R cookbook for the casual dabbler (2nd edition)

Charles Coverdale

2025-02-15

Introduction

G'day and welcome to *R cookbook for the casual dabbler (2nd edition).*

RCCD 1st edition was originally published in 2020 as a side project during the COVID-19 pandemic in Melbourne.

As I wrote in the midst of lockdown:

> I use R a lot for work and for side projects. Over the years I've collated a bunch of useful scripts, from macroeconomic analysis to quick hacks for making map legends format properly.
>
> Historically my code has been stored in random Rpubs documents, medium articles, and a bunch of .Rmd files on my hardrive. Occasionally I feel like doing things properly - and upload code to a repository on github.
>
> It doesn't take a genius to realize this isn't a very sustainable solution - and it also isn't very useful for sharing code with others. It turns out 2-years of lockdown in Melbourne was enough incentive to sit down and collate my best and most useful code into a single place. In the spirit of open source, a book seemed like the most logical format. The following is a very rough book written in **markdown** - R's very own publishing language.

RCCD1e had surprisingly good (and long-lasting reviews). Alas, five-years on, a lot has changed - in both the R community, and the world more broadly. As such, RCCD2e includes significant revisions. The most major of these is a restructure to make the chapters flow more logically. The Australian specific chapters have also been grouped together (e.g. election data).

Usage

In each chapter I've written up the background, methodology and code for a separate piece of analysis.

Most of this code will not be extraordinary to the seasoned R aficionado. However I find that in classic Pareto style ~20% of my code contributes to the **vast** majority of my work output. Having this 20% on hand will hopefully be useful to both myself and others.

Additional resources

The R community is continually writing new packages and tools. Many of these are covered extensively in various free books available on the bookdown.org website.

The rise of LLM's over the past 2-years has also made it significantly easier to find, refine, and expand on R code. I recommend using them as a first point of call.

Limitations

If you find a bug (along with spelling errors etc) please email me at charlesfcoverdale@gmail.com with the subject line 'RCCD2e'.

About the author

Charles Coverdale is an economist based in Melbourne, Australia. He is passionate about economics, climate science, and building talented teams. You can get in touch with Charles on twitter to hear more about his current projects.

Making maps beautiful

Why use a map

Maps are a great way to communicate data.

They're easily understandable, flexible, and more intuitive than a chart. There's been numerous studies showing that the average reader often struggles to interpret the units on a y-axis, let alone understand trends in scatter or line graphs.

Making maps in R takes some initial investment. However once you have some code you know and understand, spinning up new pieces of analysis can happen in minutes, rather than hours or days.

The aim of this chapter is to get you from 'I can make a map in R' to something more like 'I can conduct spatial analysis and produce a visual which is ready for publication'.

Getting started

First up, we need to load a bunch of mapping packages. The tidyverse package is a classic for just about everything data manipulation, while, the `sf` and `ggspatial` packages are essential for making maps.

```
# Load required packages
library(tidyverse)  # Includes ggplot2, dplyr, tidyr, readxl, purrr
library(ggmap)
library(sf)
library(ggspatial)
library(rlang)
library(broom)
library(Census2016)
library(strayr)
library(absmapsdata)
library(officer)
```

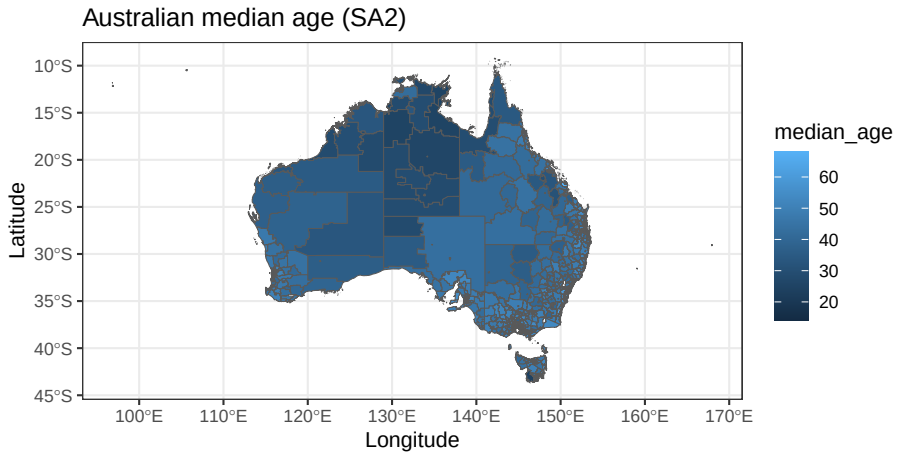
Will Mackey's `absmapsdata` package contains all the ABS' ASGS shapefiles. The data is now also callable through the (thoroughly updated and expanded) `strayr` package. For example:

```
strayr::read_absmap("sa12016")
```

To get a basic demographic map up and running, we will splice together the shapefile (in this case the SA2 map of Austrlaia) and some data from the 2016 Australian Census.

Hugh Parsonage put together a fantastic packaged called `Census2016` which makes downloading this data in a clean format easy.

```
#Get the shapefile form the absmapsdata package (predefined in the list above)

#Get the 2016 census dataset
census2016_wide <- Census2016_wide_by_SA2_year

#Select the key demographic columns from the census data (i.e. the first 8 variables)
census_short <- census2016_wide[,1:8]

#Filter for a single year
census_short_2016 <- census_short %>%
    filter(year==2016)

#Use the inner_join function to get the shapefile and census wide data into a single d
SA2_shp_census_2016 <- inner_join(strayr::read_absmap("sa22016"),census_short_2016,
                          by = c("sa2_name_2016" = "sa2_name"))

#Plot a map that uses census data
  map1 <- ggplot() +
          geom_sf(data = SA2_shp_census_2016, aes(fill = median_age)) +
          ggtitle("Australian median age (SA2)") +
          xlab("Longitude") +
          ylab("Latitude") +
          theme_bw() +
          theme(legend.position = "right")

  map1
```



There we go! A map. This looks 'okay'... but it can be much better.

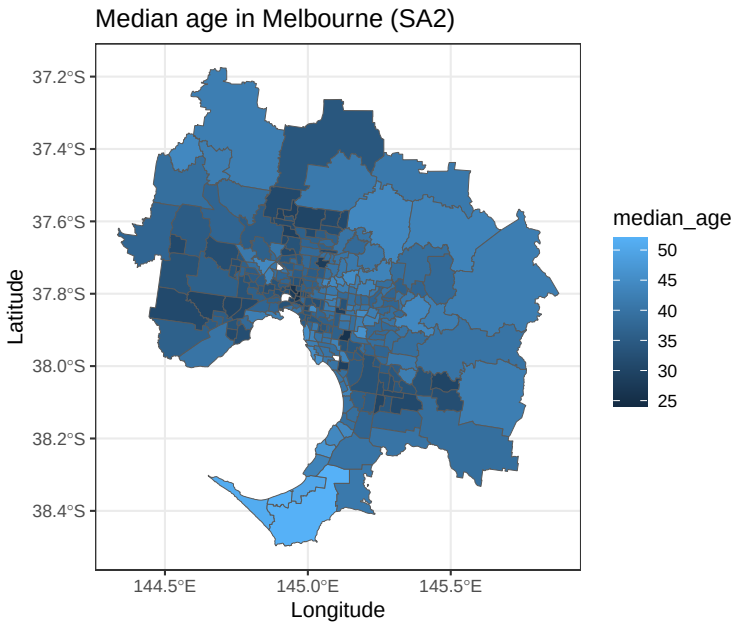
From okay to good

Heat maps don't really show too much interesting data on such a large scale, so let's filter down to Greater Melbourne.

Seeing we have a bunch of census data in our dataframe, we can also do some basic analysis (e.g. population density).

```
#As a bit of an added extra, we can create a new population density column
SA2_shp_census_2016 <- SA2_shp_census_2016 %>%
                       mutate(pop_density=persons/areasqkm_2016)

#Filter for Greater Melbourne
MEL_SA2_shp_census_2016 <- SA2_shp_census_2016 %>%
 filter(gcc_name_2016=="Greater Melbourne")

#Plot the new map just for Greater Melbourne
map2 <- ggplot() +
          geom_sf(data = MEL_SA2_shp_census_2016, aes(fill = median_age, border=NA)) +
          ggtitle("Median age in Melbourne (SA2)") +
          xlab("Longitude") +
          ylab("Latitude") +
```

```
          theme_bw() +
          theme(legend.position = "right")
map2
```



Much better. We can start to see some trends in this map. It looks like younger people tend to live closer to the city center. This seems logical.

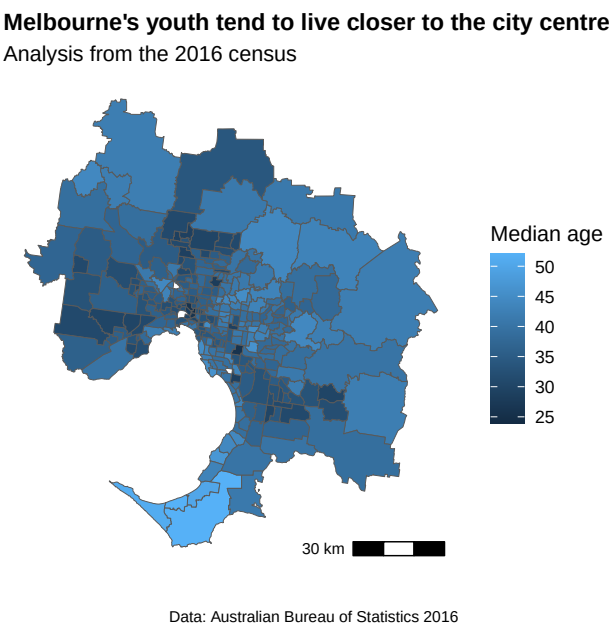
From good to great

The map above is a good start! However, how do we turn this from something 'good', into something that is 100% ready to share?

We see our 'ink to chart ratio' (i.e. the amount of non-data stuff that is on the page) is still pretty high. Is the latitude of Melbourne useful for this analysis...? Not really. Let's get rid of it and the axis labels. A few lines of code adjusting the axis, titles, and theme of the plot will go a long way. Because my geography Professor drilled it into me, I will also add a low-key scale bar.

```
map3 <- ggplot() +
          geom_sf(data = MEL_SA2_shp_census_2016, aes(fill = median_age)) +
          labs(title="Melbourne's youth tend to live closer to the city centre",
               subtitle = "Analysis from the 2016 census",
               caption = "Data: Australian Bureau of Statistics 2016",
```

```
              x="",
              y="",
              fill="Median age") +
          ggspatial::annotation_scale(location="br")+
          theme_minimal() +
            theme(axis.ticks.x = element_blank(),axis.text.x = element_blank())+
            theme(axis.ticks.y = element_blank(),axis.text.y = element_blank())+
            theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())+
            theme(legend.position = "right")+
            theme(plot.title=element_text(face="bold",size=12))+
            theme(plot.subtitle=element_text(size=11))+
            theme(plot.caption=element_text(size=8))

map3
```

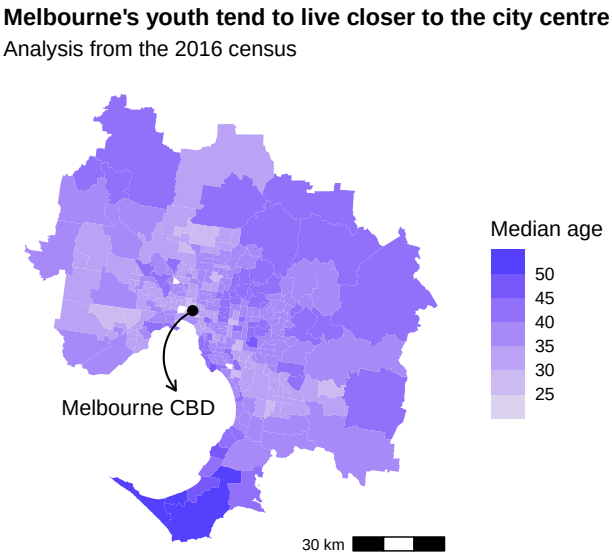


From great to fantastic

The above is perfectly reasonable and looks professionally designed. However, this is where we can get really special.

Let's add a custom colour scheme, drop the boundary edges for the SA2's, and add in a dot and label for Melbourne CBD.

```
#Add in a point for the Melbourne CBD
MEL_location <- data.frame(town_name = c("Melbourne"),
                           x = c(144.9631),
                           y = c(-37.8136))

map4 <- ggplot() +
          geom_sf(data = MEL_SA2_shp_census_2016, aes(fill = median_age),color=NA) +
          geom_point(data=MEL_location,aes(x=x,y=y),size=2,color="black")+
          labs(title="Melbourne's youth tend to live closer to the city centre",
               subtitle = "Analysis from the 2016 census",
               caption = "Data: Australian Bureau of Statistics 2016",
               x="",
               y="",
               fill="Median age") +
          scale_fill_steps(low="#E2E0EB", high="#3C33FE")+
          annotate(geom='curve',
                   x=144.9631,
                   y=-37.8136,
                   xend=144.9,
                   yend=-38.05,
                   curvature=0.5,
                   arrow=arrow(length=unit(2,"mm")))+
          annotate(geom='text',x=144.76,y=-38.1,label="Melbourne CBD")+
          ggspatial::annotation_scale(location="br")+
          theme_minimal() +
            theme(axis.ticks.x = element_blank(),axis.text.x = element_blank())+
            theme(axis.ticks.y = element_blank(),axis.text.y = element_blank())+
            theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank
            theme(legend.position = "right")+
            theme(plot.title=element_text(face="bold",size=12))+
            theme(plot.subtitle=element_text(size=11))+
            theme(plot.caption=element_text(size=8))

map4
```



Now we're talking. A 'client-ready' looking map that can be added to a report, presentation, or with a few tweaks - a digital dashboard.

Make sure to export the map as a high quality PNG using the `ggplot2:ggsave()` function.

Charts

Packages matter

There's exceptional resources online for using the `ggplot2` package and the broader `tidyverse` suite to create production ready charts.

The R Graph Gallery is a great place to start, as is the visual storytelling blogs of The Economist and the BBC.

This chapter contains the code for some of my most used charts and visualization techniques.

```
# Load core packages
library(tidyverse)  # Includes ggplot2, dplyr, tidyr, purrr, and readr
library(lubridate)
library(readxl)
library(scales)

# Load additional visualization libraries
library(ggridges)
library(ggrepel)
library(viridis)
library(patchwork)

# Load additional data manipulation libraries
library(reshape2)
library(gapminder)
```

Make the data tidy

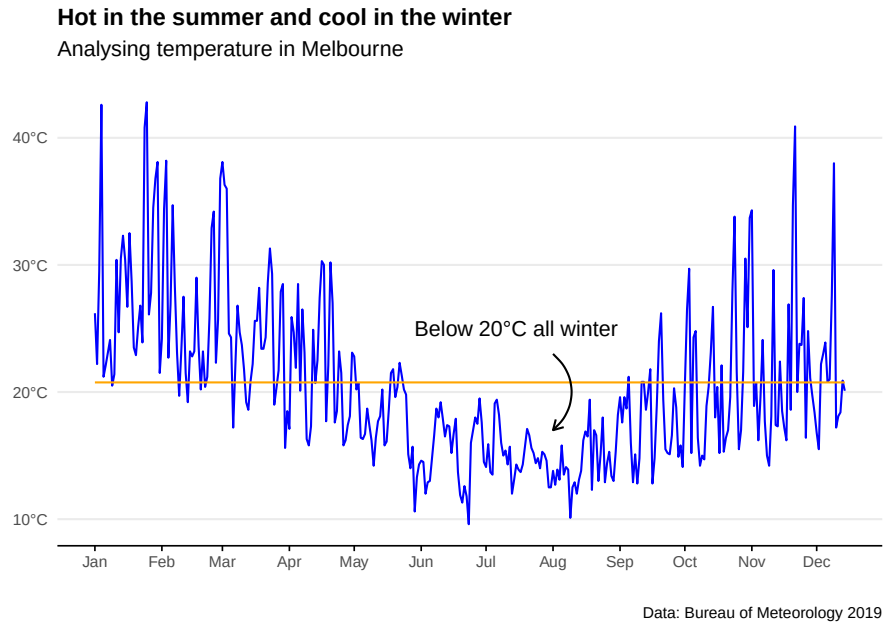
Before making a chart ensure the data is "tidy" - meaning there is a new row for every changed variable. It also doesn't hurt to remove NA's for consistency (particularly in time series).

```
#Read in data
url <-"https://raw.githubusercontent.com/charlescoverdale/ggridges/master/2019_MEL_max

#Read in with read.xlsx
MEL_temp_daily <- openxlsx::read.xlsx(url)

#Remove last 2 characters to just be left with the day number
MEL_temp_daily$Day=substr(MEL_temp_daily$Day,1,nchar(MEL_temp_daily$Day)-2)

#Make a wide format long using the gather function
MEL_temp_daily <- MEL_temp_daily %>%
                    gather(Month,Temp,Jan:Dec)

MEL_temp_daily$Month<-factor(MEL_temp_daily$Month,levels=c("Jan", "Feb", "Mar", "Apr",

#Add in a year
MEL_temp_daily["Year"]=2019

#Reorder
MEL_temp_daily <- MEL_temp_daily[,c(1,2,4,3)]

#Make a single data field using lubridate
MEL_temp_daily <- MEL_temp_daily %>% mutate(Date = make_date(Year, Month, Day))

#Drop the original date columns
MEL_temp_daily <- MEL_temp_daily %>% dplyr::select(Date, Temp) %>% drop_na()

#Add on a 7-day rolling average
MEL_temp_daily <- MEL_temp_daily %>% dplyr::mutate(Seven_day_rolling =
                                                   zoo::rollmean(Temp, k = 7, fill = NA
                                                   Mean = mean(Temp))
#Drop NA's
#MEL_temp_daily <- MEL_temp_daily %>% drop_na()
```

Line plot

```
plot_MEL_temp <- ggplot(MEL_temp_daily, aes(x = Date)) +
  geom_line(aes(y = Temp), col = "blue") +
  geom_line(aes(y = Mean), col = "orange") +
  labs(
    title = "Hot in the summer and cool in the winter",
    subtitle = "Analysing temperature in Melbourne",
```

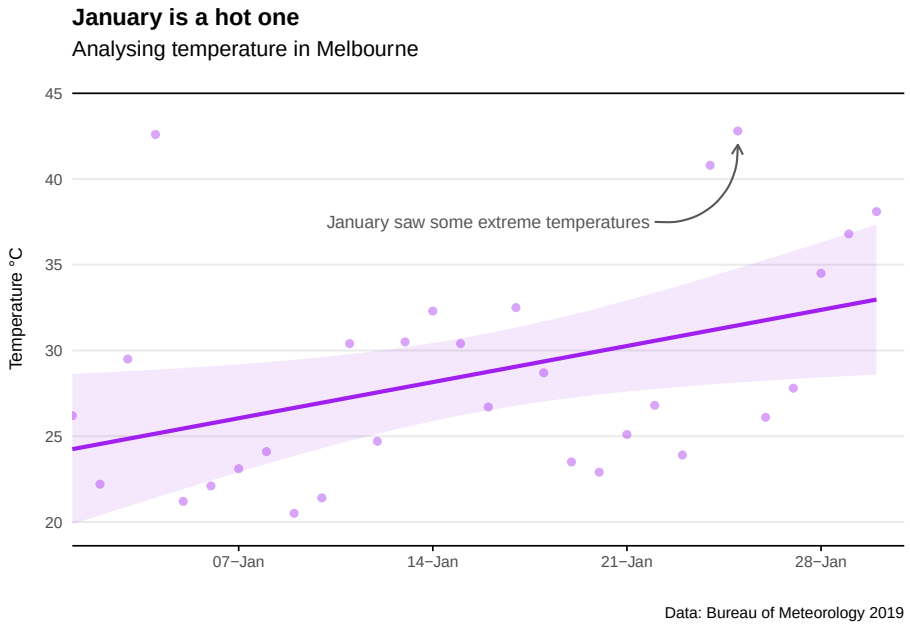
```
    caption = "Data: Bureau of Meteorology 2019",
    x = "",
    y = ""
  ) +
  scale_x_date(
    date_breaks = "1 month",
    date_labels = "%b",
    limits = as.Date(c('2019-01-01', '2019-12-14'))
  ) +
  scale_y_continuous(labels = unit_format(unit = "\u00b0C", sep = "")) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    plot.title = element_text(face = "bold", size = 12),
    plot.subtitle = element_text(size = 11),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 8),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    axis.line.x = element_line(colour = "black", size = 0.4),
    axis.ticks.x = element_line(colour = "black", size = 0.4)
  ) +
  annotate(
    geom = "curve",
    x = as.Date('2019-08-01'), y = 23,
    xend = as.Date('2019-08-01'), yend = 17,
    curvature = -0.5, arrow = arrow(length = unit(2, "mm"))
  ) +
  annotate(
    geom = "text",
    x = as.Date('2019-07-15'), y = 25,
    label = "Below 20°C all winter"
  )

plot_MEL_temp
```



Scatter and trend plot

```
MEL_temp_Jan <- MEL_temp_daily %>% filter(Date < as.Date("2019-01-31"))

ggplot(MEL_temp_Jan, aes(x = Date, y = Temp)) +
  geom_point(col = "purple", alpha = 0.4) +
  geom_smooth(col = "purple", fill = "purple", alpha = 0.1, method = "lm") +
  labs(
    title = "January is a hot one",
    subtitle = "Analysing temperature in Melbourne",
    caption = "Data: Bureau of Meteorology 2019",
    x = "",
    y = "Temperature °C"
  ) +
  scale_x_date(
    date_breaks = "1 week",
    date_labels = "%d-%b",
    limits = as.Date(c('2019-01-01', '2019-01-31')),
    expand = c(0, 0)
  ) +
  geom_hline(yintercept = 45, colour = "black", size = 0.4) +
  annotate(
```

```
    geom = "curve",
    x = as.Date('2019-01-22'), y = 37.5,
    xend = as.Date('2019-01-25'), yend = 42,
    curvature = 0.5,
    col = "#575757",
    arrow = arrow(length = unit(2, "mm"))
  ) +
  annotate(
    geom = "text",
    x = as.Date('2019-01-16'), y = 37.5,
    label = "January saw some extreme temperatures",
    size = 3.2, col = "#575757"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 12),
    plot.subtitle = element_text(size = 11),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 8),
    axis.title.y = element_text(size = 9, margin = margin(r = 10)),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    axis.line.x = element_line(colour = "black", size = 0.4),
    axis.ticks.x = element_line(colour = "black", size = 0.4)
  )
```

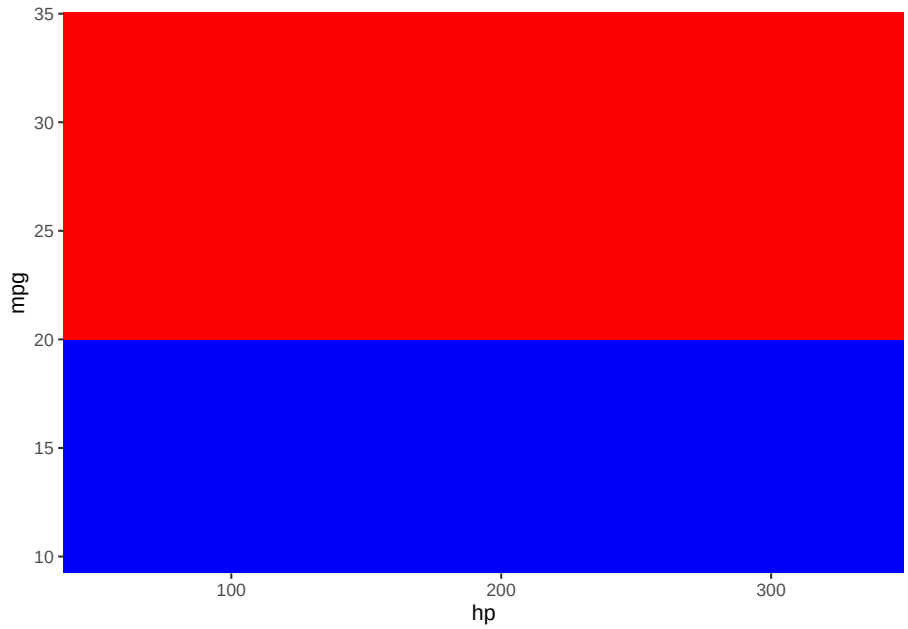


Shading areas on plots

Adding shading behind a plot area is simple using `geom_rect`.

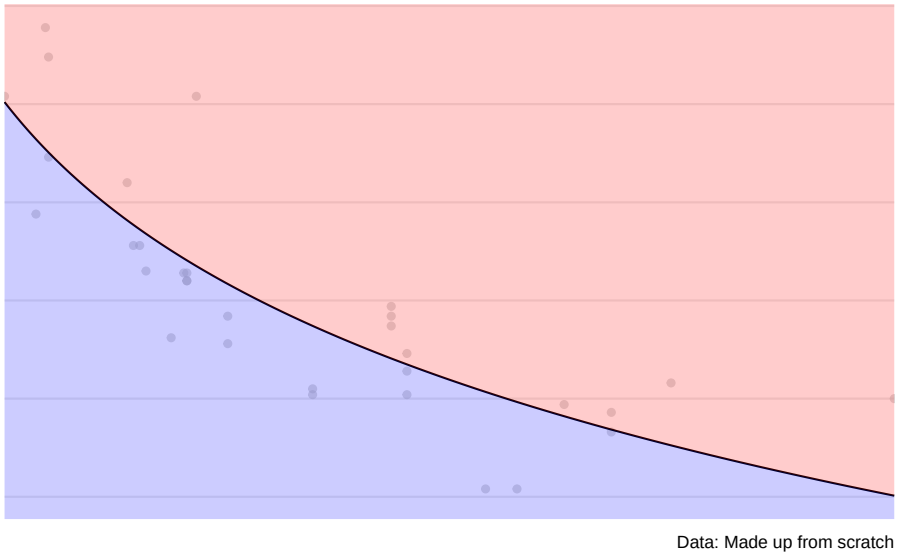
Adding shading under a particular model line? A little trickier. See both example below.

```
# Example 1: Custom y-axis threshold shading
threshold <- 20

ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  geom_hline(yintercept = threshold) +

  # Shade areas below and above threshold
  geom_rect(xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = threshold,
            fill = "blue", alpha = 0.2) +
  geom_rect(xmin = -Inf, xmax = Inf, ymin = threshold, ymax = Inf,
            fill = "red", alpha = 0.2)
```




```r
# Example 2: Model line with shaded areas
model <- lm(mpg ~ log(hp), data = mtcars)

# Generate predicted values for plotting
df_line <- data.frame(hp = seq(min(mtcars$hp), max(mtcars$hp), by = 1))
df_line$mpg <- predict(model, newdata = df_line)

# Define polygons above and below the model line
df_poly_under <- bind_rows(df_line, tibble(hp = c(max(df_line$hp), min(df_line$hp)), mpg = c(-Inf
df_poly_above <- bind_rows(df_line, tibble(hp = c(max(df_line$hp), min(df_line$hp)), mpg = c(Inf,

# Plot the data, model line, and shaded areas
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point(color = "grey", alpha = 0.5) +
  geom_line(data = df_line, aes(x = hp, y = mpg), color = "black") +

  # Shaded areas
  geom_polygon(data = df_poly_under, aes(x = hp, y = mpg), fill = "blue", alpha = 0.2) +
  geom_polygon(data = df_poly_above, aes(x = hp, y = mpg), fill = "red", alpha = 0.2) +

  scale_x_continuous(expand = c(0, 0)) +

  labs(
    title = "Look at that snazzy red/blue shaded area",
    subtitle = "Subtitle goes here",
```

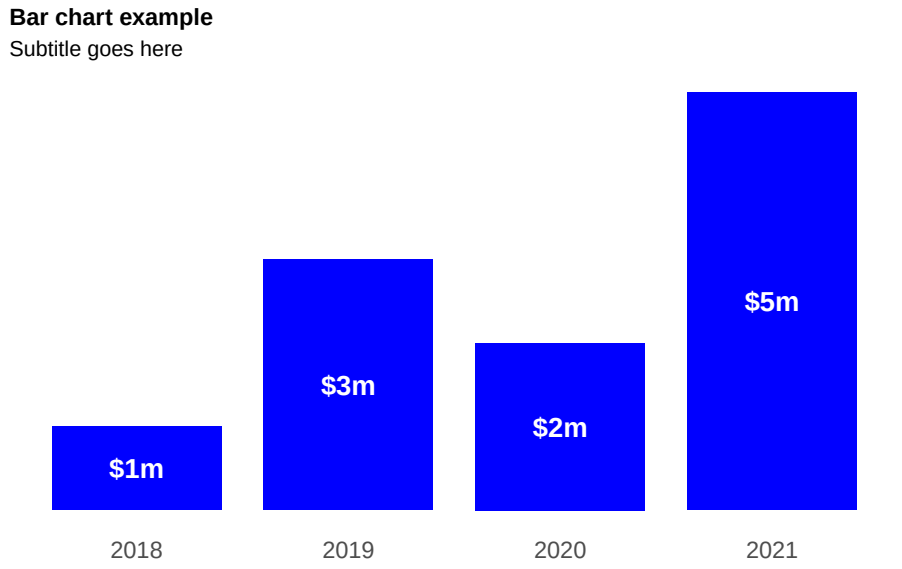
```
    caption = "Data: Made up from scratch",
    x = "",
    y = ""
  ) +

  theme_minimal() +
  theme(
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    axis.title = element_blank(),
    axis.text = element_blank(),
    axis.ticks = element_blank()
  )
```



Bar chart

```
# Create data frame directly
bar_data_single <- data.frame(
  Year = c("2018", "2019", "2020", "2021"),
  Value = c(1000000, 3000000, 2000000, 5000000)
)
```

```r
ggplot(bar_data_single, aes(x = Year, y = Value)) +
  geom_bar(stat = "identity", fill = "blue", width = 0.8) +

  # Labels in the middle of the bars
  geom_text(aes(y = Value / 2, label = scales::dollar(Value, scale = 1/1e6, suffix = "m")),
            size = 5, color = "white", fontface = "bold") +

  labs(
    title = "Bar chart example",
    subtitle = "Subtitle goes here",
    caption = "Data: Made up from scratch",
    x = "",
    y = ""
  ) +

  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 12),
    plot.subtitle = element_text(size = 11),
    plot.caption = element_text(size = 12),
    axis.text = element_text(size = 12),
    panel.grid.minor = element_blank(),
    panel.grid.major = element_blank(),
    axis.title.y = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank()
  )
```

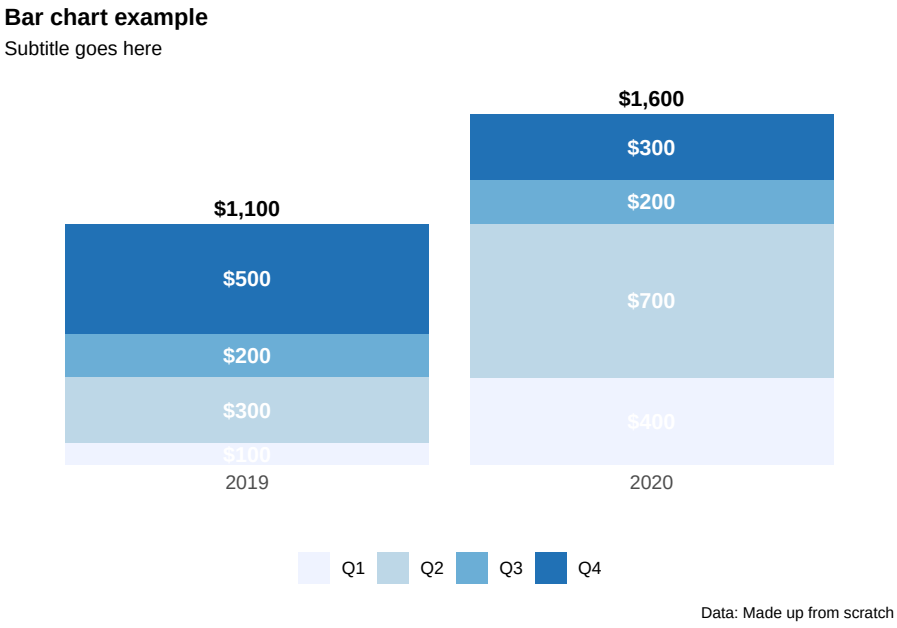



```
# Save the plot
# ggsave("test.png", width = 10, height = 10, units = "cm", dpi = 600)
```

Stacked bar chart

```
Year = c("2019", "2019", "2019", "2019", "2020", "2020", "2020", "2020")

Quarter = c("Q1","Q2","Q3", "Q4","Q1","Q2","Q3", "Q4")

Value = (c(100,300,200,500,400,700,200,300))

bar_data <- (cbind(Year, Quarter, Value))

bar_data <- as.data.frame(bar_data)

bar_data$Value = as.integer(bar_data$Value)

bar_data_totals <- bar_data %>%
                           dplyr::group_by(Year) %>%
                           dplyr:: summarise(Total = sum(Value))
```

```
ggplot(bar_data, aes(x = Year, y = Value, fill = (Quarter), label=Value)) +
  geom_bar(position = position_stack(reverse=TRUE),stat='identity')+
  geom_text(size = 4,
            col="white",
            fontface="bold",
            position = position_stack(reverse=TRUE,vjust = 0.5),
            label=scales::dollar(Value))+
  geom_text(aes(Year, Total,
                label=scales::dollar(Total),
                fill = NULL,
                vjust=-0.5),
                fontface="bold",
                size=4,
                data = bar_data_totals)+

  scale_fill_brewer(palette = "Blues") +

  labs(title="Bar chart example",
                          subtitle = "Subtitle goes here",
                          caption = "Data: Made up from scratch",
                          x="",
                          y="Units") +

  theme_minimal() +

  theme(legend.position = "bottom")+
  theme(legend.title = element_blank())+

  theme(plot.title=element_text(face="bold",size=12))+
  theme(plot.subtitle=element_text(size=10))+
  theme(plot.caption=element_text(size=8))+


  theme(axis.text=element_text(size=10))+
  theme(panel.grid.minor = element_blank())+
  theme(panel.grid.major.x = element_blank())+
  theme(panel.grid.major.y = element_blank()) +

  scale_y_continuous(expand=c(0,0),limits=c(0,1800))+

  theme(axis.title.y=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank())
```



Histogram

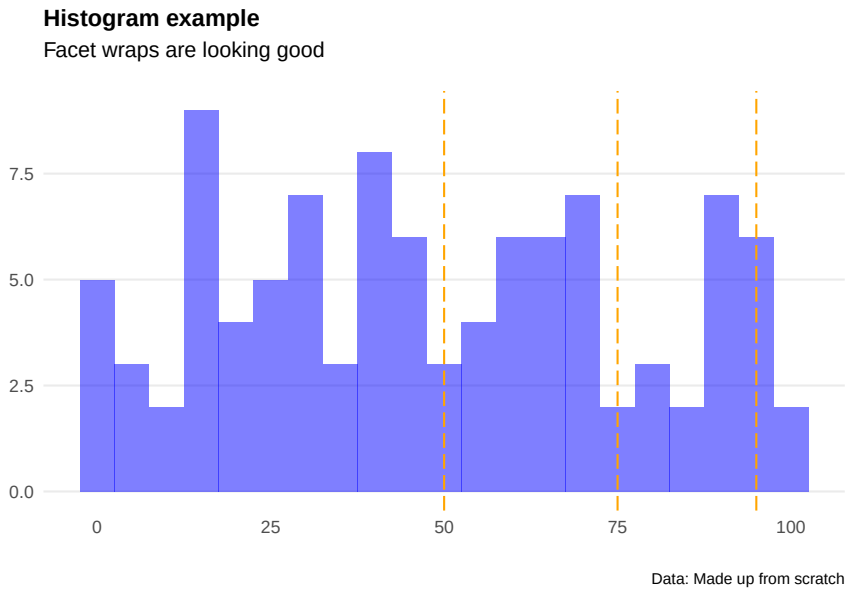
Aka. a bar chart for a continuous variable where the bars are touching. Useful to show distribution of time series or ordinal variables.

```
c("0-20", "20-40", "40-60", "60-80", "80+")
```

```
## [1] "0-20"  "20-40" "40-60" "60-80" "80+"
```

```
#Create a data set
hist_data <- data.frame(X1=sample(0:100,100,rep=TRUE))

            ggplot(hist_data)+

            geom_histogram(aes(x=X1),binwidth=5,fill="blue",alpha=0.5) +

            geom_vline(xintercept=c(50,75,95),yintercept=0,linetype="longdash",col="ora

            labs(title="Histogram example",
                         subtitle = "Facet wraps are looking good",
                         caption = "Data: Made up from scratch",
                         x="",
```

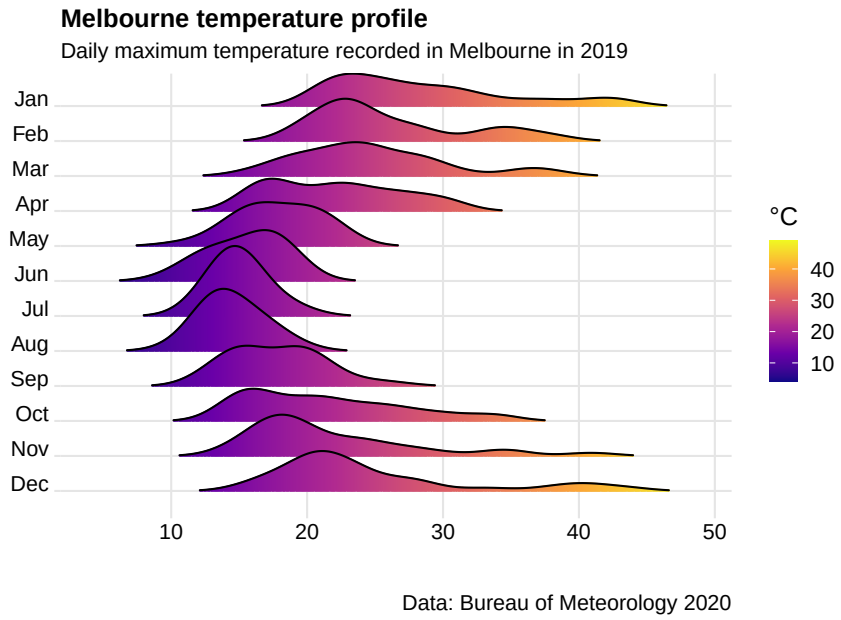
```
                    y="") +


          theme_minimal() +

          theme(panel.spacing.x = unit(10, "mm"))+

          theme(legend.position="none")+

          theme(plot.title=element_text(face="bold",size=12))+
          theme(plot.subtitle=element_text(size=11))+
          theme(plot.caption=element_text(size=8))+

          theme(axis.text=element_text(size=9))+
          theme(panel.grid.minor = element_blank())+
          theme(panel.grid.major.x = element_blank()) +

          theme(plot.subtitle = element_text(margin=margin(0,0,15,0))) +

          theme(plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"))
```



Ridge chart

Handy when working with climate variables. Particularly useful at showing the difference in range of multiples series (e.g. temperature by month).

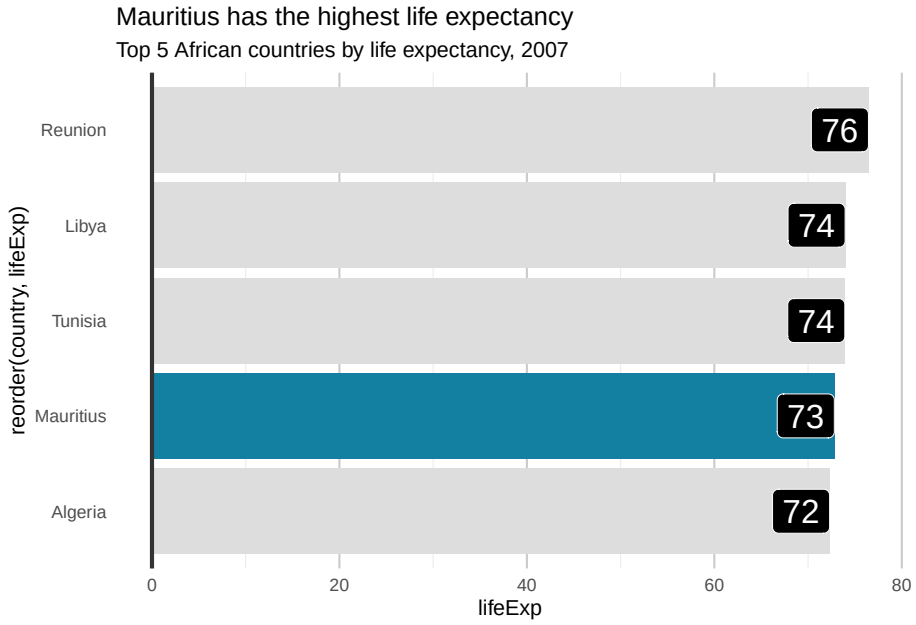
```
# Import data
url <-"https://raw.githubusercontent.com/charlescoverdale/ggridges/master/2019_MEL_max

MEL_temp_daily <- openxlsx::read.xlsx(url)

# Remove last 2 characters to just be left with the day number
MEL_temp_daily$Day=substr(MEL_temp_daily$Day,1,nchar(MEL_temp_daily$Day)-2)

# Make a wide format long using the gather function
MEL_temp_daily <- MEL_temp_daily %>%
                    gather(Month,Temp,Jan:Dec)

MEL_temp_daily$Month<-factor(MEL_temp_daily$Month,levels=c("Jan", "Feb", "Mar", "Apr",

# Plot
ggplot(MEL_temp_daily,
                           aes(x = Temp, y = Month, fill = stat(x))) +
                      geom_density_ridges_gradient(scale =2,
                                                   size=0.3,
                                                   rel_min_height = 0.01,
                                                   gradient_lwd = 1.) +
                      scale_y_discrete(limits = unique(rev(MEL_temp_daily$Month)))+
                      scale_fill_viridis_c(name = "°C", option = "C") +
                      labs(title = 'Melbourne temperature profile',
                      subtitle = 'Daily maximum temperature recorded in Melbourne in
                      caption = "Data: Bureau of Meteorology 2020") +
                      xlab(" ")+
                      ylab(" ")+
                      theme_ridges(font_size = 13, grid = TRUE)
```



BBC style: Bar charts (categorical)

```
#devtools::install_github('bbc/bbplot')
library(gapminder)
library(bbplot)

# Prepare data
bar_df <- gapminder %>%
  filter(year == 2007 & continent == "Africa") %>%
  arrange(desc(lifeExp)) %>%
  head(5)

# Make plot
bars <- ggplot(bar_df, aes(x = reorder(country, lifeExp), y = lifeExp, fill = country == "Mauriti
  geom_bar(stat="identity", position="identity") +
  geom_hline(yintercept = 0, size = 1, colour="#333333") +
  scale_fill_manual(values = c("TRUE" = "#1380A1", "FALSE" = "#dddddd")) +
  labs(title = "Mauritius has the highest life expectancy",
       subtitle = "Top 5 African countries by life expectancy, 2007") +
  coord_flip() +
  theme_minimal() +
  theme(panel.grid.major.x = element_line(color="#cbcbcb"),
```

```
        panel.grid.major.y = element_blank(),
        legend.position = "none")

# Add labels
labelled_bars <- bars +
  geom_label(aes(label = round(lifeExp, 0)),
             hjust = 1, vjust = 0.5, colour = "white",
             fill = "black", label.size = 0.2,
             family = "Helvetica", size = 6)

labelled_bars
```

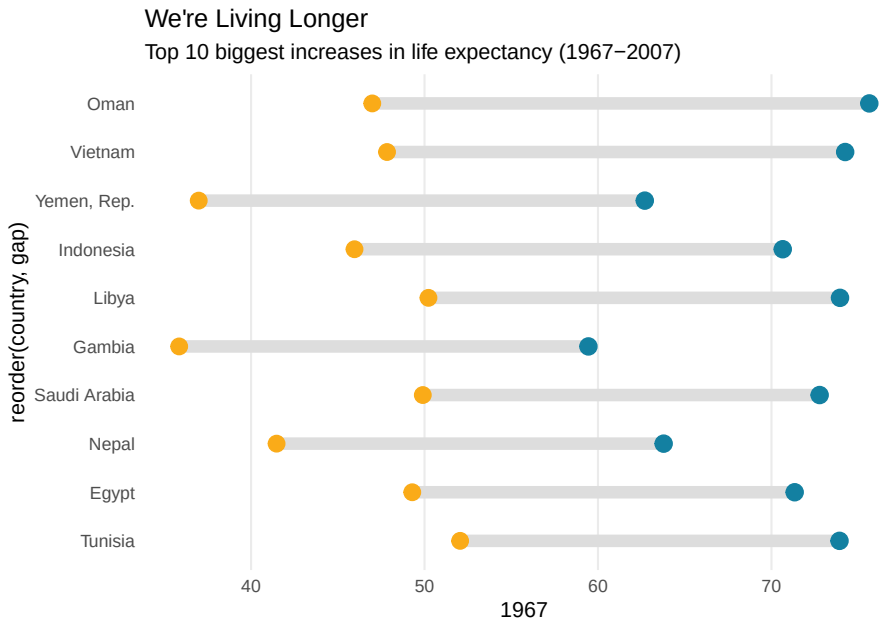


BBC style: Dumbbell charts

Dumbbell charts are handy instead of using clustered column charts with janky thinkcell labels and arrows to show the difference between the columns. Note it relies on having a 4 variable input (variable_name, value1, value2, and gap).

The `geom_dumbell`function lives inside the `ggalt` package rather than the standard ggplot2.

```r
library(ggalt)     # For geom_dumbbell
library(gapminder)
library(bbplot)    # Uncomment if using bbc_style()

# Prepare data
dumbbell_df <- gapminder %>%
  filter(year %in% c(1967, 2007)) %>%
  select(country, year, lifeExp) %>%
  pivot_wider(names_from = year, values_from = lifeExp) %>%
  mutate(gap = `2007` - `1967`) %>%
  arrange(desc(gap)) %>%
  head(10)

# Make plot
ggplot(dumbbell_df, aes(x = `1967`, xend = `2007`, y = reorder(country, gap), group = country)) +
  geom_dumbbell(colour = "#dddddd",
                size = 3,
                colour_x = "#FAAB18",
                colour_xend = "#1380A1") +
  labs(title = "We're Living Longer",
       subtitle = "Top 10 biggest increases in life expectancy (1967-2007)") +
  theme_minimal() +
  theme(panel.grid.major.y = element_blank(),
        panel.grid.minor = element_blank())
```



Facet wraps

Handy rather than showing multiple lines on the same chart.

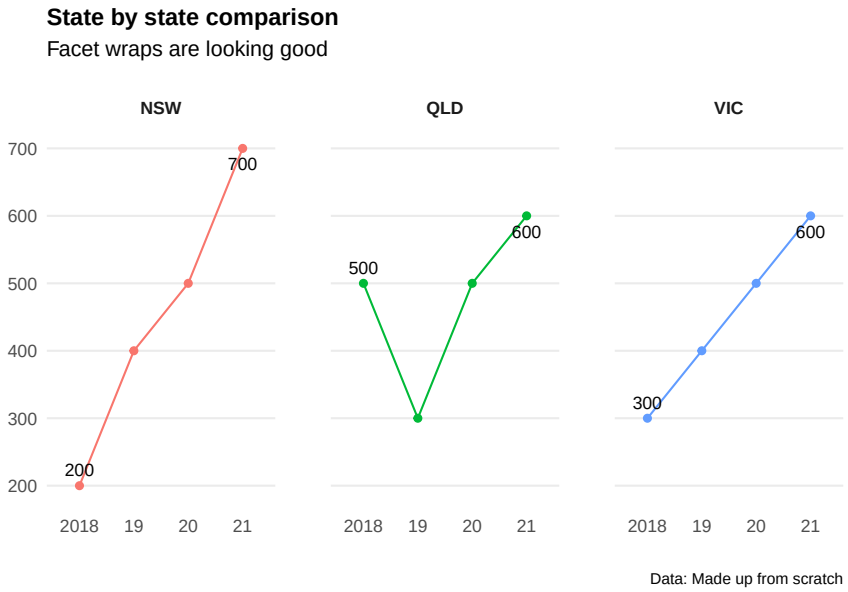
Top tips: `facet_wrap()`dataframes need to be in long form in order to be manipulated easily.

It also helps to add on separate columns for the start and end values (if you want to add data point labels).

```
#Create a data set
Year = c("2018", "2019", "2020", "2021")

QLD = (c(500,300, 500, 600))

NSW = (c(200,400, 500, 700))

VIC = (c(300,400, 500, 600))

#Combine the columns into a single dataframe
facet_data <- (cbind(Year, QLD,NSW,VIC))
facet_data <- as.data.frame(facet_data)

#Change formats to integers
facet_data$QLD = as.integer(facet_data$QLD)
facet_data$NSW = as.integer(facet_data$NSW)
facet_data$VIC = as.integer(facet_data$VIC)

#Make the wide data long
facet_data_long <- pivot_longer(facet_data,!Year, names_to="State", values_to="Value")

facet_data_long <- facet_data_long %>%
                   dplyr::mutate(start_label =
                          if_else(Year == min(Year),
                          as.integer(Value), NA_integer_))

facet_data_long <- facet_data_long %>%
                   dplyr::mutate(end_label =
                          if_else(Year == max(Year),
                          as.integer(Value), NA_integer_))



#Make the base line chart
base_chart <-   ggplot() +
```

```
                geom_line(data=facet_data_long,
                aes(x = Year,
                y = Value,
                group = State,
                colour = State)) +

                geom_point(data=facet_data_long,
                aes(x = Year,
                y = Value,
                group = State,
                colour = State)) +

                ggrepel::geom_text_repel(data=facet_data_long,
                aes(x = Year,
                y = Value,
                label = end_label),
                color = "black",
                nudge_y = -10,size=3) +

                ggrepel::geom_text_repel(data=facet_data_long,
                aes(x = Year,
                y = Value,
                label = start_label),
                color = "black",
                nudge_y = 10,size=3)


base_chart +

            scale_x_discrete(
            breaks = seq(2018, 2021, 1),
            labels = c("2018", "19", "20", "21"))+

            facet_wrap(State ~ .) +
                        #To control the grid arrangement, we can add in customer dimensions
                        #ncol = 2, nrow=2) +

            labs(title="State by state comparison",
                          subtitle = "Facet wraps are looking good",
                          caption = "Data: Made up from scratch",
                          x="",
                          y="") +


            theme_minimal() +
```

```
        theme(strip.text.x = element_text(size = 9, face = "bold")) +

        theme(panel.spacing.x = unit(10, "mm"))+

        theme(legend.position="none")+

        theme(plot.title=element_text(face="bold",size=12))+
        theme(plot.subtitle=element_text(size=11))+
        theme(plot.caption=element_text(size=8))+

        theme(axis.text=element_text(size=9))+
        theme(panel.grid.minor = element_blank())+
        theme(panel.grid.major.x = element_blank()) +

        theme(plot.subtitle = element_text(margin=margin(0,0,15,0))) +

        theme(plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"))
```

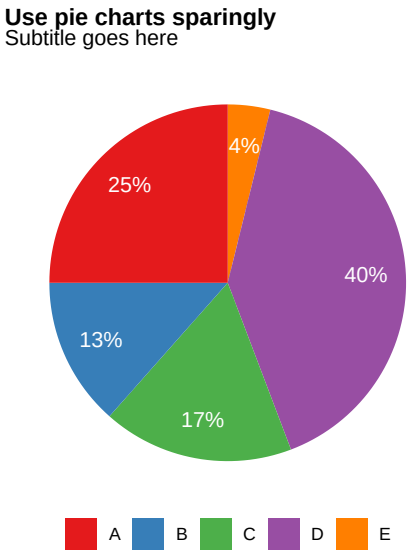


Pie chart

These should be used sparingly... but they are handy for showing proportions when the proportion of the whole is paramount (e.g. 45%) - rather than the

proportion in relation to another data point (e.g. 16% one year vs 18% the next).

```
# Create Data
pie_data <- data.frame(
  group=LETTERS[1:5],
  value=c(13,7,9,21,2))

# Compute the position of labels
pie_data <- pie_data %>%
  arrange(desc(group)) %>%
  mutate(proportion = value / sum(pie_data$value) *100) %>%
  mutate(ypos = cumsum(proportion)- 0.5*proportion )

# Basic piechart
ggplot(pie_data, aes(x="", y=proportion, fill=group)) +
  geom_bar(stat="identity")+
  coord_polar("y", start=0) +
  theme_void()+

  geom_text(aes(y = ypos,
                label = paste(round(proportion,digits=0),"%", sep = ""),x=1.25),
                color = "white",
                size=4) +

  scale_fill_brewer(palette="Set1")+

  labs(title="Use pie charts sparingly",
                          subtitle = "Subtitle goes here",
                          caption = "",
                          x="",
                          y="") +

  theme(legend.position = "bottom")+
  theme(legend.title = element_blank())+

  theme(plot.title=element_text(face="bold",size=12))+
  theme(plot.subtitle=element_text(size=11))+
  theme(plot.caption=element_text(size=8))+

  theme(plot.subtitle = element_text(margin=margin(0,0,5,0))) +
  theme(plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"))
```



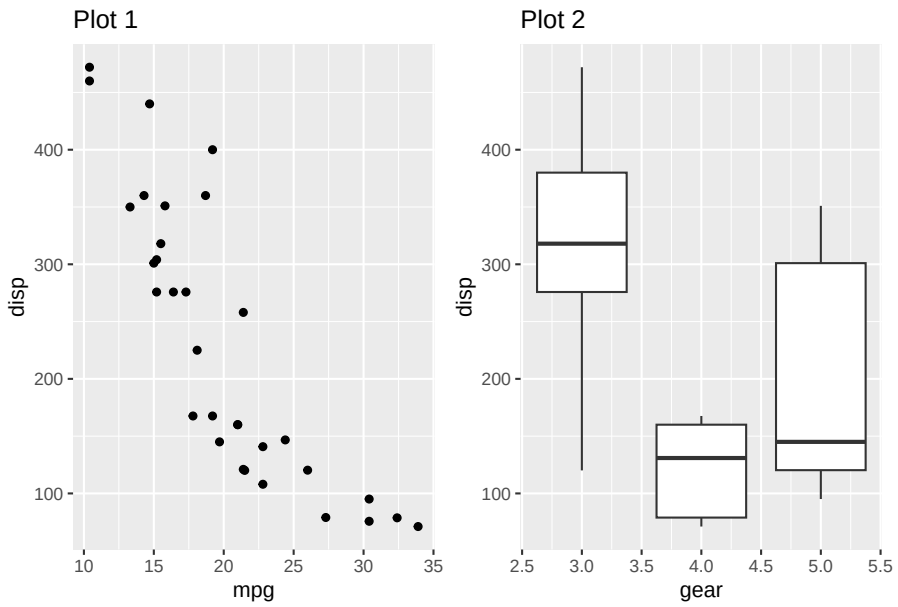

```
  # ggsave(plot=last_plot(),
  #       width=10,
  #       height=10,
  #       units="cm",
  #       dpi = 600,
  #       filename = "/Users/charlescoverdale/Desktop/pietest.png")
```

Patchwork

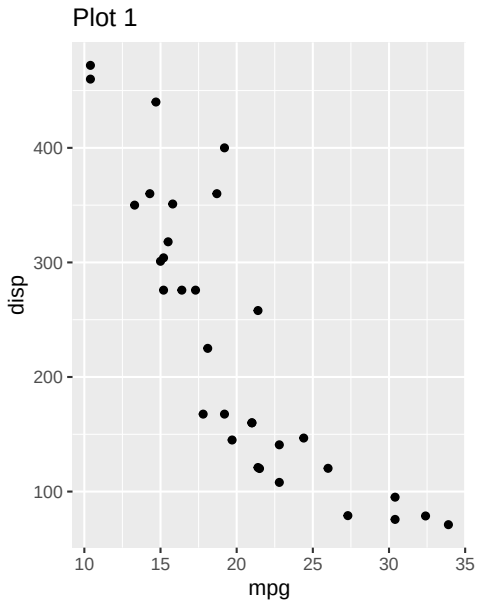
Patchwork is a nifty package for arranging plots and other graphic elements (text, tables etc) in different grid arrangements. The basic syntax is to use `plot1 | plot2` for side by side charts, and `plot1 / plot2`for top and bottom charts. You can also combine these two functions for a grid of different size columns (e.g. `plot3 / (plot1 | plot2)`

```
#Make some simply plots using the mtcars package
p1 <- ggplot(mtcars) +
  geom_point(aes(mpg, disp)) +
  ggtitle('Plot 1')

p2 <- ggplot(mtcars) +
  geom_boxplot(aes(gear, disp, group = gear)) +
  ggtitle('Plot 2')
```

```
#Example of side by side charts
library(patchwork)
p1 + p2
```




```
#Add in a table next to the plot
p1 + gridExtra::tableGrob(mtcars[1:10, c('mpg', 'disp')])
```



	mpg	disp
Mazda RX4	21	160
Mazda RX4 Wag	21	160
Datsun 710	22.8	108
Hornet 4 Drive	21.4	258
Hornet Sportabout	18.7	360
Valiant	18.1	225
Duster 360	14.3	360
Merc 240D	24.4	146.7
Merc 230	22.8	140.8
Merc 280	19.2	167.6

Saving to powerpoint

There's a bunch of ways to save ggplot graphics - but the way I find most useful is by exporting to pptx in a common 'charts' directory.

If you want to save as a png you can use the normal ggsave function - however it will not be editable (e.g. able to click and drag to rescale for a presentation).

Therefore instead we can use the grattantheme package to easily save to an editable pptx graphic.

Note: The code below has been commented out so that is will upload to bookdown.org without an error.

```
#The classic save function to png

#       ggsave(plot = ggplot2::last_plot(),
#       width = 8,
#       height = 12,
#       dpi = 600,
#       filename = "/Users/charlescoverdale/Desktop/test.png")

#Using the grattantheme package to easily safe to powerpoint

#        grattan_save_pptx(p = ggplot2::last_plot(),
```

```
#         "/Users/charlescoverdale/Desktop/test.pptx",
#         type = "wholecolumn")
```

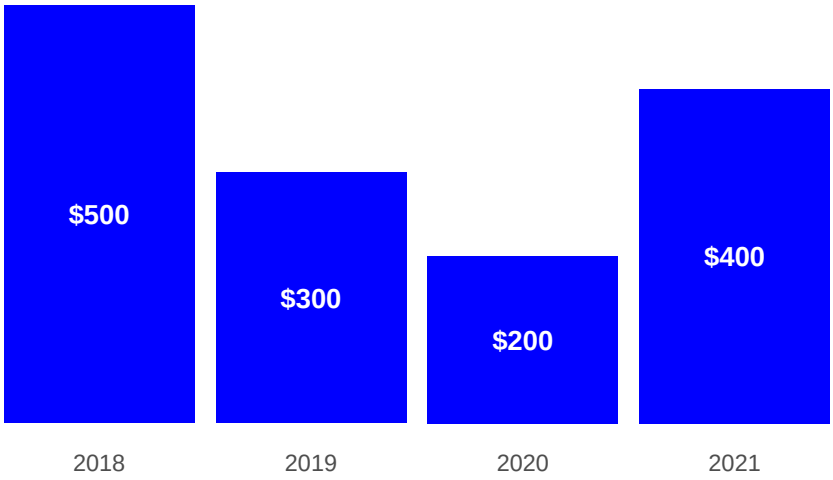
Automating chart creation

Let's say we have a dataframe of multiple variables. We want to produce simple charts of the same style for each variable (including formatting and titles etc). Sure we can change the `aes(x=)`and `aes(y=)` variables in ggplot2 manually for each column - but this is time intensive especially for large data frames. Instead, we can write a function that will loop through the whole dataframe and produce the same format of chart.

```
#Create a data set
Year = c("2018", "2019", "2020", "2021")

Variable1 = (c(500,300, 200, 400))

Variable2 = (c(200,400, 200, 700))

Variable3 = (c(300,500, 800, 1000))

#Combine the columns into a single dataframe
bar_data_multiple <- (cbind(Year, Variable1, Variable2, Variable3))
bar_data_multiple <- as.data.frame(bar_data_multiple)

#Change formats to integers
bar_data_multiple$Variable1 = as.integer(bar_data_multiple$Variable1)
bar_data_multiple$Variable2 = as.integer(bar_data_multiple$Variable2)
bar_data_multiple$Variable3 = as.integer(bar_data_multiple$Variable3)

#Define a function
loop <- function(chart_variable) {

ggplot(bar_data_multiple, aes(x = Year,
                              y = .data[[chart_variable]],
                              label = .data[[chart_variable]]))+

  geom_bar(stat='identity',fill="blue")+

  geom_text(aes(
            label = scales::dollar(.data[[chart_variable]])),
            size = 5,
            col="white",
```

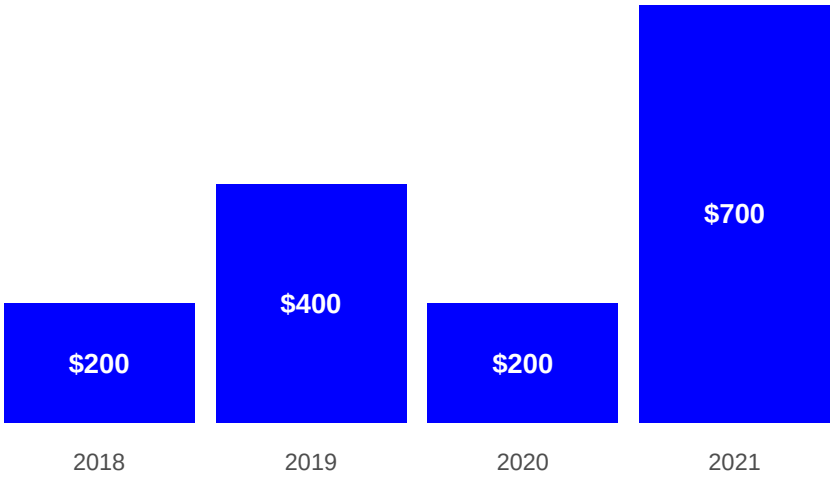
```
            fontface="bold",
            position = position_stack(vjust = 0.5))+


  labs(title=paste("Company X: ",
                   chart_variable,
                   " (",head(Year,n=1),
                   " - ",
                   tail(Year,n=1),
                   ")",
                   sep=""),
       subtitle = "Subtitle goes here",
       caption = "Data: Made up from scratch",
        x="",
        y="") +

  theme_minimal() +

  theme(plot.title=element_text(face="bold",size=12))+
  theme(plot.subtitle=element_text(size=11))+
  theme(plot.caption=element_text(size=12))+

  theme(axis.text=element_text(size=12))+
  theme(panel.grid.minor = element_blank())+
  theme(panel.grid.major.x = element_blank())+
  theme(panel.grid.major.y = element_blank()) +

  theme(axis.title.y=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank())
}

plots <- purrr::map(colnames(bar_data_multiple)[colnames(bar_data_multiple) != "Year"]
plots
```

Company X: Variable1 (2018 − 2021)
Subtitle goes here

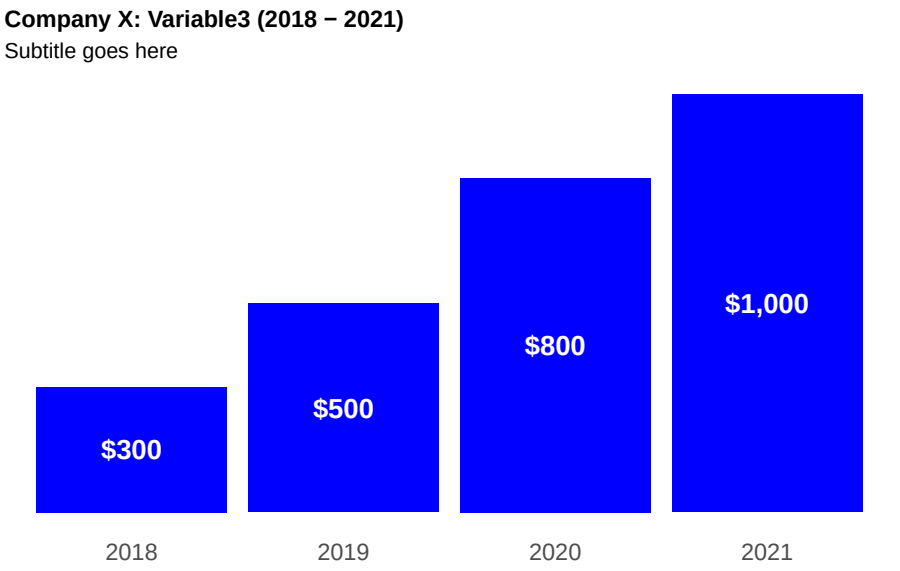


Data: Made up from scratch

Company X: Variable2 (2018 − 2021)
Subtitle goes here



Data: Made up from scratch




```
#cowplot::plot_grid(plotlist = plots)
```

Basic modelling

Creating a model is an essential part of forecasting and data analysis.

I've put together a quick guide on my process for modelling data and checking model fit.

The source data I use in this example is Melbourne's weather record over a 12 month period. Daily temperature is based on macroscale weather and climate systems, however many observable measurements are correlated (i.e. hot days tend to have lots of sunshine). This makes using weather data great for model building.

Source, format, and plot data

Before we get started, it is useful to have some packages up and running.

```
#Useful packages for regression
library(readr)
library(readxl)
library(ggplot2)
library(tidyverse)
library(lubridate)
library(modelr)
library(cowplot)
```

I've put together a csv file of weather observations in Melbourne in 2019. We begin our model by downloading the data from Github.

```
#Input data
link <- "data/MEL_weather_2019.csv"

# We'll read this data in as a dataframe
# The 'check.names' function set to false means the funny units that the BOM use for column names
```

```
MEL_weather_2019 <- read.csv(link, check.names = F)

head(MEL_weather_2019)
```

This data is relatively clean. One handy change to make is to make the date into a dynamic format (to easily switch between months, years, etc).

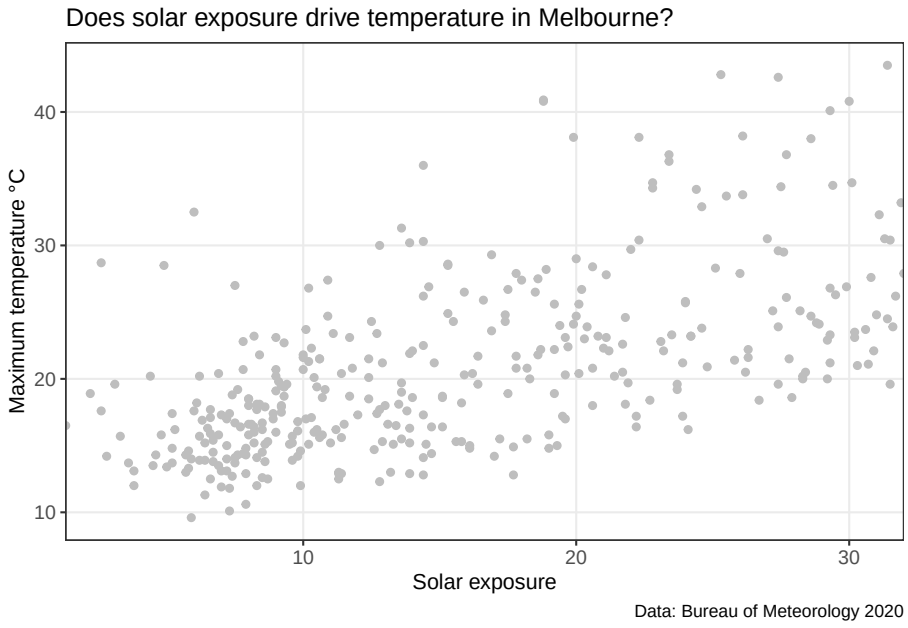
```
#Add a proper date column
MEL_weather_2019 <- MEL_weather_2019 %>%
  mutate(Date = make_date(Year, Month, Day))
```

We also notice that some of the column names have symbols in them. This can be tricky to work with, so let's rename some columns into something more manageable.

```
#Rename key df variables
names(MEL_weather_2019)[4]<- "Solar_exposure"
names(MEL_weather_2019)[5]<- "Rainfall"
names(MEL_weather_2019)[6]<- "Max_temp"

head(MEL_weather_2019)
```

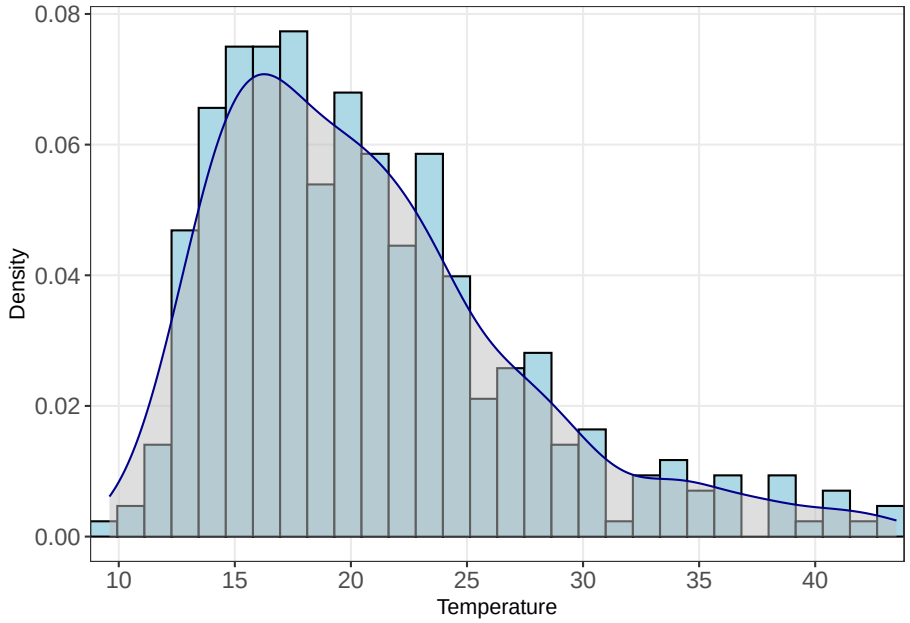
We're aiming to investigate if other weather variables can predict maximum temperatures. Solar exposure seems like a plausible place to start. We start by plotting the two variables to if there is a trend.

```
#Plot the data
MEL_temp_investigate <- ggplot(MEL_weather_2019)+
  geom_point(aes(y=Max_temp, x=Solar_exposure),col="grey")+
  labs(title = "Does solar exposure drive temperature in Melbourne?",
  caption = "Data: Bureau of Meteorology 2020") +
  xlab("Solar exposure")+
  ylab("Maximum temperature °C")+
  scale_x_continuous(expand=c(0,0))+
  theme_bw()+
  theme(axis.text=element_text(size=10))+
  theme(panel.grid.minor = element_blank())

MEL_temp_investigate
```



Eyeballing the chart above, there seems to be a correlation between the two data sets. We'll do one more quick plot to analyse the data. What is the distribution of temperature?

```
ggplot(MEL_weather_2019, aes(x=Max_temp)) +
 geom_histogram(aes(y=..density..), colour="black", fill="lightblue")+
 geom_density(alpha=.5, fill="grey",colour="darkblue")+

 scale_x_continuous(breaks=c(5,10,15,20,25,30,35,40,45),
                    expand=c(0,0))+
  xlab("Temperature")+
  ylab("Density")+

 theme_bw()+
 theme(axis.text=element_text(size=12))+
 theme(panel.grid.minor = element_blank())
```



We can see here the data is right skewed (i.e. the mean will be greater than the median). We'll need to keep this in mind. Let's start building a model.

Build a linear model

We start by looking whether a simple linear regression of solar exposure seems to be correlated with temperature. In R, we can use the linear model (lm) function.

```
#Create a straight line estimate to fit the data
temp_model <- lm(Max_temp~Solar_exposure, data=MEL_weather_2019)
```

Analyse the model fit

Let's see how well solar exposure explains changes in temperature

```
#Call a summary of the model
summary(temp_model)
```

The adjusted R squared value (one measure of model fit) is 0.3596. Furthermore the coefficient of our solar_exposure variable is statistically significant.

Compare the predicted values with the actual values

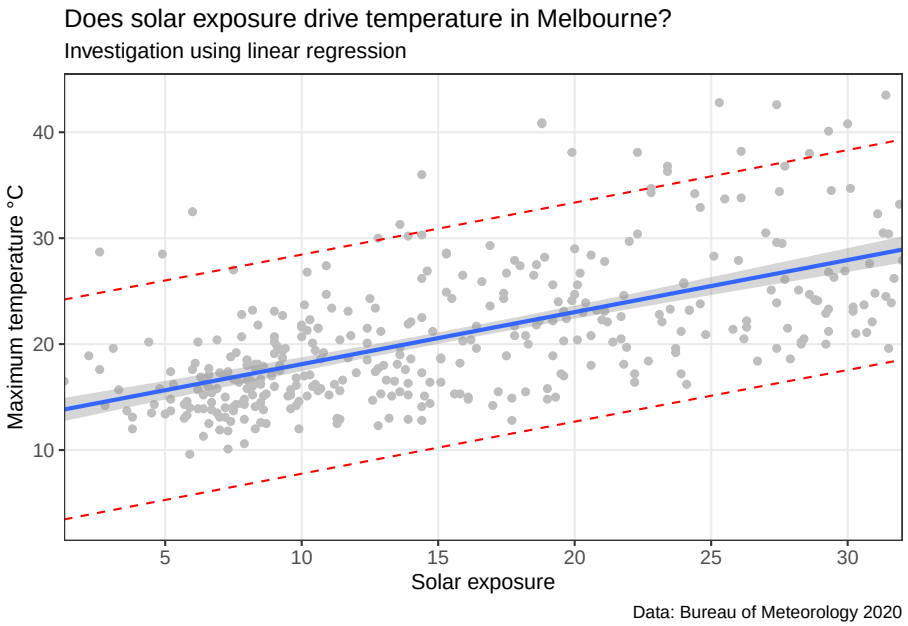
We can use this lm function to predict values of temperature based on the level of solar exposure. We can then compare this to the actual temperature record, and see how well the model fits the data set.

```
#Use this lm model to predict the values
MEL_weather_2019 <- MEL_weather_2019 %>%
 mutate(predicted_temp=predict(temp_model,newdata=MEL_weather_2019))

#Calculate the prediction interval
prediction_interval <- predict(temp_model,
                               newdata=MEL_weather_2019,
                               interval = "prediction")
summary(prediction_interval)

#Bind this prediction interval data back to the main set
MEL_weather_2019 <- cbind(MEL_weather_2019,prediction_interval)
MEL_weather_2019
```

Model fit is easier to interpret graphically. Let's plot the data with the model overlaid.

```
#Plot a chart with data and model on it
MEL_temp_predicted <-
ggplot(MEL_weather_2019)+
  geom_point(aes(y=Max_temp, x=Solar_exposure),
                 col="grey")+
  geom_line(aes(y=predicted_temp,x=Solar_exposure),
                col="blue")+
  geom_smooth(aes(y=Max_temp, x= Solar_exposure),
              method=lm)+
  geom_line(aes(y=lwr,x=Solar_exposure),
            colour="red", linetype="dashed")+
  geom_line(aes(y=upr,x=Solar_exposure),
            colour="red", linetype="dashed")+
  labs(title =
       "Does solar exposure drive temperature in Melbourne?",
       subtitle = 'Investigation using linear regression',
       caption = "Data: Bureau of Meteorology 2020") +
  xlab("Solar exposure")+
  ylab("Maximum temperature °C")+

  scale_x_continuous(expand=c(0,0),
```

```
                    breaks=c(0,5,10,15,20,25,30,35,40))+

  theme_bw()+
  theme(axis.text=element_text(size=10))+
  theme(panel.grid.minor = element_blank())

MEL_temp_predicted
```



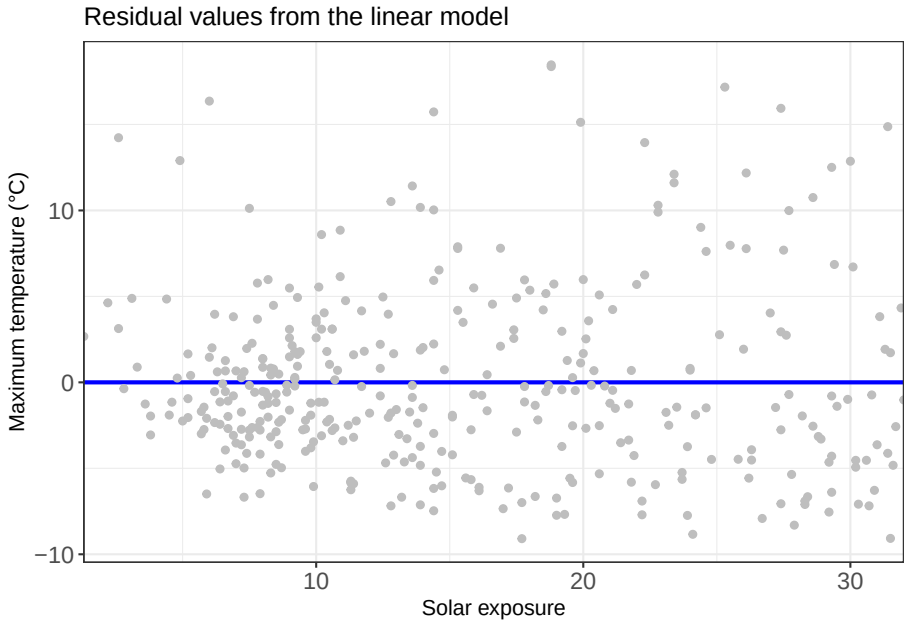
This chart includes the model (blue line), confidence interval (grey band around the blue line), and a prediction interval (red dotted line). A prediction interval reflects the uncertainty around a single value (put simple: what is the reasonable upper and lower bound that this data point could be estimated at?). A confidence interval reflects the uncertainty around the mean prediction values (put simply: what is a reasonable upper and lower bound for the blue line at this x value?). Therefore, a prediction interval will be generally much wider than a confidence interval for the same value.

Analyse the residuals

```
#Add the residuals to the series
residuals_temp_predict <- MEL_weather_2019 %>%
  add_residuals(temp_model)
```

Plot these residuals in a chart.

```
residuals_temp_predict_chart <-
  ggplot(data=residuals_temp_predict,
         aes(x=Solar_exposure, y=resid), col="grey")+

         geom_ref_line(h=0,colour="blue", size=1)+
         geom_point(col="grey")+

         xlab("Solar exposure")+
         ylab("Maximum temperature (°C)")+
         theme_bw() +
         labs(title = "Residual values from the linear model")+

         theme(axis.text=element_text(size=12))+

         scale_x_continuous(expand=c(0,0))
residuals_temp_predict_chart
```



Linear regression with more than one variable

The linear model above is *okay*, but can we make it better? Let's start by adding in some more variables into the linear regression.

Rainfall data might assist our model in predicting temperature. Let's add in that variable and analyse the results.

```
temp_model_2 <-
lm(Max_temp ~ Solar_exposure + Rainfall, data=MEL_weather_2019)
summary(temp_model_2)
```

We can see that adding in rainfall made the model better (R squared value has increased to 0.4338).

Next, we consider whether solar exposure and rainfall might be related to each other, as well as to temperature. For our third temperature model, we add an interaction variable between solar exposure and rainfall.

```
temp_model_3 <- lm(Max_temp ~ Solar_exposure +
                              Rainfall +
                              Solar_exposure:Rainfall,
                              data=MEL_weather_2019)
summary(temp_model_3)
```

We now see this variable is significant, and improves the model slightly (seen by an adjusted R squared of 0.4529).

Fitting a polynomial regression

When analysing the above data set, we see the issue is the sheer variance of temperatures associated with every other variable (it turns out weather forecasting is notoriously difficult).

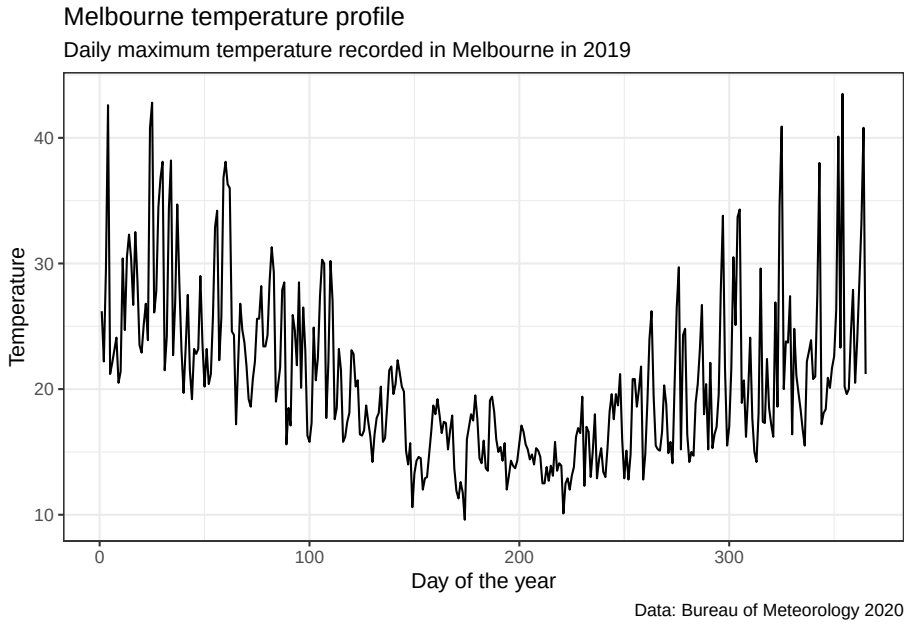
However we can expect that temperature follows a non-linear pattern throughout the year (in Australia it is hot in January-March, cold in June-August, then starts to warm up again). A linear model (e.g. a straight line) will be a very bad model for temperature — we need to introduce polynomials.

For simplicity, we will introduce a new variable (Day_number) which is the day of the year (e.g. 1 January is #1, 31 December is #366).

```
MEL_weather_2019 <- MEL_weather_2019 %>%
    mutate(Day_number=row_number())
head(MEL_weather_2019)
```

Using the same dataset as above, let's plot temperature in Melbourne in 2019.

```
MEL_temp_chart <-
ggplot(MEL_weather_2019)+
  geom_line(aes(x = Day_number, y = Max_temp)) +
  labs(title = 'Melbourne temperature profile',
  subtitle = 'Daily maximum temperature recorded in Melbourne in 2019',
  caption = "Data: Bureau of Meteorology 2020") +
  xlab("Day of the year")+
  ylab("Temperature")+
  theme_bw()

MEL_temp_chart
```



We can see we'll need a non-linear model to fit this data.

Below we create a few different models. We start with a normal straight line model, then add an x^2 and x^3 model. We then use these models and the 'predict' function to see what temperatures they forecast based on the input data.

```
#Create a straight line estimate to fit the data
poly1 <- lm(Max_temp ~ poly(Day_number,1,raw=TRUE),
         data=MEL_weather_2019)
summary(poly1)
#Create a polynominal of order 2 to fit this data
poly2 <- lm(Max_temp ~ poly(Day_number,2,raw=TRUE),
         data=MEL_weather_2019)
```

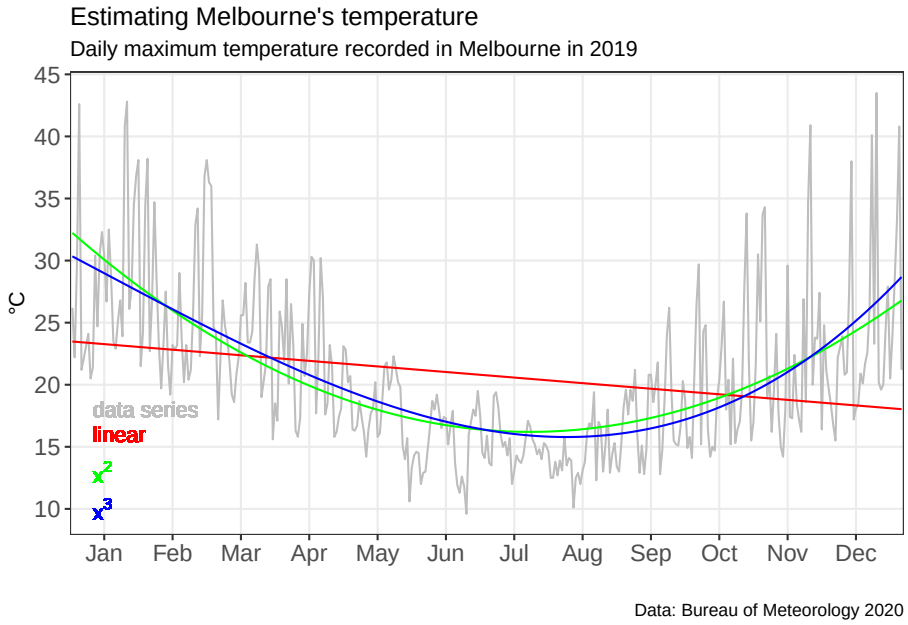
```
summary(poly2)
#Create a polynominal of order 3 to fit this data
poly3 <- lm(Max_temp ~ poly(Day_number,3,raw=TRUE),
         data=MEL_weather_2019)
summary(poly3)

#Use these models to predict
MEL_weather_2019 <- MEL_weather_2019 %>%
  mutate(poly1values=predict(poly1,newdata=MEL_weather_2019))%>%
  mutate(poly2values=predict(poly2,newdata=MEL_weather_2019))%>%
  mutate(poly3values=predict(poly3,newdata=MEL_weather_2019))

head(MEL_weather_2019)
```

In the table above we can see the estimates for that data point from the various models.

To see how well the models did graphically, we can plot the original data series with the polynominal models overlaid.

```
#Plot a chart with all models on it
MEL_weather_model_chart <-
ggplot(MEL_weather_2019)+
  geom_line(aes(x=Day_number, y= Max_temp),col="grey")+
  geom_line(aes(x=Day_number, y= poly1values),col="red") +
  geom_line(aes(x=Day_number, y= poly2values),col="green")+
  geom_line(aes(x=Day_number, y= poly3values),col="blue")+

  #Add text annotations
  geom_text(x=10,y=18,label="data series",col="grey",hjust=0)+
  geom_text(x=10,y=16,label="linear",col="red",hjust=0)+
  geom_text(x=10,y=13,label=parse(text="x^2"),col="green",hjust=0)+
  geom_text(x=10,y=10,label=parse(text="x^3"),col="blue",hjust=0)+

  labs(title = "Estimating Melbourne's temperature",
       subtitle = 'Daily maximum temperature recorded in Melbourne in 2019',
       caption = "Data: Bureau of Meteorology 2020") +

  xlim(0,366)+
  ylim(10,45)+
scale_x_continuous(breaks=
  c(15,45,75,105,135,165,195,225,255,285,315,345),
  labels=c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov"
  expand=c(0,0),
  limits=c(0,366)) +
```

```
  scale_y_continuous(breaks=c(10,15,20,25,30,35,40,45)) +
  xlab("")+
  ylab("°C")+
  theme_bw()+
  theme(axis.text=element_text(size=12))+
  theme(panel.grid.minor = element_blank())

MEL_weather_model_chart
```

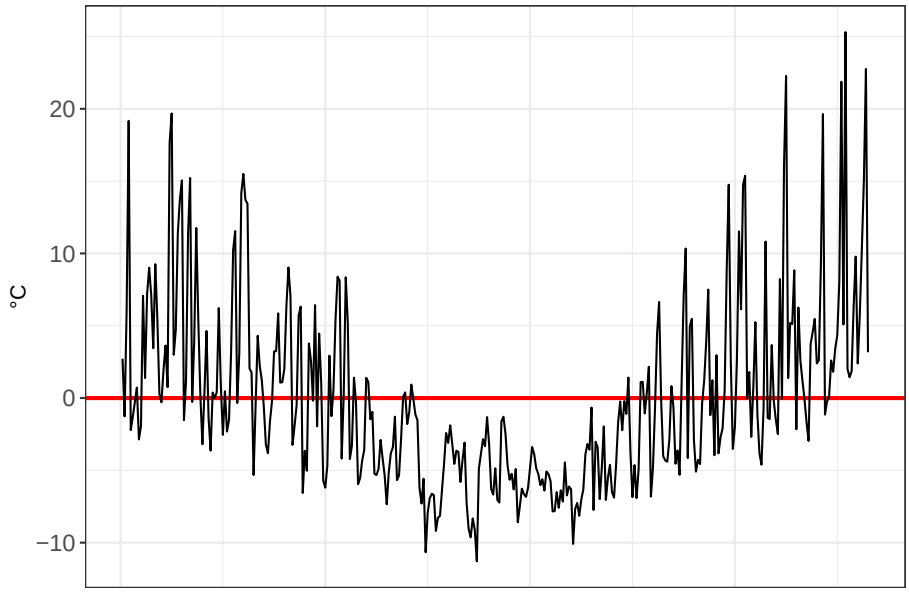


We can see in the chart above the polynomial models do much better at fitting the data. However, they are still highly variant.

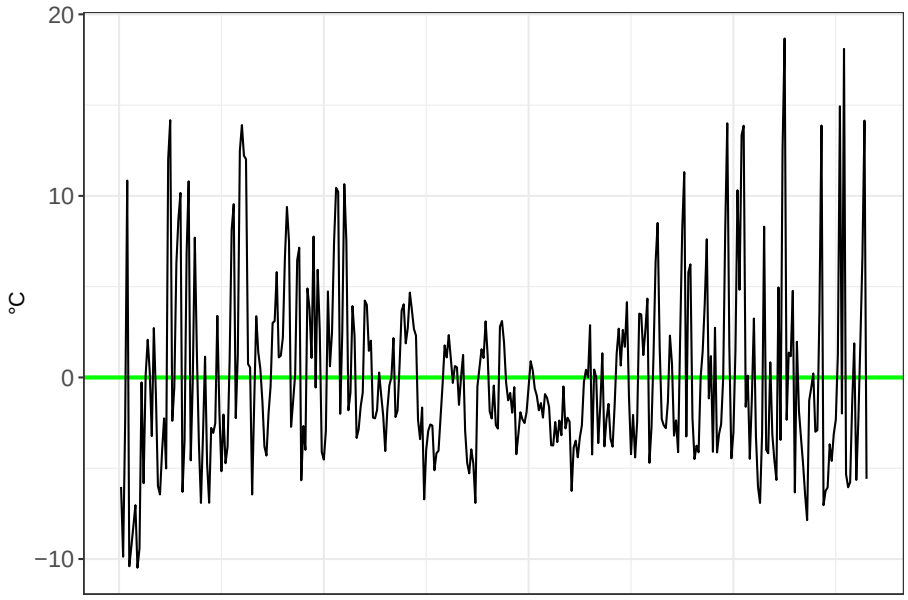
Just how variant are they? We can look at the residuals to find out. The residuals is the gap between the observed data point (i.e. the grey line) and our model.

```
#Get the residuals for poly1
residuals_poly1 <- MEL_weather_2019 %>%
  add_residuals(poly1)
residuals_poly1_chart <-
ggplot(data=residuals_poly1,aes(x=Day_number, y=resid))+
  geom_ref_line(h=0,colour="red", size=1)+
  geom_line()+
  xlab("")+
  ylab("°C")+
```

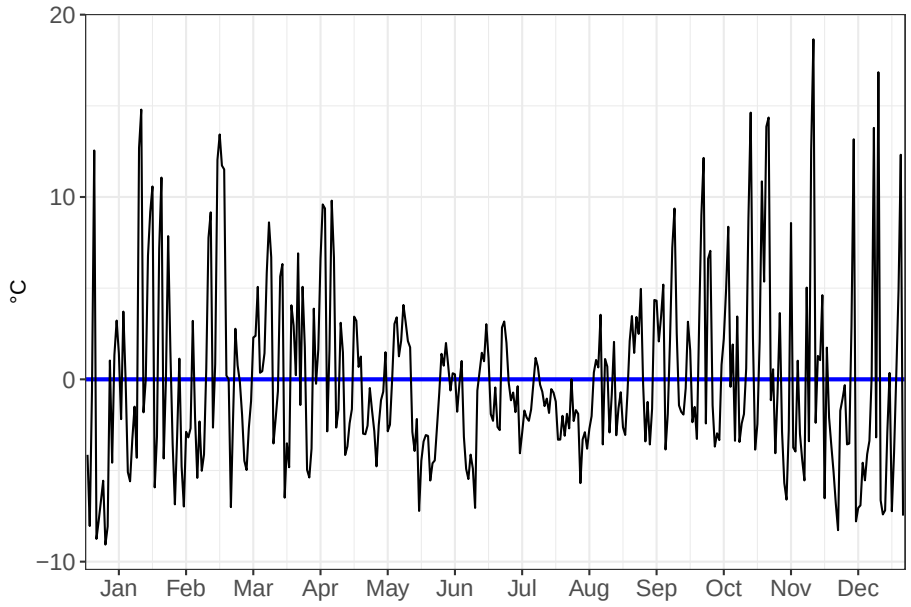
```
  theme_bw()+
  theme(axis.text=element_text(size=12))+
  theme(axis.ticks.x=element_blank(),
  axis.text.x=element_blank())
residuals_poly1_chart
```



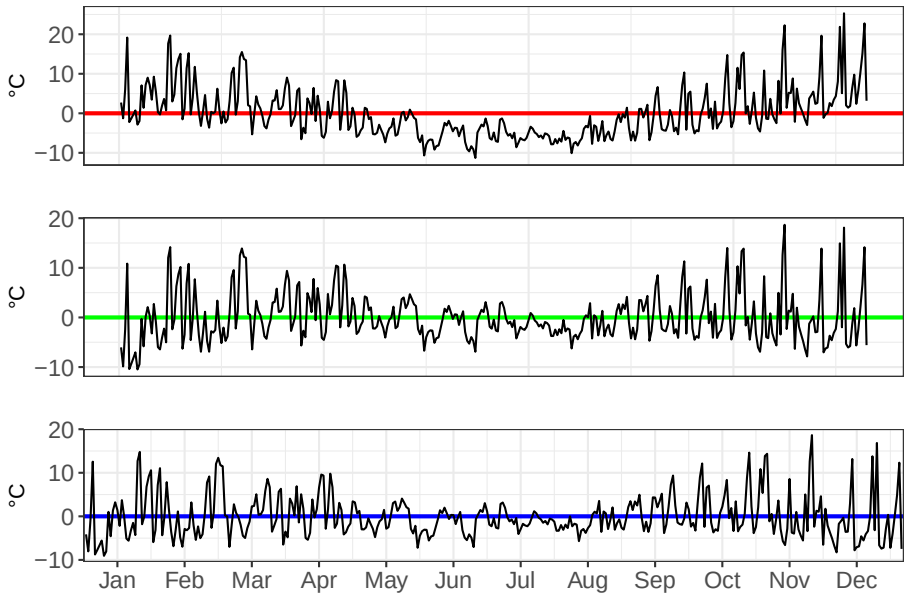

```
#Get the residuals for poly2
residuals_poly2 <- MEL_weather_2019%>%
  add_residuals(poly2)
residuals_poly2_chart <- ggplot(data=residuals_poly2,aes(x=Day_number, y=resid))+
  geom_ref_line(h=0,colour="green", size=1)+
  geom_line()+
  xlab("")+
  ylab("°C")+
  theme_bw()+
  theme(axis.text=element_text(size=12))+
  theme(axis.ticks.x=element_blank(),
  axis.text.x=element_blank())

residuals_poly2_chart
```




```
#Get the residuals for poly3
residuals_poly3 <- MEL_weather_2019 %>%
  add_residuals(poly3)
residuals_poly3_chart <- ggplot(data=residuals_poly3,aes(x=Day_number, y=resid))+
  geom_ref_line(h=0,colour="blue", size=1)+
  geom_line()+
  theme_bw()+
  theme(axis.text=element_text(size=12))+
 scale_x_continuous(breaks=
 c(15,45,75,105,135,165,195,225,255,285,315,345),
 labels=c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"),
 expand=c(0,0),
 limits=c(0,366))+
 xlab("")+
 ylab("°C")

residuals_poly3_chart
```




```
three_charts_single_page <- plot_grid(
  residuals_poly1_chart,
  residuals_poly2_chart,
  residuals_poly3_chart,
  ncol=1,nrow=3,label_size=16)
three_charts_single_page
```



As we move from a linear, to a x^2, to a x^3 model, we see the residuals decrease in volatility.

Hypothesis testing

A quick refresher

Hypothesis testing is a way of validating if a claim about a population (e.g. a data set) is correct. Getting data on a whole population (e.g. everyone in Australia) is hard. So, to validate a hypothesis, we use random samples from a population instead.

The language when dealing with hypothesis testing is purposefully janky.

When looking at the outputs of our hypothesis test, we consider p-values. Note: There's ***lots wrong with p-values*** that we won't bother getting into right now. The long story short is if you make your null hypothesis ultra specific and only report when your p-value on your millionth iteration of a test is below 0.05... bad science is likely to get published and cited.

What we need to know:

- A small p-value (typically less than or equal to 0.05) indicates strong evidence ***against*** the null hypothesis, so we ***reject*** it.
- A large p-value (greater than 0.05) indicates weak evidence ***against*** the null hypothesis, so you ***fail to reject*** it.

Let's load in some packages and get started.

```
# Load necessary packages
library(ggridges)
library(ggplot2)
library(ggrepel)       # Avoid overlapping text in plots
library(viridis)       # Color scales
library(readxl)        # Read Excel files
library(dplyr)         # Data manipulation
library(stringr)       # String operations
library(tidyr)         # Data tidying (replaces reshape)
```

```
library(lubridate)      # Work with dates
library(gapminder)      # Gapminder dataset
library(ggalt)          # Extensions to ggplot (dumbbell plots, etc.)
library(purrr)          # Functional programming
library(scales)         # Scaling tools for ggplot
library(aTSA)           # Time series analysis
library(readrba)        # For working with Reserve Bank of Australia data
```

T-testing our first hypothesis

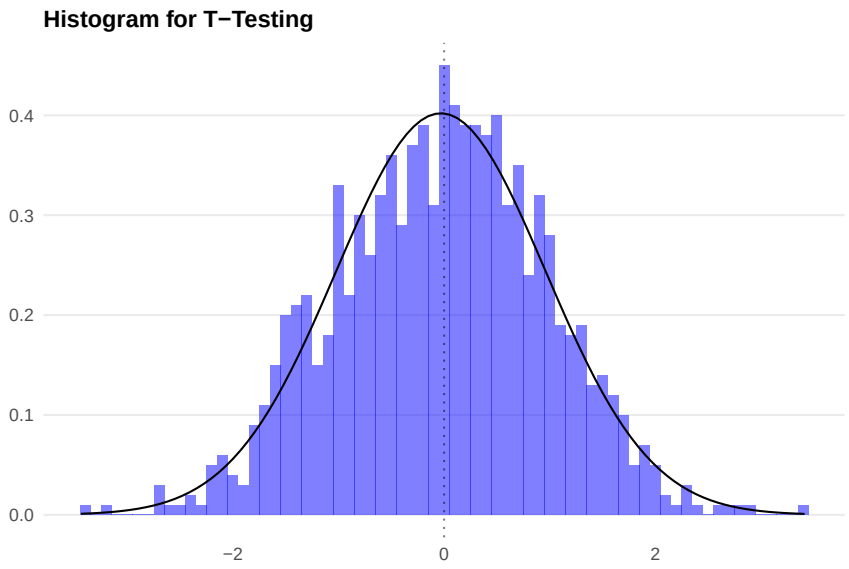
We'll start by creating a normally distributed random dataset using `rnorm`.

By default the `rnorm` function will generate a dataset that has a mean of 0 and a standard deviation of 1, but let's state it explicitly to keep things simple.

```
set.seed(40)
dataset1 <- data.frame(variable1=rnorm(1000,mean=0,sd=1))
```

Let's chart the distribution.

```
ggplot() +
  geom_histogram(aes(x = dataset1$variable1, y = ..density..),
                 binwidth = 0.1, fill = "blue", alpha = 0.5) +
  stat_function(fun = dnorm, args = list(mean = mean(dataset1$variable1),
                                         sd = sd(dataset1$variable1))) +
  geom_vline(xintercept = 0, linetype = "dotted", alpha = 0.5) +
  labs(title = "Histogram for T-Testing", x = "", y = "") +
  theme_minimal() +
  theme(plot.title=element_text(face="bold",size=12))+
  theme(plot.subtitle=element_text(size=11))+
  theme(plot.caption=element_text(size=8))+
  theme(axis.text=element_text(size=9))+
  theme(panel.grid.minor = element_blank())+
  theme(panel.grid.major.x = element_blank()) +
  theme(plot.subtitle = element_text(margin=ggplot2::margin(0,0,15,0))) +
  theme(plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"))
```



We know that the mean of dataset1 will be approximately zero (because we set it)... but let's check anyway.

```
# Find the mean of dataset1
mean(dataset1$variable1)
```

Now let's run our first hypothesis test. We'll use the `t.test` function. This is in the format of t.test(data, null_hypothesis).

We'll start with the null hypothesis that the mean for dataset1$variable1 is 5. This is a two tailed t-test, as we will reject the null if we're confident the mean is *either* above or below 5.

```
# Hypothesis test
t.test(dataset1$variable1, mu = 5)
```

We see the p-value here is tiny, meaning we reject the null hypothesis. That is to say, the mean for dataset1$variable1 is not 5.

Understanding tailed tests

By default, `t.test` assumes a two-tailed test with a 95% confidence level. However, sometimes we need to test if one variable is greater or smaller than another (rather than just different from the null). This is when we use one-tailed tests.

Next, we test whether the mean is -0.03 using a one-tailed test:

```
# Hypothesis test
t.test(dataset1$variable1, mu = -0.03, alternative="greater")
```

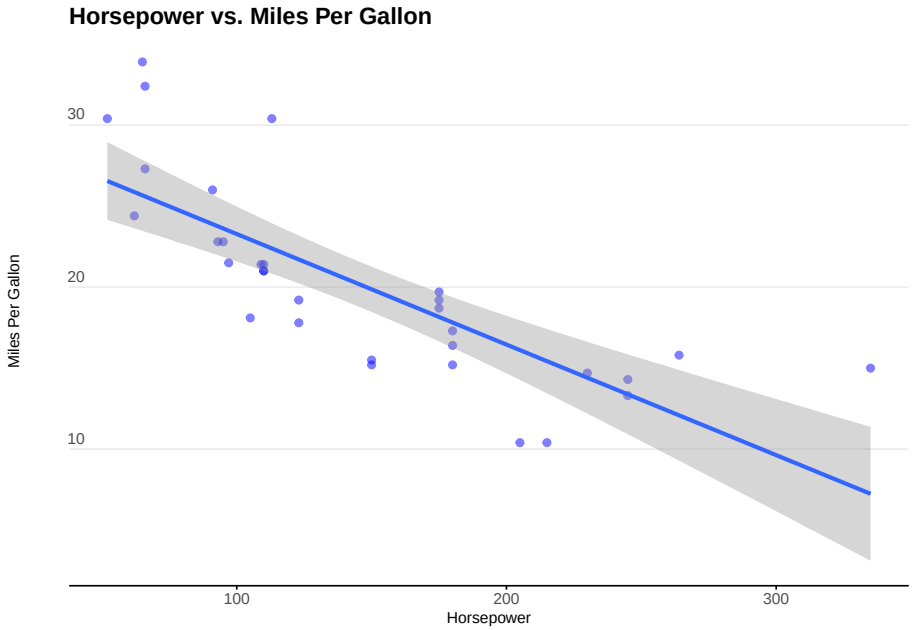
We see here the p-value is greater than 0.05, leading us to *fail to reject* the null hypothesis. In a sentence, we cannot say that the mean of dataset1$variable1 is different to 0.01.

Correlation

A correlation coefficient measures the direction and strength of the relationship between two variables. However, correlation calculations assume normal distributions.

Let's examine the relationship between `mpg` and `hp` in the `mtcars` dataset.

```
ggplot2::ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point(color = "blue", alpha = 0.5) +
  geom_smooth(method = 'lm') +
  labs(
    title = "Horsepower vs. Miles Per Gallon",
    x = "Horsepower",
    y = "Miles Per Gallon"
  ) +
  theme_minimal(base_size = 8) +
  theme(
    legend.position = "bottom",
    plot.title = element_text(face = "bold", size = 12),
    plot.subtitle = element_text(size = 11, margin = margin(0, 0, 25, 0)),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 8),
    axis.title.y = element_text(margin = margin(r = 3)),
    axis.text.y = element_text(vjust = -0.5, margin = margin(l = 20, r = -10)),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    axis.line.x = element_line(colour = "black", size = 0.4),
    axis.ticks.x = element_line(colour = "black", size = 0.4)
  )
```



We see that miles per gallon is correlated with horsepower. It's a negative relationship, meaning the more horsepower in a car, the less miles per gallon the car exhibits.

We can use the `cor.test` to tell us the correlation coefficient and the p-value of the correlation.

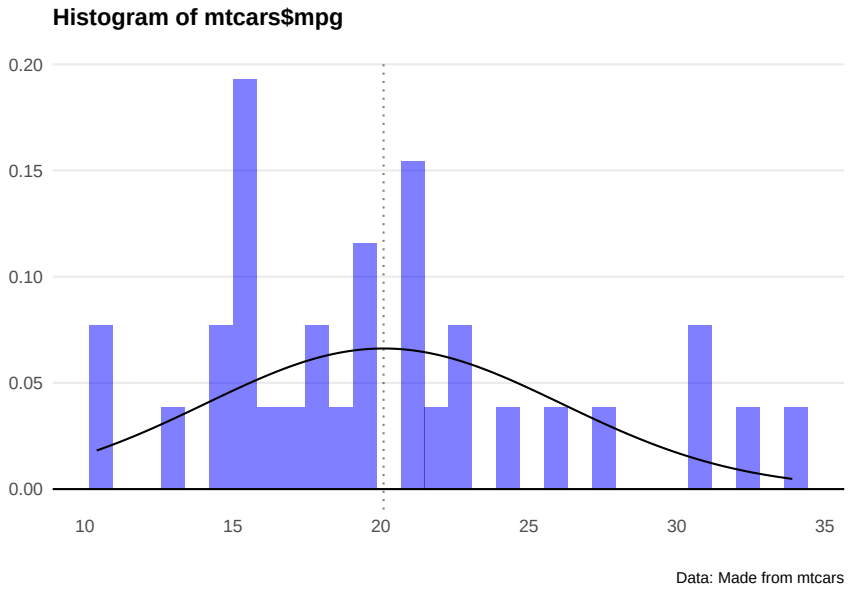
We specify the method as 'pearson' for the Pearson correlation coefficient.

```
cor.test(mtcars$hp, mtcars$mpg, method="pearson")
```

Since Pearson's method assumes normality, we check the distributions. Let's plot a histogram for both hp and mpg.

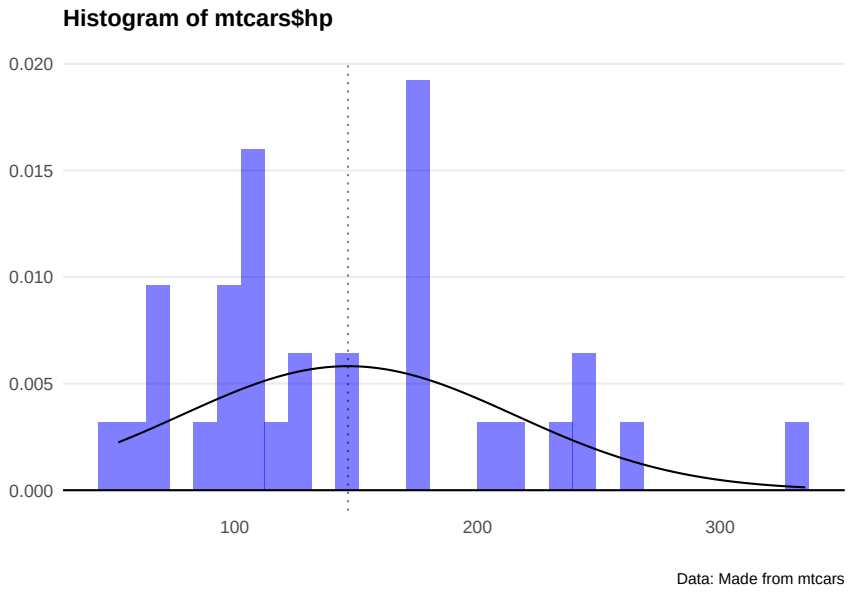
```
ggplot()+
  geom_histogram(aes(x=mtcars$mpg,y=..density..),fill="blue",alpha=0.5) +
  stat_function(fun = dnorm,
                args = list(mean = mean(mtcars$mpg), sd = sd(mtcars$mpg)))+
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = mean(mtcars$mpg), linetype="dotted",alpha=0.5)+
            labs(title="Histogram of mtcars$mpg",
                          caption = "Data: Made from mtcars",
                          x="",
                          y="") +
            theme_minimal() +
            theme(panel.spacing.x = unit(10, "mm"))+
```

```
            theme(legend.position="none")+
            theme(plot.title=element_text(face="bold",size=12))+
            theme(plot.subtitle=element_text(size=11))+
            theme(plot.caption=element_text(size=8))+
            theme(axis.text=element_text(size=9))+
            theme(panel.grid.minor = element_blank())+
            theme(panel.grid.major.x = element_blank()) +
            theme(plot.title = element_text(margin=ggplot2::margin(0,0,15,0))) +
            theme(plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"))
```




```
ggplot()+
  geom_histogram(aes(x=mtcars$hp,y=..density..),fill="blue",alpha=0.5) +
  stat_function(fun = dnorm,
                args = list(mean = mean(mtcars$hp), sd = sd(mtcars$hp)))+
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = mean(mtcars$hp), linetype="dotted",alpha=0.5)+

            labs(title="Histogram of mtcars$hp",
                         caption = "Data: Made from mtcars",
                         x="",
                         y="") +

            theme_minimal() +
            theme(panel.spacing.x = unit(10, "mm"))+
```

```
            theme(legend.position="none")+
            theme(plot.title=element_text(face="bold",size=12))+
            theme(plot.subtitle=element_text(size=11))+
            theme(plot.caption=element_text(size=8))+
            theme(axis.text=element_text(size=9))+
            theme(panel.grid.minor = element_blank())+
            theme(panel.grid.major.x = element_blank()) +
            theme(plot.title = element_text(margin=ggplot2::margin(0,0,15,0))) +
            theme(plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"))
```

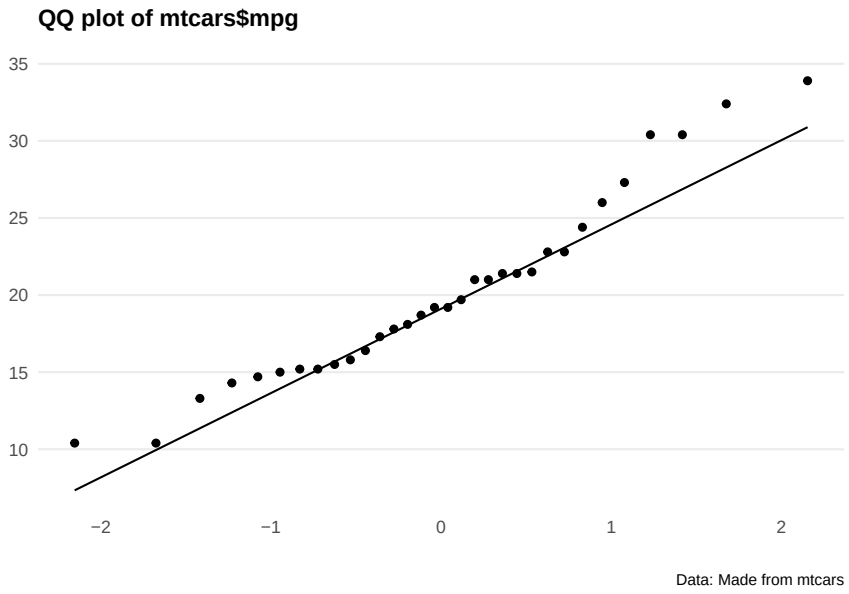


Crikey... they don't look very normal at all.

Let's plot QQ plots of our variables and see what's going on.

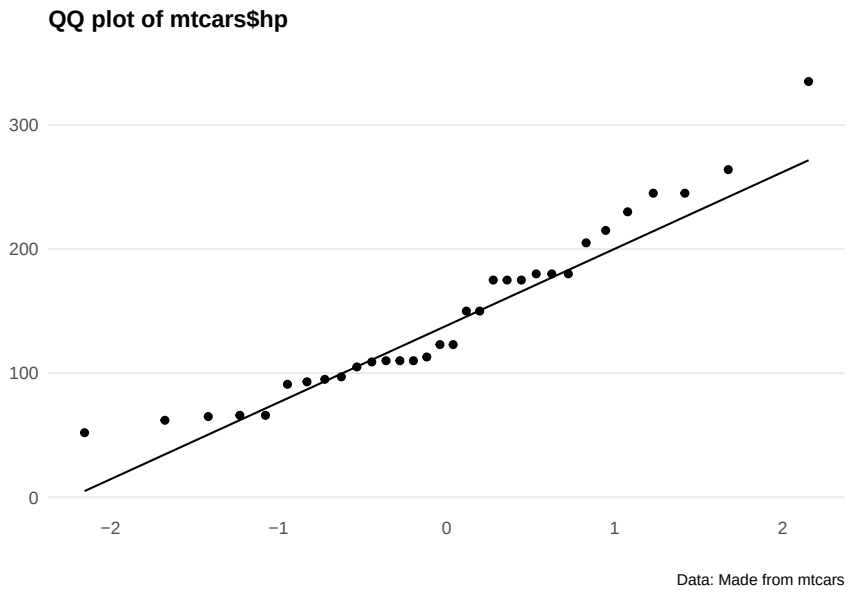
```
ggplot(mtcars, aes(sample = mpg)) +
  geom_qq()+
  geom_qq_line()+
            labs(title="QQ plot of mtcars$mpg",
                         caption = "Data: Made from mtcars",
                         x="",
                         y="") +
            theme_minimal() +
            theme(panel.spacing.x = unit(10, "mm"))+
            theme(legend.position="none")+
            theme(plot.title=element_text(face="bold",size=12))+
```

```
          theme(plot.subtitle=element_text(size=11))+
          theme(plot.caption=element_text(size=8))+
          theme(axis.text=element_text(size=9))+
          theme(panel.grid.minor = element_blank())+
          theme(panel.grid.major.x = element_blank()) +
          theme(plot.title = element_text(margin=ggplot2::margin(0,0,15,0))) +
          theme(plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"))
```




```
ggplot(mtcars, aes(sample = hp)) +
  geom_qq()+
  geom_qq_line()+
          labs(title="QQ plot of mtcars$hp",
                       caption = "Data: Made from mtcars",
                       x="",
                       y="") +
          theme_minimal() +
          theme(panel.spacing.x = unit(10, "mm"))+
          theme(legend.position="none")+
          theme(plot.title=element_text(face="bold",size=12))+
          theme(plot.subtitle=element_text(size=11))+
          theme(plot.caption=element_text(size=8))+
          theme(axis.text=element_text(size=9))+
          theme(panel.grid.minor = element_blank())+
          theme(panel.grid.major.x = element_blank()) +
```

```
            theme(plot.title = element_text(margin=ggplot2::margin(0,0,15,0))) +
            theme(plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"))
```



Hmm okay, both series are a bit all over the shop. Let's do a statistical test to confirm.

The Shapiro-Wilk's method is widely used for normality testing. The null hypothesis of this tests is that the sample distribution is normal. If the test is significant, the distribution is non-normal.

```
shapiro.test(mtcars$mpg)
shapiro.test(mtcars$hp)
```

Confidence intervals for the mean

We can calkcualte the confidence interval for the mean. This measures the variance of the normal distribution, and gives us an idea of how 'clustered' the values are to the mean.

There are 4 steps to do this:

1. Calculate the mean
2. Calculate the standard error of the mean

3. Find the t-score that corresponds to the confidence level
4. Calculate the margin of error and construct the confidence interval

```
mpg.mean <- mean(mtcars$mpg)
print(mpg.mean)

mpg.n <- length(mtcars$mpg)
mpg.sd <- sd(mtcars$mpg)
mpg.se <- mpg.sd/sqrt(mpg.n)
print(mpg.se)

alpha = 0.05
degrees.freedom = mpg.n - 1
t.score = qt(p=alpha/2, df=degrees.freedom,lower.tail=F)
print(t.score)

mpg.error <- t.score * mpg.se

lower.bound <- mpg.mean - mpg.error
upper.bound <- mpg.mean + mpg.error
print(c(lower.bound,upper.bound))
```

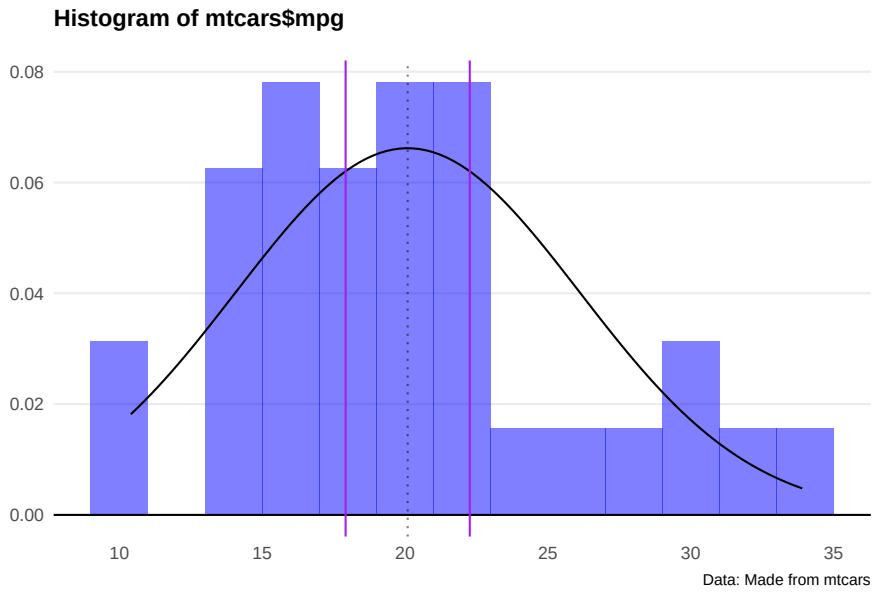
For the lazy folks among us - there's also this quick and dirty way of doing it.

```
# Calculate the mean and standard error
mpg.model <- lm(mpg ~ 1, mtcars)

# Calculate the confidence interval
confint(mpg.model, level=0.95)
```

Great. Let's plot this interval on the distribution.

```
ggplot(mtcars, aes(x = mpg)) +
  geom_histogram(aes(y = ..density..), binwidth = 2, fill = "blue", alpha = 0.5) +
  stat_function(fun = dnorm, args = list(mean = mean(mtcars$mpg), sd = sd(mtcars$mpg))) +
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = mean(mtcars$mpg), linetype = "dotted", alpha = 0.5) +
  geom_vline(xintercept = c(lower.bound, upper.bound), col = "purple") +
  labs(
    title = "Histogram of mtcars$mpg",
    caption = "Data: Made from mtcars",
    x = NULL,
    y = NULL
  ) +
```

```
  theme_minimal() +
  theme(
    panel.spacing.x = unit(10, "mm"),
    legend.position = "none",
    plot.title = element_text(face = "bold", size = 12, margin = margin(0, 0, 15, 0)),
    plot.subtitle = element_text(size = 11),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 9),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    plot.margin = unit(c(0.5, 0.5, 0.5, 0.5), "cm")
  )
```

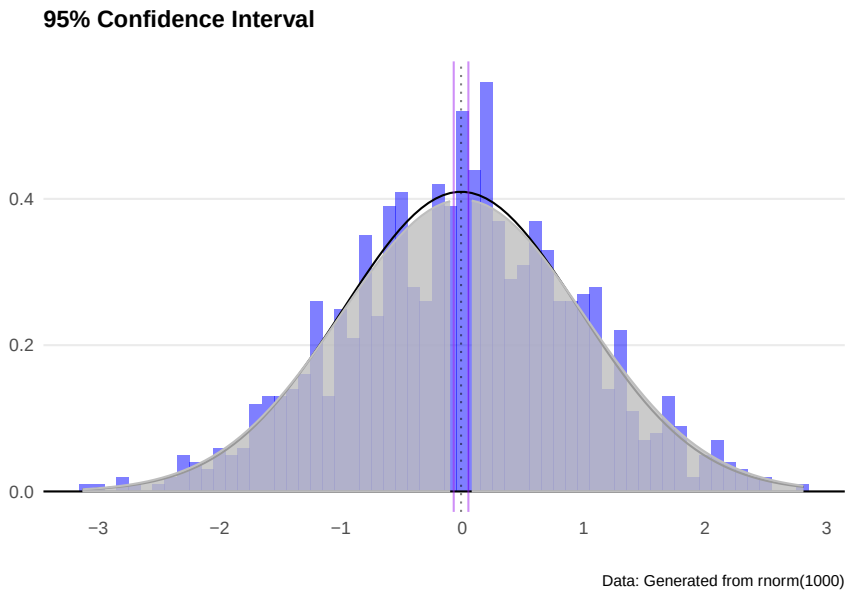


Two things we note here: Firstly, the distribution doesn't look *that* normal. Secondly, the 95% confidence interval looks narrow as a result.

Let's do the same analysis with an actual normal distribution and see what happens.

```
set.seed(404)
dataset2 <- data.frame(variable1 = rnorm(1000, mean = 0, sd = 1))

df2.mean <- mean(dataset2$variable1)
df2.n <- length(dataset2$variable1)
df2.sd <- sd(dataset2$variable1)
```

```r
df2.se <- df2.sd / sqrt(df2.n)  # Fixed variable name typo

alpha <- 0.05
t.score <- qt(p = alpha / 2, df = df2.n - 1, lower.tail = FALSE)

df2.error <- t.score * df2.se
lower.bound.df2 <- df2.mean - df2.error
upper.bound.df2 <- df2.mean + df2.error

# Functions to shade the tails
shade_tail <- function(x, bound, direction) {
  y <- dnorm(x, mean = 0, sd = 1)
  y[(direction == "upper" & x < bound) | (direction == "lower" & x > bound)] <- NA
  return(y)
}

# Plot
ggplot(dataset2, aes(x = variable1)) +
  geom_histogram(aes(y = ..density..), binwidth = 0.1, fill = "blue", alpha = 0.5) +
  stat_function(fun = dnorm, args = list(mean = df2.mean, sd = df2.sd)) +
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = c(df2.mean, lower.bound.df2, upper.bound.df2),
             linetype = c("dotted", "solid", "solid"), col = c("black", "purple", "pur
  stat_function(fun = function(x) shade_tail(x, upper.bound.df2, "upper"),
                geom = "area", fill = "grey", col = "grey", alpha = 0.8) +
  stat_function(fun = function(x) shade_tail(x, lower.bound.df2, "lower"),
                geom = "area", fill = "grey", col = "grey", alpha = 0.8) +
  labs(
    title = "95% Confidence Interval",
    caption = "Data: Generated from rnorm(1000)",
    x = "", y = ""
  ) +
  theme_minimal() +
  scale_x_continuous(breaks = seq(-3, 3, by = 1)) +
  theme(
    panel.spacing.x = unit(10, "mm"),
    legend.position = "none",
    plot.title = element_text(face = "bold", size = 12, margin = margin(0, 0, 15, 0)),
    plot.subtitle = element_text(size = 11),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 9),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    plot.margin = unit(c(0.5, 0.5, 0.5, 0.5), "cm")
  )
```



Great - we've got a more sensible looking plot, and greyed out the tails where our confidence interval excludes. We expect out observation to fall somewhere between the two purple lines (or more exactly between -2.1 and 2.1)

Confidence intervals for a model

We can also calculate the confidence interval around a linear model. This process shows how confident we can be about any single point in the linear estimate. If the confidence interval is wide, the estimate at that point is likely unreliable.

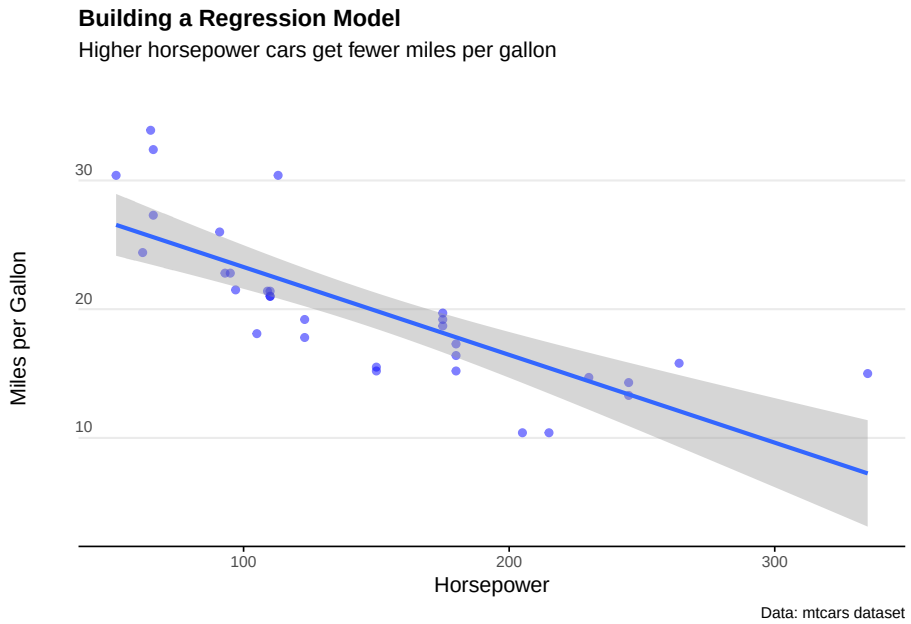
We'll create a linear model of mpg on the y-axis and horsepower on the x-axis.

```
mtcars.lm <- lm(mpg ~ hp, data = mtcars)
summary(mtcars.lm)

predict(mtcars.lm, newdata = mtcars, interval = 'confidence')
```

The`geom_smooth()`function presents an easy way to plot a confidence interval on a chart.

The syntax in this example is:
`geom_smooth(aes(x = hp, y = mpg), method='lm', level=0.95)`

```r
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point(col = "blue", alpha = 0.5) +
  geom_smooth(method = "lm", level = 0.95) +
  labs(
    title = "Building a Regression Model",
    subtitle = "Higher horsepower cars get fewer miles per gallon",
    caption = "Data: mtcars dataset",
    x = "Horsepower",
    y = "Miles per Gallon"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    plot.title = element_text(face = "bold", size = 12),
    plot.subtitle = element_text(size = 11, margin = margin(0, 0, 25, 0)),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 8),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    axis.title.y = element_text(margin = margin(r = 3)),
    axis.text.y = element_text(vjust = -0.5, margin = margin(l = 20, r = -10)),
    axis.line.x = element_line(colour = "black", size = 0.4),
    axis.ticks.x = element_line(colour = "black", size = 0.4)
  )
```



For a sanity check, let's crank up the confidence level to 0.999 (meaning our interval should capture just about all the observations). We see the confidence interval band increases... but not by *that* much. Why?

Well remember how the data isn't a very good normal distribution? That means the confidence interval function won't be super accurate - especially at the extremes.

Forecasting

So, we've got a time series dataset... but what is a reasonable forecast for how it might behave in the future?

Sure we can build a confidence interval (as we learned in the previous chapter) and figure out a reasonable value - but what about forecasting for multiple periods into the future?

That's where we need to build models. Let's load in some packages.

```
# Load in required packages
library(tidyverse)   # Includes ggplot2, dplyr, purrr, stringr, etc.
library(ggridges)    # For ridge plots
library(forecast)    # Time series forecasting
library(ggrepel)     # For better text label placement
library(viridis)     # Color scales
library(readxl)      # Excel file reading
library(lubridate)   # Date and time manipulation
library(gapminder)   # Gapminder data for examples
library(ggalt)       # Additional ggplot2 geoms
library(scales)      # Scale functions for ggplot2
```

We'll start with some pre-loaded time series data. The `ggplot2` package includes a data set called 'economics' that contains US economic indicators from the 1960's to 2015.

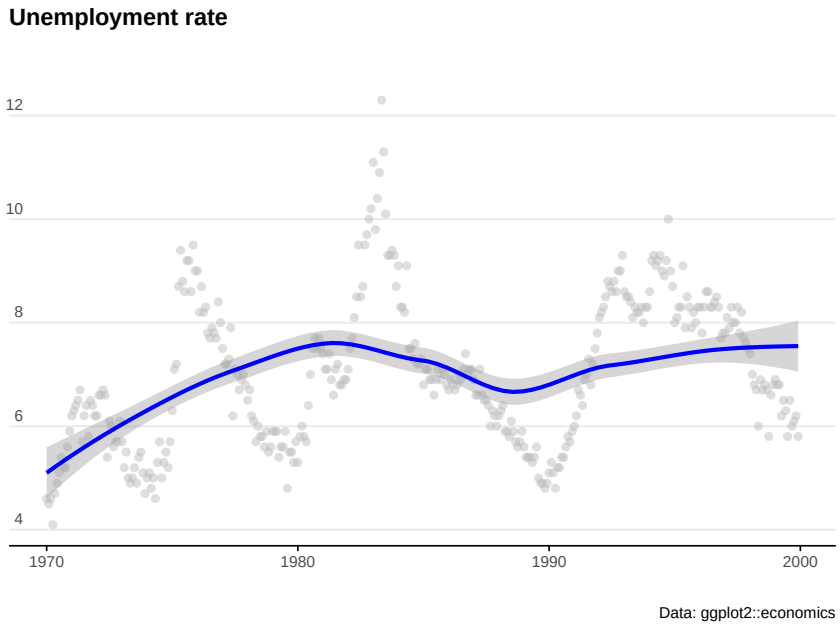
```
econ_data <- economics %>% dplyr::select(c("date", "uempmed"))
econ_data <- econ_data %>% dplyr::filter((date >= as.Date("1970-01-01")
                                          & date <= as.Date("1999-12-31")))
```

As a side note: We can also get Australian unemployment rate data using the `readrba` function.

```
aus_unemp_rate <- read_rba(series_id = "GLFSURSA")
head(aus_unemp_rate)
```

Let's plot the US data to see what we are working with.

```
ggplot(econ_data) +
  geom_point(aes(x = date, y = uempmed), col = "grey", alpha = 0.5) +
  geom_smooth(aes(x = date, y = uempmed), col = "blue") +

  labs(title = "Unemployment rate",
       caption = "Data: ggplot2::economics",
       x = "",
       y = "") +

  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 12, margin = margin(0, 0, 25, 0)),
    plot.subtitle = element_text(size = 11),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 8),
    axis.title.y = element_text(margin = margin(t = 0, r = 3, b = 0, l = 0)),
    axis.text.y = element_text(vjust = -0.5, margin = margin(l = 20, r = -10)),
    axis.line.x = element_line(colour = "black", size = 0.4),
    axis.ticks.x = element_line(colour = "black", size = 0.4),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    legend.position = "bottom"
  )
```



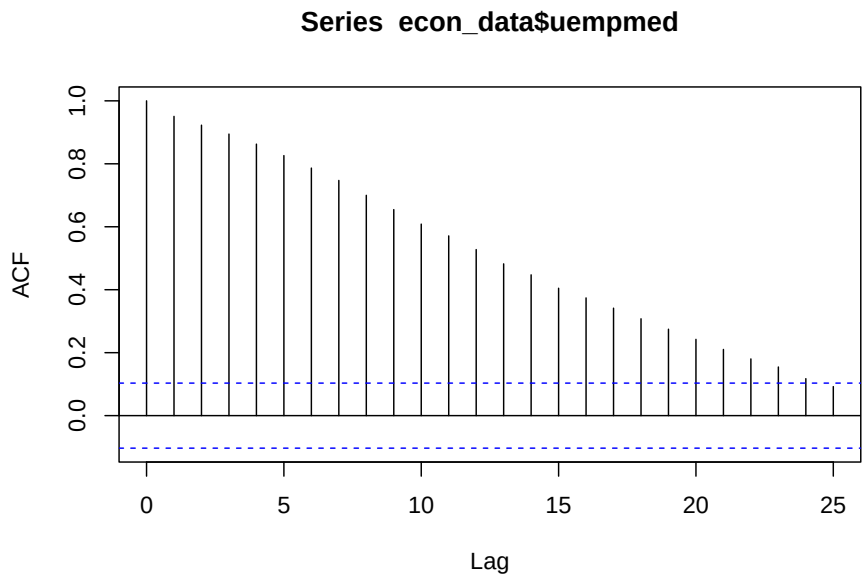
ARIMA models

AutoRegressive Integrated Moving Average (ARIMA) models are a handy tool to have in the toolbox. An ARIMA model describes where Yt depends on its own lags. A moving average (MA only) model is one where Yt depends only on the lagged forecast errors. We combine these together (technically we integrate them) and get ARIMA.

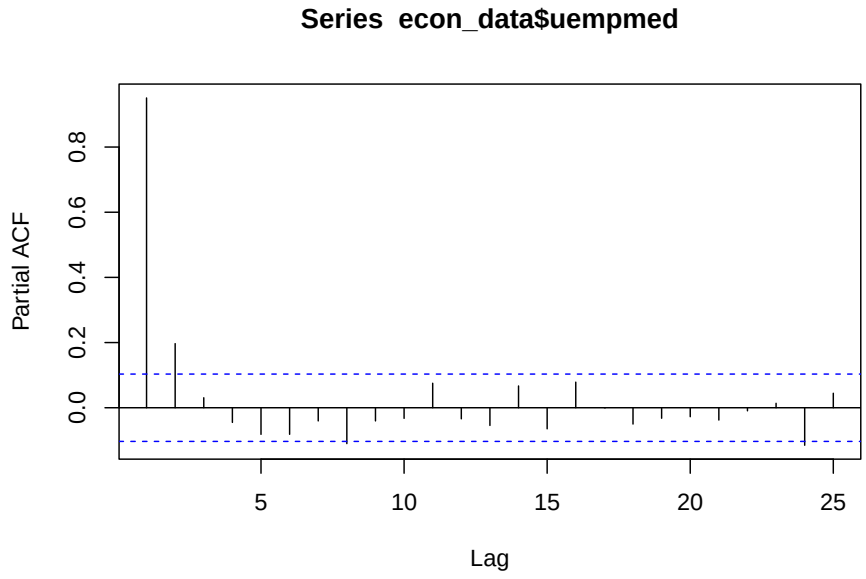
When working with ARIMAs, we need to 'difference' our series to make it stationary.

We check if it is stationary using the augmented Dickey-Fuller test. The null hypothesis assumes that the series is non-stationary. A series is said to be stationary when its mean, variance, and autocovariance don't change much over time.

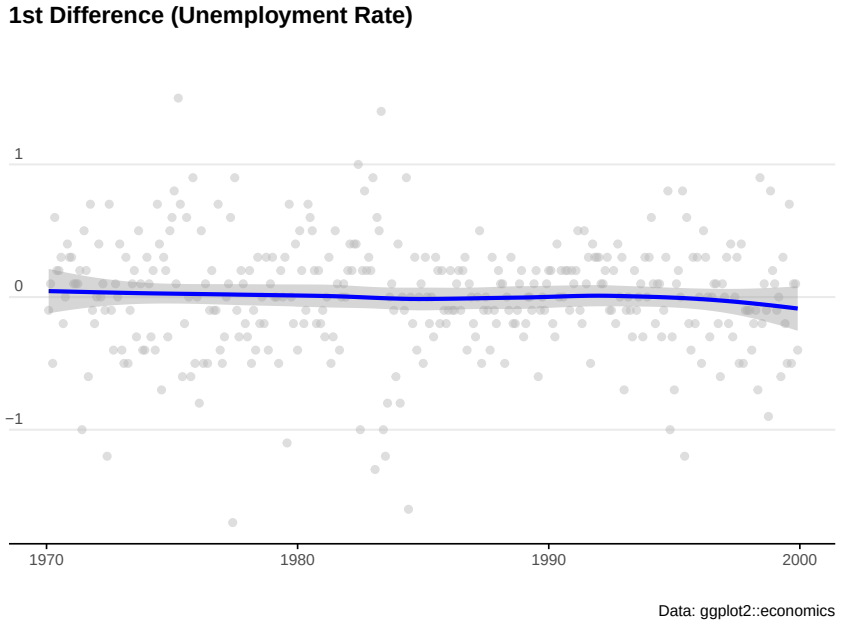
```
# Test for stationarity
aTSA::adf.test(econ_data$uempmed)

# See the auto correlation
acf(econ_data$uempmed)
```



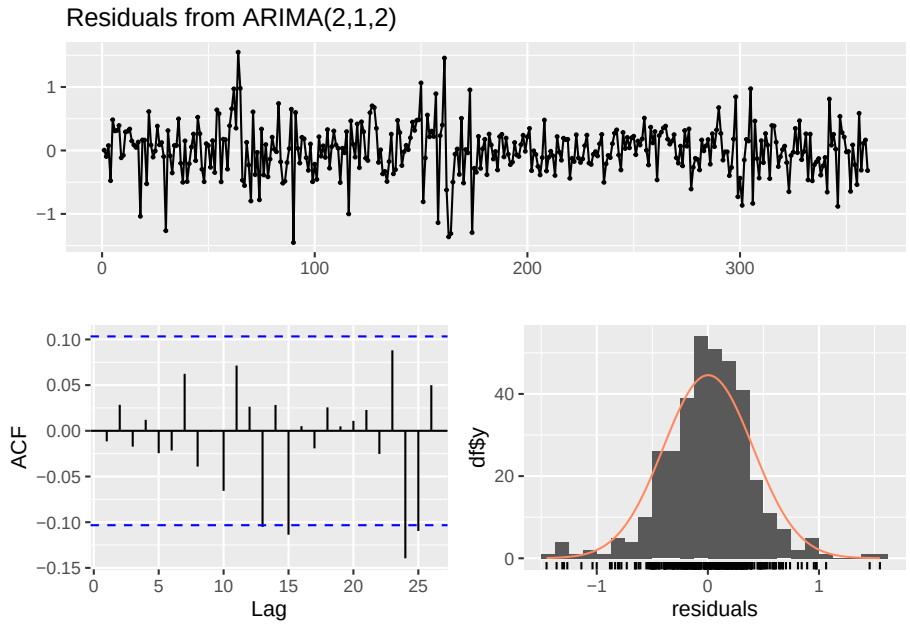

```
# Identify partial auto correlation
pacf(econ_data$uempmed)
```



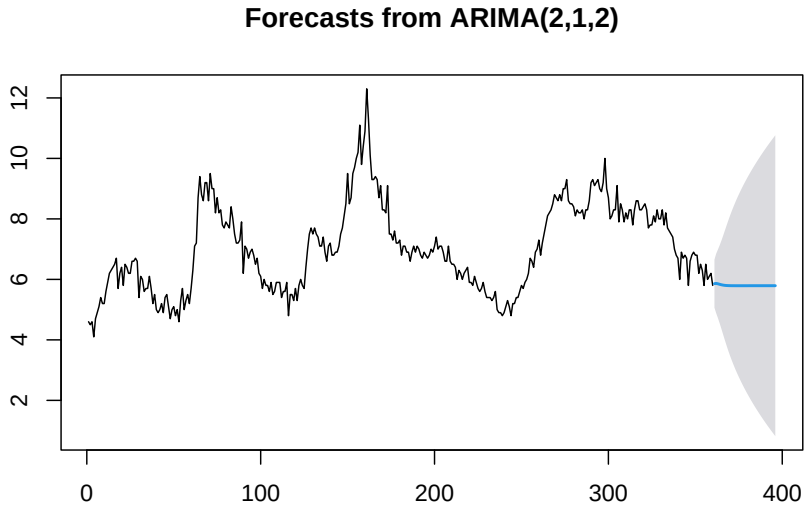
```r
# Take the first differences of the series
econ_data <- econ_data %>% mutate(diff = uempmed - lag(uempmed))

# Plot the first differences
ggplot(econ_data, aes(x = date, y = diff)) +
  geom_point(col = "grey", alpha = 0.5) +
  geom_smooth(col = "blue") +

  labs(
    title = "1st Difference (Unemployment Rate)",
    caption = "Data: ggplot2::economics",
    x = "",
    y = ""
  ) +

  theme_minimal() +
  theme(
    legend.position = "bottom",
    plot.title = element_text(face = "bold", size = 12, margin = margin(0, 0, 25, 0)),
    plot.subtitle = element_text(size = 11),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 8),
    axis.title.y = element_text(margin = margin(t = 0, r = 3, b = 0, l = 0)),
    axis.text.y = element_text(vjust = -0.5, margin = margin(l = 20, r = -10)),
    axis.line.x = element_line(colour = "black", size = 0.4),
    axis.ticks.x = element_line(colour = "black", size = 0.4),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank()
  )
```




```r
# Fit an ARIMA model
ARIMA_model <- forecast::auto.arima(econ_data$uempmed)

# Display model summary and residual diagnostics
summary(ARIMA_model)
forecast::checkresiduals(ARIMA_model)
```




```r
# Forecast for the next 36 time periods
ARIMA_forecast <- forecast::forecast(ARIMA_model, h = 36, level = c(95))

# Plot the forecast
plot(ARIMA_forecast)
```


Forecasts from ARIMA(2,1,2)
2
4
6
8
10
12
0
100
200
300
400

Web scraping

Why it matters

Collecting data off websites can be a nightmare. The worst case is manually typing data from a web-page into spreadsheets... but there are *many* steps we can do before resorting to that.

This chapter will outline the process for pulling data off the web, and particularly for understanding the exact web-page element we want to extract.

The notes and code loosely follow the fabulous data tutorial by Grant R. McDermott in his *Data Science for Economists* series. It has been updated to scrape the most recent version and structure of the relevant Wikipedia pages.

First up, let's load some packages.

```
# Install development version of rvest if necessary
if (numeric_version(packageVersion("rvest")) < numeric_version('0.99.0')) {
  remotes::install_github('tidyverse/rvest')
}

# Load and install the packages that we'll be using today
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse, rvest, lubridate, janitor, data.table, hrbrthemes)

library(ggplot2)
library(dplyr)
library(tidyverse)
```

Anatomy of a webpage

Web pages can be categorized as either *server-side rendered* (where content is embedded in the HTML) or *client-side rendered* (where content loads dynami-

cally using JavaScript). When scraping server-side rendered pages, locating the correct CSS or XPath selectors is crucial.

Trawling through CSS code on a webpage is a bit of a nightmare - so we'll use a chrome extension called SelectGadget to help.

The R package that's going to do the heavy lifting is called `rvest` and is based on the python package called Beauty Soup.

Scraping a table

Let's use this wikipedia page as a starting example. It contains various entries for the men's 100m running record.

We can start by pulling all the data from the webpage.

```
m100 <- rvest:: read_html(
               "http://en.wikipedia.org/wiki/Men%27s_100_metres_world_record_progressio
m100
```

```
## {html_document}
## <html class="client-nojs vector-feature-language-in-header-enabled vector-feature-la
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="skin--responsive skin-vector skin-vector-search-vue mediawik ...
```

. . . and we get a whole heap of mumbo jumbo.

To get the table of 'Unofficial progression before the IAAF' we're going to have to be more specific.

Using the SelectGadget tool we can click around and identify that that specific table.

```
pre_iaaf =
  m100 %>%
  html_element("div+ .wikitable :nth-child(1)") %>% ## select table element
  html_table()                                     ## convert to data frame

pre_iaaf
```

```
## # A tibble: 21 x 5
##     Time Athlete              Nationality    'Location of races'     Date
##    <dbl> <chr>                <chr>          <chr>                   <chr>
##  1  10.8 Luther Cary          United States  Paris, France           July 4, 1~
##  2  10.8 Cecil Lee            United Kingdom Brussels, Belgium       September~
```

```
##  3  10.8 Étienne De Ré         Belgium        Brussels, Belgium       August 4,~
##  4  10.8 L. Atcherley          United Kingdom Frankfurt/Main, Germany April 13,~
##  5  10.8 Harry Beaton          United Kingdom Rotterdam, Netherlands  August 28~
##  6  10.8 Harald Anderson-Arbin Sweden         Helsingborg, Sweden     August 9,~
##  7  10.8 Isaac Westergren      Sweden         Gävle, Sweden           September~
##  8  10.8 Isaac Westergren      Sweden         Gävle, Sweden           September~
##  9  10.8 Frank Jarvis          United States  Paris, France           July 14, ~
## 10  10.8 Walter Tewksbury      United States  Paris, France           July 14, ~
## # i 11 more rows
```

Niiiiice - now that's better. Let's do some quick data cleaning.

```
pre_iaaf <- pre_iaaf %>%
            clean_names() %>%
            mutate(date = mdy(date))

pre_iaaf
```

```
## # A tibble: 21 x 5
##     time athlete               nationality    location_of_races       date
##    <dbl> <chr>                 <chr>          <chr>                   <date>
##  1  10.8 Luther Cary           United States  Paris, France           1891-07-04
##  2  10.8 Cecil Lee             United Kingdom Brussels, Belgium       1892-09-25
##  3  10.8 Étienne De Ré         Belgium        Brussels, Belgium       1893-08-04
##  4  10.8 L. Atcherley          United Kingdom Frankfurt/Main, Germany 1895-04-13
##  5  10.8 Harry Beaton          United Kingdom Rotterdam, Netherlands  1895-08-28
##  6  10.8 Harald Anderson-Arbin Sweden         Helsingborg, Sweden     1896-08-09
##  7  10.8 Isaac Westergren      Sweden         Gävle, Sweden           1898-09-11
##  8  10.8 Isaac Westergren      Sweden         Gävle, Sweden           1899-09-10
##  9  10.8 Frank Jarvis          United States  Paris, France           1900-07-14
## 10  10.8 Walter Tewksbury      United States  Paris, France           1900-07-14
## # i 11 more rows
```

Let's also scrape the data for the more recent running records. That's the tables named 'Records (1912-1976)' and 'Records since 1977'.

For the second table:

```
iaaf_76 = m100 %>%
  html_element("#mw-content-text > div > table:nth-child(17)") %>%
  html_table()

iaaf_76 <-iaaf_76 %>%
  clean_names() %>%
  mutate(date = mdy(date))
```

```
iaaf_76
```

```
## # A tibble: 54 x 8
##     time wind   auto athlete       nationality location_of_race date       ref
##    <dbl> <chr> <dbl> <chr>         <chr>       <chr>            <date>     <chr>
##  1  10.6 ""     NA   Donald Lippi~ United Sta~ Stockholm, Swed~ 1912-07-06 [2]
##  2  10.6 ""     NA   Jackson Scho~ United Sta~ Stockholm, Swed~ 1920-09-16 [2]
##  3  10.4 ""     NA   Charley Padd~ United Sta~ Redlands, USA    1921-04-23 [2]
##  4  10.4 "0.0"  NA   Eddie Tolan   United Sta~ Stockholm, Swed~ 1929-08-08 [2]
##  5  10.4 ""     NA   Eddie Tolan   United Sta~ Copenhagen, Den~ 1929-08-25 [2]
##  6  10.3 ""     NA   Percy Willia~ Canada      Toronto, Canada  1930-08-09 [2]
##  7  10.3 "0.4"  10.4 Eddie Tolan   United Sta~ Los Angeles, USA 1932-08-01 [2]
##  8  10.3 ""     NA   Ralph Metcal~ United Sta~ Budapest, Hunga~ 1933-08-12 [2]
##  9  10.3 ""     NA   Eulace Peaco~ United Sta~ Oslo, Norway     1934-08-06 [2]
## 10  10.3 ""     NA   Chris Berger  Netherlands Amsterdam, Neth~ 1934-08-26 [2]
## # i 44 more rows
```

And now for the third table:

```
iaaf <- m100 %>%
        html_element("#mw-content-text > div.mw-parser-output > table:nth-child(23)") %
        html_table() %>%
        clean_names() %>%
        mutate(date = mdy(date))
iaaf
```

```
## # A tibble: 24 x 9
##     time wind   auto athlete        nationality   location_of_race    date
##    <dbl> <chr> <dbl> <chr>          <chr>         <chr>               <date>
##  1 10.1  1.3      NA Bob Hayes      United States Tokyo, Japan        1964-10-15
##  2 10.0  0.8      NA Jim Hines      United States Sacramento, USA     1968-06-20
##  3 10.0  2.0      NA Charles Greene United States Mexico City, Mexico 1968-10-13
##  4  9.95 0.3      NA Jim Hines      United States Mexico City, Mexico 1968-10-14
##  5  9.93 1.4      NA Calvin Smith   United States Colorado Springs, ~ 1983-07-03
##  6  9.83 1.0      NA Ben Johnson    Canada        Rome, Italy         1987-08-30
##  7  9.93 1.0      NA Carl Lewis     United States Rome, Italy         1987-08-30
##  8  9.93 1.1      NA Carl Lewis     United States Zürich, Switzerland 1988-08-17
##  9  9.79 1.1      NA Ben Johnson    Canada        Seoul, South Korea  1988-09-24
## 10  9.92 1.1      NA Carl Lewis     United States Seoul, South Korea  1988-09-24
## # i 14 more rows
## # i 2 more variables: notes_note_2 <chr>, duration_of_record <chr>
```

How good. Now let's bind the rows together to make a master data set.

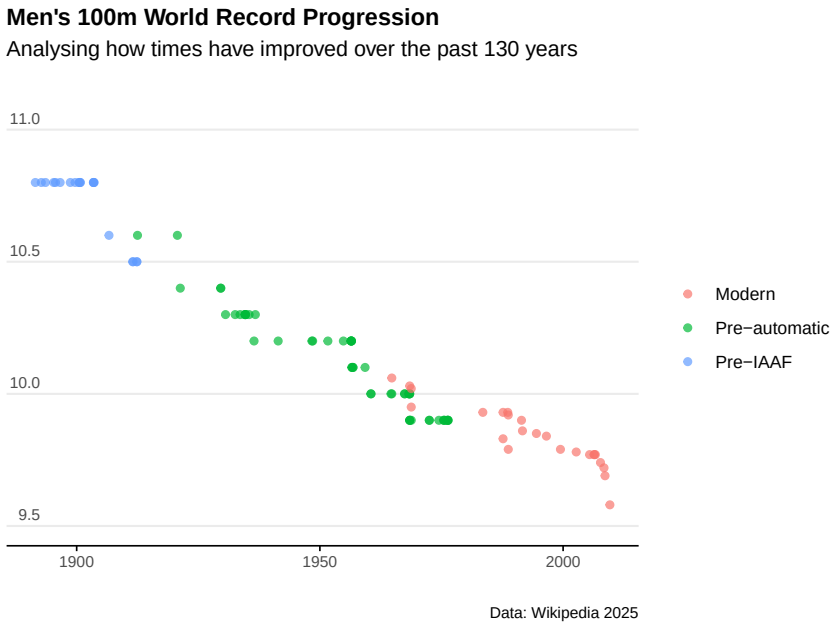
```
wr100 <- rbind(
    pre_iaaf %>% dplyr::select(time, athlete, nationality, date) %>%
    mutate(era = "Pre-IAAF"),
    iaaf_76 %>% dplyr::select(time, athlete, nationality, date) %>%
    mutate(era = "Pre-automatic"),
    iaaf %>% dplyr::select(time, athlete, nationality, date) %>%
    mutate(era = "Modern")
    )

wr100
```

```
## # A tibble: 99 x 5
##     time athlete               nationality    date       era
##    <dbl> <chr>                 <chr>          <date>     <chr>
##  1  10.8 Luther Cary           United States  1891-07-04 Pre-IAAF
##  2  10.8 Cecil Lee             United Kingdom 1892-09-25 Pre-IAAF
##  3  10.8 Étienne De Ré         Belgium        1893-08-04 Pre-IAAF
##  4  10.8 L. Atcherley          United Kingdom 1895-04-13 Pre-IAAF
##  5  10.8 Harry Beaton          United Kingdom 1895-08-28 Pre-IAAF
##  6  10.8 Harald Anderson-Arbin Sweden         1896-08-09 Pre-IAAF
##  7  10.8 Isaac Westergren      Sweden         1898-09-11 Pre-IAAF
##  8  10.8 Isaac Westergren      Sweden         1899-09-10 Pre-IAAF
##  9  10.8 Frank Jarvis          United States  1900-07-14 Pre-IAAF
## 10  10.8 Walter Tewksbury      United States  1900-07-14 Pre-IAAF
## # i 89 more rows
```

Excellent. Let's plot the results.

```
ggplot(wr100) +
  geom_point(aes(x = date, y = time, col = era), alpha = 0.7) +

  labs(
    title = "Men's 100m World Record Progression",
    subtitle = "Analysing how times have improved over the past 130 years",
    caption = "Data: Wikipedia 2025",
    x = "",
    y = ""
  ) +

  theme_minimal() +

  scale_y_continuous(limits = c(9.5, 11), breaks = c(9.5, 10, 10.5, 11)) +

  theme(
    axis.text.y = element_text(vjust = -0.5, margin = margin(l = 20, r = -20)),
```

```
    plot.subtitle = element_text(margin = margin(0, 0, 25, 0),size=11),
    legend.title = element_blank(),
    plot.title = element_text(face = "bold", size = 12),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 8),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    axis.line.x = element_line(colour = "black", size = 0.4),
    axis.ticks.x = element_line(colour = "black", size = 0.4)
)
```



Text mining

Numbers are great... but words literally tell a story. Analysing text (e.g. books, tweets, survey responses) in a quantitative format is naturally challenging - however there's a few tricks which can simplify the process.

This chapter outlines the process for inputting text data, and running some simple analysis. The notes and code loosely follow the fabulous book *Text Mining with R* by Julia Silge and David Robinson.

First up, let's load some packages.

```
library(ggplot2)
library(dplyr)
library(tidyverse)
library(tidytext)
library(textdata)
```

Frequency analysis

There's a online depository called Project Gutenberg which catalogue texts that have lost their copyright.

It just so happens that The Bible is on this list. Let's check out the most frequent words.

```
library(tidyverse)
library(tidytext)

# Correct URL for the raw text file
bible_url <- "https://raw.githubusercontent.com/charlescoverdale/casualdabbler2e/main/data/bible.

# Read the text file directly from the URL
bible <- read_lines(bible_url)
```

```
# Convert to a tibble
bible_df <- tibble(text = bible)

# Tokenize words and remove stop words
bible_tidy <- bible_df %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word")

# Find and display the most common words
common_words <- bible_tidy %>%
  count(word, sort = TRUE) %>%
  head(20)  # Show the top 20 words

print(common_words)
```

```
## # A tibble: 20 x 2
##    word       n
##    <chr>  <int>
##  1 lord    7830
##  2 thou    5474
##  3 thy     4600
##  4 god     4446
##  5 ye      3983
##  6 thee    3827
##  7 1       2830
##  8 2       2724
##  9 3       2570
## 10 israel  2565
## 11 4       2476
## 12 son     2370
## 13 7       2351
## 14 5       2308
## 15 6       2297
## 16 hath    2264
## 17 king    2264
## 18 9       2210
## 19 8       2193
## 20 people  2142
```

Somewhat unsurprisingly - “lord” wins it by a country mile.

Sentiment analysis

Just like a frequency analysis, we can do a 'vibe' analysis (i.e. sentiment of a text) using a clever thesaurus matching technique.

In the `tidytext` package are lexicons which include the general sentiment of words (e.g. the emotion you can use to describe that word).

Let's see the count of words most associated with 'joy' in the bible.

```
# Tokenize words
bible_tidy <- bible_df %>%
  unnest_tokens(word, text) %>%
  mutate(word = tolower(word))  # Ensure lowercase matching

# Get NRC lexicon & filter for "joy"
nrcjoy <- tidytext::get_sentiments("nrc") %>%
  filter(sentiment == "joy")

# Join words with NRC joy sentiment list & count occurrences
bible_joy_words <- bible_tidy %>%
  inner_join(nrcjoy, by = "word") %>%
  count(word, sort = TRUE)

# View top joyful words
print(bible_joy_words)
```

```
## # A tibble: 264 x 2
##    word         n
##    <chr>    <int>
##  1 god       4446
##  2 good       720
##  3 art        494
##  4 peace      429
##  5 found      404
##  6 glory      402
##  7 daughter   324
##  8 pray       313
##  9 love       310
## 10 blessed    302
## # i 254 more rows
```

Geocoding

Drivetime analysis

Raster data

ABS economic indicators

Australian election data

Bayesian for the common man