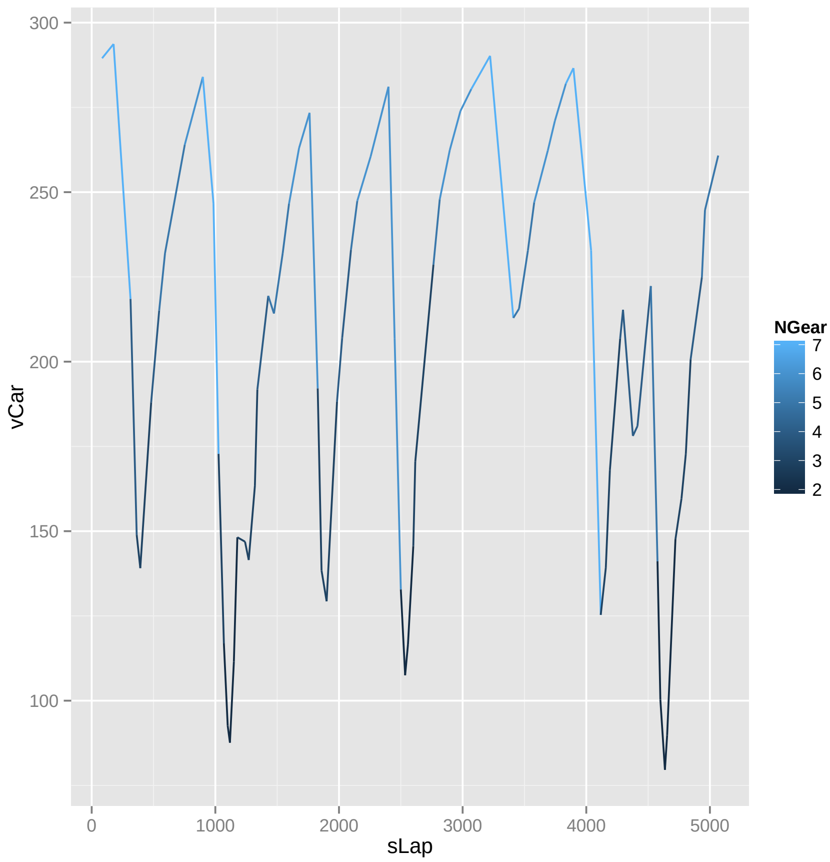


Based on Tony Hirst's Blog Post, March 2012

41 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))
```

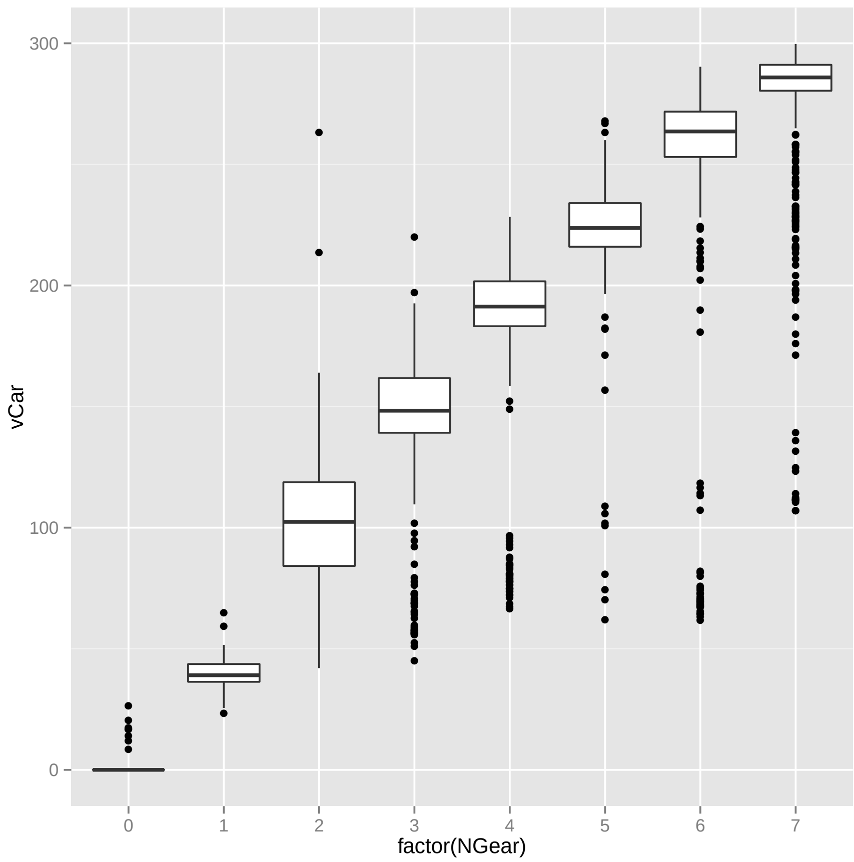


Based on Tony Hirst's Blog Post, March 2012

42 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()
```

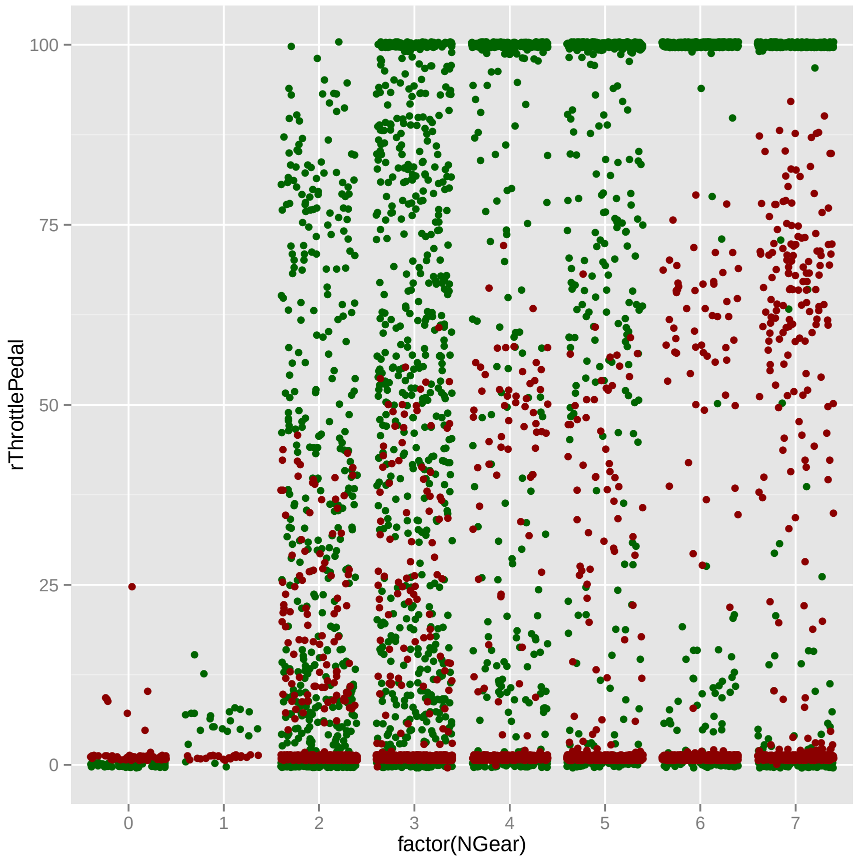


Based on Tony Hirst's Blog Post, March 2012

43 F1: Footwork

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

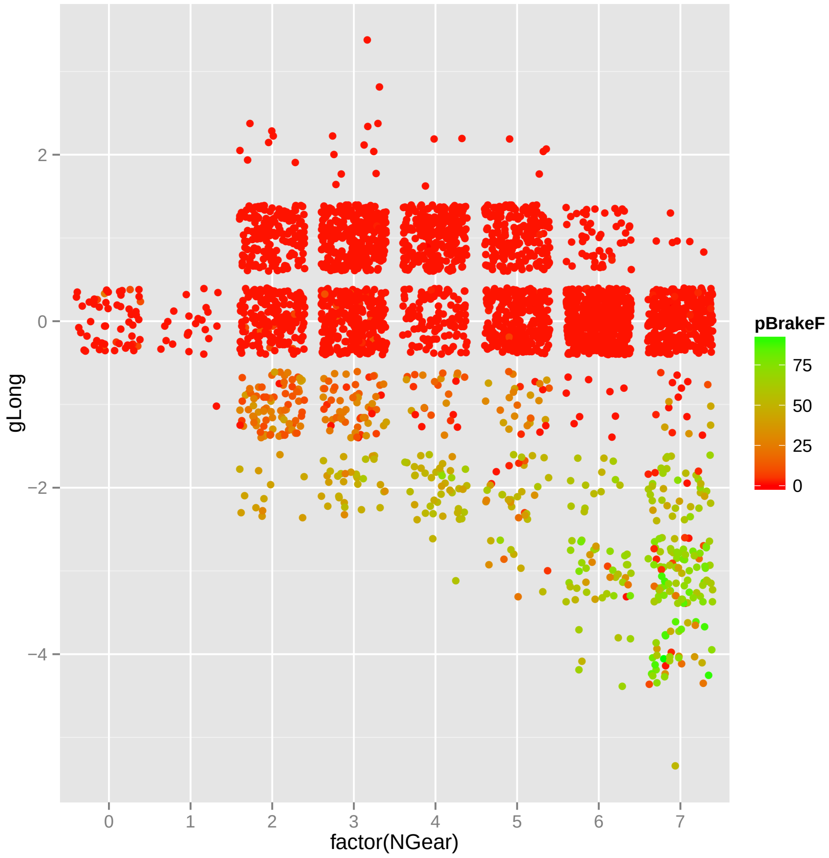
```
ggplot(f1, aes(factor(NGear))) +
  geom_jitter(aes(y=rThrottlePedal), colour='darkgreen') +
  geom_jitter(aes(y=pBrakeF), colour='darkred')
```



Based on Tony Hirst's Blog Post, March 2012

44 F1: Forces on the Driver

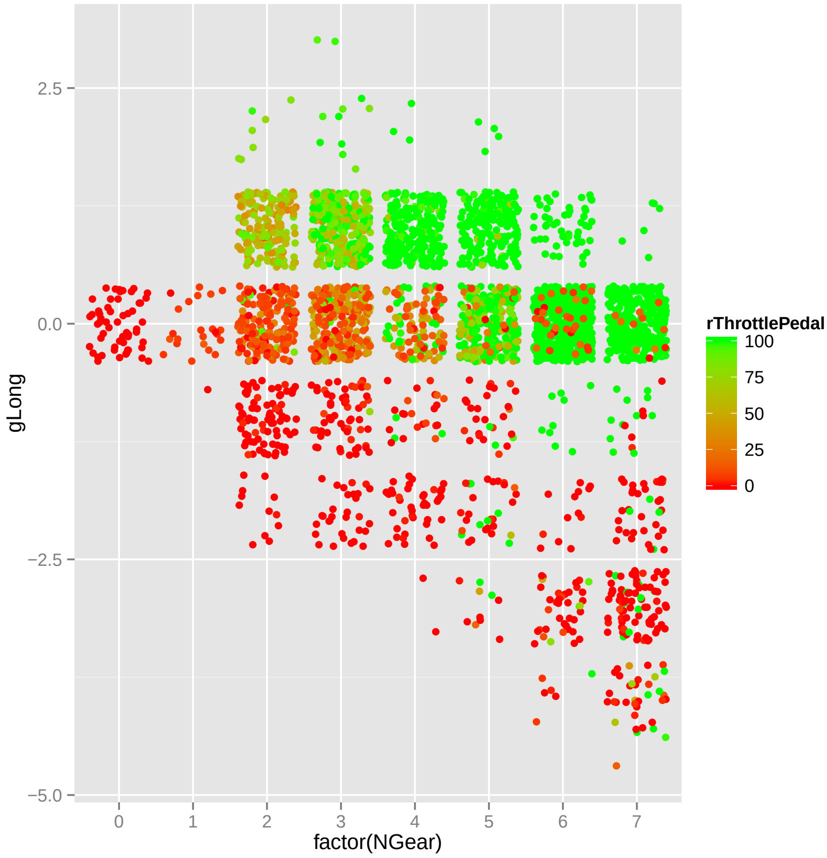
```
ggplot(f1, aes(factor(NGear), gLong)) +
  geom_jitter(aes(col=pBrakeF)) +
  scale_colour_gradient(low='red', high='green')
```



Based on Tony Hirst's Blog Post, March 2012

45 F1: More Forces

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

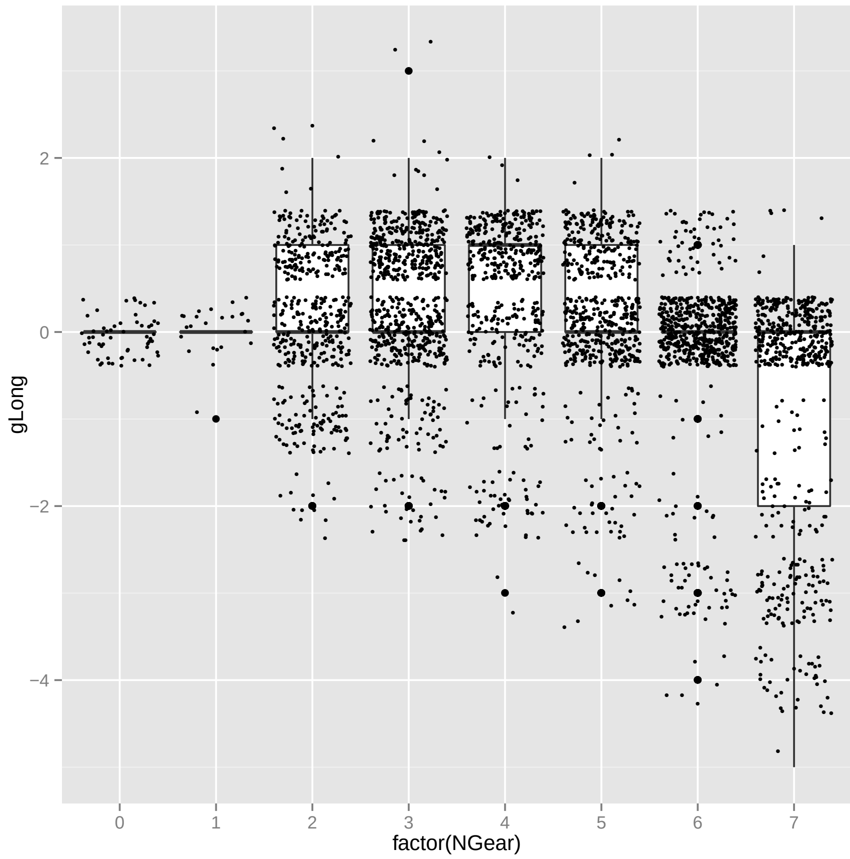


Based on Tony Hirst's Blog Post, March 2012

46 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.

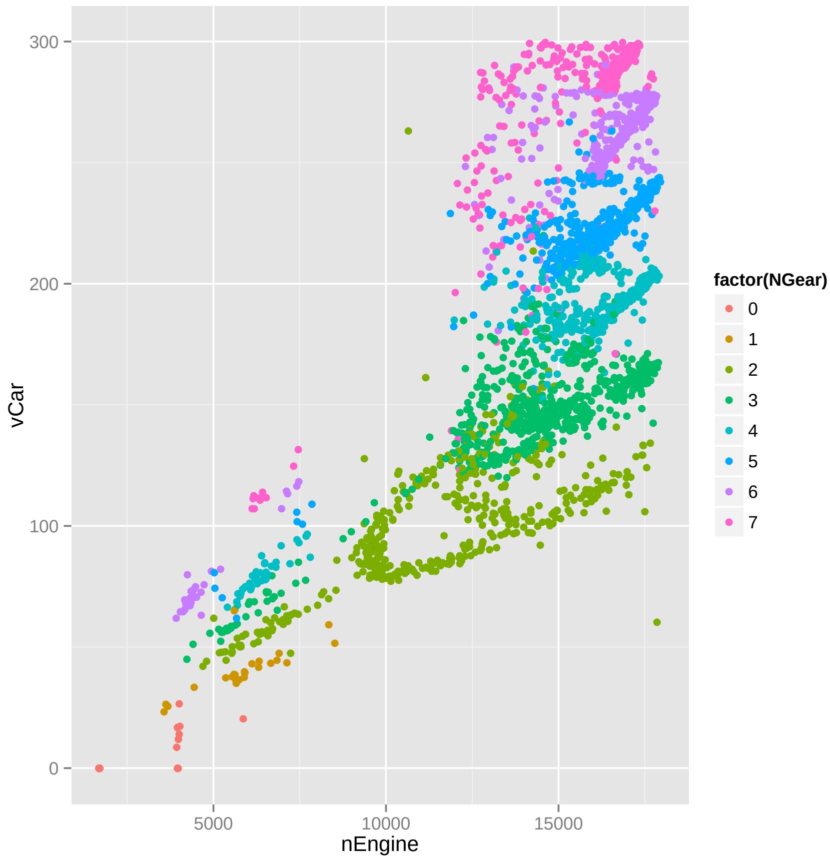
```
ggplot(f1, aes(factor(NGear), gLong)) +
  geom_boxplot() +
  geom_jitter(size=1)
```



Based on Tony Hirst's Blog Post, March 2012

47 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

48 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.

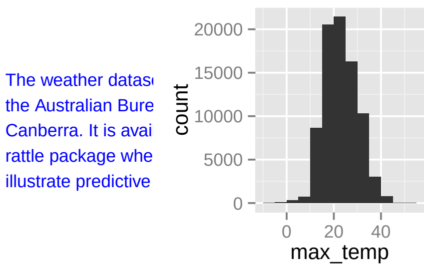
49 Plotting a Table and Text

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(ds[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(ds, aes(max_temp))
t3 <- t3 + geom_histogram(binwidth=5)
grid.arrange(t1, t2, arrangeGrob(t2, t3, ncol=3), ncol=1)
```

1	2008-12-01	Albury	13.4	22.9	0.6	4.6	8.4
2	2008-12-02	Albury	7.4	25.1	0.0	4.6	8.4
3	2008-12-03	Albury	12.9	25.7	0.0	4.6	8.4
4	2008-12-04	Albury	9.2	28.0	0.0	4.6	8.4
5	2008-12-05	Albury	17.5	32.3	1.0	4.6	8.4
6	2008-12-06	Albury	14.6	29.7	0.2	4.6	8.4
7	2008-12-07	Albury	14.3	25.0	0.0	4.6	8.4
8	2008-12-08	Albury	7.7	26.7	0.0	4.6	8.4
9	2008-12-09	Albury	9.7	31.9	0.0	4.6	8.4

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.



50 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.

51 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.

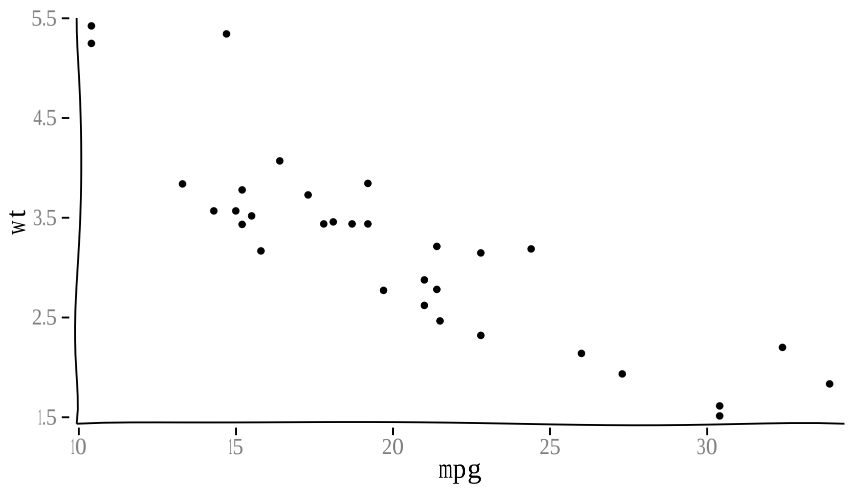
```
library(extrafont)
download.file("http://simonsoftware.se/other/xkcd.ttf", dest="xkcd.ttf")
system("mkdir ~/.fonts")
system("mv xkcd.ttf ~/.fonts")
font_import()
loadfonts()
```

The installation can be uninstalled with:

```
remove.packages(c("extrafont","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.

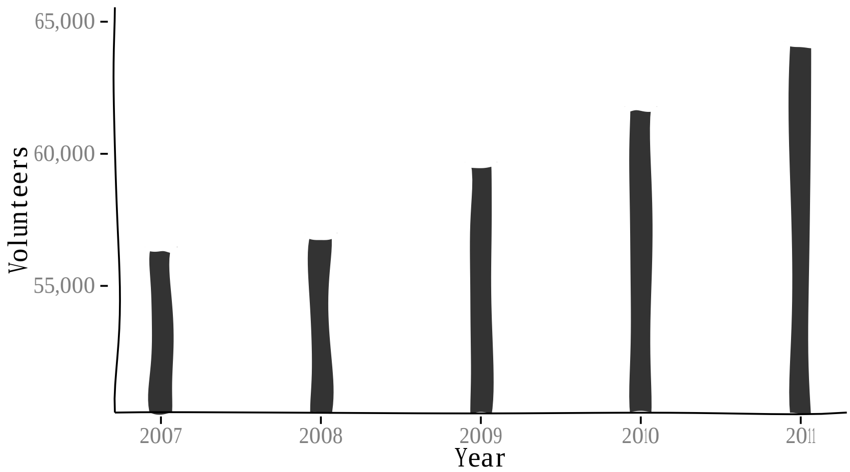
```
library(xkcd)
xrange <- range(mtcars$mpg)
yrange <- range(mtcars$wt)
p <- ggplot(mtcars, aes(mpg, wt))
p <- p + geom_point()
p <- p + xkcdaxis(xrange, yrange)
p
```



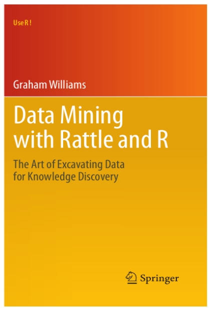
52 Having Some More Fun: xkcd Bar Chart

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



53 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 follow the links on the website with a * which indicates 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.

54 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 (2012). *wq: Exploring water quality monitoring data*. R package version 0.3-8, 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/.

Schloerke B, Crowley J, Cook D, Hofmann H, Wickham H, Briatte F, Marbach M, Thoen E (2014). *GGally: Extension to ggplot2*. R package version 0.4.5, URL http://CRAN.R-project.org/package=GGally.

Wickham H (2012a). *plyr: Tools for splitting, applying and combining data*. R package version 1.8, URL http://CRAN.R-project.org/package=plyr.

Wickham H (2012b). *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.

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.

Williams GJ (2014). *rattle: Graphical user interface for data mining in R*. R package version 3.0.2, URL http://rattle.togaware.com/.

This document, sourced from GGPlot2O.Rnw revision 282, was processed by KnitR version 1.5 of 2013-09-28 and took 91.2 seconds to process. It was generated by gjw on nyx running Ubuntu 13.10 with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-02-14 06:10:59.