Data Science with R
Exploring Data with GGPlot2

Graham.Williams@togaware.com

16th August 2014

Visit http://HandsOnDataScience.com/ for more Chapters.

The ggplot2 (Wickham and Chang, 2014) 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)          # Visualise data.
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.
library(dplyr)            # Data manipulation.
```

As we work through this chapter, 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 chapter 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.

1 Preparing the Dataset

We use the relatively large **weatherAUS** dataset from rattle (Williams, 2014) to illustrate the capabilities of ggplot2. For plots which generate large images we might use random subsets of the same dataset.

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

The dataset is summarised below.

```
dim(ds)

## [1] 91953    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)

## Source: local data frame [6 x 24]
##
##         Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-01   Albury    13.4    22.9      0.6          NA       NA
....

tail(ds)

## Source: local data frame [6 x 24]
##
##             Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 91948 2014-06-24    Uluru     7.3    18.0        0          NA       NA
....

str(ds)

## Classes 'tbl_df', 'tbl' and 'data.frame': 91953 obs. of  24 variables:
##  $ Date         : Date, format: "2008-12-01" "2008-12-02" ...
##  $ Location     : Factor w/ 49 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: 2344   Min.   :-8.5   Min.   :-3.8
##  1st Qu.:2010-03-26   Sydney  : 2252   1st Qu.: 7.6   1st Qu.:17.9
##  Median :2011-09-07   Adelaide: 2101   Median :11.9   Median :22.4
....
```

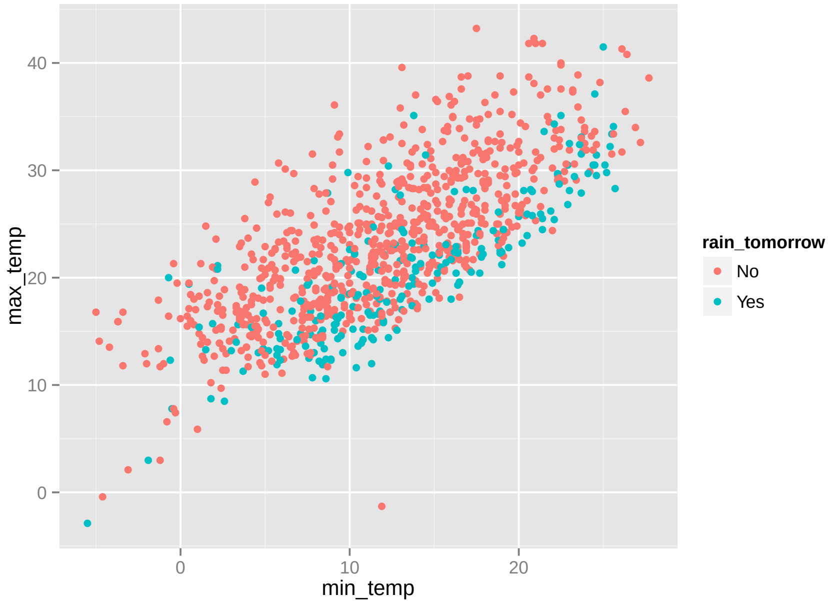
2 Collecting Information

```
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] 194876

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

## [1] 0
```

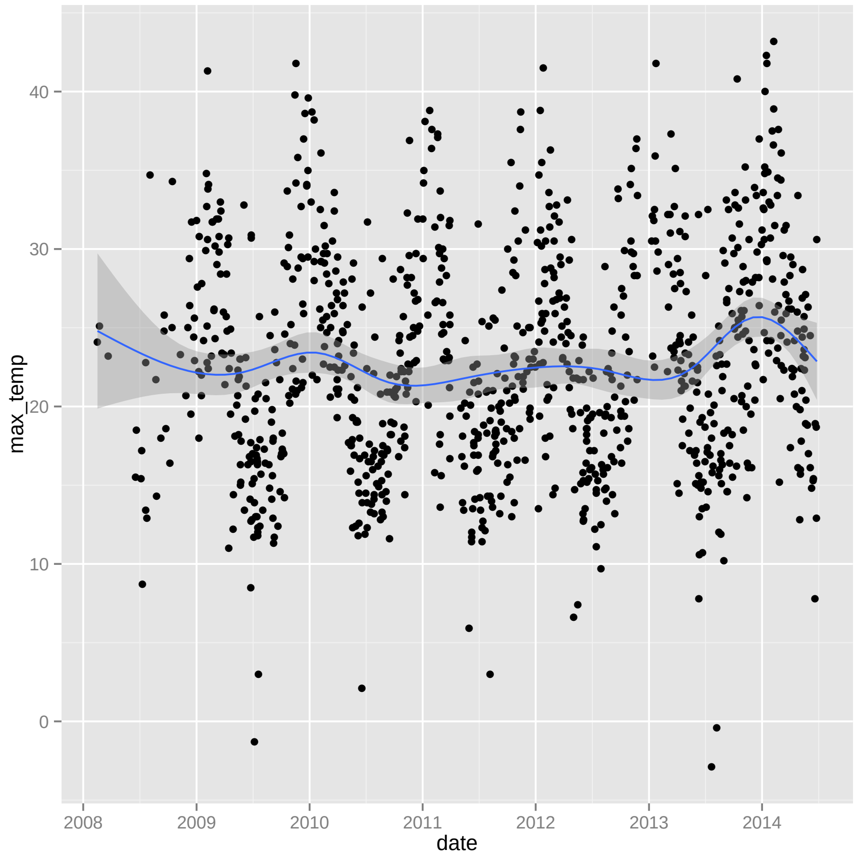
3 Scatter Plot



A scatter plot displays points scattered over a plot. The points are specified as x and y for a two dimensional plot, as specified by the aesthetics. We use `geom_point()` to plot the points.

```
sobs <- sample(nrow(ds), 1000)

ds[sobs,]                                                        %>%
  ggplot(aes(x=min_temp, y=max_temp, colour=rain_tomorrow))       +
  geom_point()
```

3.1 Adding a Smooth Fitted Curve

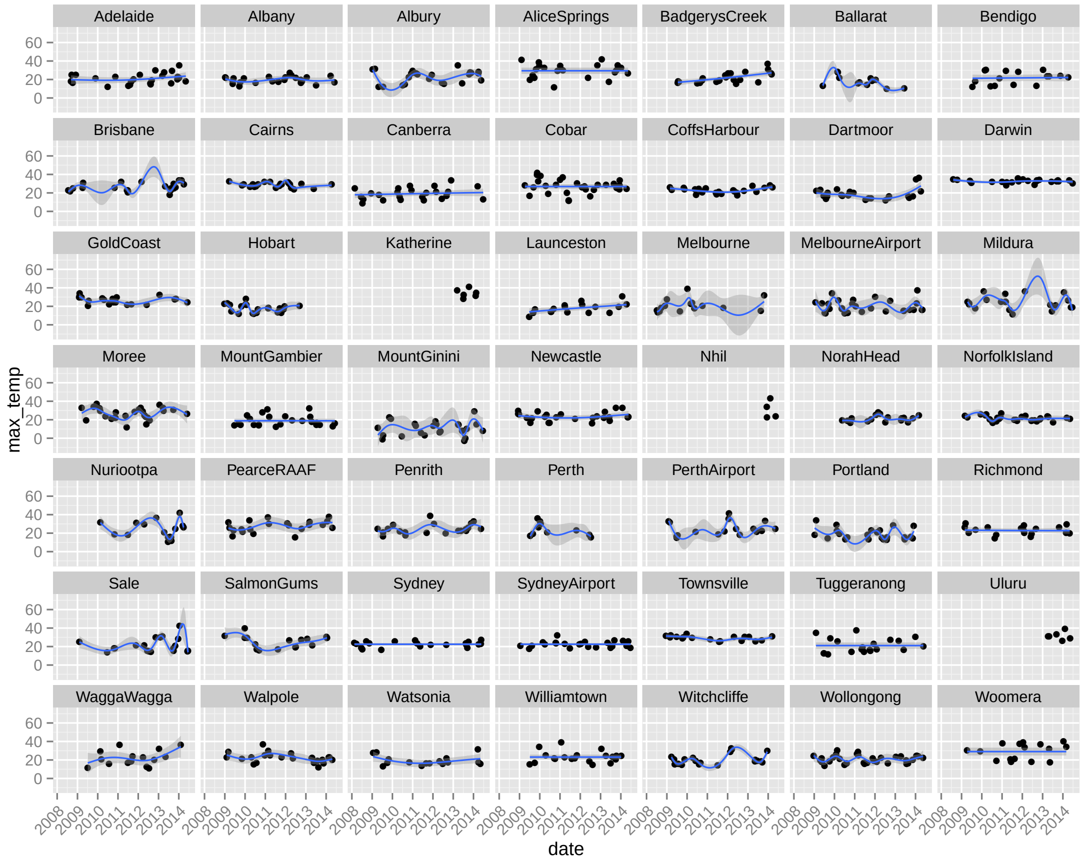


Here we have added a smooth fitted curve using `geom_smooth()`. Since the dataset has many points the smoothing method recommend is `method="gam"` and will be automatically chosen if not specified, with a message displayed to inform us of this. The formula specified, `formula=y~s(x, bs="cs")` is also the default for `method="gam"`.

Typical of scatter plots of big data there is a mass of overlaid points. We can see patterns though, noting the obvious pattern of seasonality. Notice also the three or four white vertical bands—this is indicative of some systemic issue with the data in that for specific days it would seem there is no data collected. Something to explore and understand further, but our scope for now.

```
ds[sobs,]                                                    %>%
  ggplot(aes(x=date, y=max_temp))                             +
  geom_point()                                                +
  geom_smooth(method="gam", formula=y~s(x, bs="cs"))
```

3.2 Using Facet to Plot Multiple Locations



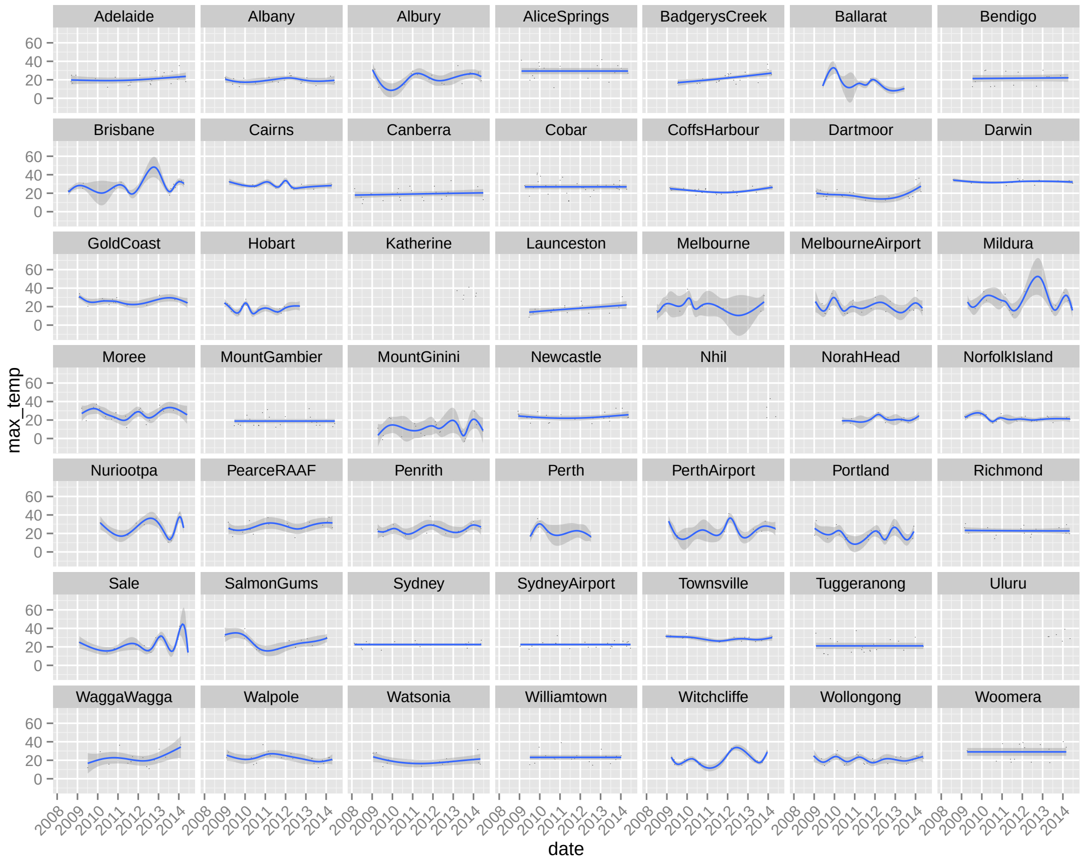
Partitioning the dataset by a categoric variable will reduce the blob effect for big data. For the above plot we use `facet_wrap()` to separately plot each location's maximum temperature over time. The x axis tick labels are rotated 45°.

The seasonal effect is again very clear, and a mild increase in the maximum temperature over the small period is evident in many locations.

Using large dots on the smaller plots still leads to much overlaid plotting.

```
ds[sobs,]                                                    %>%
  ggplot(aes(x=date, y=max_temp))                            +
  geom_point()                                               +
  geom_smooth(method="gam", formula=y~s(x, bs="cs"))        +
  facet_wrap(~location)                                      +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```

3.3 Facet over Locations with Small Dots

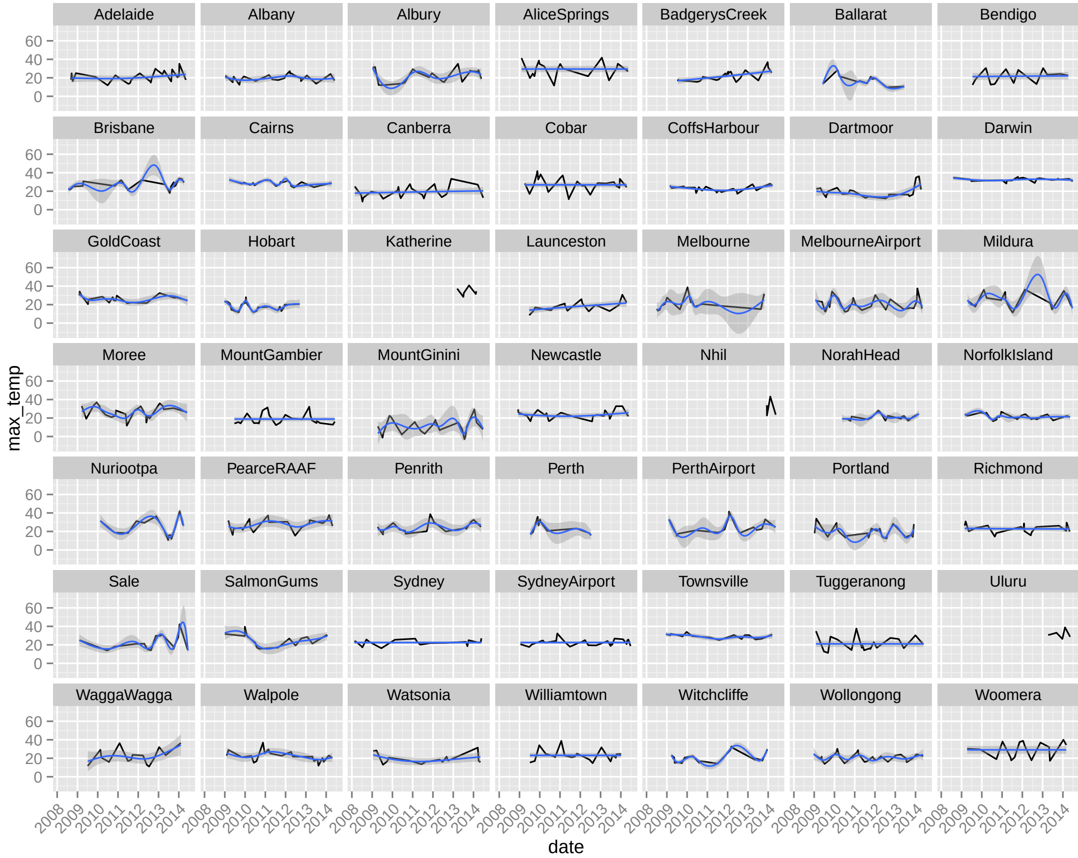


Here we have changed the size of the points to be `size=0.2` instead of the default `size=0.5`. The points then play less of a role in the display.

Changing to plotted points to be much smaller de-clutters the plot significantly and improves the presentation.

```
ds[sobs,]                                                    %>%
  ggplot(aes(x=date, y=max_temp))                             +
  geom_point(size=0.2)                                        +
  geom_smooth(method="gam", formula=y~s(x, bs="cs"))          +
  facet_wrap(~location)                                       +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```

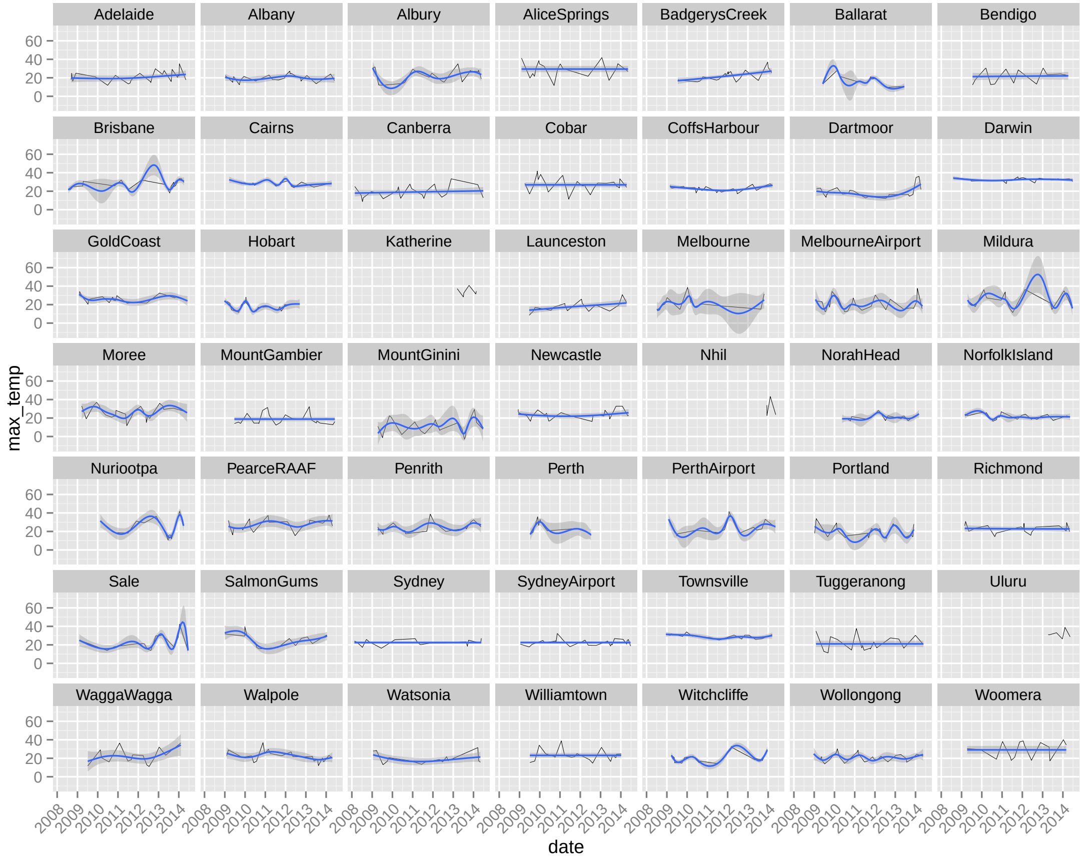
3.4 Facet over Locations with Lines



Changing to lines using `geom_line()` instead of points using `geom_points()` might improve the presentation over the big dots, somewhat. Though it is still a little dense.

```
ds[sobs,]                                                        %>%
  ggplot(aes(x=date, y=max_temp))                                 +
  geom_line()                                                     +
  geom_smooth(method="gam", formula=y~s(x, bs="cs"))              +
  facet_wrap(~location)                                           +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```

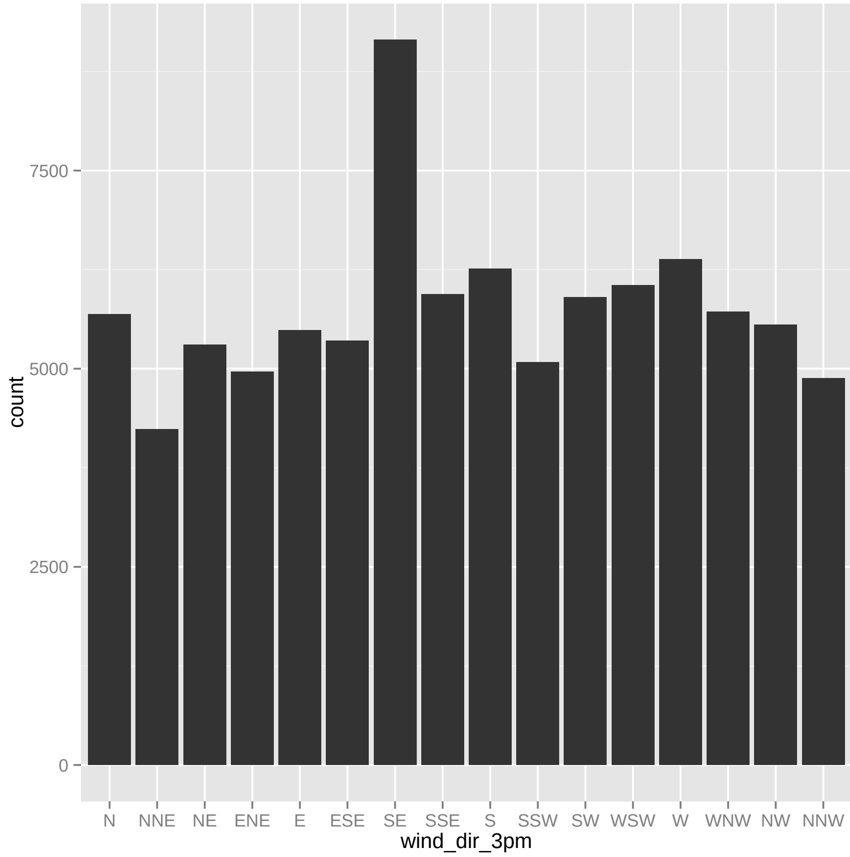
3.5 Facet over Locations with Thin Lines



Finally, we use a smaller point size to draw the lines. You can decide which looks best for the data you have. Compare the use of lines here and above using `geom_point()`. Often the decision comes down to what you are aiming to achieve with the plots or what story you will tell.

```
ds[sobs,]                                                    %>%
  ggplot(aes(x=date, y=max_temp))                             +
  geom_line(size=0.1)                                         +
  geom_smooth(method="gam", formula=y~s(x, bs="cs"))          +
  facet_wrap(~location)                                       +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```

4 Histograms and Bar Charts



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.

```
p <- ds                                                    %>%
  ggplot(aes(x=wind_dir_3pm))                               +
  geom_bar()
p
```

4.1 Saving Plots to File

Once we have a plot displayed we can save the plot to file quite simply using `ggsave()`. The format is determined automatically by the name of the file to which we save the plot. Here we save the plot as a PDF:

```
ggsave("barchart.pdf", width=11, height=7)
```

The default `width=` and `height=` are those of the current plotting window. For saving the plot above we have specified a particular width and height, presumably to suit our requirements for placing the plot in a document or in some presentation.

Essentially, `ggsave()` is a wrapper around the various functions that save plots to files, and adds some value to these default plotting functions.

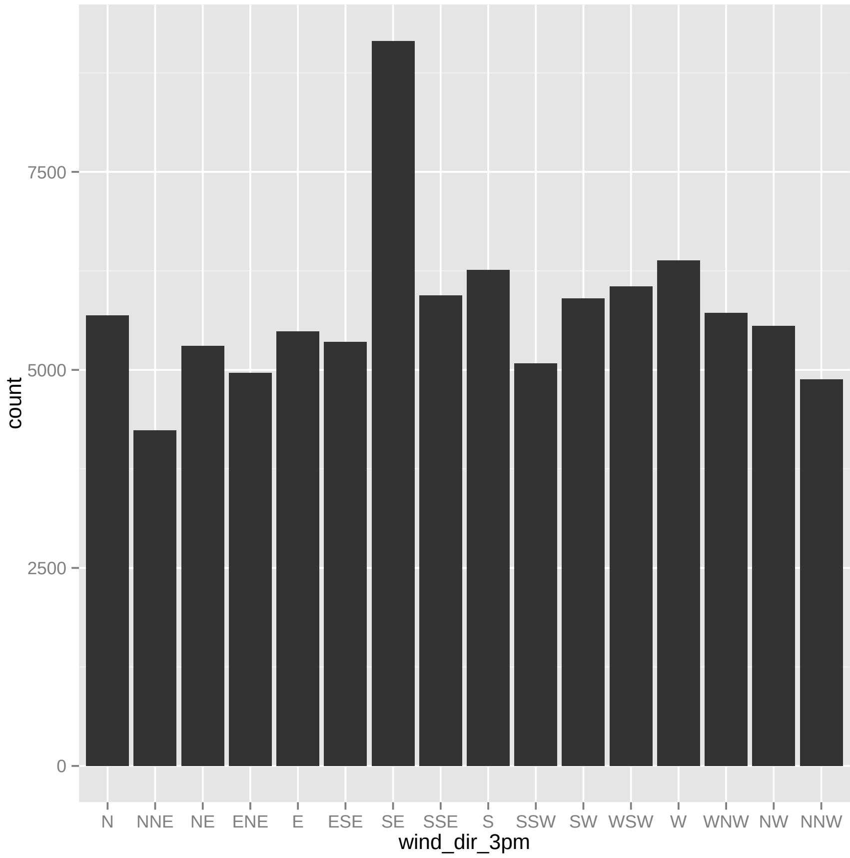
The traditional approach is to initiate a device, print the plot, and the close the device.

```
pdf("barchart.pdf", width=11, height=7)
p
dev.off()
```

Note that for a `pdf()` device the default `width=` and `height=` are 7 (inches). Thus, for both of the above examples we are widening the plot whilst retaining the height.

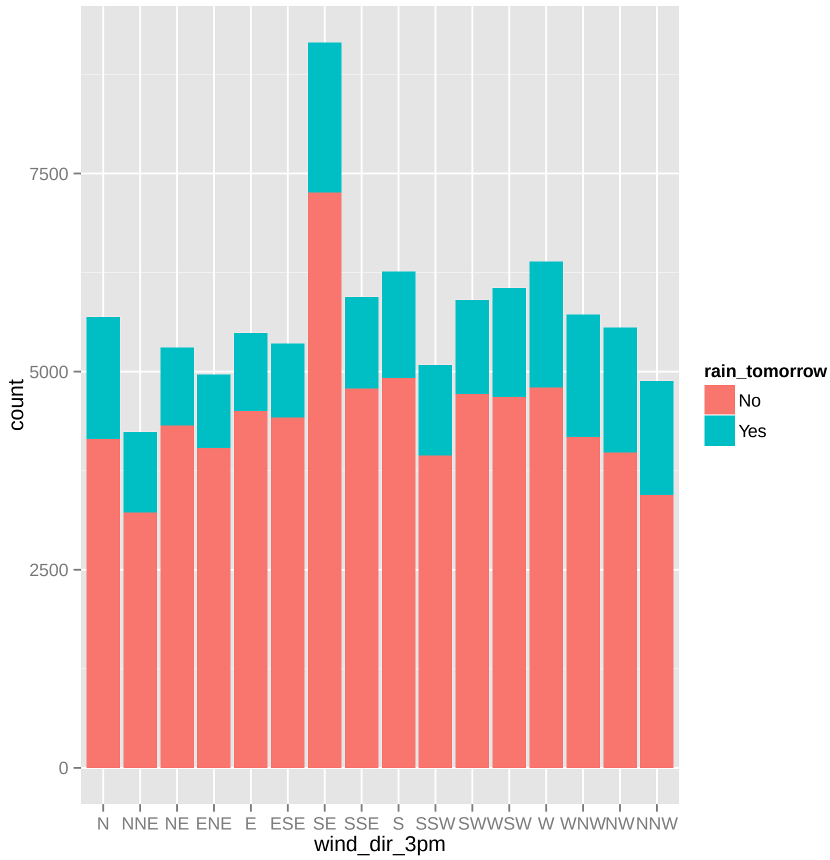
There is some art required in choosing a good width and height. By increasing the height or width any text that is displayed on the plot, essentially stays the same size. Thus by increasing the plot size, the text will appear smaller. By decreasing the plot size, the text becomes larger. Some experimentation is often required to get the right size for any particular purpose.

4.2 Bar Chart



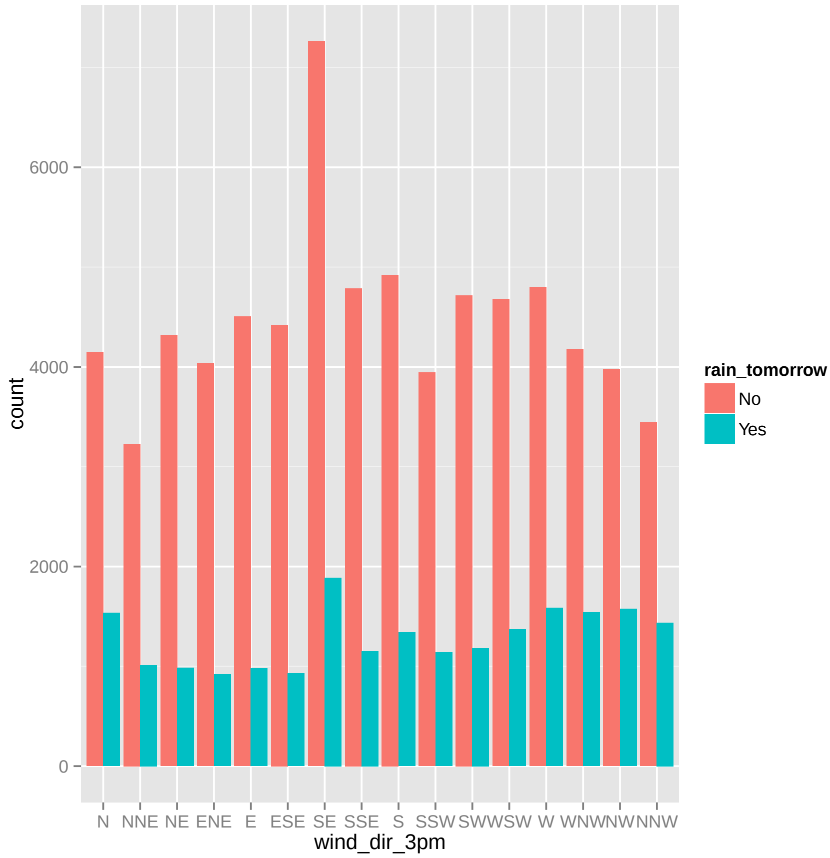

```
ds                                                    %>%
  ggplot(aes(x=wind_dir_3pm))                          +
  geom_bar()
```

4.3 Stacked Bar Chart



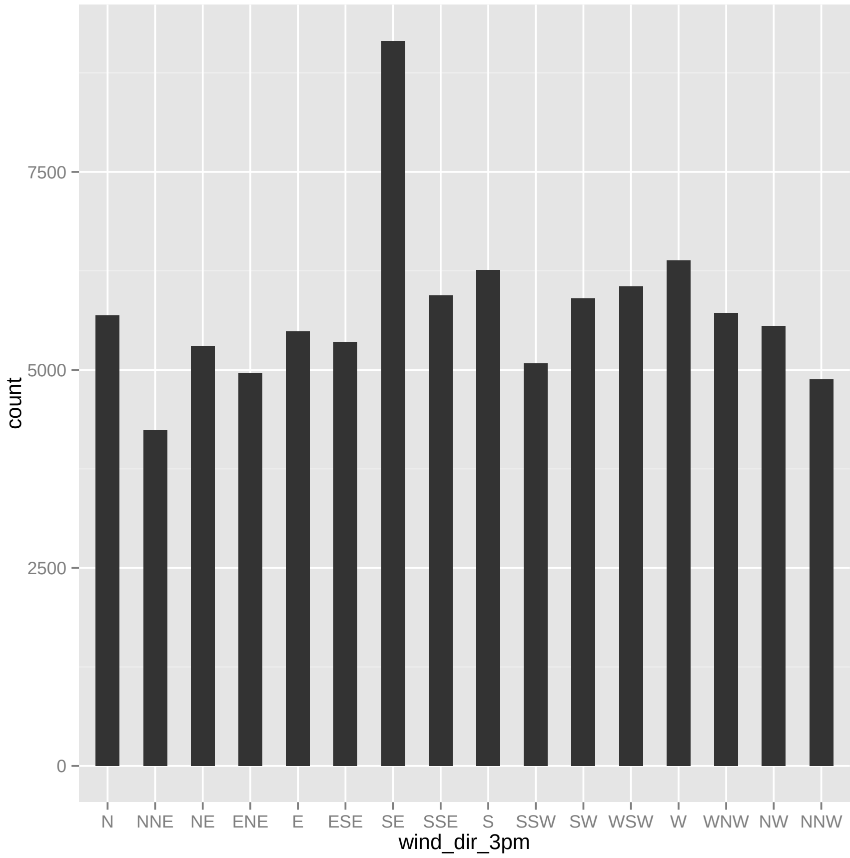

```
ds                                                        %>%
  ggplot(aes(x=wind_dir_3pm, fill=rain_tomorrow))          +
  geom_bar()
```

4.4 Dodged Bar Chart




```
ds                                                                %>%
  ggplot(aes(x=wind_dir_3pm, fill=rain_tomorrow))                  +
  geom_bar(position="dodge")
```

4.5 Narrow Bars

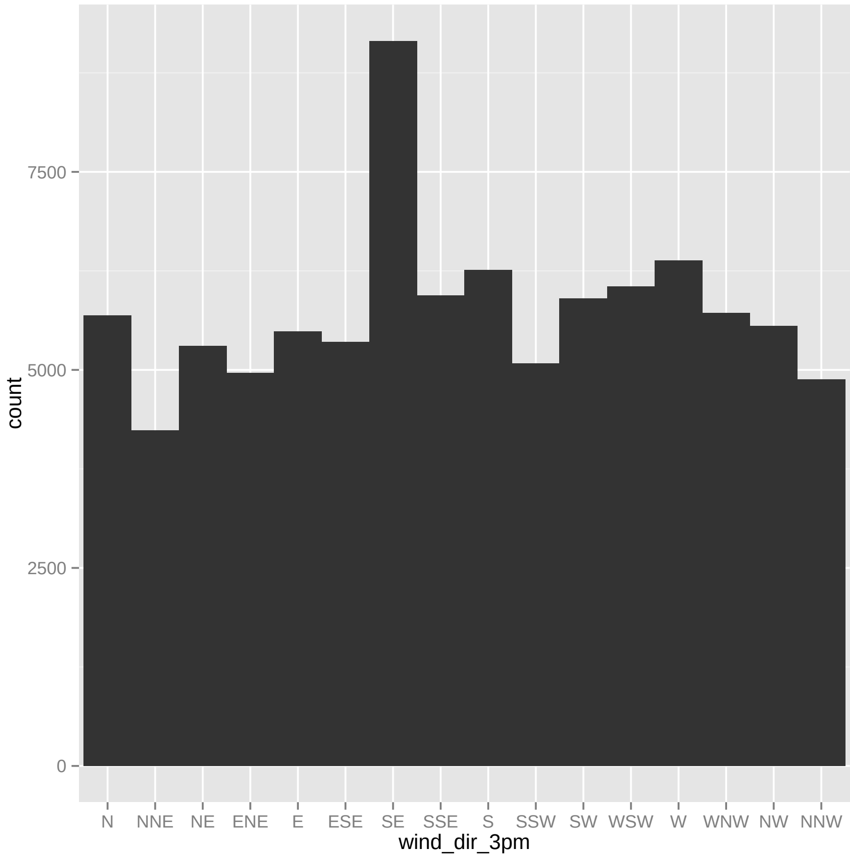


There are many options available to change the appearance of the histogram to make it look like almost anything we could want. In the following pages we will illustrate a few simpler modifications.

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!

```
ds                                                                    %>%
  ggplot(aes(wind_dir_3pm))                                            +
  geom_bar(width=0.5)
```

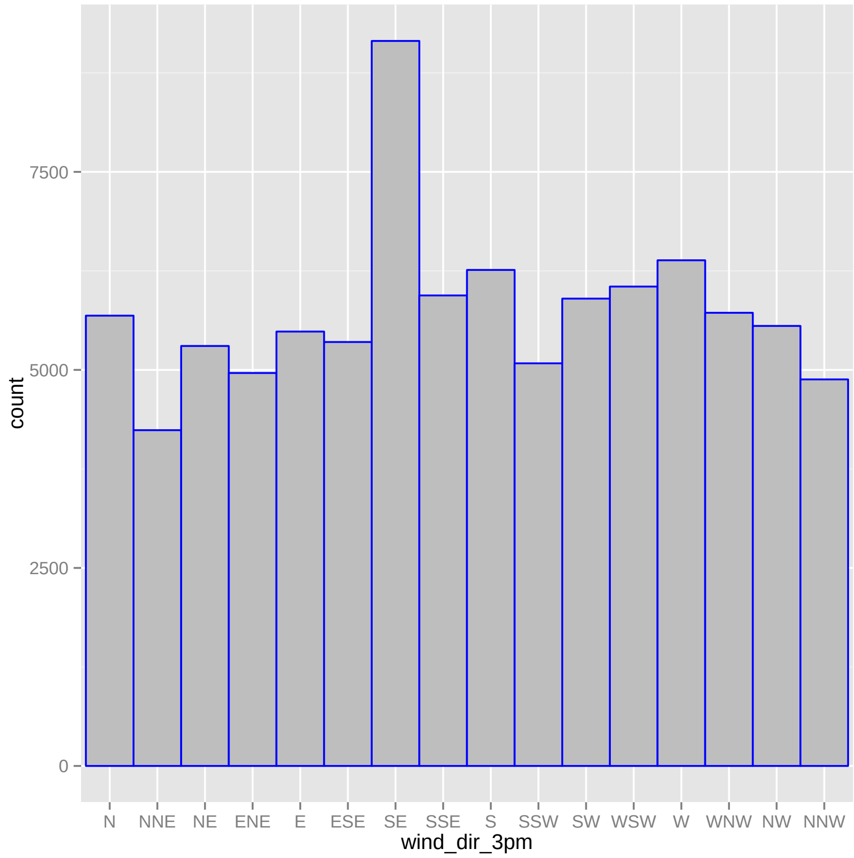
4.6 Full Width Bars



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

```
ds                                                          %>%
  ggplot(aes(wind_dir_3pm))                                  +
  geom_bar(width=1)
```

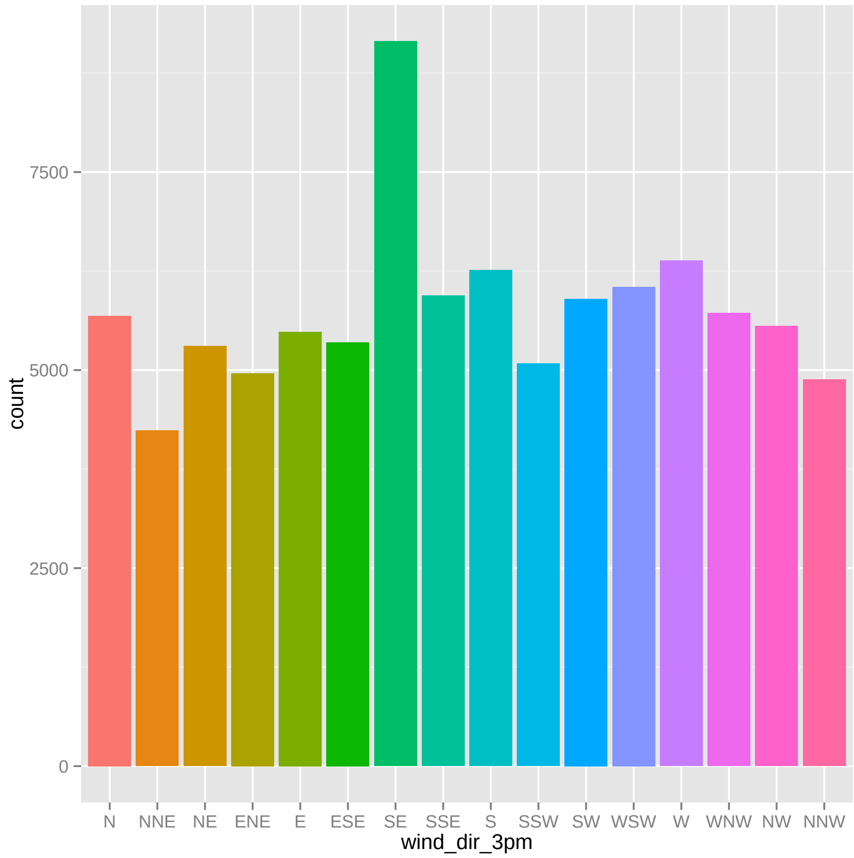
4.7 Full Width Bars with Borders



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.

```
ds                                                                    %>%
  ggplot(aes(wind_dir_3pm))                                            +
  geom_bar(width=1, colour="blue", fill="grey")
```

4.8 Coloured Histogram Without a Legend

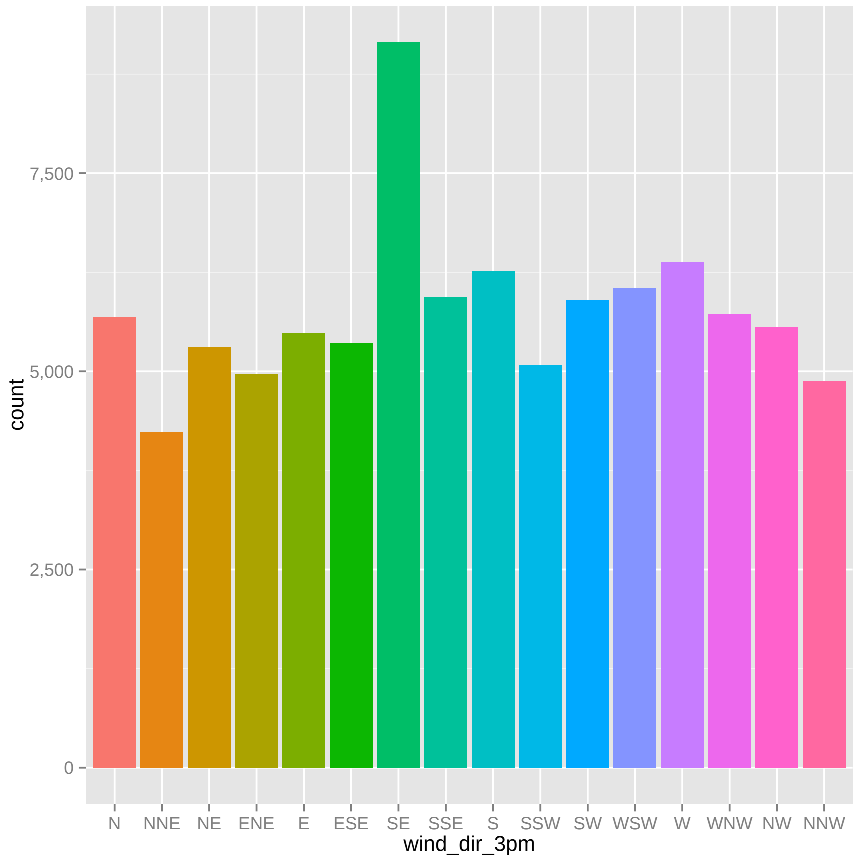


Now we really add a flamboyant streak to our plot by adding quite a spread of colour. To do so we simply specify a `fill=` aesthetic to be controlled by the values of the variable `wind_dir_3pm` which of course is the variable being plotted on the x-axis. A good set of colours is chosen by default.

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

```
ds                                                                    %>%
  ggplot(aes(wind_dir_3pm, fill=wind_dir_3pm))                         +
  geom_bar()                                                           +
  theme(legend.position="none")
```

4.9 Comma Formatted Labels

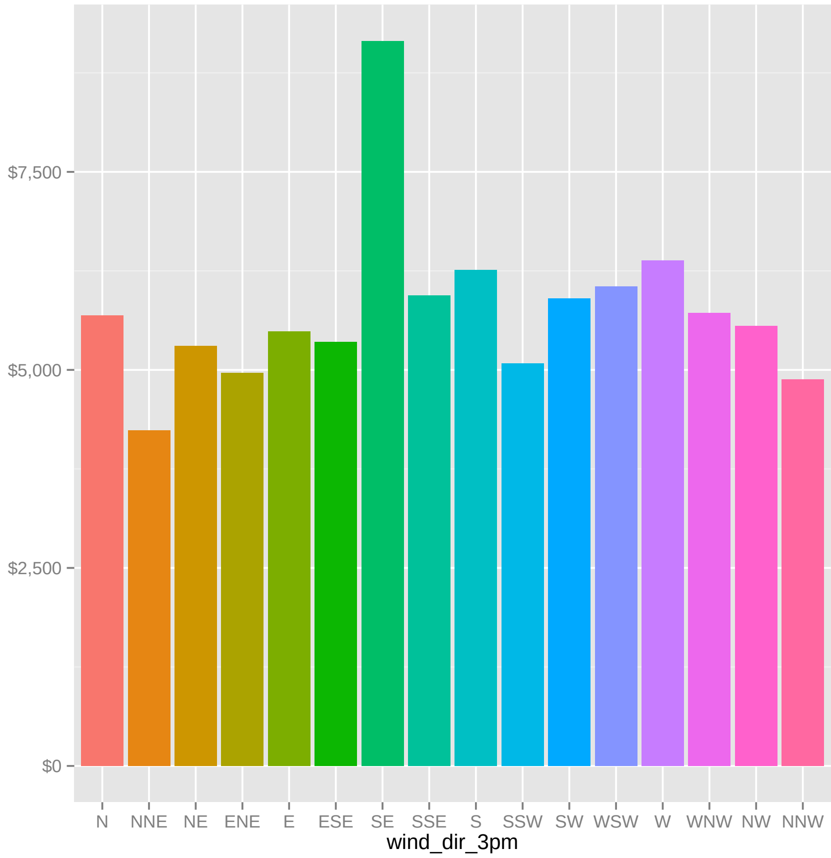


Since ggplot2 Version 0.9.0 the scales (Wickham, 2014) package has been introduced to handle many of the scale operations, in such a way as to support base and lattice graphics, as well as ggplot2 graphics. Scale operations include position guides, as in the axes, and aesthetic guides, as in the legend.

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`.

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

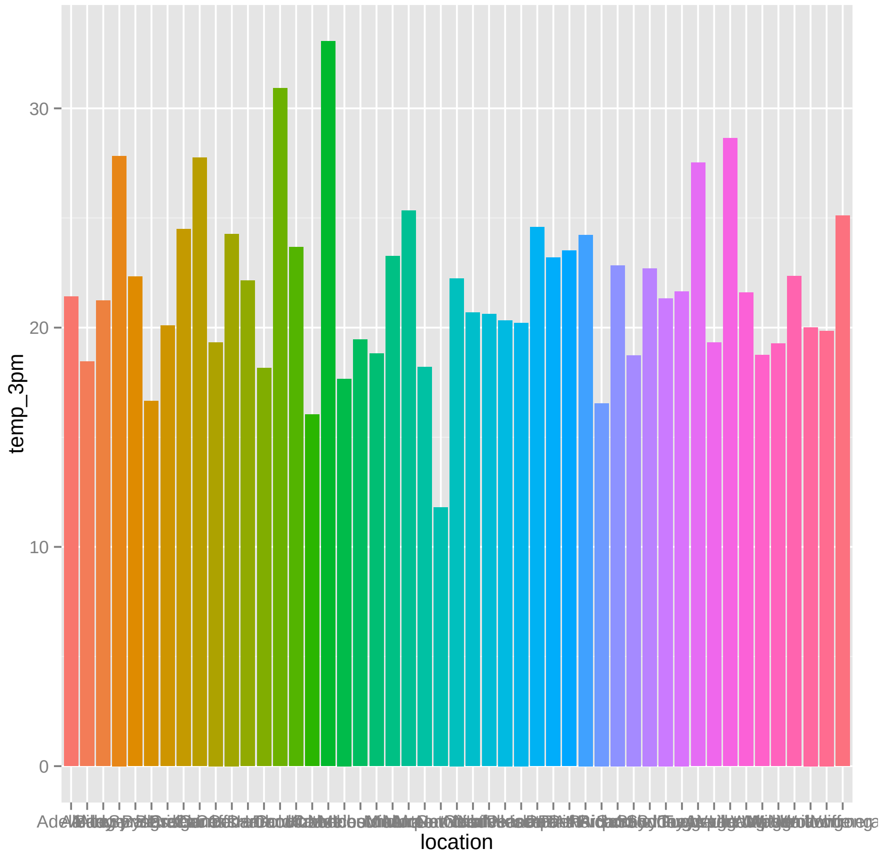
4.10 Dollar Formatted Labels



Though we are not actually plotting currency here, one of the supported scales is `dollar`. Choosing this will add the comma but also prefix the amount with `$`.

```
ds                                                                  %>%
  ggplot(aes(wind_dir_3pm, fill=wind_dir_3pm))                      +
  geom_bar()                                                        +
  scale_y_continuous(labels=dollar)                                 +
  labs(y="")                                                        +
  theme(legend.position="none")
```

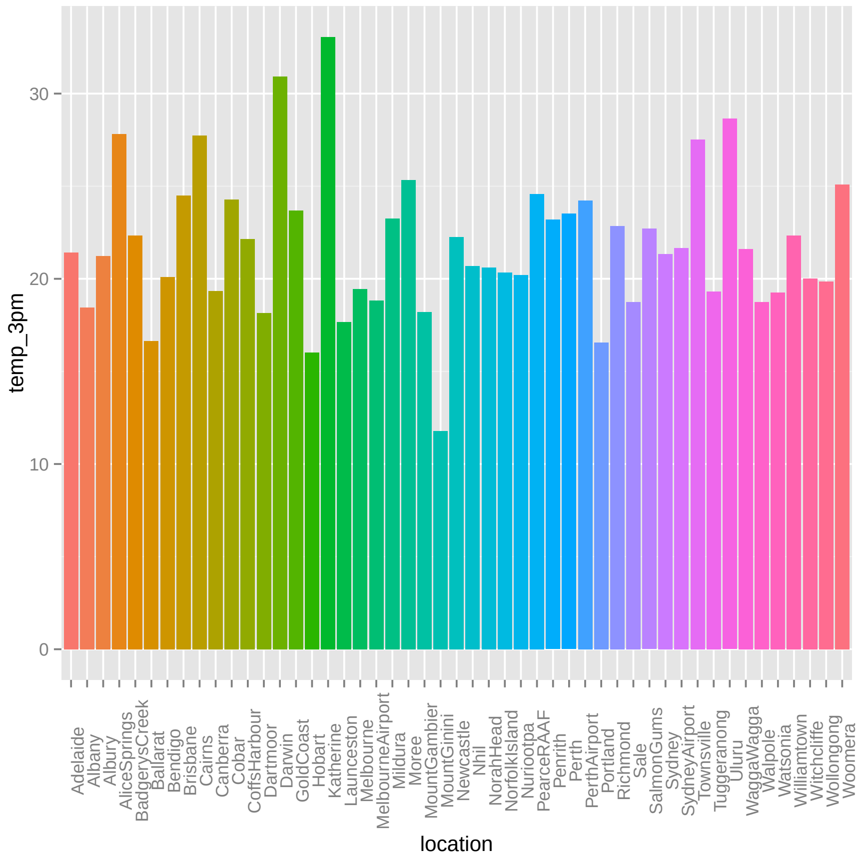
4.11 Multiple Bars



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.

```
ds                                                              %>%
  ggplot(aes(x=location, y=temp_3pm, fill=location))             +
  stat_summary(fun.y="mean", geom="bar")                         +
  theme(legend.position="none")
```

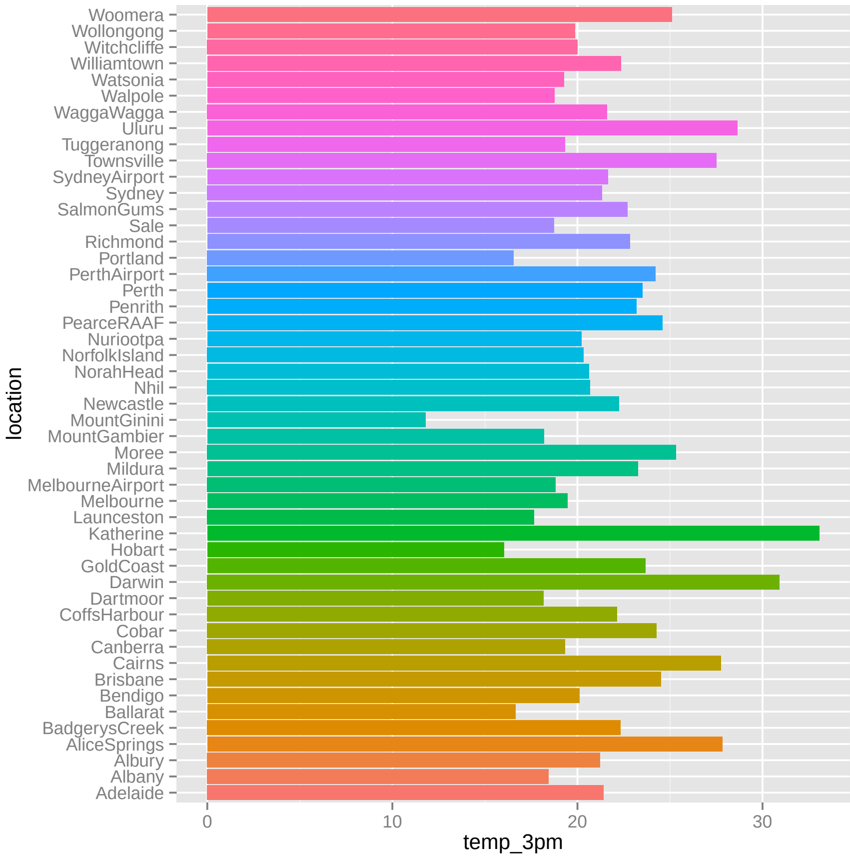
4.12 Rotated Labels



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

```
ds                                                          %>%
  ggplot(aes(location, temp_3pm, fill=location))              +
  stat_summary(fun.y="mean", geom="bar")                      +
  theme(legend.position="none",
        axis.text.x=element_text(angle=90))
```

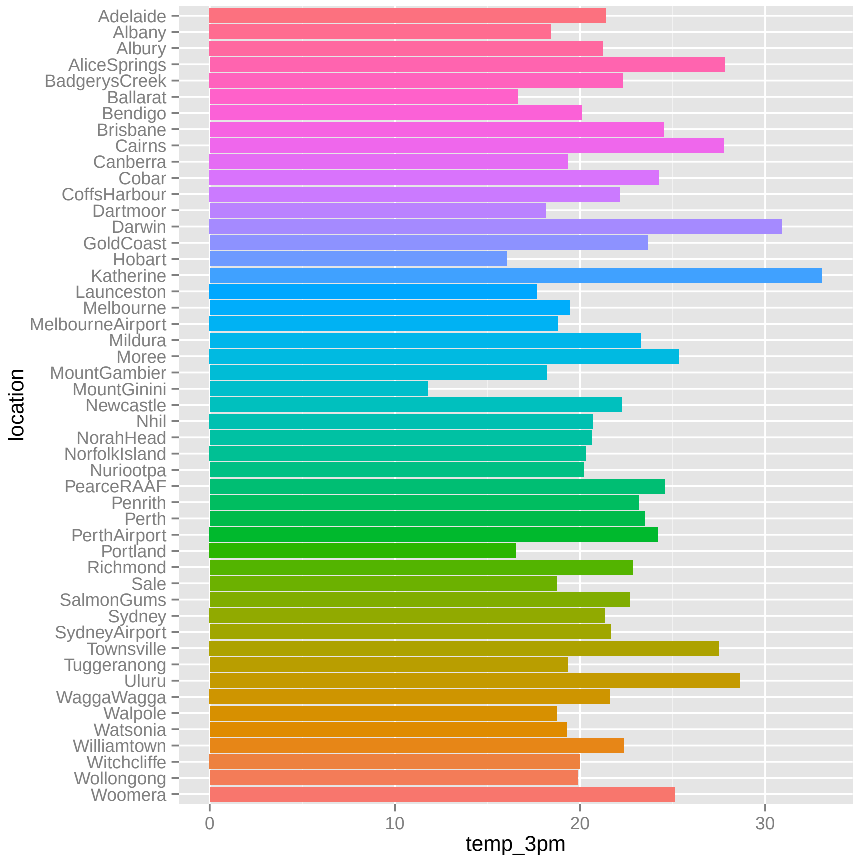
4.13 Horizontal Histogram



Alternatively here we flip the coordinates and produce a horizontal histogram.

```
ds                                                        %>%
  ggplot(aes(location, temp_3pm, fill=location))            +
  stat_summary(fun.y="mean", geom="bar")                    +
  theme(legend.position="none")                             +
  coord_flip()
```

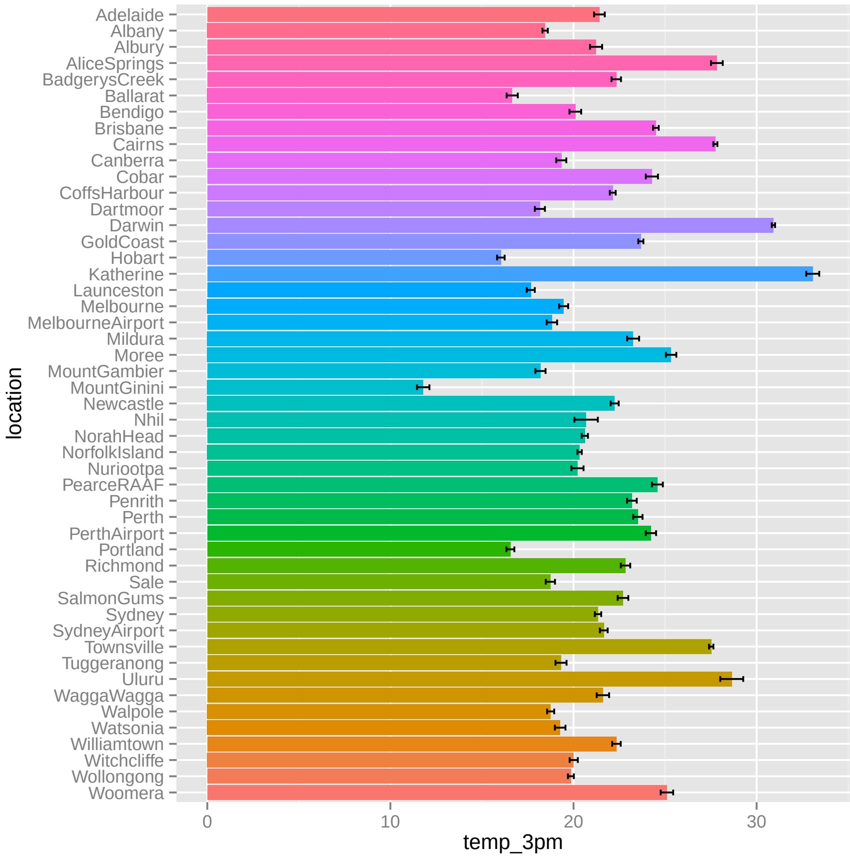
4.14 Reorder the Levels



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.

```
ds                                                                    %>%
  mutate(location=factor(location, levels=rev(levels(location))))    %>%
  ggplot(aes(location, temp_3pm, fill=location))                       +
  stat_summary(fun.y="mean", geom="bar")                               +
  theme(legend.position="none")                                        +
  coord_flip()
```

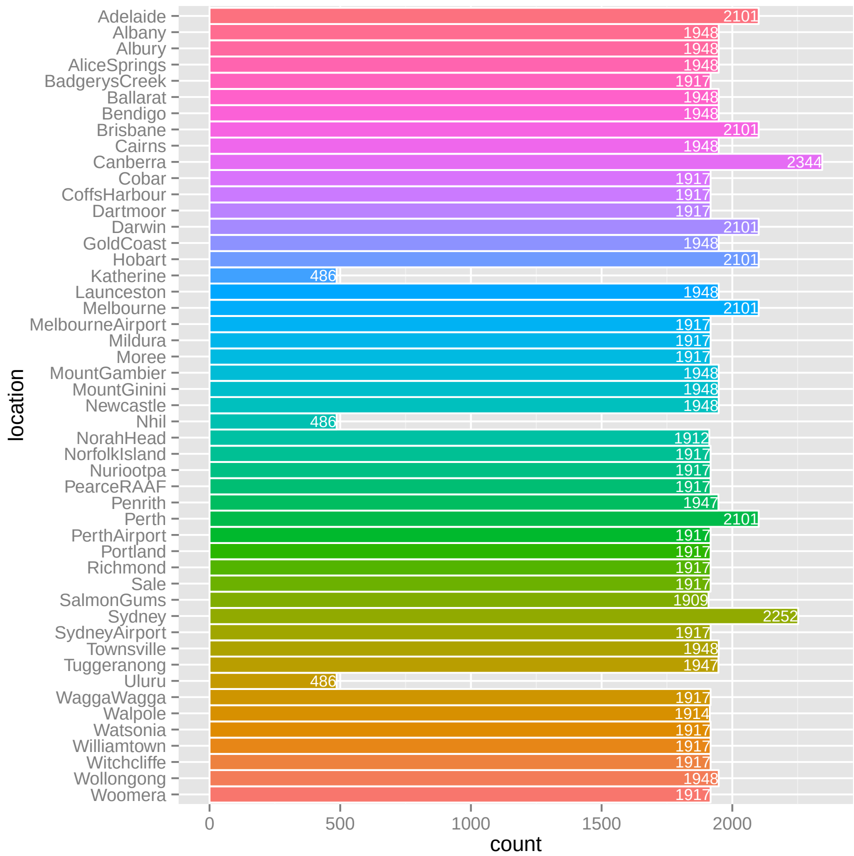
4.15 Plot the Mean with CI



Here we add a confidence interval around the mean.

```
ds                                                               %>%
  mutate(location=factor(location, levels=rev(levels(location))))  %>%
  ggplot(aes(location, temp_3pm, fill=location))                  +
  stat_summary(fun.y="mean", geom="bar")                          +
  stat_summary(fun.data="mean_cl_normal", geom="errorbar",
               conf.int=0.95, width=0.35)                         +
  theme(legend.position="none")                                   +
  coord_flip()
```

4.16 Text Annotations

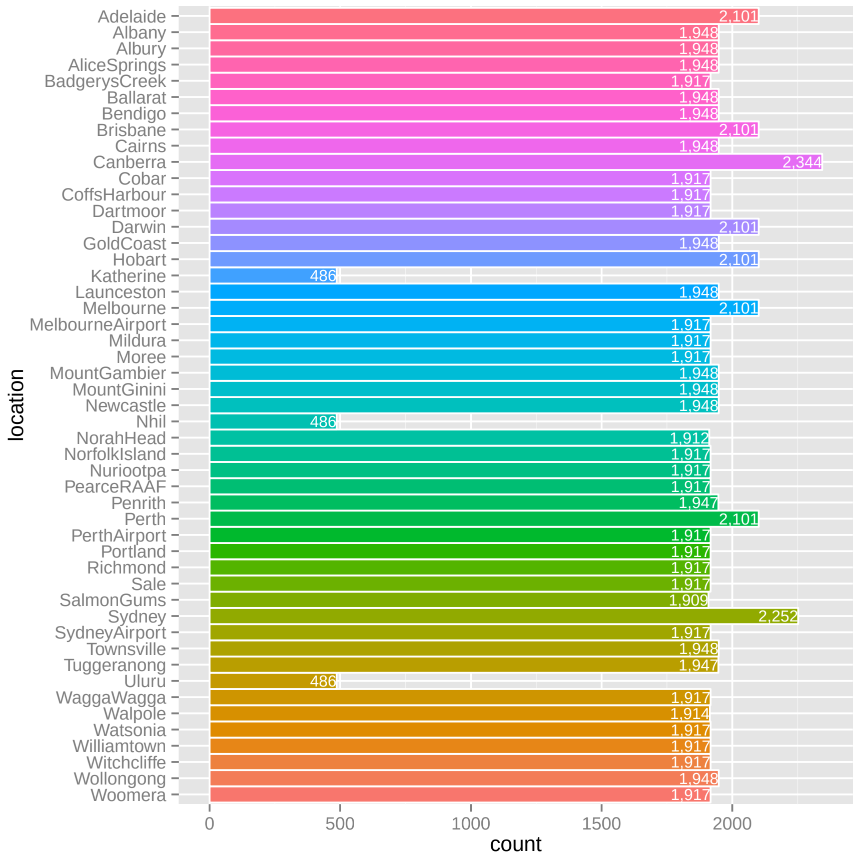


It would be informative to also show the actual numeric values on the plot. This plot shows the counts.

```
ds                                                                    %>%
  mutate(location=factor(location, levels=rev(levels(location))))    %>%
  ggplot(aes(location, fill=location))                                  +
  geom_bar(width=1, colour="white")                                     +
  theme(legend.position="none")                                         +
  coord_flip()                                                          +
  geom_text(stat="bin", color="white", hjust=1.0, size=3,
            aes(y=..count.., label=..count..))
```

Exercise: Instead of plotting the counts, plot the mean `temp_3pm`, and include the textual value.

4.17 Text Annotations with Commas

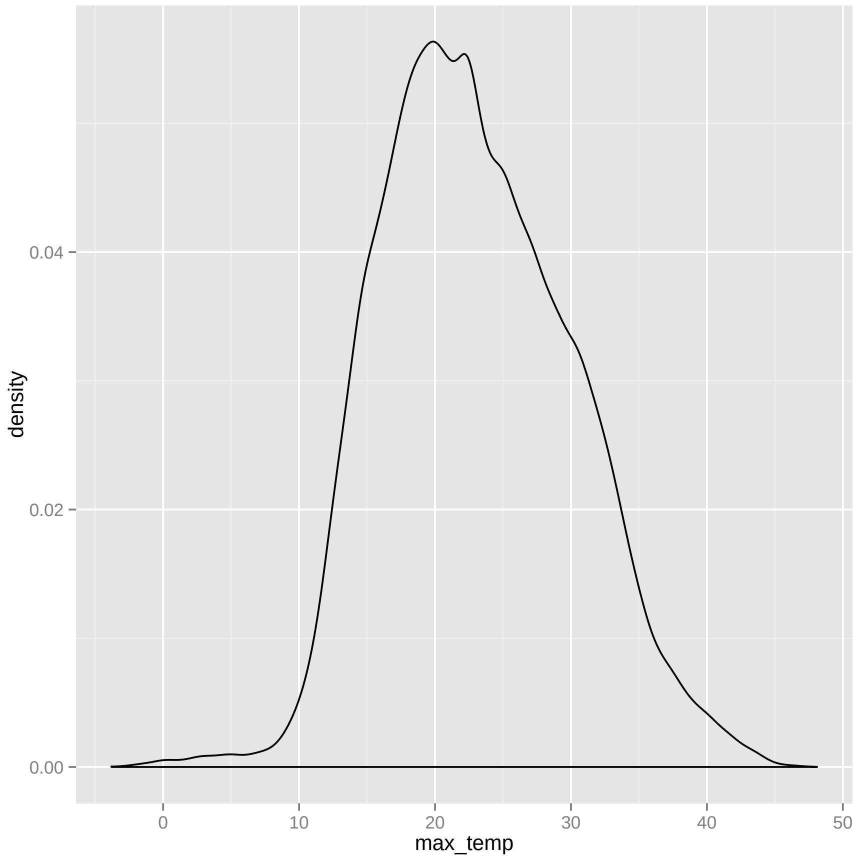


A small variation is to add commas to the numeric annotations to separate the thousands (Will Beasley 121230 email).

```
ds                                                                    %>%
  mutate(location=factor(location, levels=rev(levels(location))))    %>%
  ggplot(aes(location, fill=location))                                 +
  geom_bar(width=1, colour="white")                                    +
  theme(legend.position="none")                                        +
  coord_flip()                                                         +
  geom_text(stat="bin", color="white", hjust=1.0, size=3,
            aes(y=..count.., label=scales::comma(..count..)))
```

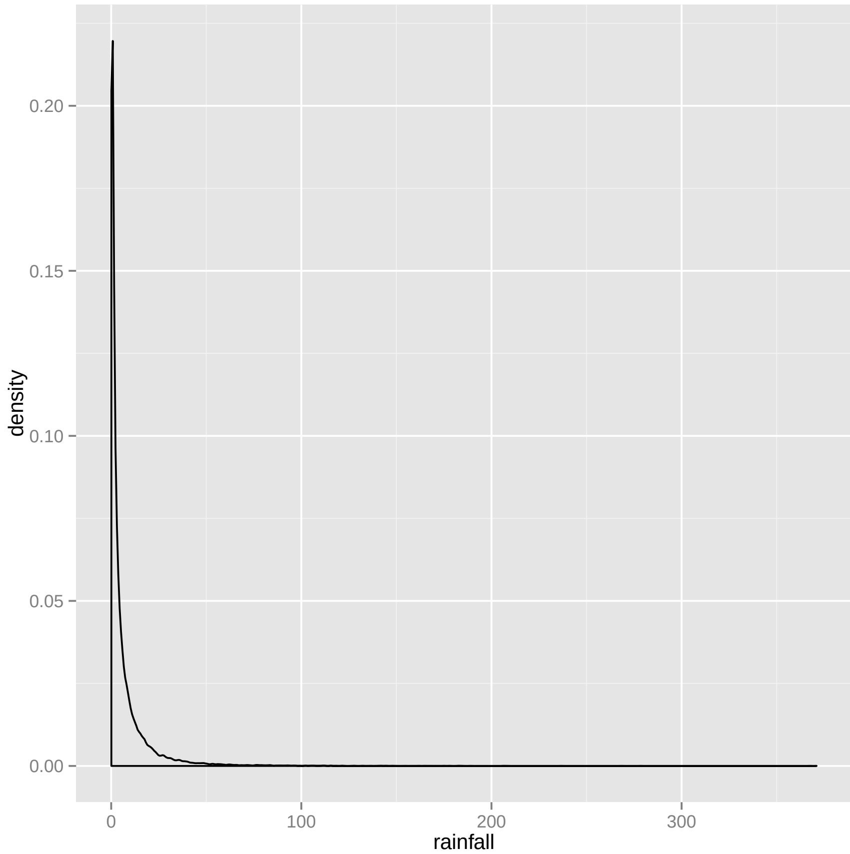
Exercise: Do this without having to use scales::, perhaps using a format.

5 Density Distributions




```
ds                                                          %>%
  ggplot(aes(x=max_temp))                                    +
  geom_density()
```

5.1 Skewed Distributions

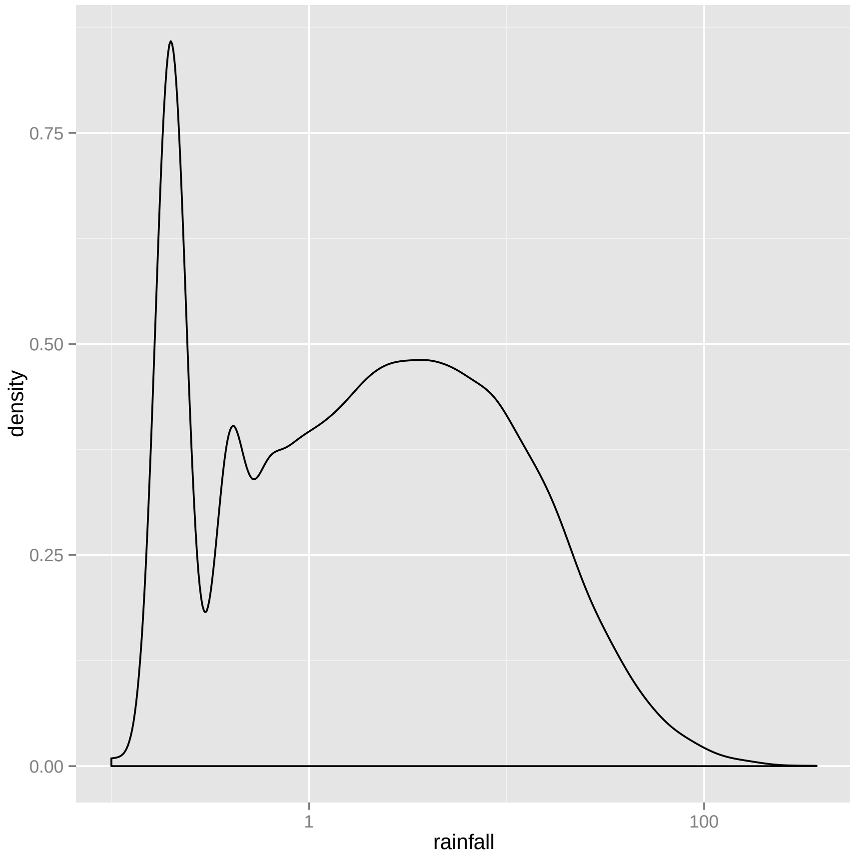


This plot certainly informs us about the skewed nature of the amount of rainfall recorded for any one day, but we lose a lot of resolution at the low end.

Note that we use a subset of the dataset to include only those observations of rainfall (i.e., where the rainfall is non-zero). Otherwise a warning will note many rows contain non-finite values in calculating the density statistic.

```
ds                                                    %>%
  filter(rainfall != 0)                               %>%
  ggplot(aes(x=rainfall))                              +
  geom_density()                                       +
  scale_y_continuous(labels=comma)                     +
  theme(legend.position="none")
```

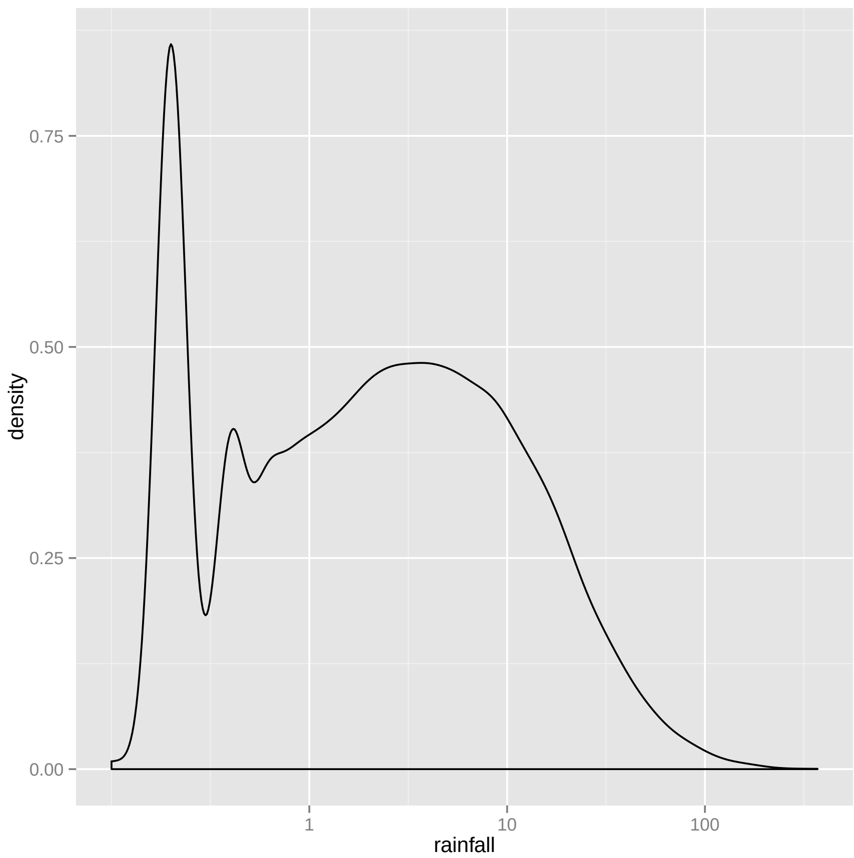
5.2 Log Transform X Axis



A common approach to dealing with the skewed distribution is to transform the scale to be a logarithmic scale, base 10.

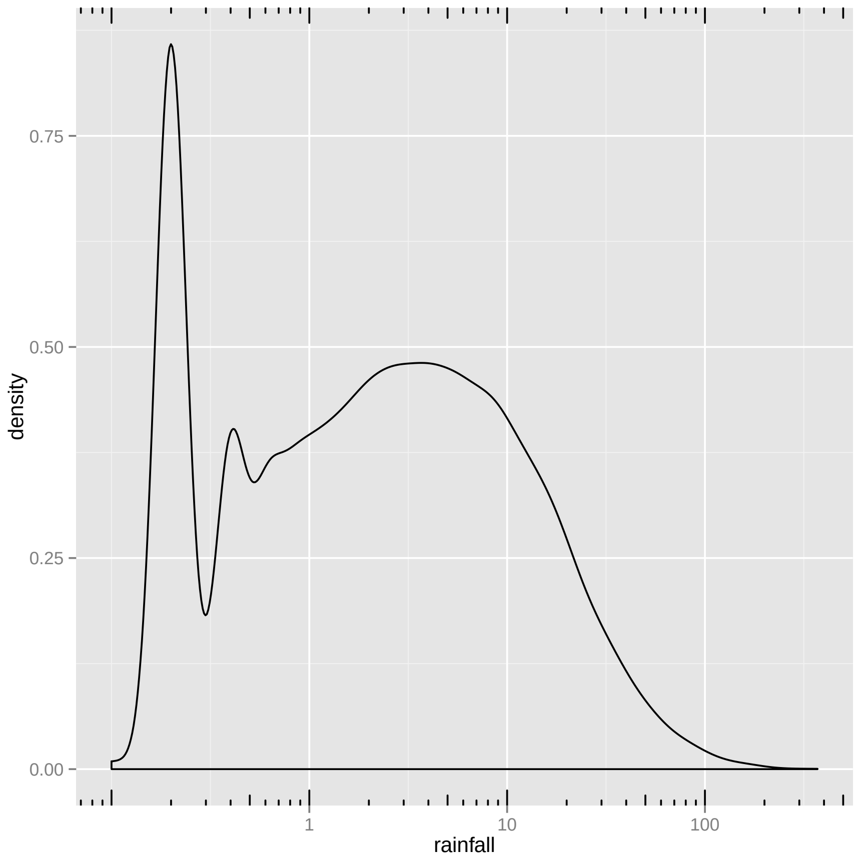
```
ds                                                              %>%
  filter(rainfall != 0)                                         %>%
  ggplot(aes(x=rainfall))                                        +
  geom_density()                                                 +
  scale_x_log10()                                                +
  theme(legend.position="none")
```

5.3 Log Transform X Axis with Breaks



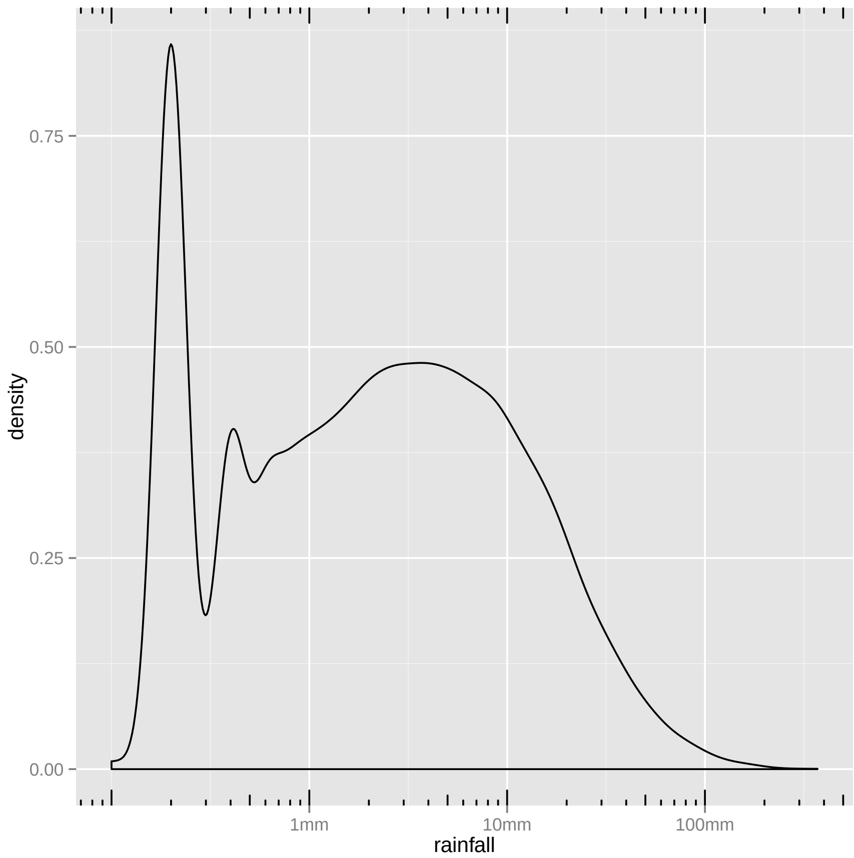

```
ds                                                                    %>%
  filter(rainfall != 0)                                               %>%
  ggplot(aes(x=rainfall))                                              +
  geom_density(binwidth=0.1)                                           +
  scale_x_log10(breaks=c(1, 10, 100), labels=comma)                    +
  theme(legend.position="none")
```

5.4 Log Transform X Axis with Ticks



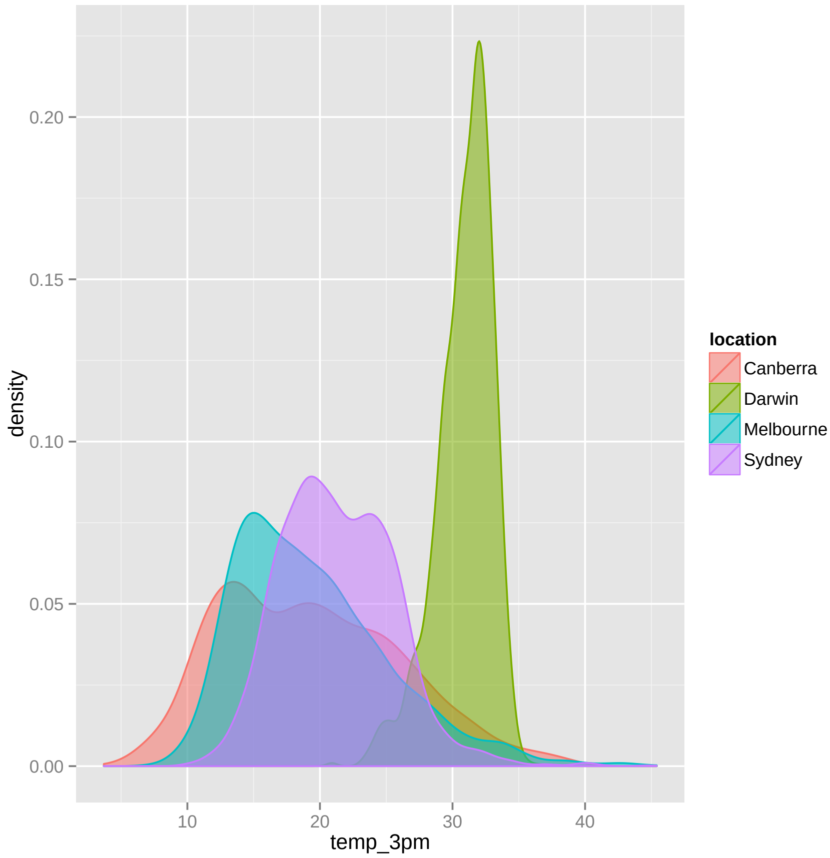

```
ds                                                              %>%
  filter(rainfall != 0)                                         %>%
  ggplot(aes(x=rainfall))                                       +
  geom_density(binwidth=0.1)                                    +
  scale_x_log10(breaks=c(1, 10, 100), labels=comma)             +
  annotation_logticks(sides="bt")                               +
  theme(legend.position="none")
```

5.5 Log Transform X Axis with Custom Label



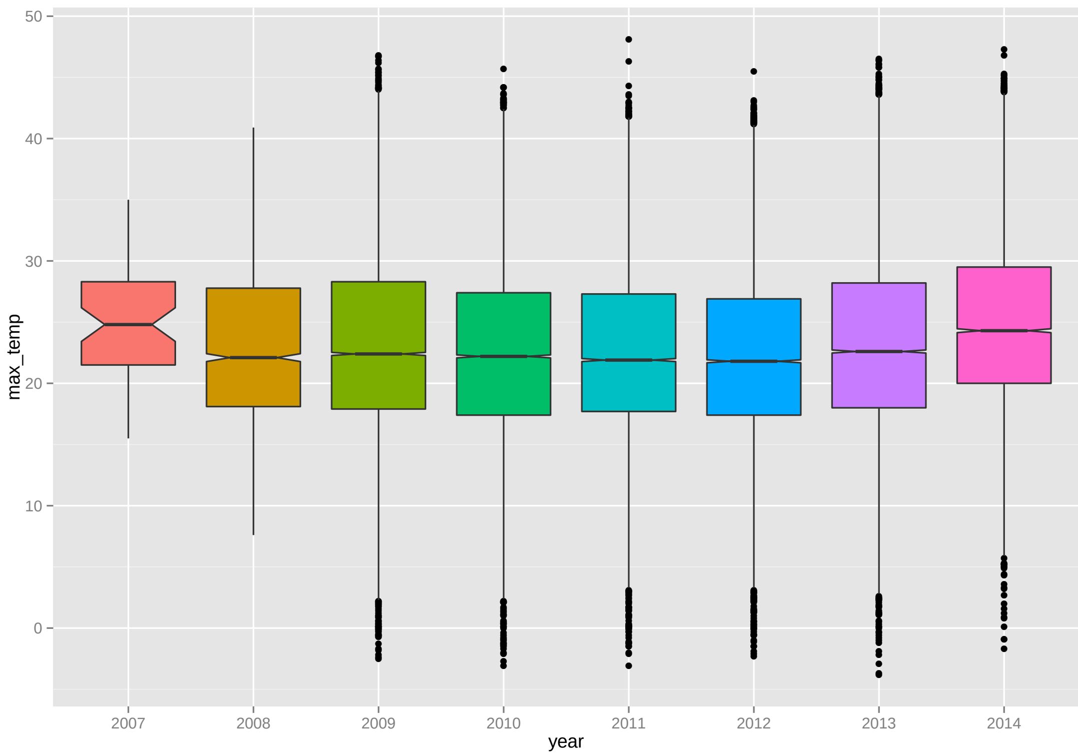

```
mm <- function(x) { sprintf("%smm", x) }

ds                                                          %>%
  filter(rainfall != 0)                                     %>%
  ggplot(aes(x=rainfall))                                   +
  geom_density()                                            +
  scale_x_log10(breaks=c(1, 10, 100), labels=mm)            +
  annotation_logticks(sides="bt")                           +
  theme(legend.position="none")
```

5.6 Transparent Plots




```
cities <- c("Canberra", "Darwin", "Melbourne", "Sydney")

ds                                                                    %>%
  filter(location %in% cities)                                        %>%
  ggplot(aes(temp_3pm, colour = location, fill = location))            +
  geom_density(alpha = 0.55)
```

6 Box Plot Distributions



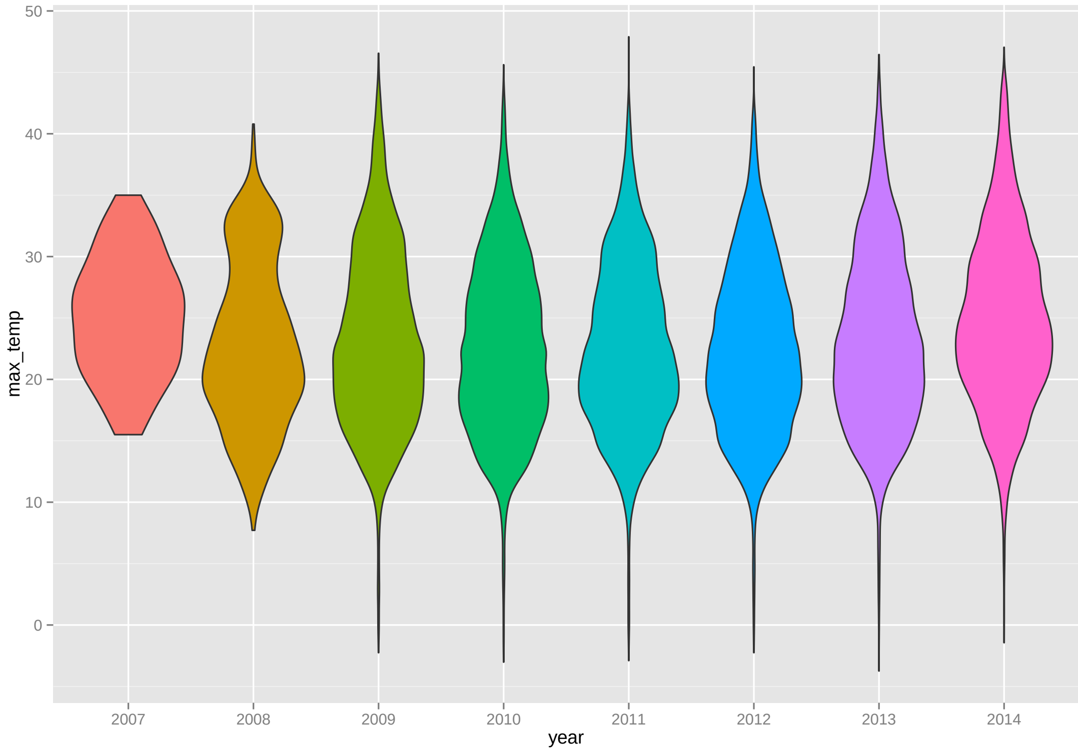
A box plot, also known as a box and whiskers plot, shows the median (the second quartile) within a box which extends to the first and third quartiles. We note that each quartile delimits one quarter of the dataset and hence the box itself contains half the dataset.

Colour is added simply to improve the visual appeal of the plot rather than to convey new information. Since we include `fill=` we also turn off the otherwise included legend.

Here we observe the overall change in the maximum temperature over the years. Notice the first and last plots which probably reflect truncated data, providing motivation to confirm this in the data, before making significant statements regarding these observations.

```
ds                                                          %>%
  mutate(year=factor(format(ds$date, "%Y")))                %>%
  ggplot(aes(x=year, y=max_temp, fill=year))                 +
  geom_boxplot(notch=TRUE)                                   +
  theme(legend.position="none")
```

6.1 Violin Plot

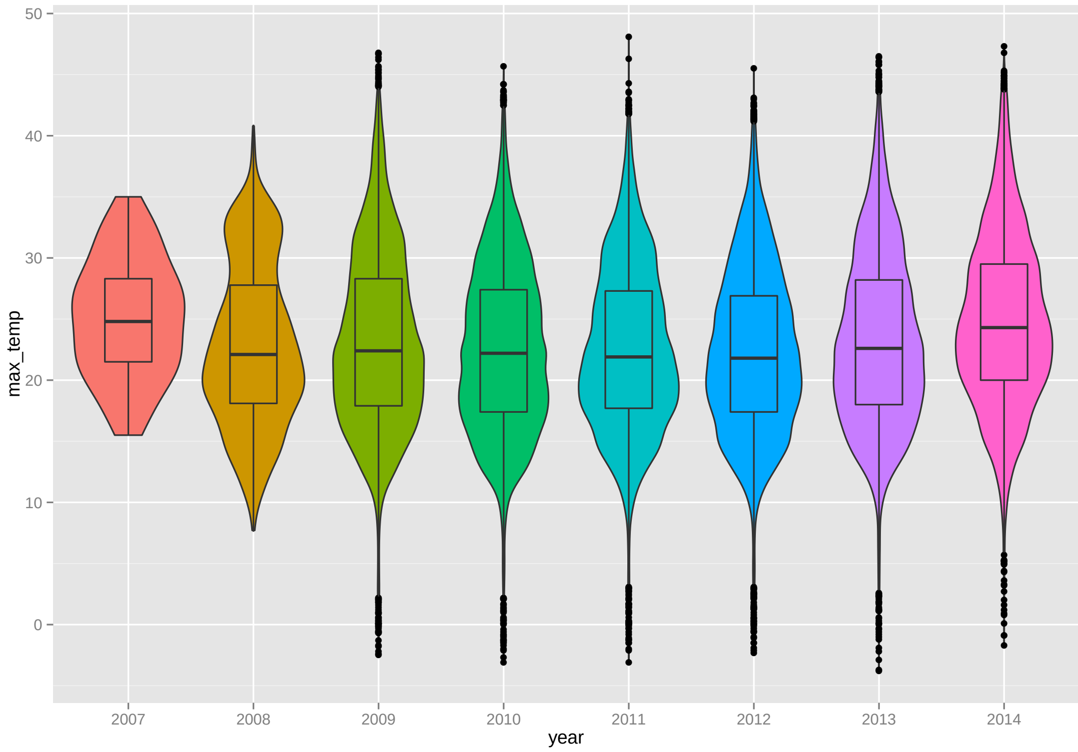


A violin plot is another interesting way to present a distribution, using a shape that resembles a violin.

We again use colour to improve the visual appeal of the plot.

```
ds                                                              %>%
  mutate(year=factor(format(ds$date, "%Y")))                    %>%
  ggplot(aes(x=year, y=max_temp, fill=year))                     +
  geom_violin()                                                  +
  theme(legend.position="none")
```

6.2 Violin with Box Plot



We can overlay the violin plot with a box plot to show the quartiles.

```
ds                                                              %>%
  mutate(year=factor(format(ds$date, "%Y")))                    %>%
  ggplot(aes(x=year, y=max_temp, fill=year))                    +
  geom_violin()                                                 +
  geom_boxplot(width=.5, position=position_dodge(width=0))      +
  theme(legend.position="none")
```

6.3 Violin with Box Plot by Location

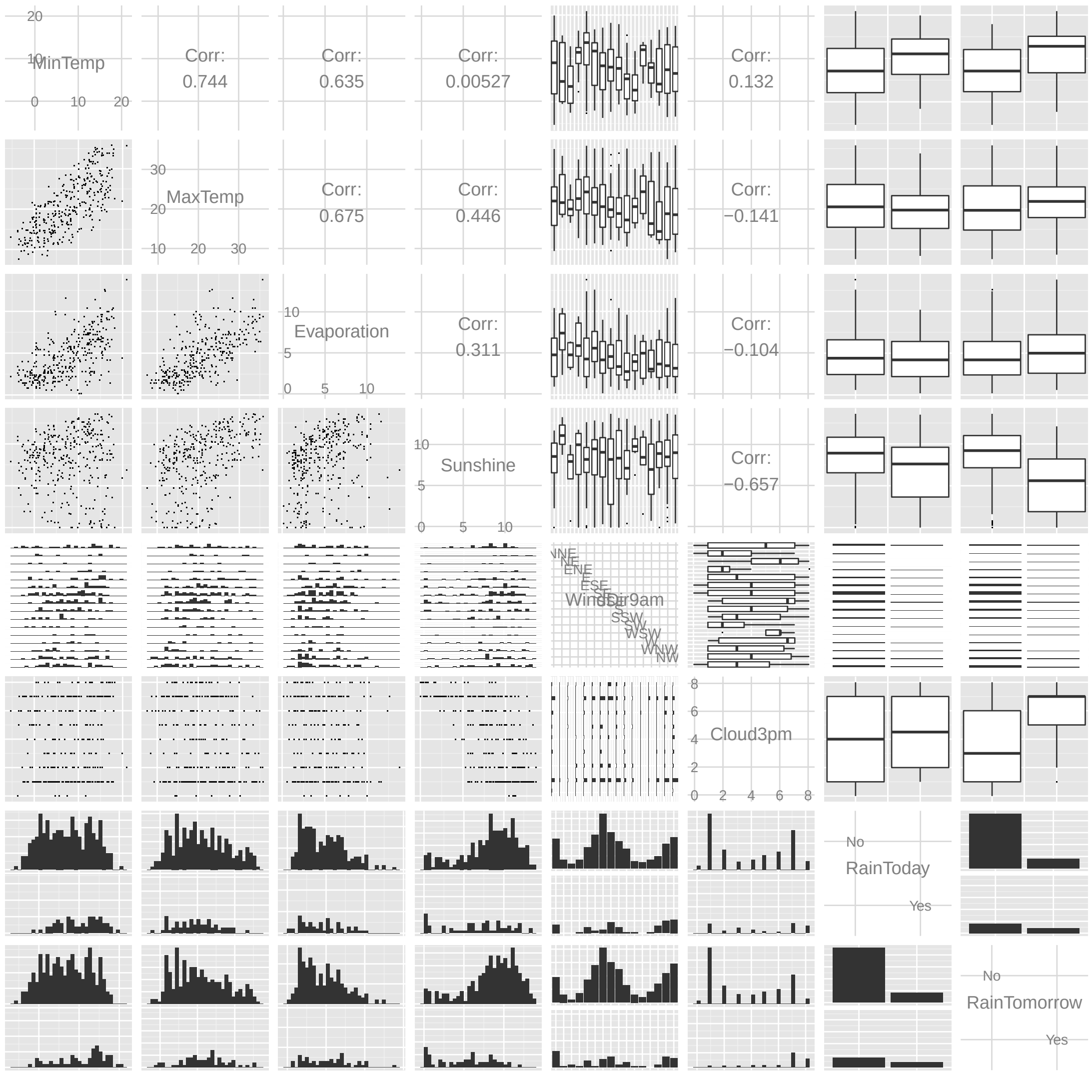
We can readily split the plot across the locations. Things get a little crowded, but we get an overall view across all of the different weather stations. Notice we also rotated the x-axis labels so that they don't overlap.

We can immediately see one of the issues with this dataset, noting that three weather stations have fewer observations that then others.

Various other observations are also interesting. Some locations have little variation in their maximum temperatures over the years.

```
ds                                                                      %>%
  mutate(year=factor(format(ds$date, "%Y")))                            %>%
  ggplot(aes(x=year, y=max_temp, fill=year))                             +
  geom_violin()                                                          +
  geom_boxplot(width=.5, position=position_dodge(width=0))               +
  theme(legend.position="none",
        axis.text.x=element_text(angle=45, hjust=1))                     +
  facet_wrap(~location)
```

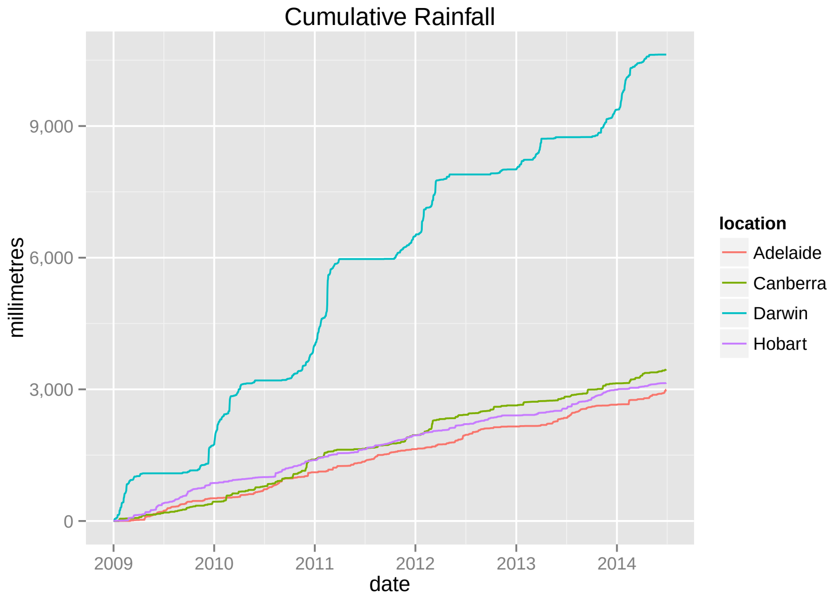
7 Pairs Plot: Using ggpairs()



A sophisticated pairs plot, as delivered by **ggpairs()** from GGally (Schloerke *et al.*, 2014), brings together many of the kinds of plots we have already seen to compare pairs of variables. Here we have used it to plot the **weather** dataset from rattle (Williams, 2014). We see scatter plotes, histograms, and box plots in the pairs plot.

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

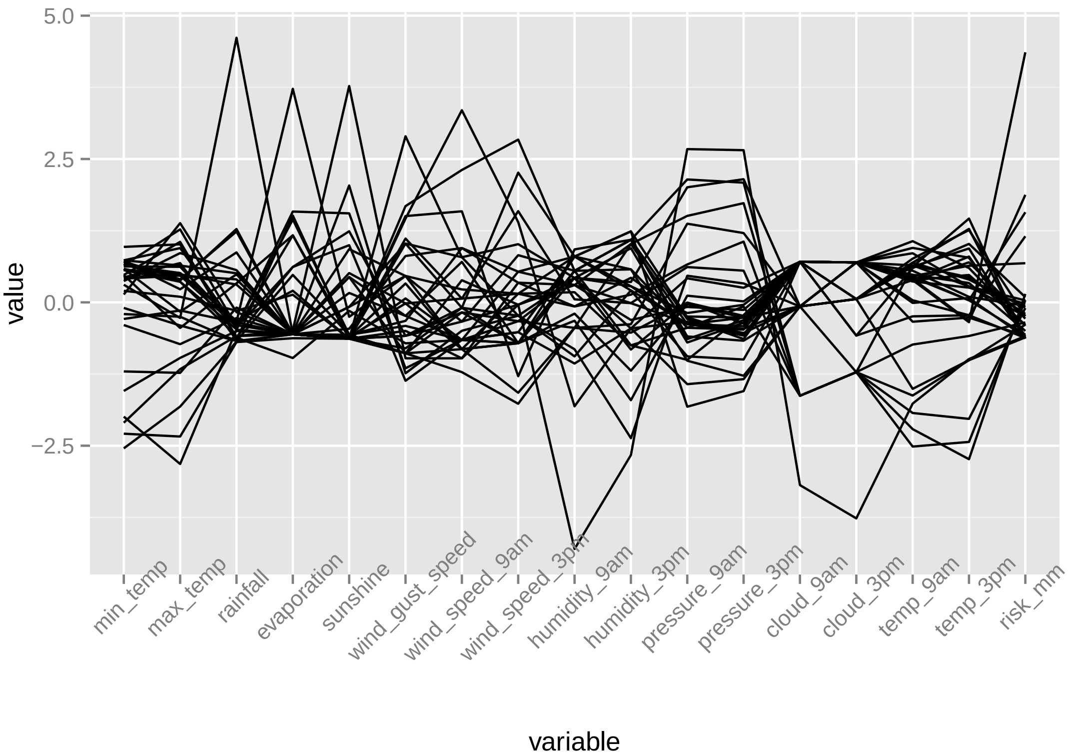
8 Cumulative Distribution Plot



Here we show the cumulative sum of a variable for different locations.

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

ds                                                                       %>%
  filter(location %in% cities & date >= "2009-01-01")                    %>%
  group_by(location)                                                     %>%
  mutate(cumRainfall=order_by(date, cumsum(rainfall)))                   %>%
  ggplot(aes(x=date, y=cumRainfall, colour=location))                     +
  geom_line()                                                             +
  ylab("millimetres")                                                     +
  scale_y_continuous(labels=comma)                                        +
  ggtitle("Cumulative Rainfall")
```

9 Parallel Coordinates Plot



A parallel coordinates plot provides a graphical summary of multivariate data. Generally it works best with 10 or fewer variables. The variables will be scaled to a common range (e.g., by performing a z-score transform which subtracts the mean and divides by the standard deviation) and each observation is plot as a line running horizontally across the plot.

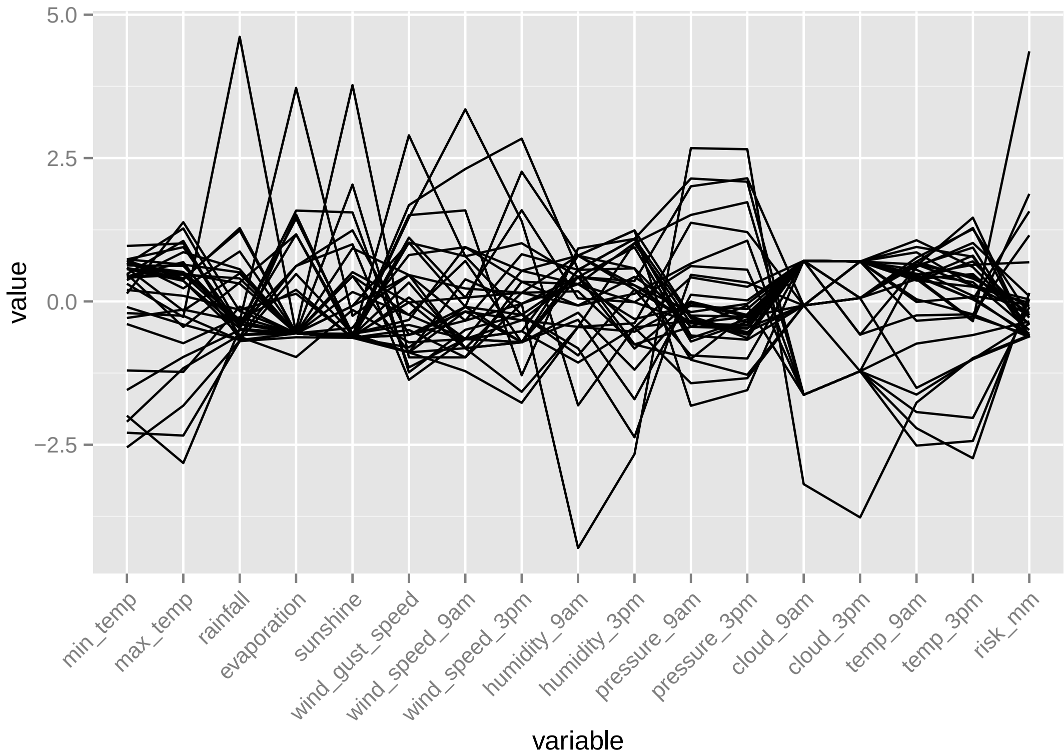
One of the main uses of parallel coordinate plots is to identify common groups of observations through their common patterns of variable values. We can use parallel coordinate plots to identify patterns of systematic relationships between variables over the observations.

For more than about 10 variables consider the use of heatmaps or fluctuation plots.

Here we use `ggparcoord()` from GGally (Schloerke *et al.*, 2014) to generate the parallel coordinates plot. The underlying plotting is performed by ggplot2 (Wickham and Chang, 2014).

```
library(GGally)

cities <- c("Canberra", "Darwin", "Melbourne", "Sydney")

ds                                                          %>%
  filter(location %in% cities & rainfall>75)                %>%
  ggparcoord(columns=numi)                                   +
  theme(axis.text.x=element_text(angle=45))
```

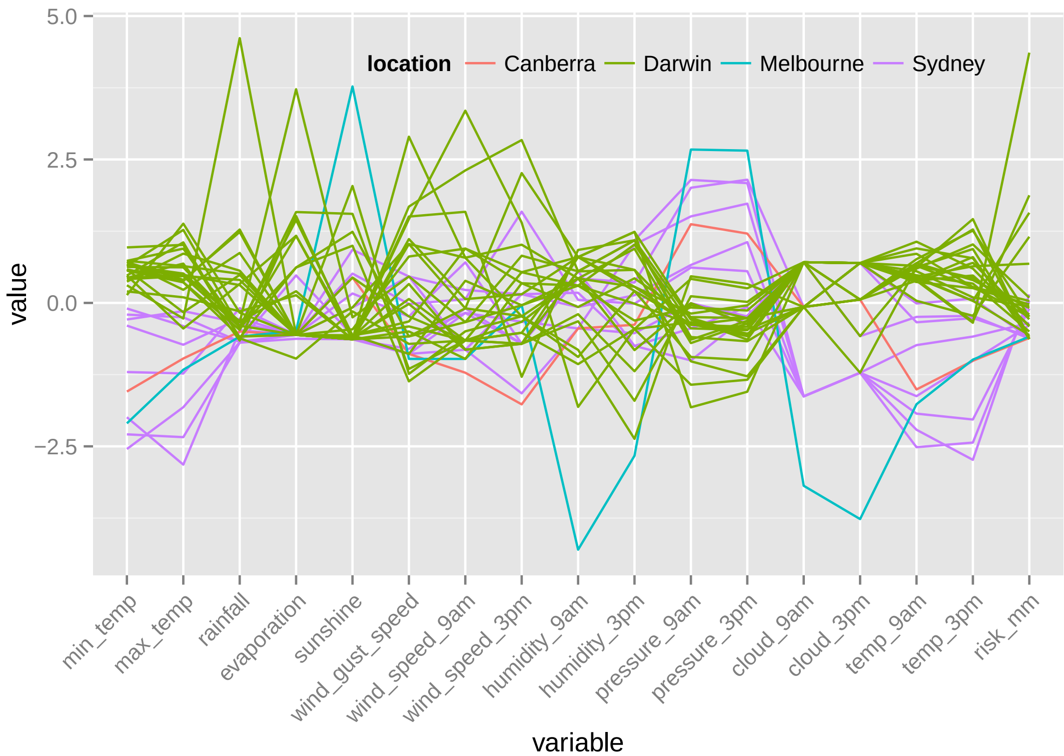
9.1 Labels Aligned



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`.

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

9.2 Colour and Label



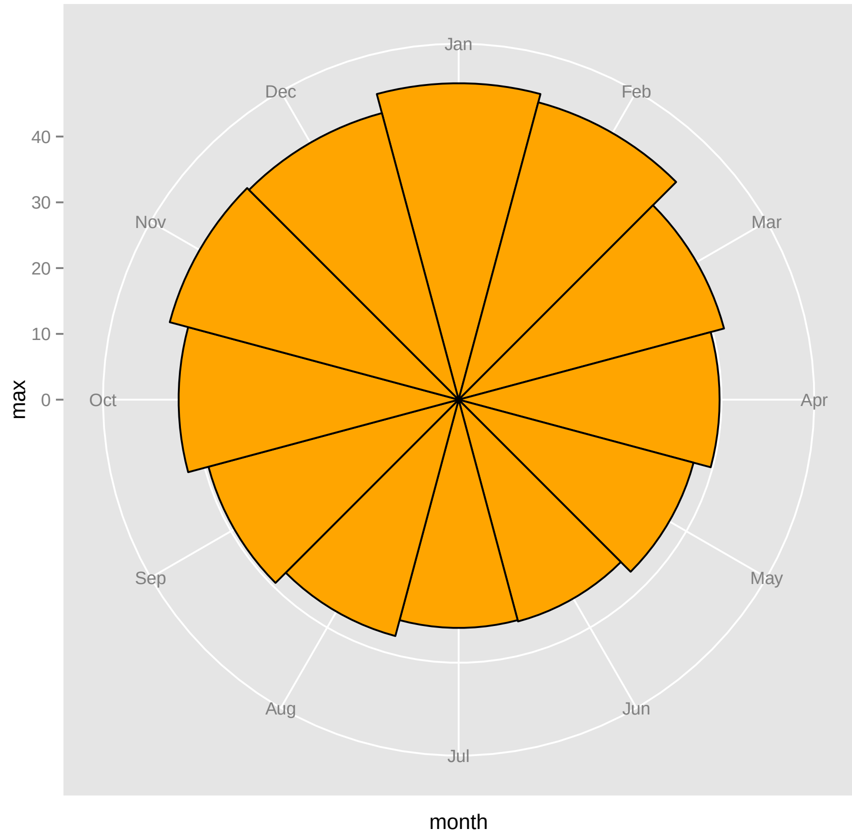
Here we add some colour and force the legend to be within the plot rather than, by default, reducing the plot size and adding the legend to the right.

We can discern just a little structure relating to locations. We have limited the data to those days where more than 74mm of rain is recorded and clearly Darwin becomes 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. Apart from this the parallel coordinates in this instance is not showing much structure.

The example code illustrates several aspects of placing the legend. We specified a specific location within the plot itself, justified as top left, laid out horizontally, and transparent.

```
ds                                                                  %>%
  filter(location %in% cities & rainfall>75)                        %>%
  ggparcoord(columns=numi, group="location")                         +
  theme(axis.text.x=element_text(angle=45, hjust=1),
        legend.position=c(0.25, 1),
        legend.justification=c(0.0, 1.0),
        legend.direction="horizontal",
        legend.background=element_rect(fill="transparent"),
        legend.key=element_rect(fill="transparent",
            colour="transparent"))
```

10 Polar Coordinates



Here we use a polar coordinate to plot what is otherwise a bar chart. The data preparation begins with converting the date to a month, then grouping by the month, after which we obtain the maximum *max_temp* for each group. The months are then ordered correctly. We then generate a bar plot and place the plot on a polar coordinate. Not necessarily particularly easy to read, and probably better to not use polar coordinates in this case!

```
ds                                                                    %>%
  mutate(month=format(date, "%b"))                                    %>%
  group_by(month)                                                     %>%
  summarise(max=max(max_temp))                                        %>%
  mutate(month=factor(month, levels=month.abb))                       %>%
  ggplot(aes(x=month, y=max, group=1))                                 +
  geom_bar(width=1, stat="identity", fill="orange", color="black")     +
  coord_polar(theta="x", start=-pi/12)
```

11 Multiple Plots: Using grid.arrange()

12 Multiple Plots: Using grid.arrange()

13 Multiple Plots: Alternative Arrangements

14 Multiple Plots: Sharing a Legend

15 Multiple Plots: 2D Histogram

16 Multiple Plots: 2D Histogram Plot

17 Multiple Plots: Using layOut()

17.1 Arranging Plots with layOut()

17.2 Arranging Plots Through Viewports

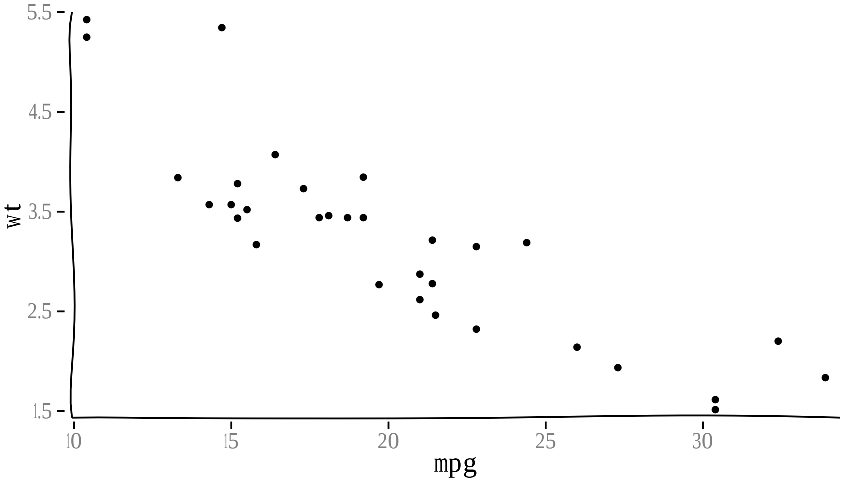
17.3 Plot

18 Plotting a Table

19 Plotting a Table and Text

20 Interactive Plot Building

21 Having Some Fun with xkcd



For a bit of fun we can generate plots in the style of the popular online comic strip *xkcd* using xkcd (Manzanera, 2014). 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()
```

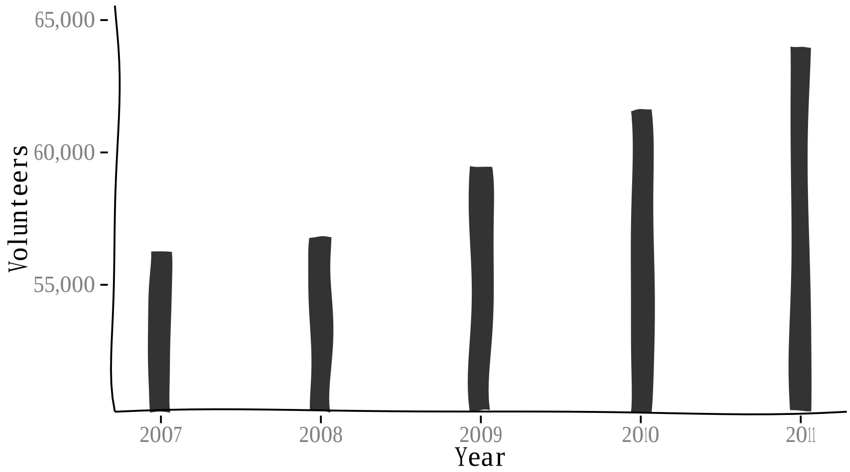
The installation can be uninstalled with:

```
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)
mtcars                                                              %>%
  ggplot(aes(mpg, wt))                                               +
  geom_point()                                                       +
  xkcdaxis(xrange, yrange)
```

21.1 Bar Chart

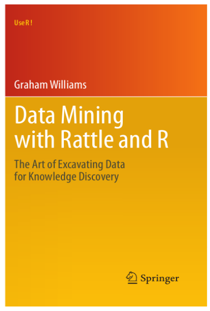


We can do a bar chart in the same style.

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

ds <- dsname                                                          %>%
  get()                                                               %>%
  mutate(xmin=year-0.1, xmax=year+0.1, ymin=50000, ymax=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)

ggplot()                                                               +
  xkcdrect(mapping, ds)                                                +
  xkcdaxis(xrange, yrange)                                             +
  xlab("Year")                                                         +
  ylab("Volunteers")                                                   +
  scale_y_continuous(labels=comma)
```

22 Further Reading and Acknowledgements

The Rattle Book, published by Springer, provides a comprehensive introduction to 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 chapter is one of many chapters available from http://HandsOnDataScience.com. In particular follow the links on the website with a * which indicates the generally more developed chapters.

Other resources include:

- The GGPlot2 documentation is quite extensive and useful
- The R Cookbook is a great resource explaining how to do many types of plots using ggplot2.

23 References

Manzanera ET (2014). *xkcd: Plotting ggplot2 graphics in a XKCD style.* R package version 0.0.3, URL http://CRAN.R-project.org/package=xkcd.

R Core Team (2014). *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 (2014). *scales: Scale functions for graphics.* R package version 0.2.4, URL http://CRAN.R-project.org/package=scales.

Wickham H, Chang W (2014). *ggplot2: An implementation of the Grammar of Graphics.* R package version 1.0.0, 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 3.1.4, URL http://rattle.togaware.com/.

This document, sourced from GGPlot2O.Rnw revision 486, was processed by KnitR version 1.6 of 2014-05-24 and took 86 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04.1 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-08-16 21:13:30.