# 4

# *Visualising Data*

One of the most important tasks for any data scientist is to visualise data. Presenting data visually will often lead to new insights and discoveries, as well as providing clear evidence of any issues with the data. A visual presentation is also often the most effective means for communicating insight to the business problem owners.

R offers a comprehensive suite of tools to visualise data, with ggplot2 (Wickham and Chang, 2016) being dominate amongst them. The ggplot2 package implements a grammar for writing sentences describing the graphics. Using this package we construct a plot beginning with the dataset and the aesthetics (e.g., x-axis and y-axis) and then add geometric elements, statistical operations, scales, facets, coordinates, and numerous other components. In this chapter we explore data using ggplot2 to gain new insights into our data. The package provides an extensive collection of capabilities offering an infinite variety of visual possibilities. We will present some basics as a launching pad for plotting data but note that further opportunities abound and are well covered in many other resources and online.

Packages used in this chapter include GGally (Schloerke *et al.*, 2016), RColorBrewer (Neuwirth, 2014), dplyr (Wickham *et al.*, 2017a), ggplot2 (Wickham and Chang, 2016), gridExtra (Auguie, 2016), lubridate (Grolemund *et al.*, 2016), magrittr (Bache and Wickham, 2014), randomForest (Breiman *et al.*, 2015), rattle (Williams, 2017), scales (Wickham, 2016), stringi (Gagolewski *et al.*, 2017), and stringr (Wickham, 2017a).

```
# Load required packages from local library into the R session.

library(GGally)       # Parallel coordinates.
library(RColorBrewer) # Choose different colors.
library(dplyr)        # Data wrangling.
```

```
library(ggplot2)      # Visualise data.
library(gridExtra)    # Layout multiple plots.
library(lubridate)    # Dates and time.
library(magrittr)     # Pipelines for data processing.
library(randomForest) # Deal with missing data.
library(rattle)       # weatherAUS dataset and normVarNames().
library(scales)       # Include commas in numbers.
library(stringi)      # String concat operator %s+%.
library(stringr)      # Strings: str_replace().
```

## 4.1 Preparing the Dataset

We use the modestly large **weatherAUS** dataset from rattle (Williams, 2017) to illustrate the capabilities of ggplot2. For plots that generate large images we might use random subsets of the same dataset to allow replication in a timely manner. We begin by loading the dataset saved through the template we developed in Chapter 3.

```
# Build the filename used to previously store the data.

fpath  <- "data"
dsname <- "weatherAUS"
dsdate <- "_20170702"
dsfile <- dsname %s+% dsdate %s+% ".RData"

fpath %>%
  file.path(dsfile) %>%
  print() ->
dsrdata

## [1] "data/weatherAUS_20170702.RData"

# Load the R objects from file and list them.

load(dsrdata) %>% print()

##  [1] "ds"        "dsname"    "dspath"    "dsdate"
##  [5] "nobs"      "vars"      "target"    "risk"
```

```
##  [9] "id"        "ignore"    "omit"      "inputi"
## [13] "inputs"    "numi"      "numc"      "cati"
## [17] "catc"      "form"      "seed"      "train"
## [21] "validate"  "test"      "tr_target" "tr_risk"
## [25] "va_target" "va_risk"   "te_target" "te_risk"
```

We will perform missing value imputation but note that this is not something we should be doing lightly (inventing new data). We do so here simply to avoid warnings that would otherwise advise us of missing data when using ggplot2. Note the use of randomForest::**na.roughfix()** to perform missing value imputation as discussed in Chapter 3. Alternatively we could remove observations with missing values using stats::**na.omit()**.

```
# Count the number of missing values.

ds[vars] %>% is.na() %>% sum() %>% comcat()

## 278,293

# Impute missing values.

ds[vars] %<>% na.roughfix()

# Confirm that no missing values remain.

ds[vars] %>% is.na() %>% sum() %>% comcat()

## 0
```

We are now in a position to visually explore our dataset. We begin with a textual dplyr:: **glimpse()** of the dataset for reference.

```
glimpse(ds)

## Observations: 134,776
## Variables: 27
## $ date            <date> 2008-12-01, 2008-12-02, 2008-12-0...
## $ location        <fctr> albury, albury, albury, albury, a...
## $ min_temp        <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, ...
## $ max_temp        <dbl> 22.9, 25.1, 25.7, 28.0, 32.3, 29.7...
```

```
## $ rainfall        <dbl> 0.6, 0.0, 0.0, 0.0, 1.0, 0.2, 0.0,...
## $ evaporation     <dbl> 4.8, 4.8, 4.8, 4.8, 4.8, 4.8, 4.8,...
## $ sunshine        <dbl> 8.5, 8.5, 8.5, 8.5, 8.5, 8.5, 8.5,...
## $ wind_gust_dir   <ord> w, wnw, wsw, ne, w, wnw, w, w, nnw...
## $ wind_gust_speed <dbl> 44, 44, 46, 24, 41, 56, 50, 35, 80...
## $ wind_dir_9am    <ord> w, nnw, w, se, ene, w, sw, sse, se...
## $ wind_dir_3pm    <ord> wnw, wsw, wsw, e, nw, w, w, w, nw,...
## $ wind_speed_9am  <int> 20, 4, 19, 11, 7, 19, 20, 6, 7, 15...
## $ wind_speed_3pm  <dbl> 24, 22, 26, 9, 20, 24, 24, 17, 28,...
## $ humidity_9am    <int> 71, 44, 38, 45, 82, 55, 49, 48, 42...
## $ humidity_3pm    <int> 22, 25, 30, 16, 33, 23, 19, 19, 9,...
## $ pressure_9am    <dbl> 1007.7, 1010.6, 1007.6, 1017.6, 10...
## $ pressure_3pm    <dbl> 1007.1, 1007.8, 1008.7, 1012.8, 10...
## $ cloud_9am       <dbl> 8, 5, 5, 5, 7, 5, 1, 5, 5, 5, 5, 8...
## $ cloud_3pm       <dbl> 5, 5, 2, 5, 8, 5, 5, 5, 5, 5, 5, 8...
## $ temp_9am        <dbl> 16.9, 17.2, 21.0, 18.1, 17.8, 20.6...
## $ temp_3pm        <dbl> 21.8, 24.3, 23.2, 26.5, 29.7, 28.9...
## $ rain_today      <fctr> no, no, no, no, no, no, no, no, n...
## $ risk_mm         <dbl> 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0.0,...
## $ rain_tomorrow   <fctr> no, no, no, no, no, no, no, no, y...
## $ year            <fctr> 2008, 2008, 2008, 2008, 2008, 200...
## $ season          <fctr> summer, summer, summer, summer, s...
## $ cluster         <fctr> area2, area2, area2, area2, area2...
```

## 4.2 Scatter Plot

Our first plot is a simple scatter plot which displays points scattered over a plot. A difficulty with scatter plots (and indeed with many types of plots) is that for large datasets we end up with rather dense plots. For illustrative purposes we will identify a random subset of just 1,000 observations to plot. We thus avoid filling the plot completely with points as might otherwise happen as in Figure 2.3.

```
ds %>%
  nrow() %>%
  sample(1000) ->
sobs
```

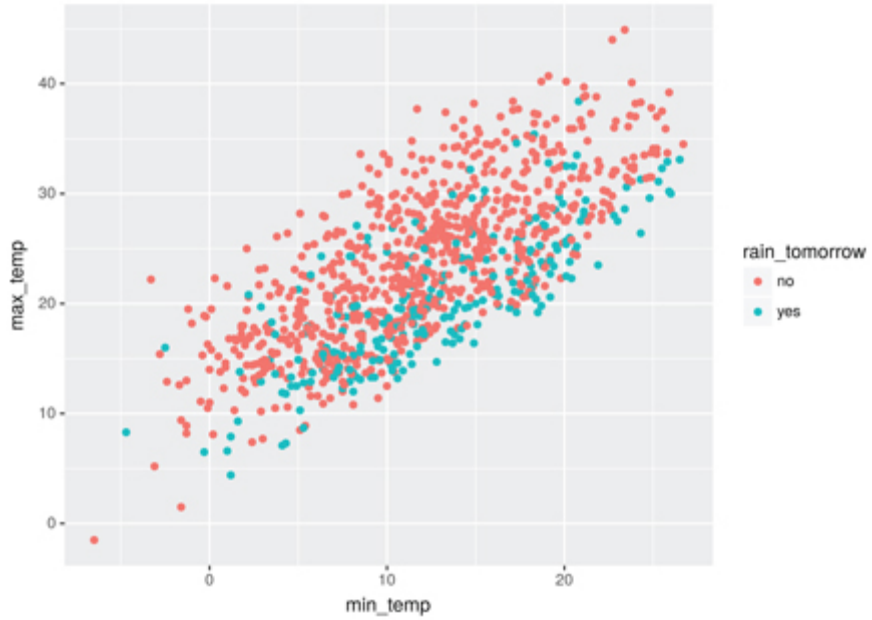


**Figure 4.1** *Scatter plot of the weatherAUS dataset.*

We are now ready to generate the plot. We choose just the random sample of rows whose indices are stored in the variable sobs. This subset is piped through to ggplot2::**ggplot()** which initialises the plot. To the plot we add points using ggplot2::**geom_point()**. The resulting plot is displayed in Figure 4.1.

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

The call to ggplot2::**ggplot()** includes as its argument the aesthetics of the plot. We identify for the x= axis the variable min_temp and for the y= axis the variable max_temp as the y- axis. In addition to minimally identifying the x and y mapping we add a colour= option to distinguish between those days where rain_tomorrow is true from those where it is false.

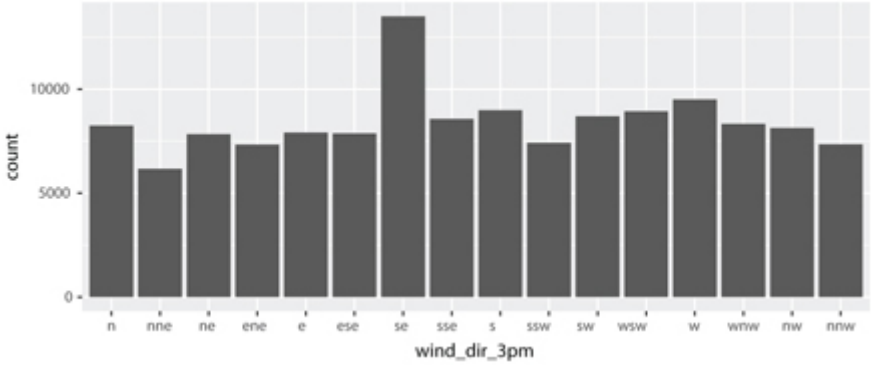


**Figure 4.2** *Bar chart showing the relative occurrence of different wind directions as recorded at 3pm.*

Having set up the aesthetics of the plot we can add a graphical layer. The simplest is just to plot the points (*x*, *y*) coloured appropriately. All that is required is to add (i.e., "+") a call to ggplot2:: **geom_point()**.

## 4.3 Bar Chart

Another common plot is the bar chart which displays the count of observations using bars. Such plots are generated by ggplot2 using ggplot2::**geom_bar()**. Figure 4.2 is generated with:

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

Here we only require an x-axis to be specified which in our example is wind_dir_3pm. To the base plot created by ggplot2::**ggplot()** we add a so-called bar geometric to construct the required plot. The resulting plot shows the frequency of the levels of the categoric variable wind_dir_3pm across the whole dataset.

## 4.4 Saving Plots to File

Generating a plot is one thing but we will want to make use of the plot possibly in multiple ways. Once we have a plot displayed we can save the plot to file quite simply using ggplot2::**ggsave()**. The format is determined automatically by the name of the file to which we save the plot. Here, for example, we save the plot as a PDF that we might include in other documents or share with a colleague for discussion.

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

Notice the use of width= and height=. The default values are those of the current plotting window so for saving the plot we have specified a particular width and height. By trial and error or by experience we have found the proportions used here to suit our requirements.

There is some art required in choosing a good width and height as we discuss in . 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.

## 4.5 Adding Spice to the Bar Chart

A bar chart can be enhanced in many ways to demonstrate different characteristics of the data. A stacked bar chart is commonly used to identify the distribution of the observations over another

variable, like the target variable. Our target is rain_tomorrow and we obtain a stacked bar chart by filling the bars with colour based on this variable using fill=. This is implemented as follows with the result displayed in Figure 4.3.

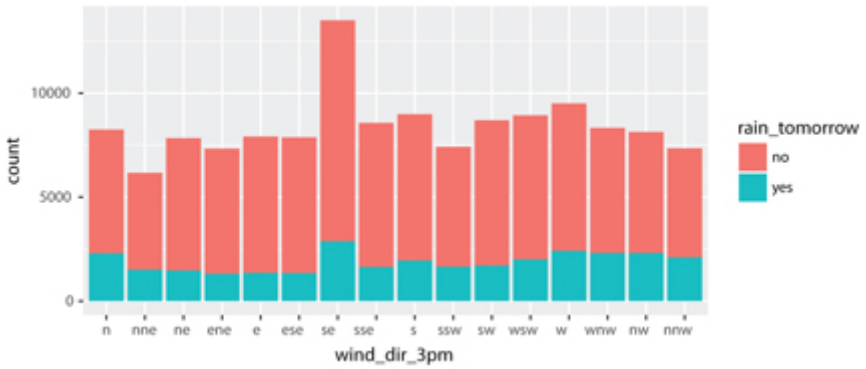


**Figure 4.3** *Stacked bar chart.*

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

There are many options available to tune how we present our plots using ggplot2. In a similar way to building pipelines of functions to achieve our data processing as we saw in Chapters 2 and 3, we build our plots incrementally. We will illustrate a number of options in the following codes as we build a more interesting presentation of the data as in Figure 4.4. Be sure to replicate the plot by adding one line of the following code at a time and studying its impact before moving on to the next line/option/layer. We detail some of the options below.

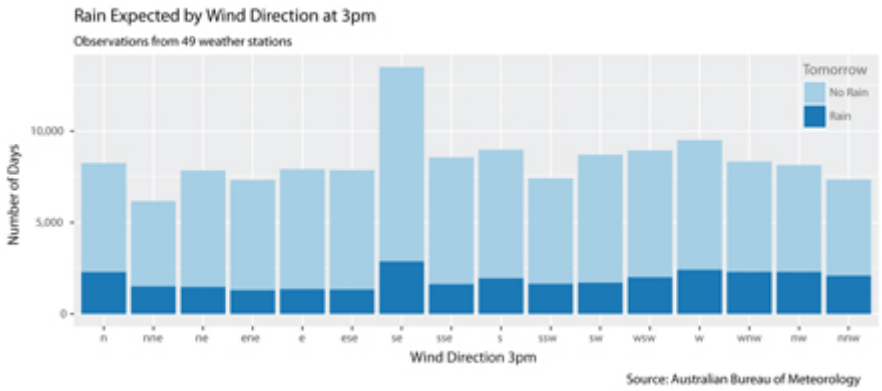


**Figure 4.4** *A decorated stacked bar chart.*

```
blues2 <- brewer.pal(4, "Paired")[1:2] %T>% print()

## [1] "#A6CEE3" "#1F78B4"

ds$location %>%
  unique() %>%
  length() %T>%
  print() ->
num_locations

## [1] 49
```

```
ds %>%
  ggplot(aes(x=wind_dir_3pm, fill=rain_tomorrow)) +
  geom_bar() +
  scale_fill_manual(values = blues2,
                    labels = c("No Rain", "Rain")) +
  scale_y_continuous(labels=comma) +
  theme(legend.position   = c(.95, .85),
        legend.title      = element_text(colour="grey40"),
        legend.text       = element_text(colour="grey40"),
        legend.background = element_rect(fill="transparent")) +
  labs(title    = "Rain Expected by Wind Direction at 3pm",
       subtitle = "Observations from " %s+%
                  num_locations %s+%
                  " weather stations",
       caption  = "Source: Australian Bureau of Meteorology",
       x        = "Wind Direction 3pm",
       y        = "Number of Days",
       fill     = "Tomorrow")
```

The most obvious change is to the colouring which is supported by RColorBrewer::**brewer.pal()**. This is used to generate a dark/light pair of colours that are softer in presentation. We use only the first two colours from the generated palette of four colours.

In reviewing the original plot we might also notice that the y scale is in the thousands and yet no comma is used to emphasise that. It is always a good idea to include commas in large numbers to denote the thousands and avoid misreading. We do so by specifying that the y scale labels should use scales::**comma()**. We also include more informative labels through the use of ggplot2::**labs()**.

The new plot is arguably more appealing and marginally more informative than the original. We might note though that the most interesting question we can ask in relation to the data behind this plot is whether there is any differences in the distribution between rain and no rain across the different wind directions. Our chosen plot does not facilitate answering this question. A simple change to ggplot2::**geom_bar()** by adding position="fill" results in Figure 4.5. Notice that we moved the legend back to its original position as there is now no empty space within the plot.

```
ds %>%
  ggplot(aes(x=wind_dir_3pm, fill=rain_tomorrow)) +
  geom_bar(position="fill") +
  scale_fill_manual(values = blues2,
                    labels = c("No Rain", "Rain")) +
  scale_y_continuous(labels=comma) +
  theme(legend.title      = element_text(colour="grey40"),
        legend.text       = element_text(colour="grey40"),
        legend.background = element_rect(fill="transparent")) +
  labs(title    = "Rain Expected by Wind Direction at 3pm",
       subtitle = "Observations from " %s+%
                  num_locations %s+%
                  " weather stations",
       caption  = "Source: Australian Bureau of Meteorology",
       x        = "Wind Direction 3pm",
       y        = "Number of Days",
       fill     = "Tomorrow")
```

Observe now that indeed wind direction appears to have an influence on whether it rains the following day. Northerly winds appear to have a higher proportion of following days on which it rains. Any such observation requires further statisical confirmation to be sure.

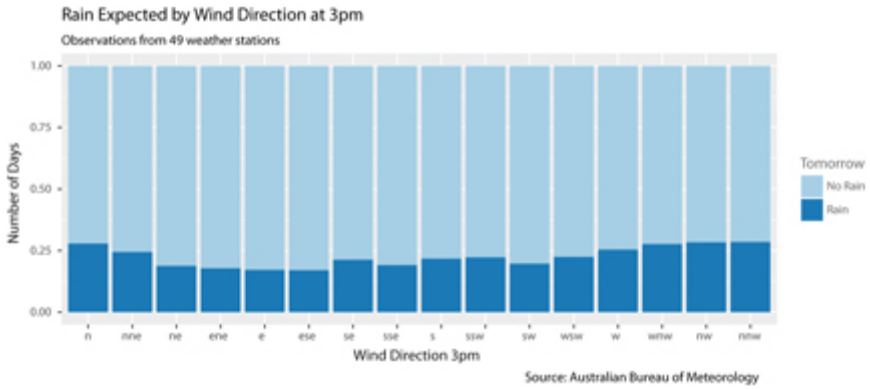


**Figure 4.5** *A decorated stacked fil led bar chart.*

## 4.6 Alternative Bar Charts

There are a variety of options available to tune how we present a bar chart using ggplot2:: **ggplot()** . We illustrate here dealing with a few more bars in the plot by visualising the mean temperature at 3pm for locations in the dataset.

We start with the plot shown in Figure 4.6. The plot has x= location and y= temp_3pm with a fill= set to the location. The choice of fill simply results in each bar being rendered with a different colour but adds no real value to the plot other than, arguably, its appeal. To this basic setup we add a stat= summary and calculate the bars as the mean value of the y axis variable using fun.y=.

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

There are 49 locations represented in the dataset resulting in quite a clutter of location names along the x axis. The obvious solution is to rotate the labels which we achieve by modifying the ggplot2::**theme()** through setting the axis.text= to be rotated by an angle= of 90° . The result is shown in Figure 4.7.

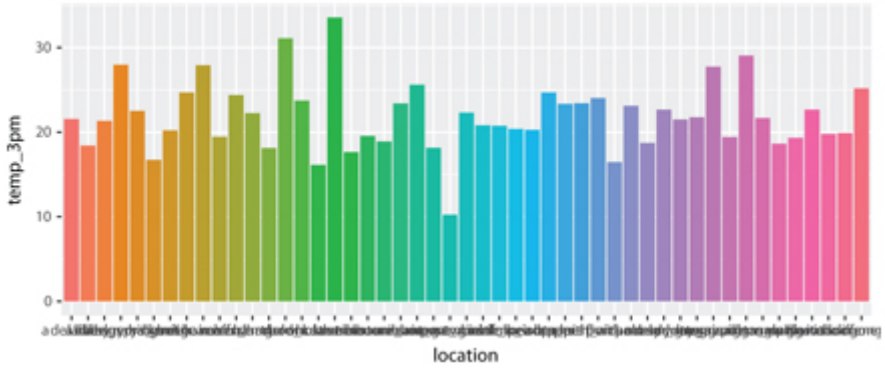


**Figure 4.6** *Multiple bars with overlapping labels.*

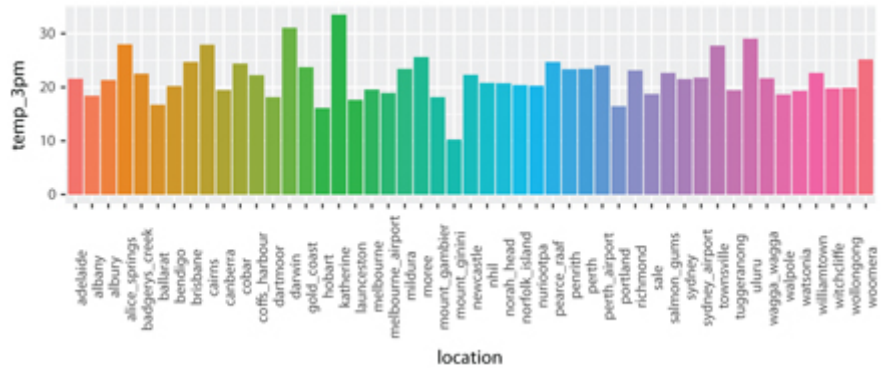


**Figure 4.7** *Rotating labels in a plot.*

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

Instead of flipping the labels we could flip the coordinates and produce a horizontal bar chart as in Figure 4.8. Rotating the plot allows more bars to be readily added down the page than we might across the page. It is also easier for us to read the labels left to right rather than bottom up. However the plot is less compact.

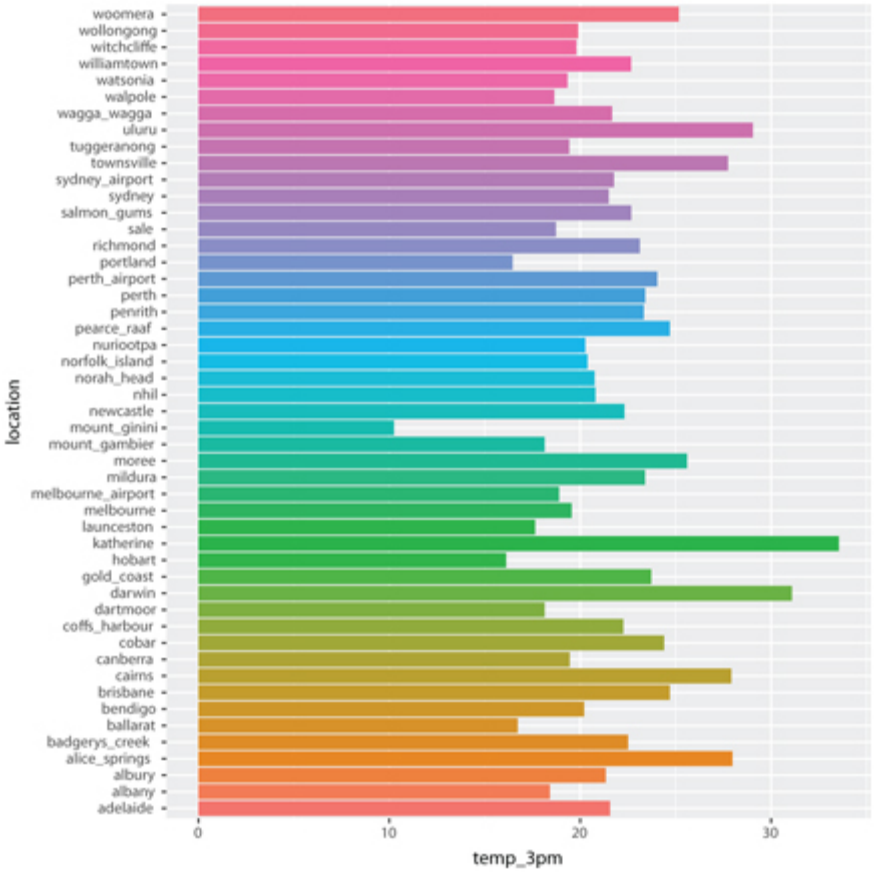


**Figure 4.8** *Rotating the plot.*

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

We would also naturally be inclined to expect the labels to appear in alphabetic order rather than the reverse as it appears by default. One approach is to reverse the order of the levels in the original dataset. We can use dplyr::**mutate()** within a pipeline to temporarily do this and pass the modified dataset on to ggplot2::**ggplot()**. The result can be seen in Figure 4.9.

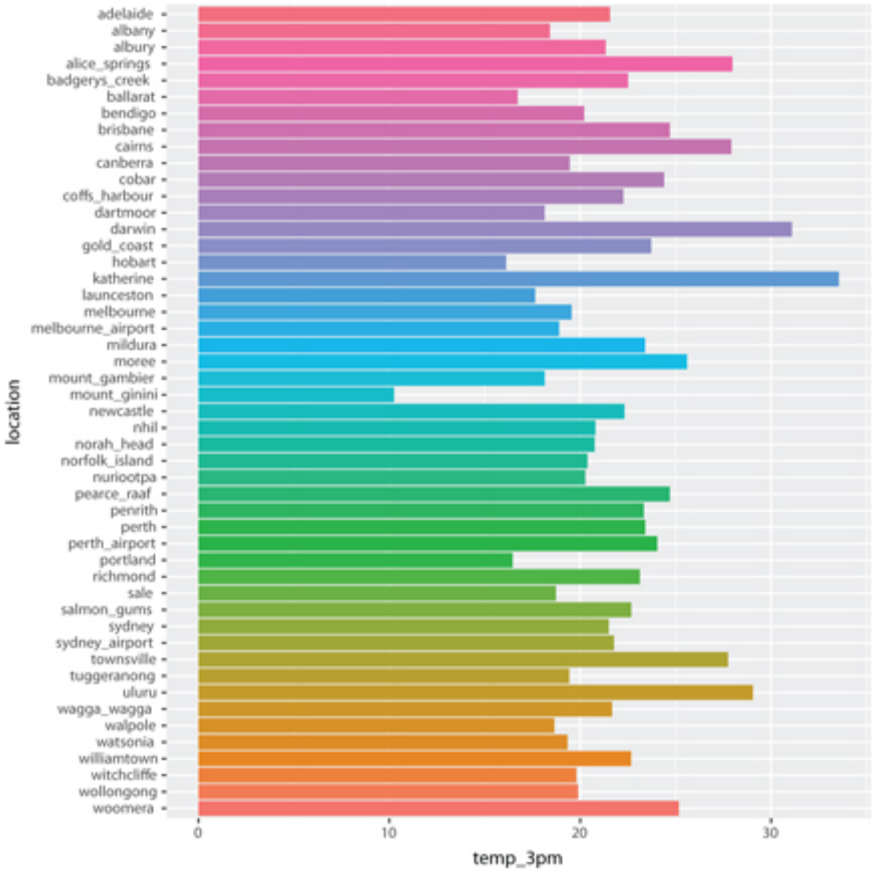


**Figure 4.9** *Reordering labels.*

```
ds$location %>%
  levels() %>%
  rev() ->
loc

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

  coord_flip()
```

## 4.7 Box Plots

A box plot, also known as a box and whiskers plot, is another tool to visualise the distribution of our data. The box plot of Figure 4.10 shows the median value (the midpoint) of the variable as the horizontal line within the box. The box itself contains half the observations with one quarter of the remaining observations shown below and one quarter of the remaining observations shown

above the box. Specific points are displayed as outliers at the extremes. An outlier is a value that is some distance from the bulk of the data. We use ggplot2::**geom_boxplot()** to add a box plot to out canvas.

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

Here the aesthetics for ggplot2::**ggplot()** are set with x= year and y= max_temp. Colour is added using fill= year. The colour (arguably) improves the visual appeal of the plot—it conveys little if any information. Since we have included fill= we must also turn off the otherwise included but redundant legend through the ggplot2::**theme()** with a legend.position= none.

Now that we have presented the data we begin our observations of the data. It is noted that the first and last verticals look different to the others due to the data likely being truncated. Our task is to confirm this indeed is a fact of the data itself. The 2017 data is truncated to the beginning of the year in this particular dataset. It is clear that the year has begun with very hot temperatures, remembering that being from Australia this data is recorded for a summer month.

A variation of the box plot is the violin plot. The violin plot adds information about the distribution of the data. The resulting shape often reflects that of a violin. In Figure 4.11 we again use colour to improve the visual appeal of the plot.

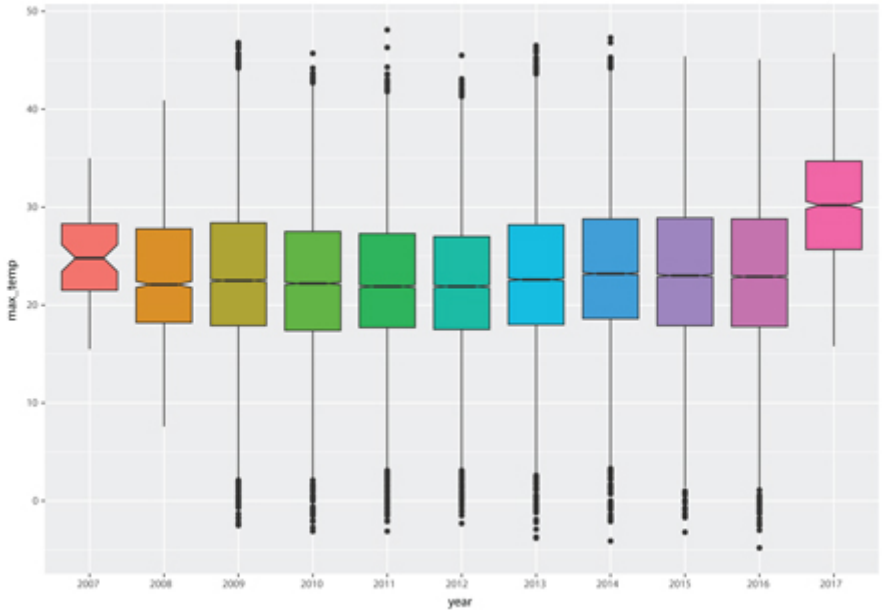


**Figure 4.10** *A traditional box and wiskers plot.*

```
ds %>%
  ggplot(aes(x=year, y=max_temp, fill=year)) +
  geom_violin() +
  theme(legend.position="none")
```

The amount of information we capture in the plot is increased by overlaying a box plot onto the violin plot. This could result in information overload or else it might convey concisely the story that the data tells. Often we need to make a trade off.

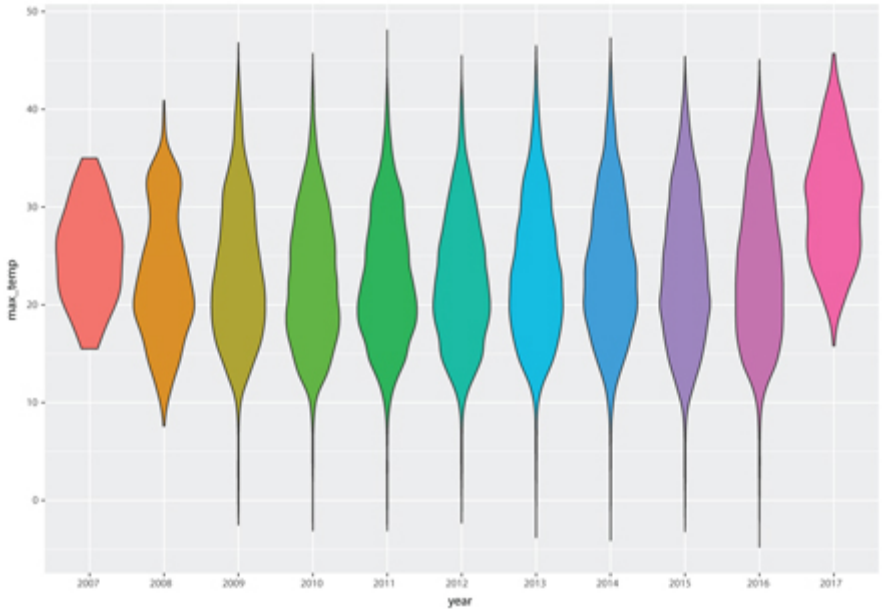


**Figure 4.11** *A violin plot.*

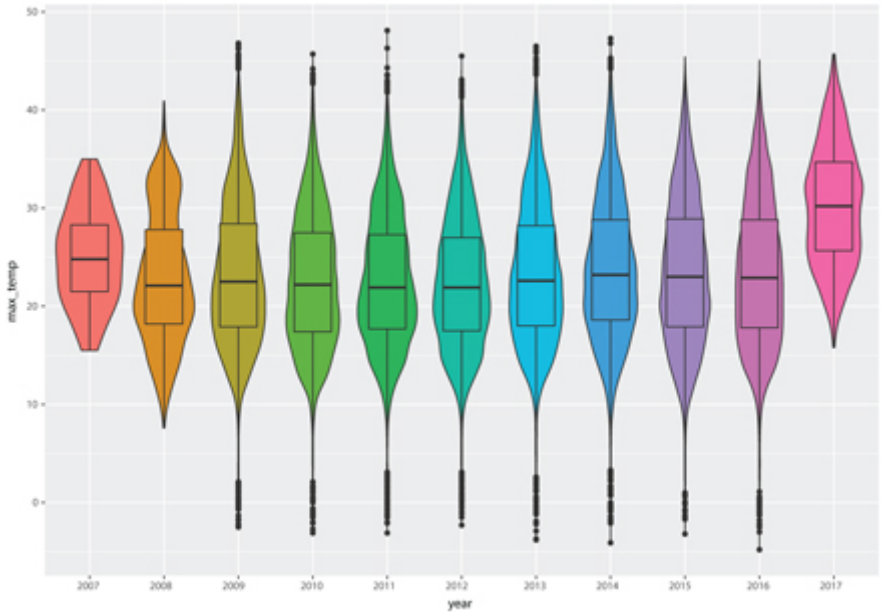


**Figure 4.12** *A violin plot with a box plot overlay.*

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

We can also readily split our plot across locations. The resulting plot (Figure 4.13) is a little crowded but we get an overall view across all of the different weather stations. Notice that we also rotated the x-axis labels so that they don't overlap.

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

By creating this plot we can identify further issues exhibited by the data. For example, it is clear that three weather stations have fewer observations than the others as their plots are obviously truncated on the left. This may be a data collection issue or simply that those weather stations are newly installed. The actual reasoning may or may not be important but it is important to question and to understand what we observe from the data. Often we will identify systemic issues with the data that need to be addressed in modelling.

We notice also the issue we identified earlier of apparently few observations before 2009 with only the location Canberra having any observations in 2007. In seeking an explanation we come to understand this was simply the nature of how the data was collected from the Bureau. In any modelling over all locations we may decide to eliminate pre-2009 observations.

Further observing the characteristics of the data we note that some locations have quite minimal variation in their maximum temperatures over the years (e.g., Darwin) whilst others can swing between extremes. We also observe that most outliers appear to be at the warmer end of the scale rather than the colder end. There is also a collection of locations which appear to have no outliers. And so on. We are beginning on a journey of discovery—to discover our data—to live and breathe the data (Williams, 2011).

We follow up our observations with modelling to cluster the locations, for example, according to the visual patterns we have just observed. In particular, as our datasets increase in size (this current dataset has just 134,776 observations) we often need to subset the data in various ways in order to visualise the observations.


**Figure 4.13** *Violin/box plot by location*

A cluster analysis provides a simple mechanism to identify groups and allows us to then visualise within those groups. We illustrated the process of a cluster analysis over the numeric variables in Chapter 3. The aim of the cluster analysis, introducing a new feature, was to group the locations so that within a group the locations have similar values for the numeric variables and between the groups the numeric variables are more dissimilar. The resulting feature cluster allocated each location to a region and we can use that here.

We will construct a plot using exactly the same sequence of commands we used for Figure 4.13 . On the principle of avoiding repeating ourselves we create a function to capture the sequence of commands and thus allow multiple plots to be generated by simply calling the function rather than writing out the code each time.

```
# For convenience define a function to generate the plot.

myplot <- function(ds, n)
{
  ds %>%
    filter(cluster==n) %>%
    ggplot(aes(x=year, y=max_temp, fill=year)) +
    geom_violin() +
    geom_boxplot(width=.5, position=position_dodge(width=0)) +
    theme(legend.position="none") +
    theme(axis.text.x=element_text(angle=45, hjust=1)) +
    facet_wrap(~location)
}
```

We can now plot specific clusters with the results in Figures 4.14 and 4.15.

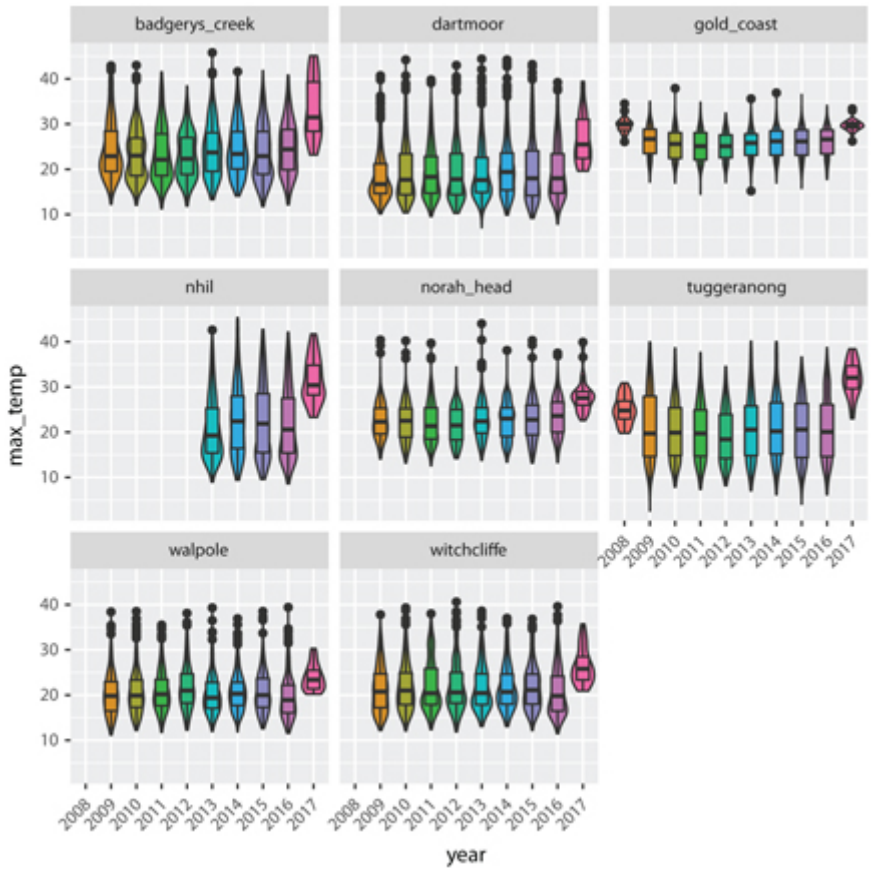


**Figure 4.14** *Visualise the first set of clustered locations.*

```
# Visualise specific cluster of locations.

myplot(ds, "area4")
```

```
# Visualise specific cluster of locations.

myplot(ds, "area5")
```

Being a grammar for graphics, or essentially a language in which to express our graphics, there is an infinite variety of possibilities with ggplot2. We will see further examples in the following chapters and extensively on the Internet.

As we explore the dataset and observe and question the characteristics of the dataset, we will move from visual observation to programmatic exploration and statistical confirmation.

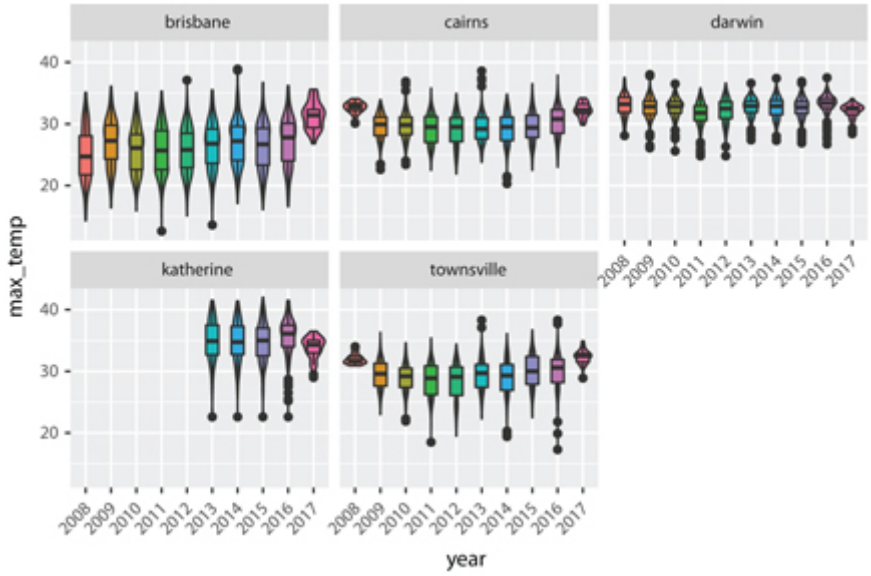


**Figure 4.15** *Visualise the second set of clustered locations.*

## 4.8 Exercises

### Exercise 4.1 Dashboard

Your computer collects information about its usage as log files. Identify some log files that you have access to on your own computer. Build a dashboard using ggplot2 to provide informative plots of the information contained in the log data. Include at least a bar plot and line chart, illustrating activity over time. Explore options for how you might turn these plots into a live dashboard.

### Exercise 4.2 Visualising Ferries

In the Chapter 3 exercises we introduced the ferry dataset. Continue your observations of that dataset through a visual exploration. Build on the narrative developed there to produce an enhanced narrative that includes visualisations that support the narrative.