

One Page R Data Science Visual Discovery

Graham.Williams@togaware.com

3rd June 2018

Visit <https://essentials.togaware.com/onepagers> for more Essentials.

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 glimpses in any issues with the data itself. Sometimes these will stand out in a visual presentation whilst scanning the data textually can often hide distributional wobble. A visual presentation is also an effective means for communicating insight to others.

20180603

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 along with the aesthetics (e.g., the x-axis and y-axis) and then adding geometric elements, statistical operations, scales, facets, coordinates, and numerous other components to the plot.

In this guide we explore data using **ggplot2** and affiliated packages to gain insight 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 many of which are available online.

Through this guide new R commands will be introduced. The reader is encouraged to review the command's documentation and understand what the command does. Help is obtained using the `? command` as in:

```
?read.csv
```

Documentation on a particular package can be obtained using the `help=` option of `library()`:

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

This chapter is intended to be hands on. To learn effectively the reader is encouraged to run R locally (e.g., RStudio or Emacs with ESS mode) and to replicate all commands as they appear here. Check that output is the same and it is clear how it is generated. Try some variations. Explore.

Copyright © 2000-2018 Graham Williams. This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/) allowing this work to be copied, distributed, or adapted, with attribution and provided under the same license.



1 Packages from the R Library

Packages used in this chapter include dplyr (Wickham *et al.*, 2018), ggplot2, magrittr (Bache and Wickham, 2014), randomForest (Breiman *et al.*, 2018), rattle.data (Williams, 2017c), scales (Wickham, 2017), stringr (Wickham, 2018), and, rattle (Williams, 2017b),

20180603

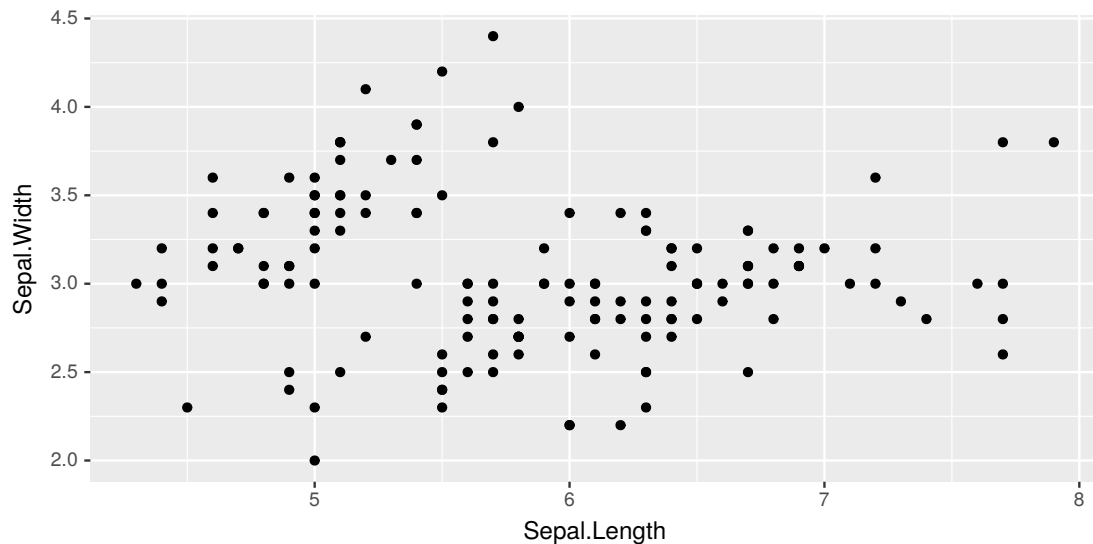
Packages are loaded here into the R session from the local library folders. Any packages that are missing can be installed using `utils::install.packages()`.

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

library(dplyr)           # glimpse().
library(ggplot2)        # Visualise data.
library(magrittr)       # Data pipelines: %>% %<>% %T>% equals().
library(randomForest)   # na.roughfix() for missing data.
library(rattle)         # normVarNames().
library(rattle.data)    # weatherAUS.
library(scales)         # commas(), percent().
library(stringr)       # str_replace_all().
```

2 Quickstart Scatter Plot

20180603



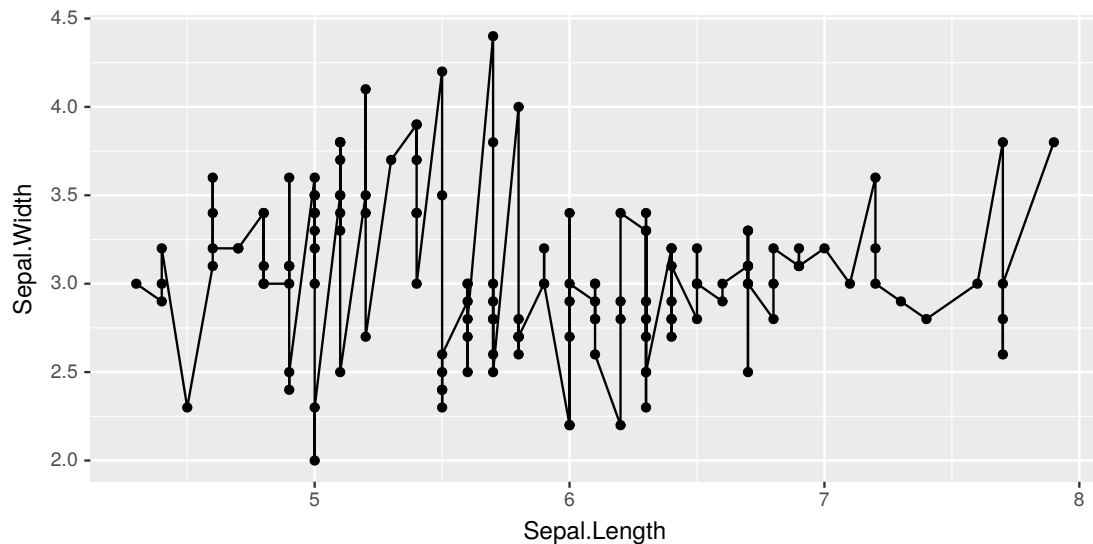
Here we illustrate the simplest of examples. The `datasets::iris` dataset is a widely used sample dataset in statistics and commonly used to illustrate concepts in statistical model building. Above we see a simple [scatter plot](#) which displays a two dimensional plot. By dimensions we refer to the x and y axes and associate one dimension (i.e., variable) to each axis. The observations from the dataset, which are the rows in the dataset, are then plotted, scattering the points over the plot.

To build this plot we reference the `datasets::iris` dataset in the computer's memory and pipe (`stringr::%>%`) it into `ggplot2::ggplot2()`. The aesthetics `ggplot2::aes()` are set up with `Sepal.Length` set as the x= axis and `Sepal.Width` as the y= axis. To add the points (observations) to the plot we employ `ggplot2::geom_point()`.

```
iris %>%  
  ggplot(aes(x=Sepal.Length, y=Sepal.Width)) +  
  geom_point()
```

3 Adding a Line

20180603

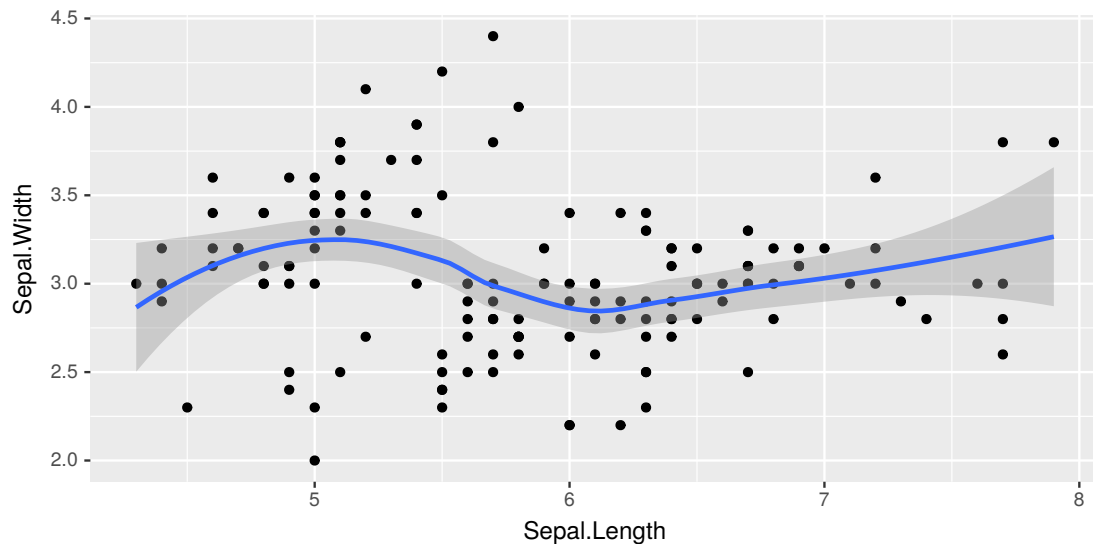


As an alternative to plotting the points we could join the dots and plot a line. One reason for doing so might be to attempt to observe more clearly any relationship between the two dimensions (i.e., the two variables we are plotting). Add a `ggplot2::geom_line()` to the canvas we can draw a line from left to right the follows the points exactly. The result is a rather jugged line and it is debatable whether it has added any further insight into our understanding of the data.

```
iris %>%  
  ggplot(aes(x=Sepal.Length, y=Sepal.Width)) +  
  geom_point() +  
  geom_line()
```

4 Adding a Smooth Line

20180603



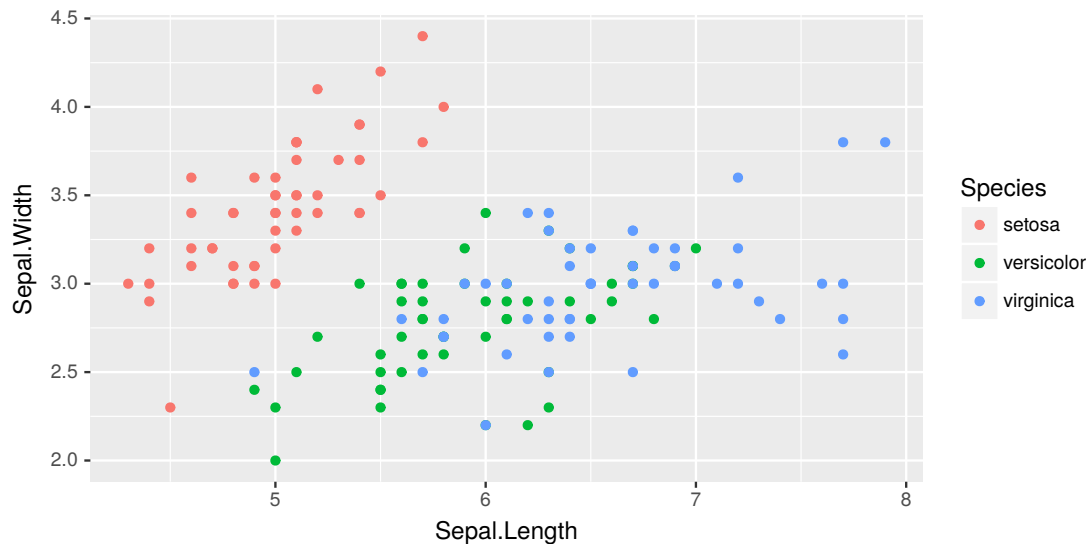
The rather jagged line from point to point does not give a very convincing picture of any relationship between the two variables. A common approach is to introduce a statistical function to “fit” a single smoother line to the points. We aim to draw a smoother line from left to right the follows the points but without the requirement of going through every point.

The resulting plot here uses `ggplot2::geom_smooth()` with a so called [locally weighted scatterplot smoothing](#) method or LOESS to produce the smoothed line. As a result we can observe statistically something of a relationship between the two variables although it is not a simple relationship.

```
iris %>%  
  ggplot(aes(x=Sepal.Length, y=Sepal.Width)) +  
  geom_point() +  
  stat_smooth(method="loess")
```

5 Colour the Points

20180603



A key variable in the `datasets::iris` dataset is the **Species**. By colouring the dots according to the **Species** we suddenly identify a quite significant relationship in the data. Indeed we might call this knowledge discovery. It is very clear that all of the observations in the top left belong to the **setosa** species whilst the next clump is primarily **versicolor** and the right hand observations are **virginica**. It is clear that there is some so-called *structure* in this dataset.

```
iris %>%  
  ggplot(aes(x=Sepal.Length, y=Sepal.Width, colour=Species)) +  
  geom_point()
```

The colour is added simply by specifying a further aesthetic, `colour=Species`. This results in the different values of the variable **Species** being coloured differently.

6 Initialise the Weather Dataset

The modestly large **weatherAUS** dataset from **rattle.data** is used to further illustrate the capabilities of **ggplot2**. For plots which generate large images a random subset of the same dataset is deployed to allow replication in a timely manner. The dataset is loaded and processed following the template introduced in [Williams \(2017a\)](#).

```
# Initialise the dataset as per the template.
```

```
library(rattle)
```

```
dsname <- "weatherAUS"
```

```
ds <- get(dsname)
```

The dataset can be summarised using **tibble::glimpse()**.

```
glimpse(ds)
```

```
## Observations: 145,460
```

```
## Variables: 24
```

```
## $ Date <date> 2008-12-01, 2008-12-02, 2008-12-03,...
```

```
## $ Location <fct> Albury, Albury, Albury, Albury, Albu...
```

```
## $ MinTemp <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, 14...
```

```
## $ MaxTemp <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, 0...
```

```
## $ Evaporation <dbl> NA, NA, NA, NA, NA, NA, NA, NA, ...
```

```
## $ Sunshine <dbl> NA, NA, NA, NA, NA, NA, NA, NA, ...
```

```
## $ WindGustDir <ord> W, WNW, WSW, NE, W, WNW, W, W, NNW, ...
```

```
## $ WindGustSpeed <dbl> 44, 44, 46, 24, 41, 56, 50, 35, 80, ...
```

```
## $ WindDir9am <ord> W, NNW, W, SE, ENE, W, SW, SSE, SE, ...
```

```
## $ WindDir3pm <ord> WNW, WSW, WSW, E, NW, W, W, W, NW, S...
```

```
## $ WindSpeed9am <dbl> 20, 4, 19, 11, 7, 19, 20, 6, 7, 15, ...
```

```
## $ WindSpeed3pm <dbl> 24, 22, 26, 9, 20, 24, 24, 17, 28, 1...
```

```
## $ Humidity9am <int> 71, 44, 38, 45, 82, 55, 49, 48, 42, ...
```

```
## $ Humidity3pm <int> 22, 25, 30, 16, 33, 23, 19, 19, 9, 2...
```

```
## $ Pressure9am <dbl> 1007.7, 1010.6, 1007.6, 1017.6, 1010...
```

```
## $ Pressure3pm <dbl> 1007.1, 1007.8, 1008.7, 1012.8, 1006...
```

```
## $ Cloud9am <int> 8, NA, NA, NA, 7, NA, 1, NA, NA, NA,...
```

```
## $ Cloud3pm <int> NA, NA, 2, NA, 8, NA, NA, NA, NA, NA...
```

```
## $ Temp9am <dbl> 16.9, 17.2, 21.0, 18.1, 17.8, 20.6, ...
```

```
## $ Temp3pm <dbl> 21.8, 24.3, 23.2, 26.5, 29.7, 28.9, ...
```

```
## $ RainToday <fct> No, No, No, No, No, No, No, No, No, ...
```

```
## $ RISK_MM <dbl> 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0.0, 0...
```

```
## $ RainTomorrow <fct> No, No, No, No, No, No, No, No, Yes,...
```

7 Normalise Variable Names

Variable names are normalised so as to have some certainty in interacting with the data. The convenience function `rattle::normVarNames()` can do this.

```
# Review the variables before normalising their names.

names(ds)

## [1] "Date"          "Location"      "MinTemp"
## [4] "MaxTemp"       "Rainfall"      "Evaporation"
## [7] "Sunshine"      "WindGustDir"   "WindGustSpeed"
## [10] "WindDir9am"    "WindDir3pm"    "WindSpeed9am"
....

# Capture the original variable names for use in plots.

vnames <- names(ds)

# Normalise the variable names.

names(ds) %<>% normVarNames()

# Confirm the results are as expected.

names(ds)

## [1] "date"          "location"      "min_temp"
## [4] "max_temp"      "rainfall"      "evaporation"
## [7] "sunshine"      "wind_gust_dir" "wind_gust_speed"
## [10] "wind_dir_9am"  "wind_dir_3pm"  "wind_speed_9am"
....

# Index the original variable names by the new names.

names(vnames) <- names(ds)

vnames

##          date          location          min_temp
##      "Date"      "Location"      "MinTemp"
##    max_temp    rainfall    evaporation
##  "MaxTemp"    "Rainfall"    "Evaporation"
....
```

The variable names now conform to our expectations of them and in accordance to our chosen style as documented in

8 Variables and Model Target

Reviewing the variables we will note the different roles each of the variables play. Again we make use of the template in identifying variable roles.

```
# Note the available variables.

vars <- names(ds) %T>% print()

## [1] "date"           "location"       "min_temp"
## [4] "max_temp"       "rainfall"       "evaporation"
## [7] "sunshine"       "wind_gust_dir"  "wind_gust_speed"
## [10] "wind_dir_9am"   "wind_dir_3pm"   "wind_speed_9am"
## [13] "wind_speed_3pm" "humidity_9am"   "humidity_3pm"
## [16] "pressure_9am"   "pressure_3pm"   "cloud_9am"
## [19] "cloud_3pm"      "temp_9am"       "temp_3pm"
## [22] "rain_today"     "risk_mm"        "rain_tomorrow"

# Note the target variable.

target <- "rain_tomorrow"

# Place the target variable at the beginning of the vars.

vars <- c(target, vars) %>% unique() %T>% print()

## [1] "rain_tomorrow"  "date"           "location"
## [4] "min_temp"       "max_temp"       "rainfall"
## [7] "evaporation"    "sunshine"       "wind_gust_dir"
## [10] "wind_gust_speed" "wind_dir_9am"   "wind_dir_3pm"
## [13] "wind_speed_9am" "wind_speed_3pm" "humidity_9am"
## [16] "humidity_3pm"   "pressure_9am"   "pressure_3pm"
## [19] "cloud_9am"      "cloud_3pm"      "temp_9am"
## [22] "temp_3pm"       "rain_today"     "risk_mm"
```

9 Modelling Roles

```

# Note the risk variable which measures the severity of the outcome.

risk <- "risk_mm"

# Note the identifiers.

id <- c("date", "location")

# Initialise ignored variables: identifiers.

ignore <- c(risk, id)

# Remove the variables to ignore.

vars <- setdiff(vars, ignore)

# Identify the input variables for modelling.

inputs <- setdiff(vars, target) %T>% print()

## [1] "min_temp"      "max_temp"      "rainfall"
## [4] "evaporation"   "sunshine"      "wind_gust_dir"
## [7] "wind_gust_speed" "wind_dir_9am"  "wind_dir_3pm"
## [10] "wind_speed_9am" "wind_speed_3pm" "humidity_9am"
## [13] "humidity_3pm"   "pressure_9am"   "pressure_3pm"
## [16] "cloud_9am"      "cloud_3pm"      "temp_9am"
## [19] "temp_3pm"       "rain_today"

# Also record them by indices.

inputi <-
  inputs %>%
  sapply(function(x) which(x == names(ds)), USE.NAMES=FALSE) %T>%
  print()

## [1] 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
## [20] 22

```

10 Variable Types

```
# Identify the numeric variables by index.

numi <-
  ds %>%
  sapply(is.numeric) %>%
  which() %>%
  intersect(inputi) %T>%
  print()

## [1] 3 4 5 6 7 9 12 13 14 15 16 17 18 19 20 21

# Identify the numeric variables by name.

numc <-
  ds %>%
  names() %>%
  extract(numi) %T>%
  print()

## [1] "min_temp" "max_temp" "rainfall"
## [4] "evaporation" "sunshine" "wind_gust_speed"
....

# Identify the categoric variables by index.

cati <-
  ds %>%
  sapply(is.factor) %>%
  which() %>%
  intersect(inputi) %T>%
  print()

## [1] 8 10 11 22

# Identify the categoric variables by name.

catc <-
  ds %>%
  names() %>%
  extract(cati) %T>%
  print()

## [1] "wind_gust_dir" "wind_dir_9am" "wind_dir_3pm"
## [4] "rain_today"
....

# Normalise the levels of all categoric variables.

for (v in catc)
  levels(ds[[v]]) %<>% normVarNames()
```

11 Missing Value Imputation

We can also 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 for us.

```
# Count the number of missing values.

ds[vars] %>% is.na() %>% sum()
## [1] 343248

# Impute missing values.

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

# Confirm that no missing values remain.

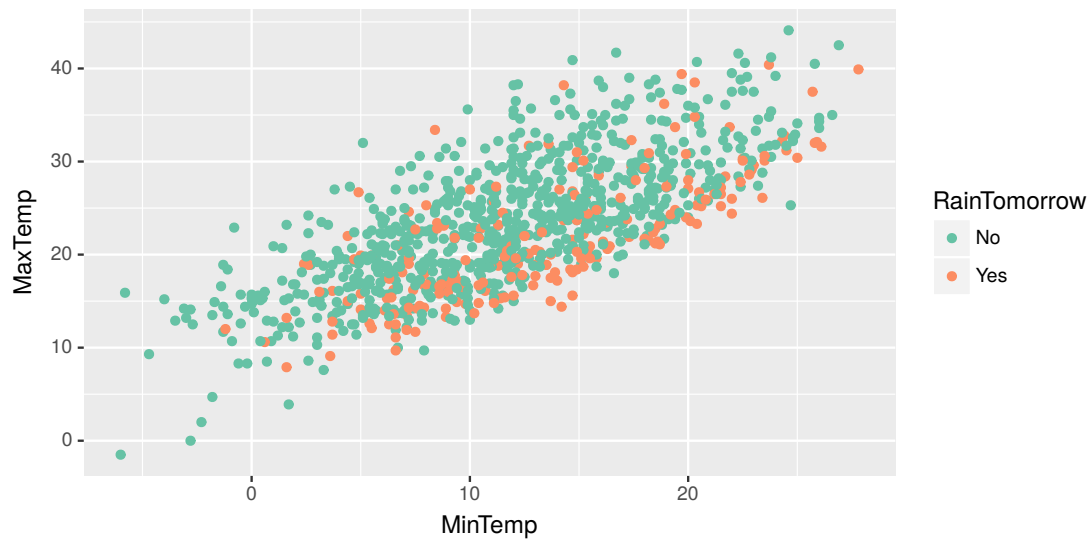
ds[vars] %>% is.na() %>% sum()
## [1] 0
```

12 Review the Dataset

The dataset is now ready to be visually explored but before doing so it is useful to have another `tibble::glimpse()`.

```
glimpse(ds)
## Observations: 145,460
## Variables: 24
## $ date          <date> 2008-12-01, 2008-12-02, 2008-12-0...
## $ location      <fct> Albury, Albury, Albury, Albury, Al...
## $ 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> 8.4, 8.4, 8.4, 8.4, 8.4, 8.4, 8.4,...
## $ 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 <dbl> 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   <dbl> 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     <fct> no, no, no, no, no, no, no, no, no...
## $ risk_mm        <dbl> 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0.0,...
## $ rain_tomorrow  <fct> No, No, No, No, No, No, No, No, Ye...
```

13 Scatter Plot



The simplest plot is a [scatter plot](#) which displays points scattered over a plot. If the dataset is large the resulting plot will be rather dense. For illustrative purposes a random subset of just 1,000 observations is used. A linear relationship between the two variables can be seen.

```
ds %>%
  sample_n(1000) %>%
  ggplot(aes(x=min_temp, y=max_temp, colour=rain_tomorrow)) +
  geom_point() +
  scale_colour_brewer(palette="Set2") +
  labs(x = vnames["min_temp"],
       y = vnames["max_temp"],
       colour = vnames["rain_tomorrow"])
```

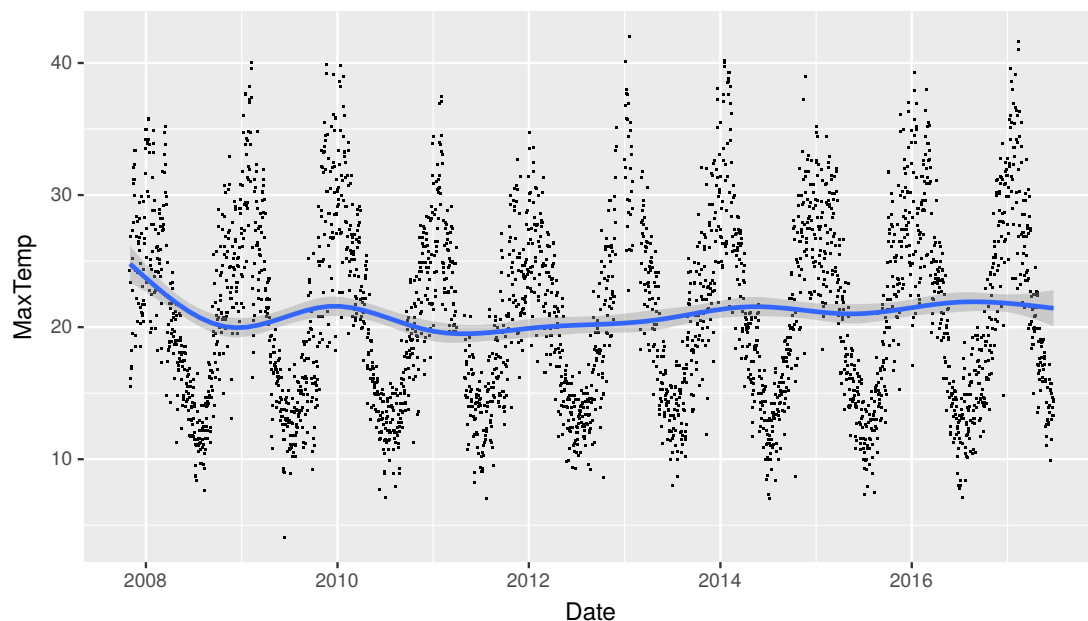
The random sample of 1,000 rows is generated using `dplyr::sample_n()` and is then piped through to `ggplot2::ggplot()`. The function argument identifies the aesthetics of the plot so that `x=` associates the variable `min_temp` with the x-axis and `y=` associates the variable `max_temp` with the y-axis.

In addition the `colour=` option provides a mechanism to distinguish between days where the observation `rain_tomorrow` is Yes and where it is No. A colour palette can be chosen using `ggplot2::scale_colour_brewer()`.

A graphical layer is added to the plot consisting of (x, y) points coloured appropriately. The function `ggplot2::geom_point()` achieves this.

The original variable names stored as `vnames` are used to label the plot using `ggplot2::labs()`. The original names will make more sense to the reader than our chosen normalised names.

14 Scatter Plot, Smooth Fitted Curve



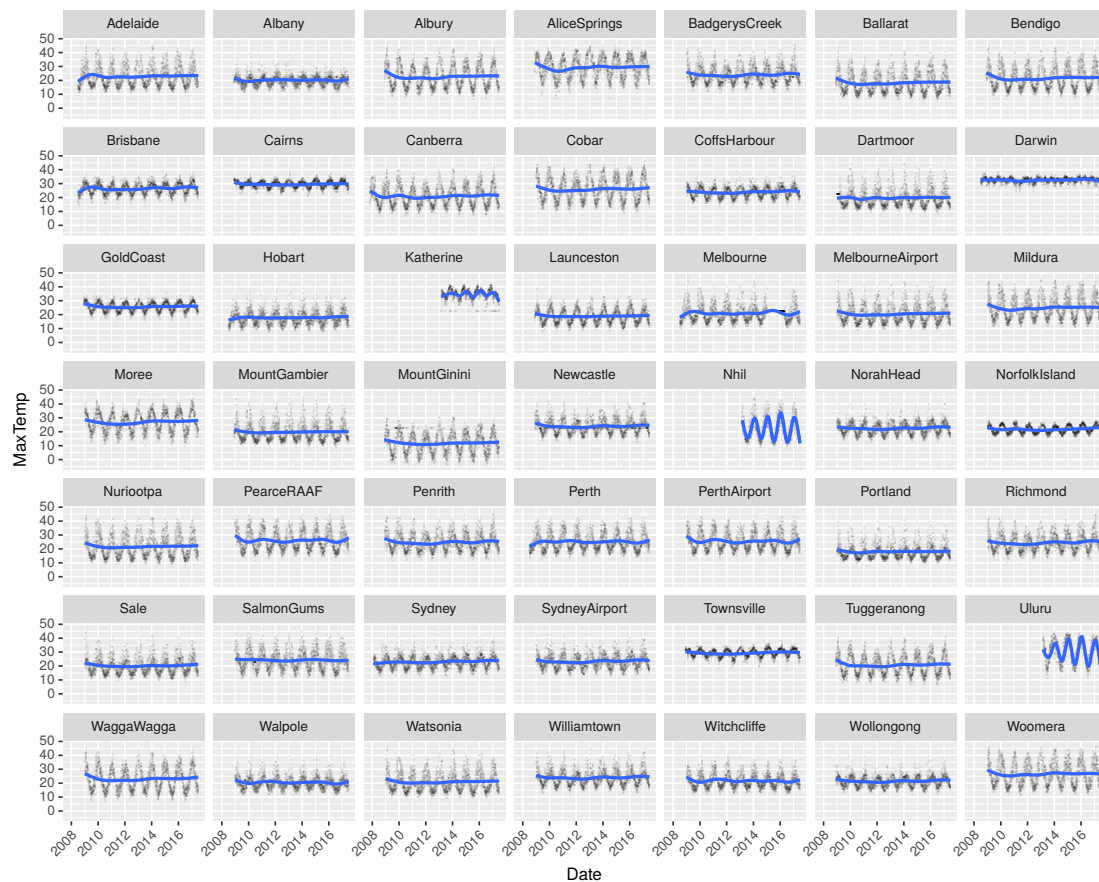
This scatter plot of `x=date` against `y=max_temp` shows a pattern of seasonality over the dataset and a trend line over the period of the dataset.

```
ds %>%
  filter(location=="Canberra") %>%
  ggplot(aes(x=date, y=max_temp)) +
  geom_point(shape=".") +
  geom_smooth(method="gam", formula=y~s(x, bs="cs")) +
  labs(x=vnames["date"], y=vnames["max_temp"])
```

The scatter plot is again created using `ggplot2::geom_point()`. Typical of scatter plots of big data there will be many overlaid points. To reduce the impact the points are reduced to a small dot using `shape="."`.

An additional layer is added consisting of a smooth fitted curve using `ggplot2::geom_smooth()`. The dataset has many points and so a smoothing method recommended is `method="gam"` which will automatically be chosen if not specified but with a message to that effect. The formula specified using `formula=` is also the default for `method="gam"`.

15 Scatter Plot, Faceted Locations

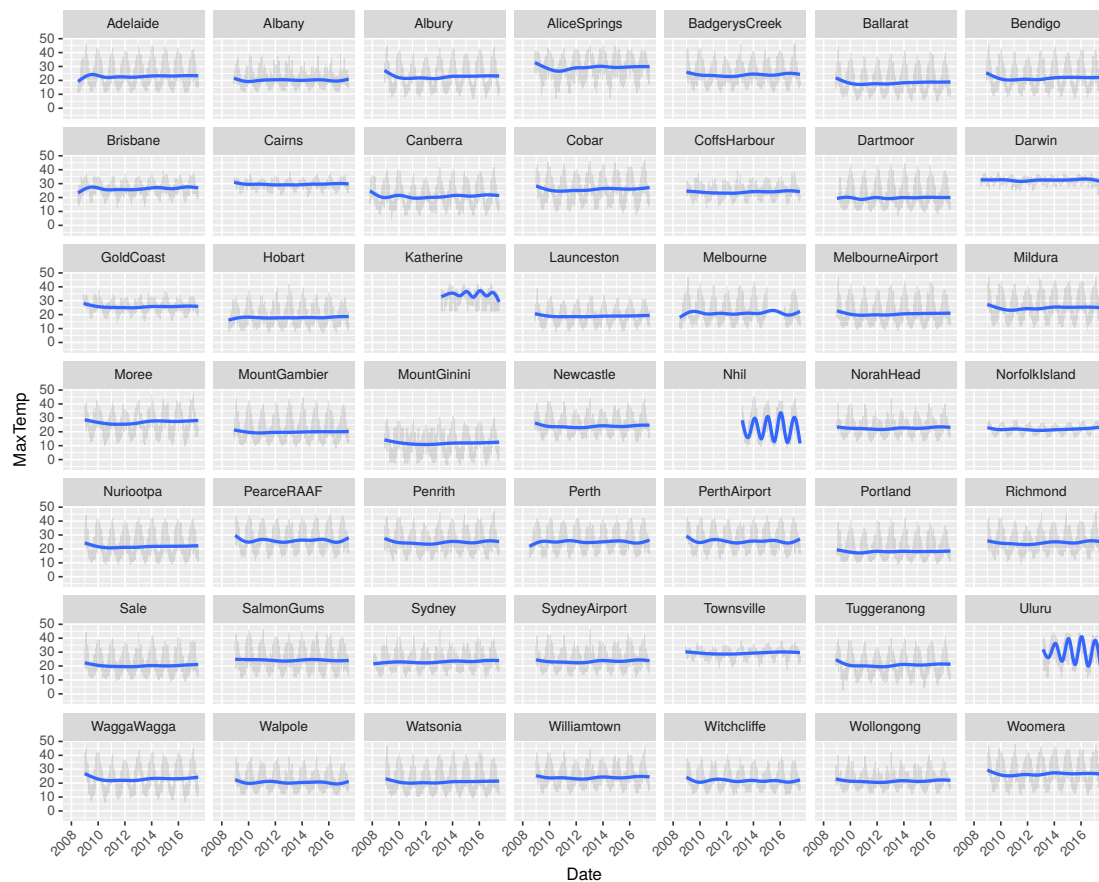


Partitioning the dataset by a categoric variable will reduce the blob effect for big data. The plot here uses `location` as the faceted variable to separately plot each location's maximum temperature over time. Notice the seasonal effect across all plots, some with quite different patterns.

```
ds %>%
  ggplot(aes(x=date, y=max_temp)) +
  geom_point(alpha=0.05, shape=".") +
  geom_smooth(method="gam", formula=y~s(x, bs="cs")) +
  facet_wrap(~location) +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  labs(x=vnames["date"], y=vnames["max_temp"])
```

The plot uses `facet_wrap()` to separately plot each location. For the points `ggplot2::geom_point()` is provided an `alpha=` to reduce the effect of overlaid points. Using smaller dots on the plots by way of `shape=` also de-clutters the plot significantly and improves the presentation and emphasises the patterns. The x axis tick labels are rotated 45° using `angle=45` within `ggplot2::element_text()` to avoid the labels overlapping. The `hjust=1` forces the labels to be right justified.

16 Line Plot, Faceted Locations, Thin Lines

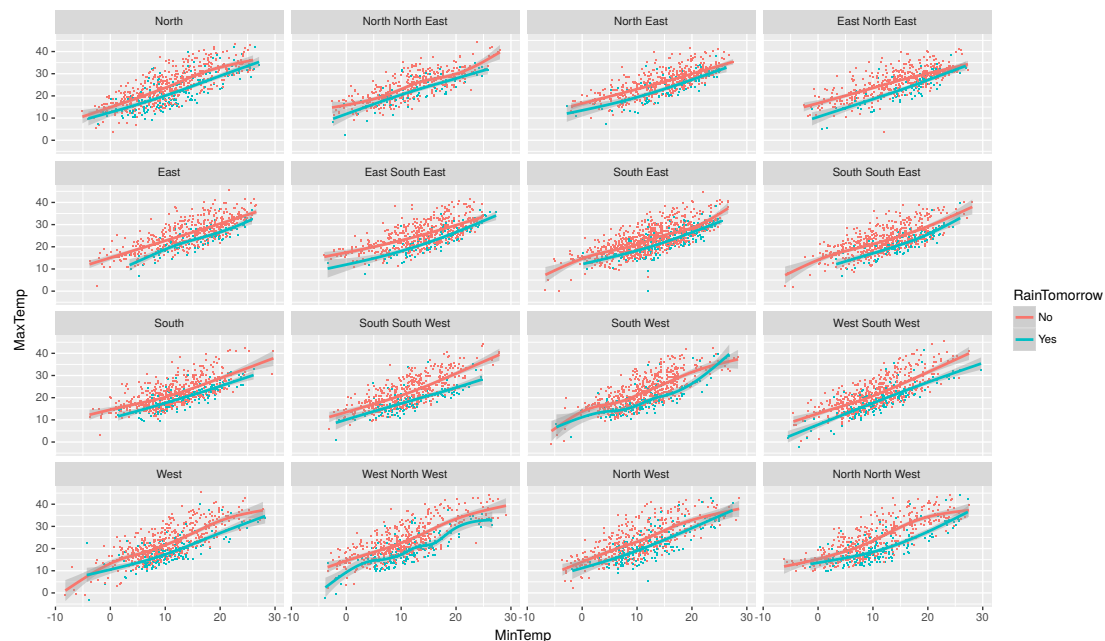


An alternative is to present the plot as a line chart rather than a scatter plot. It does make more sense for a time series plot such as this, though the effect is little changed due to the amount of data being displayed.

```
ds %>%
  ggplot(aes(x=date, y=max_temp)) +
  geom_line(alpha=0.1, size=0.05) +
  geom_smooth(method="gam", formula=y~s(x, bs="cs")) +
  facet_wrap(~location) +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  labs(x=vnames["date"], y=vnames["max_temp"])
```

Changing to lines simply uses `ggplot2::geom_line()` instead of `ggplot2::geom_point()`. Very thin lines are used as specified through the `size=` option. Nonetheless, the data remains quite dense.

17 Faceted Wind Directions



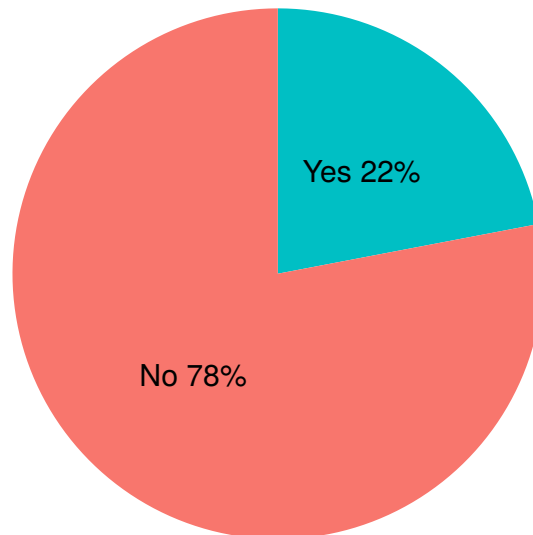
Labels of a faceted plot can be modified as here expanding n to North, s to South, etc. Observe that the linear relationship for rainy days is below that for dry days. The maximum temperature is generally closer to the minimum temperature on days where it rains the following day.

```
lblr <- function(x)
{
  x %>%
    str_replace_all("n", "North ") %>%
    str_replace_all("s", "South ") %>%
    str_replace_all("e", "East ") %>%
    str_replace_all("w", "West ") %>%
    str_replace(" $", "")
}

ds %>%
  sample_n(10000) %>%
  ggplot(aes(x=min_temp, y=max_temp, colour=rain_tomorrow)) +
  geom_point(shape=".") +
  geom_smooth(method="gam", formula=y~s(x, bs="cs")) +
  facet_wrap(~wind_dir_3pm, labeller=labeller(wind_dir_3pm=lblr)) +
  labs(x = vnames["min_temp"],
       y = vnames["max_temp"],
       colour = vnames["rain_tomorrow"])
```

The function to remap the directions uses `stringr::str_replace_all()` to do the work. It is then transformed into a `ggplot2::labeller()` for `wind_dir_3pm=`.

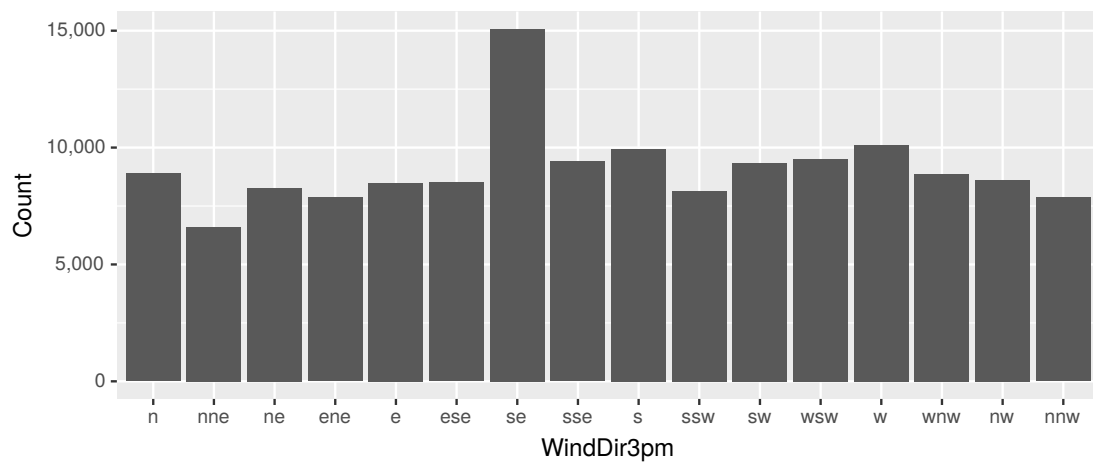
18 Pie Chart



A [pie chart](#) is a popular circular plot showing the relative proportions through angular slices. Generally, pie charts are not recommended, particularly for multiple wedges, because humans generally have difficulty perceiving the relative angular differences between slices. For two or three slices it may be argued that the pie chart is just fine, and if further information is provided, such as labelling the slices with their sizes.

```
ds %>%
  group_by(rain_tomorrow) %>%
  count() %>%
  ungroup() %>%
  mutate(per=round(`n`/sum(`n`), 2)) %>%
  mutate(label=paste(rain_tomorrow, percent(per))) %>%
  arrange(per) %>%
  ggplot(aes(x=1, y=per, fill=rain_tomorrow)) +
  geom_bar(stat="identity") +
  coord_polar(theta='y') +
  theme_void() +
  theme(legend.position="none") +
  geom_text(aes(x=1, y=cumsum(per)-per/2, label=label), size=8)
```

19 Histogram



Another common plot is the [histogram](#) or bar chart which displays the count of observations using bars. The bars present the frequency of the levels of the categoric variable `wind_dir_3pm` from the dataset.

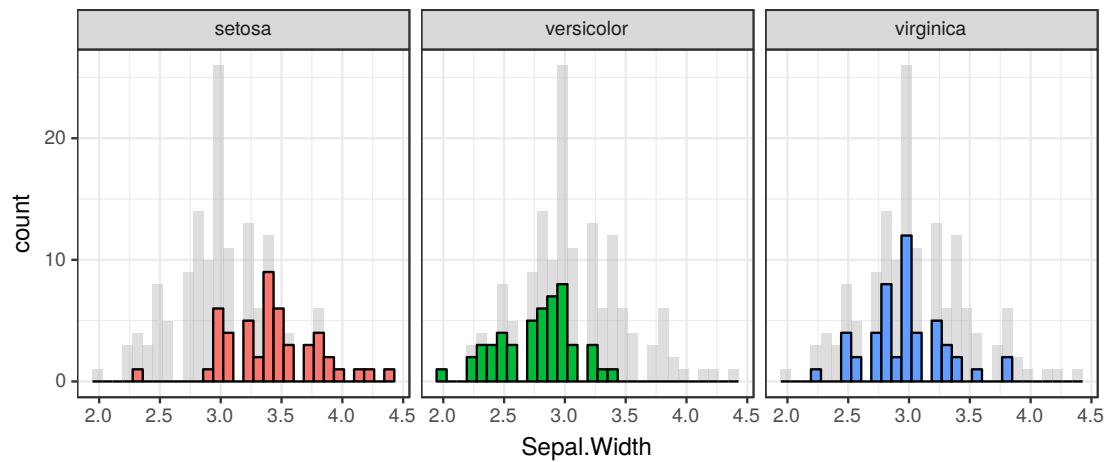
```
ds %>%  
  ggplot(aes(x=wind_dir_3pm)) +  
  geom_bar() +  
  scale_y_continuous(labels=comma) +  
  labs(x=vnames["wind_dir_3pm"], y="Count")
```

A histogram is generated using `ggplot2::geom_bar()`. Only an x-axis is required as the aesthetic and `wind_dir_3pm` is chosen.

20 Histogram with Background

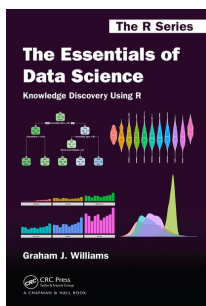
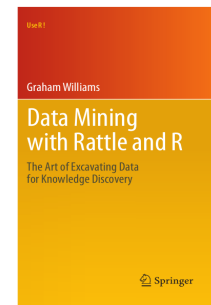
Originally from <https://drsimonj.svbtle.com/plotting-background-data-for-groups-with-ggplot2> who used the iris dataset. Convert to use the weather dataset.

```
## 'stat_bin()' using 'bins = 30'. Pick better value with  
## 'binwidth'.  
## 'stat_bin()' using 'bins = 30'. Pick better value with  
## 'binwidth'.
```



21 Further Reading and Acknowledgements

The [Rattle](#) book ([Williams, 2011](#)), published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from [Amazon](#). Rattle provides a graphical user interface through which the user is able to load, explore, visualise, and transform data, and to build, evaluate, and export models. Through its Log tab it specifically aims to provide an R template which can be exported and serve as the starting point for further programming with data in R.



The [Essentials of Data Science](#) book ([Williams, 2017a](#)), published by CRC Press, provides a comprehensive introduction to data science through programming with data using R. It is available from [Amazon](#). The book provides a template based approach to doing data science and knowledge discovery. Templates are provided for data wrangling and model building. These serve as generic starting points for programming with data, and are designed to require minimal effort to get started. Visit <https://essentials.togaware.com> for further guides and templates.

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.

22 References

- Bache SM, Wickham H (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5, URL <https://CRAN.R-project.org/package=magrittr>.
- Breiman L, Cutler A, Liaw A, Wiener M (2018). *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*. R package version 4.6-14, URL <https://CRAN.R-project.org/package=randomForest>.
- R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Wickham H (2017). *scales: Scale Functions for Visualization*. R package version 0.5.0, URL <https://CRAN.R-project.org/package=scales>.
- Wickham H (2018). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.3.1, URL <https://CRAN.R-project.org/package=stringr>.
- Wickham H, Chang W (2016). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 2.2.1, URL <https://CRAN.R-project.org/package=ggplot2>.
- Wickham H, François R, Henry L, Müller K (2018). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.5, URL <https://CRAN.R-project.org/package=dplyr>.
- 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.
- Williams GJ (2017a). *The Essentials of Data Science: Knowledge discovery using R*. The R Series. CRC Press.
- Williams GJ (2017b). *rattle: Graphical User Interface for Data Science in R*. R package version 5.1.0, URL <https://CRAN.R-project.org/package=rattle>.
- Williams GJ (2017c). *rattle.data: Rattle Datasets*. R package version 1.0.2, URL <https://CRAN.R-project.org/package=rattle.data>.

This document, sourced from VisualiseO.Rnw bitbucket revision 241, was processed by KnitR version 1.20 of 2018-02-20 10:11:46 UTC and took 21.9 seconds to process. It was generated by gjw on Ubuntu 18.04 LTS.