R cookbook for the casual dabbler (2nd edition)

Charles Coverdale

2025-02-24

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.

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. Additionally:

- The Australian specific chapters have also been grouped together (e.g. election data and ABS economic indicators).
- There are new chapters on functions, packages, as well as more advanced econometric techniques (Bayesian approaches and causal inference).
- Worked examples have been updated to reflect best (or at least better) practice.
- Many a bug and typo has been fixed.

Usage

In each chapter I've written up the background, methodology and worked example 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. RCCD2e includes optimized (and better formated) code which has gone through the scrutiny of many of the different LLM's.

I encourage users to paste in code snippets to language models (or use Cursor) for deeper explanations or alternate examples.

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 working across London and Melbourne. 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)
```

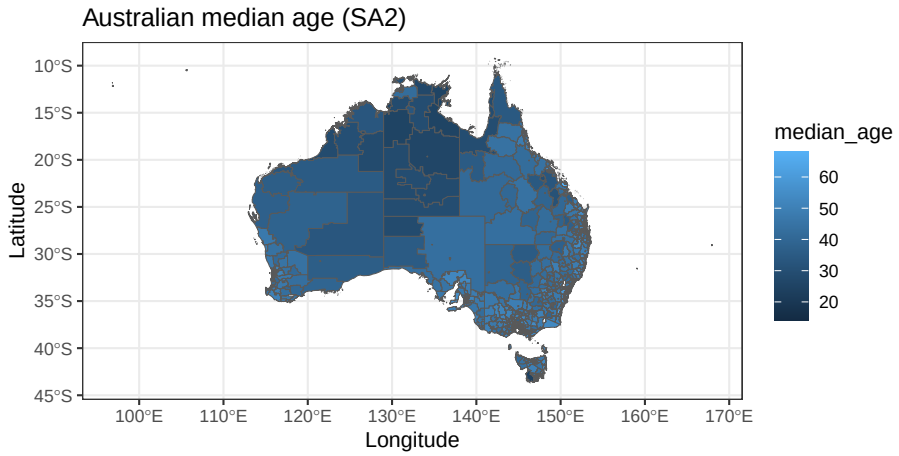
Will Mackey's `absmapsdata` package contains all the ABS' ASGS shapefiles. The data is now also callable through the (thoroughly updated and expanded) `strayr` package. For example:

```
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.

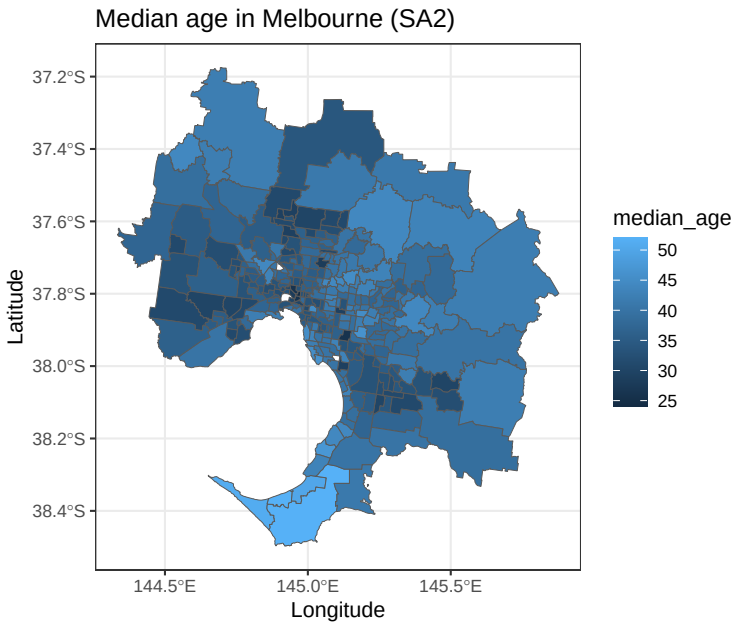
From okay to good

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.

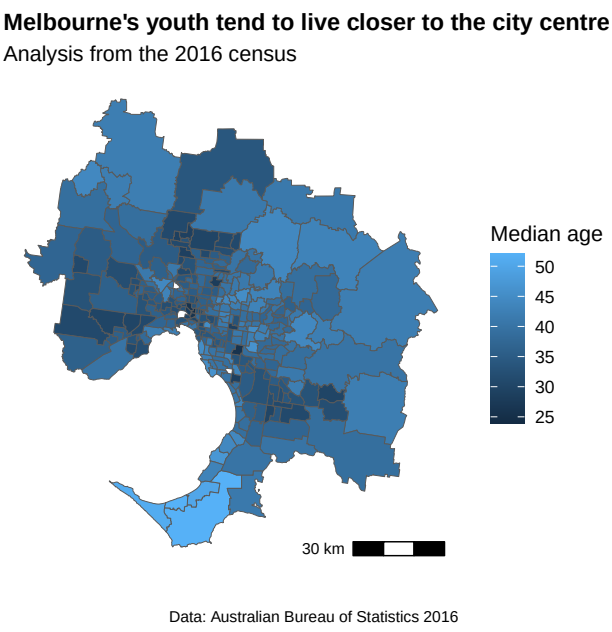
From good to great

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
```

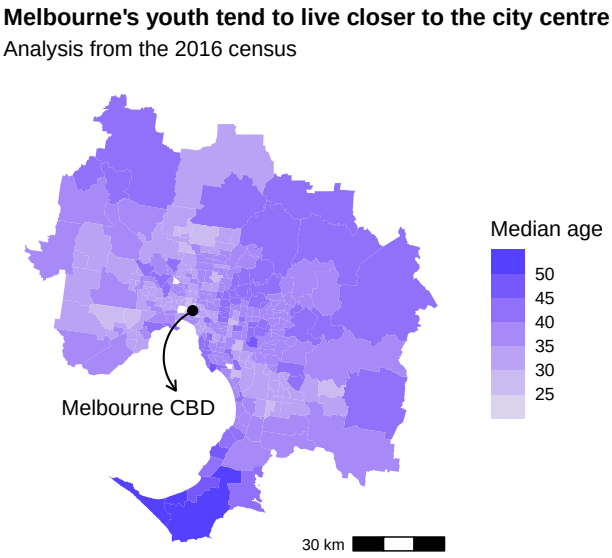


From great to fantastic

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

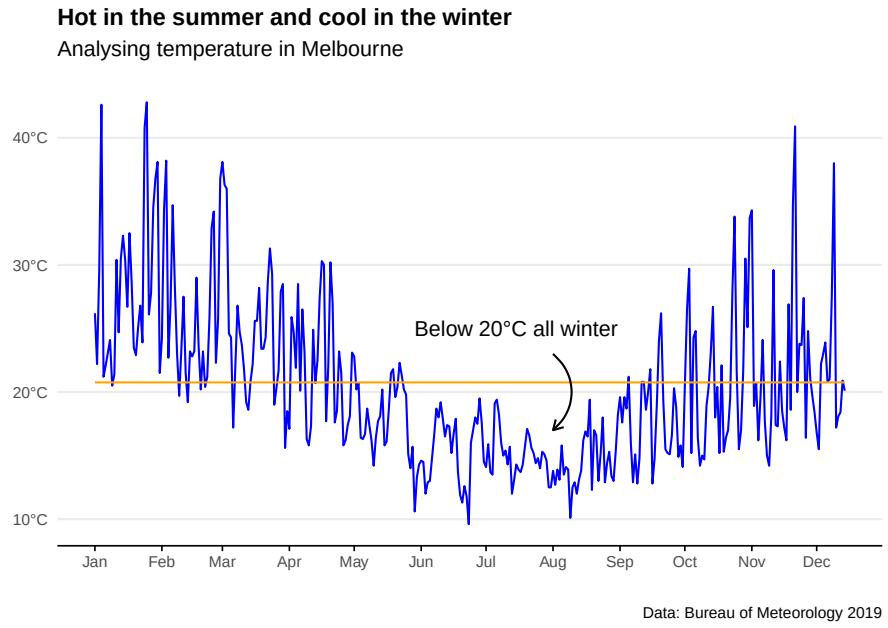
Before making a chart ensure the data is "tidy" - meaning there is a new row for every changed variable. It also doesn't hurt to remove NA's for consistency (particularly in time series).

```
#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",
```

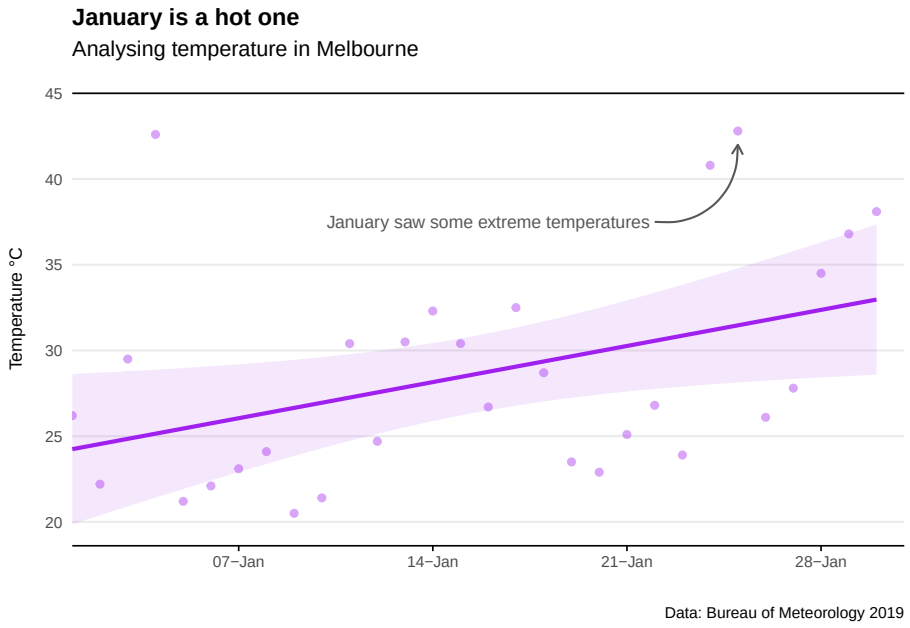
```
    caption = "Data: Bureau of Meteorology 2019",
    x = "",
    y = ""
  ) +
  scale_x_date(
    date_breaks = "1 month",
    date_labels = "%b",
    limits = as.Date(c('2019-01-01', '2019-12-14'))
  ) +
  scale_y_continuous(labels = unit_format(unit = "\u00b0C", sep = "")) +
  theme_minimal() +
  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 = ggplot2::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)
  )
```

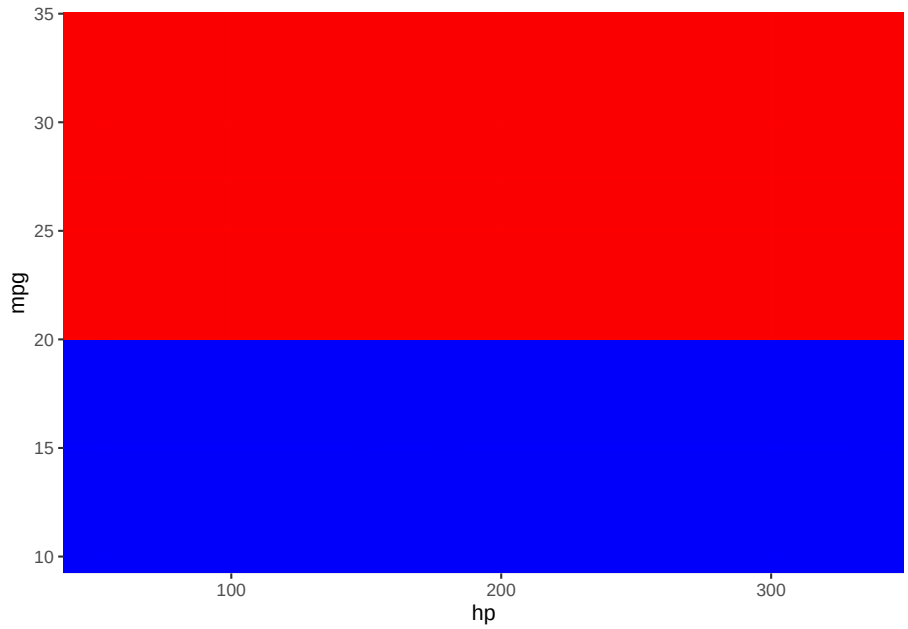


Shading areas on plots

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

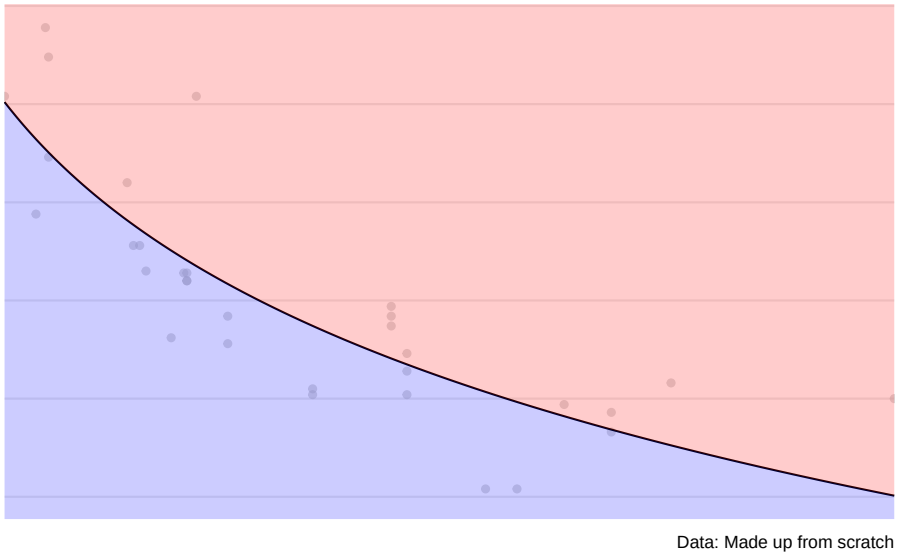
Adding shading under a particular model line? A little trickier. See both example below.

```
# 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",
```

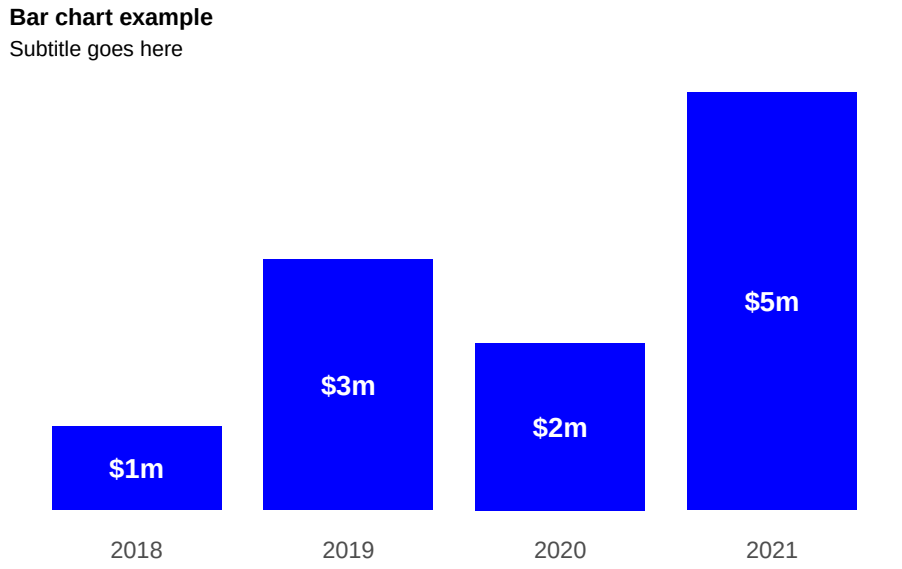
```
    caption = "Data: Made up from scratch",
    x = "",
    y = ""
  ) +

  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()
  )
```

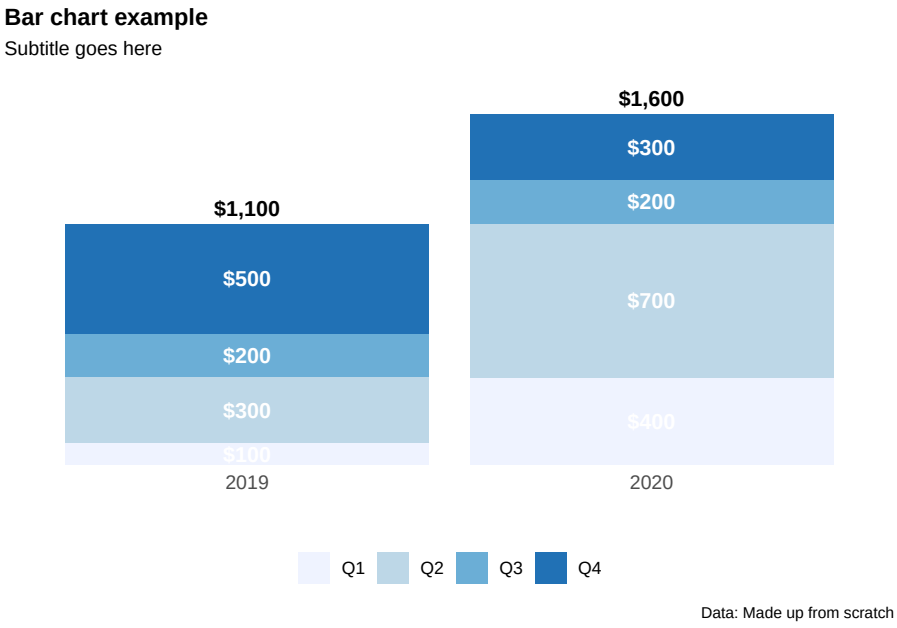



```
# Save the plot
# ggsave("test.png", width = 10, height = 10, units = "cm", dpi = 600)
```

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

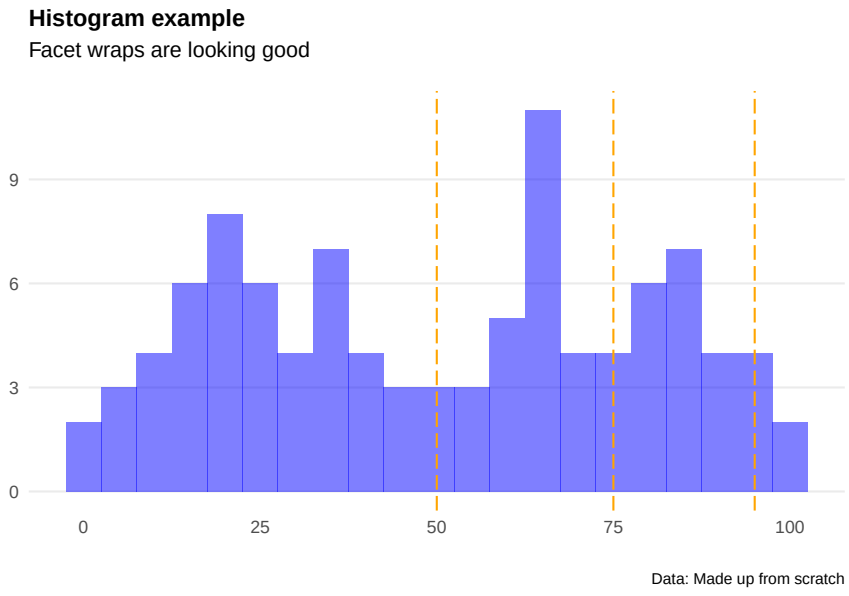
Aka. a bar chart for a continuous variable where the bars are touching. Useful to show distribution of time series or ordinal variables.

```
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="",
```

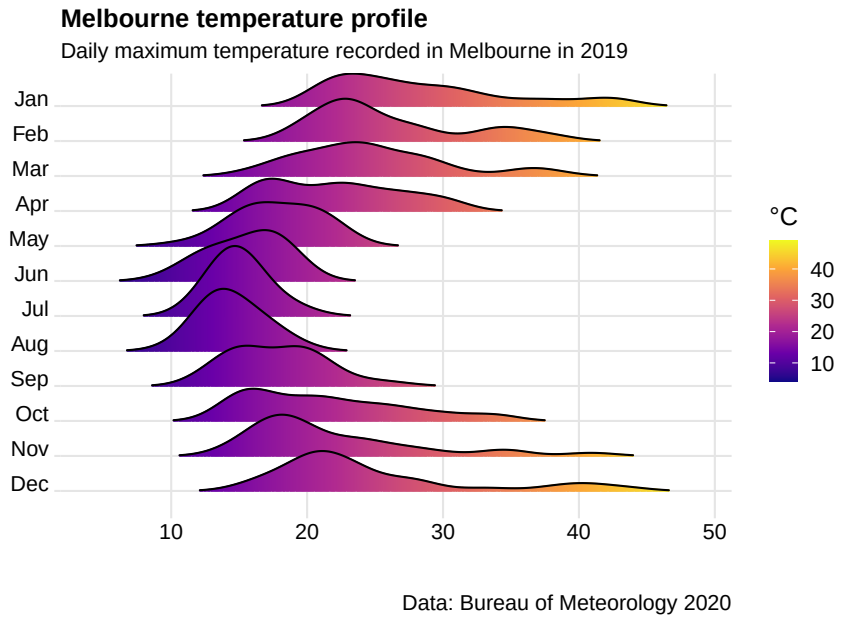
```
                    y="") +


          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=ggplot2::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).

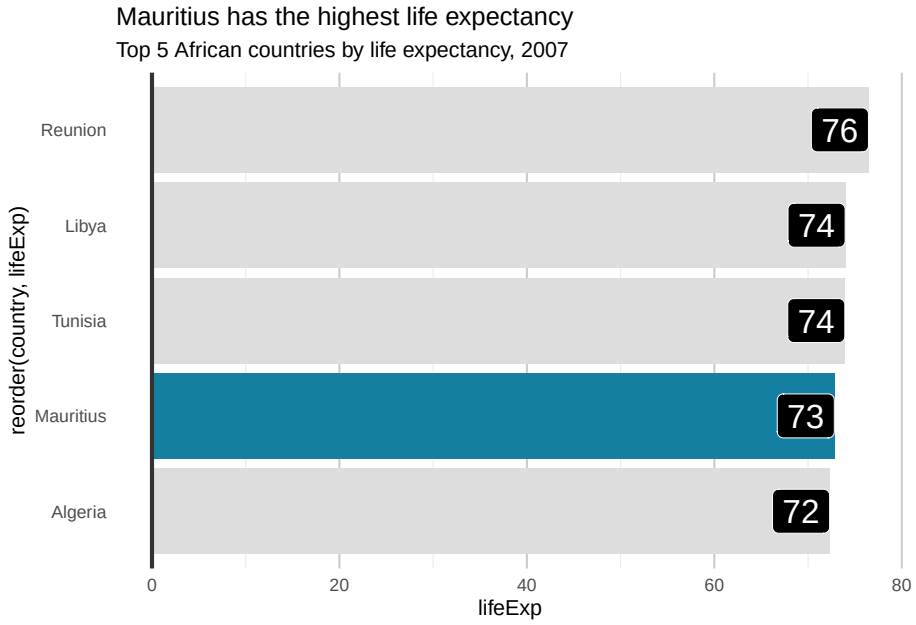
```
# Import data
url <-"https://raw.githubusercontent.com/charlescoverdale/ggridges/master/2019_MEL_max

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
```

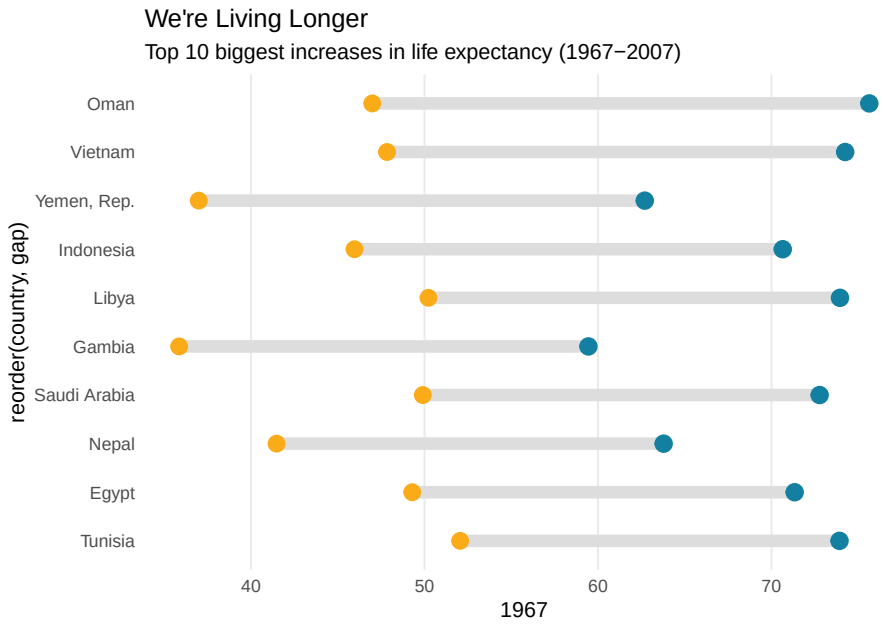


BBC style: Dumbbell charts

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.

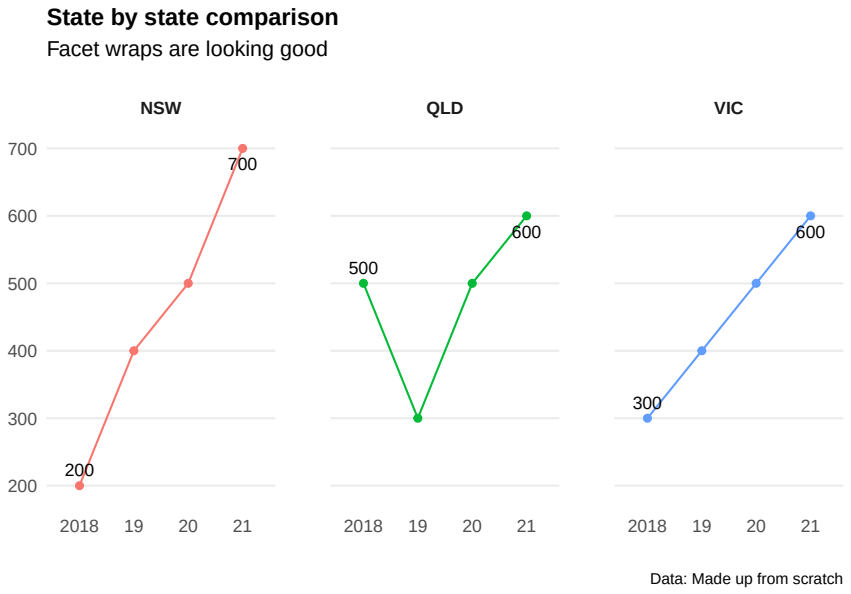
Top tips: `facet_wrap()`dataframes need to be in long form in order to be manipulated easily.

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=ggplot2::margin(0,0,15,0))) +

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

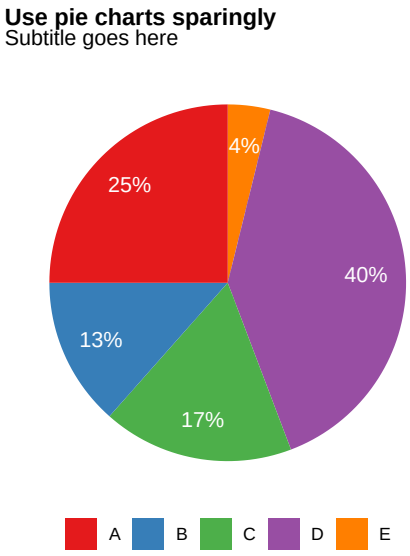


Pie chart

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=ggplot2::margin(0,0,5,0))) +
  theme(plot.margin=unit(c(0.5,0.5,0.5,0.5),"cm"))
```



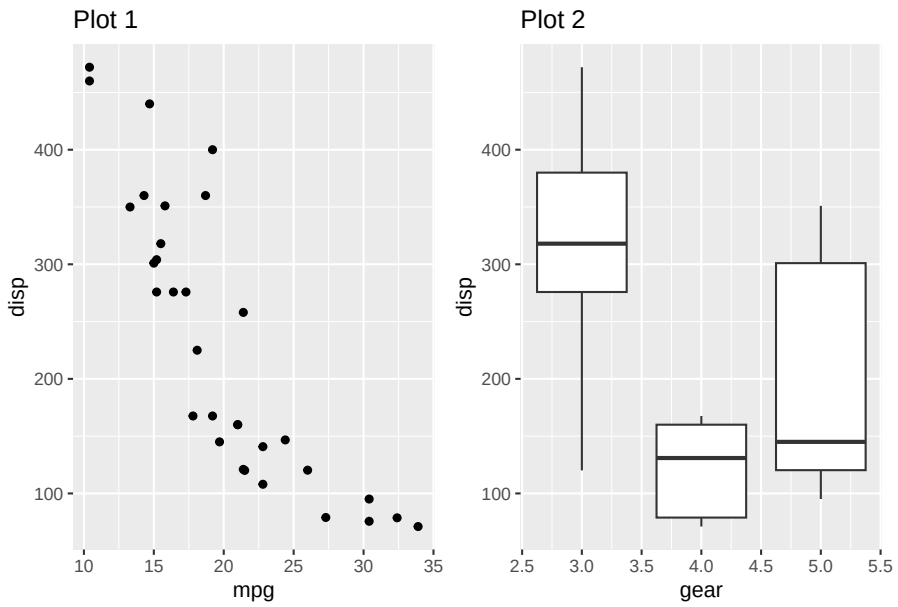

```
  # ggsave(plot=last_plot(),
  #       width=10,
  #       height=10,
  #       units="cm",
  #       dpi = 600,
  #       filename = "/Users/charlescoverdale/Desktop/pietest.png")
```

Patchwork

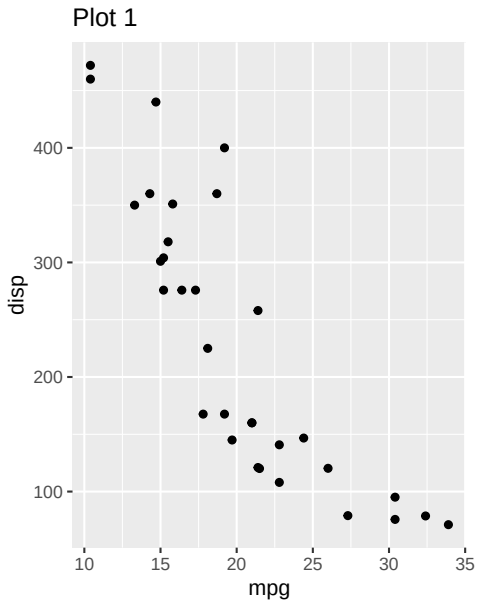
Patchwork is a nifty package for arranging plots and other graphic elements (text, tables etc) in different grid arrangements. The basic syntax is to use `plot1 | plot2` for side by side charts, and `plot1 / plot2`for top and bottom charts. You can also combine these two functions for a grid of different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

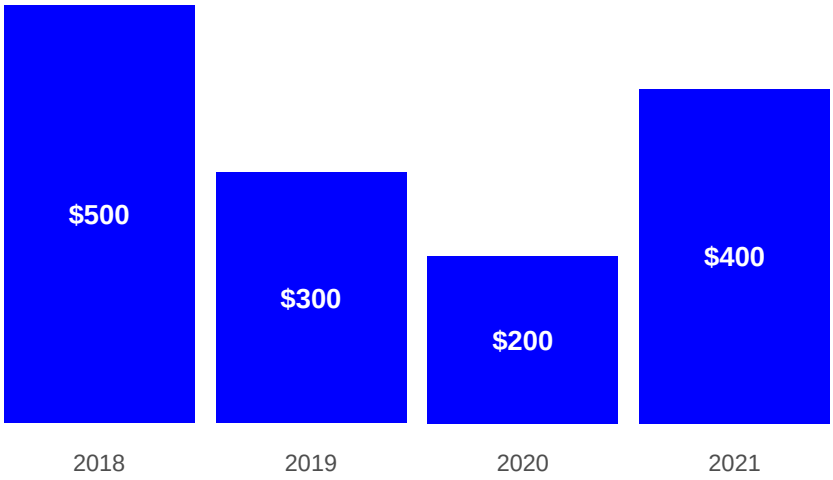
Automating chart creation

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",
```

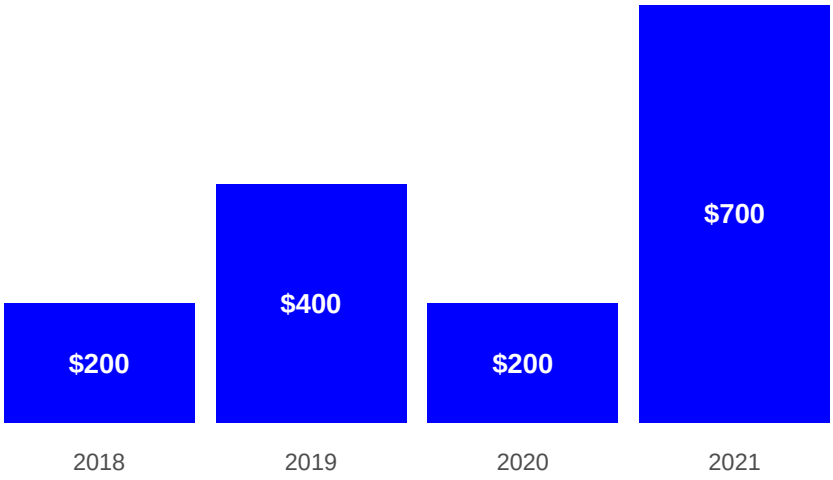
```
            fontface="bold",
            position = position_stack(vjust = 0.5))+


  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

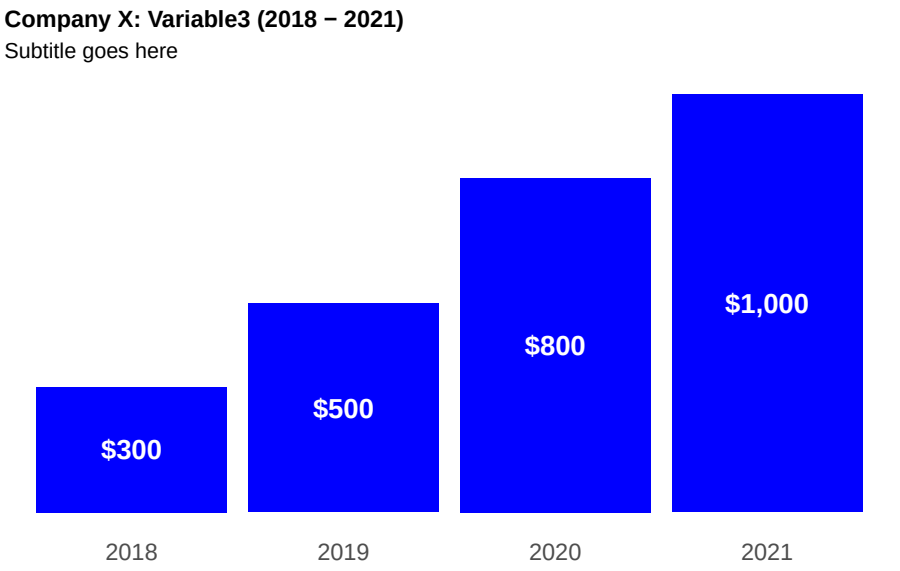


Data: Made up from scratch

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).

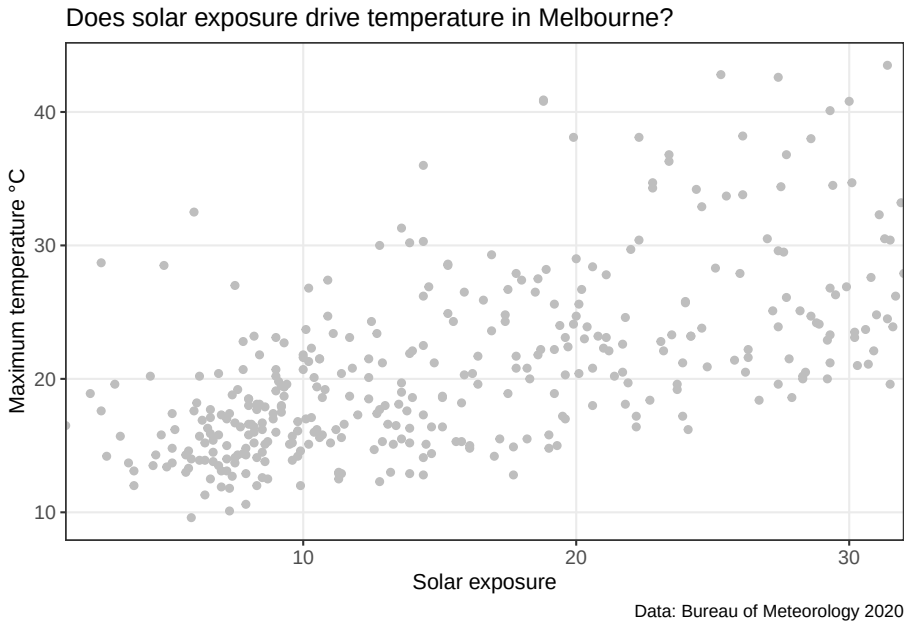
```
#Add a proper date column
MEL_weather_2019 <- MEL_weather_2019 %>%
  mutate(Date = make_date(Year, Month, Day))
```

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)
```

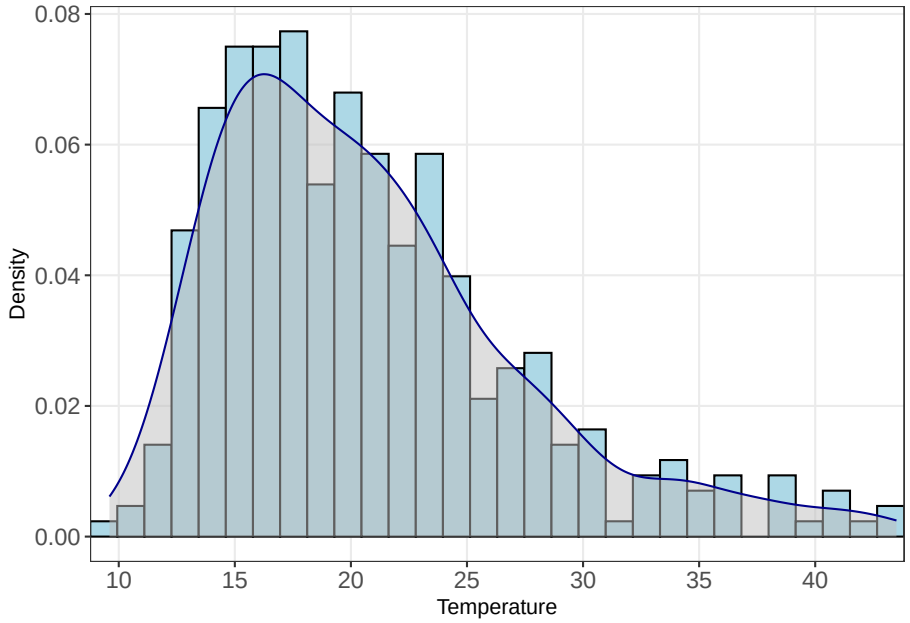
We're aiming to investigate if other weather variables can predict maximum temperatures. Solar exposure seems like a plausible place to start. We start by plotting the two variables to if there is a trend.

```
#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

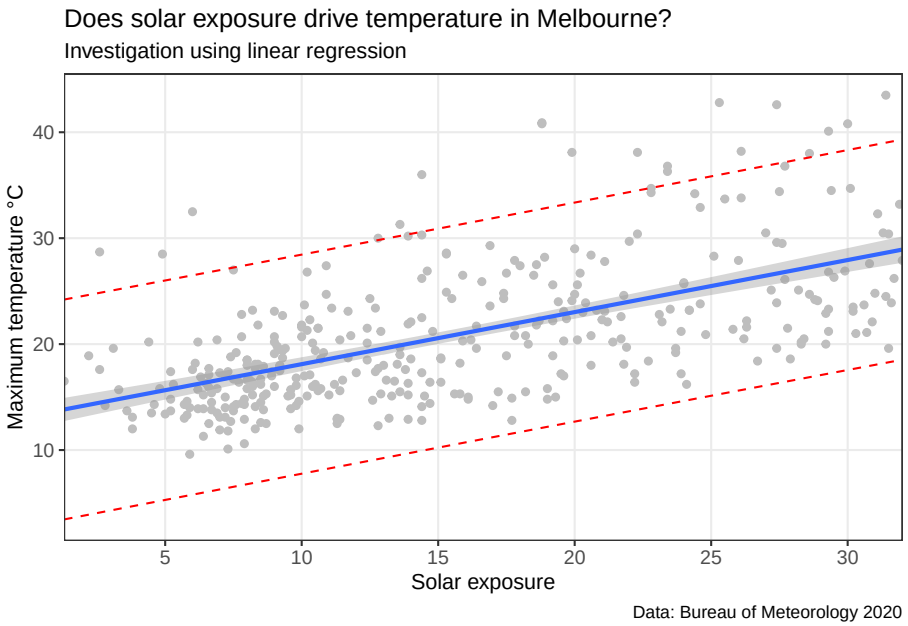
We can use this lm function to predict values of temperature based on the level of solar exposure. We can then compare this to the actual temperature record, and see how well the model fits the data set.

```
#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
```



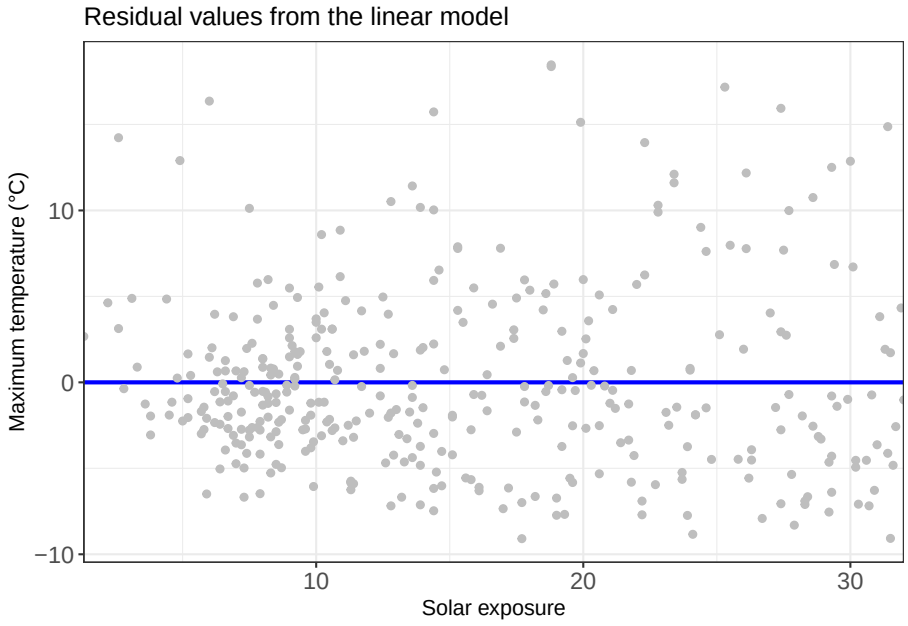
This chart includes the model (blue line), confidence interval (grey band around the blue line), and a prediction interval (red dotted line). A prediction interval reflects the uncertainty around a single value (put simple: what is the reasonable upper and lower bound that this data point could be estimated 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).

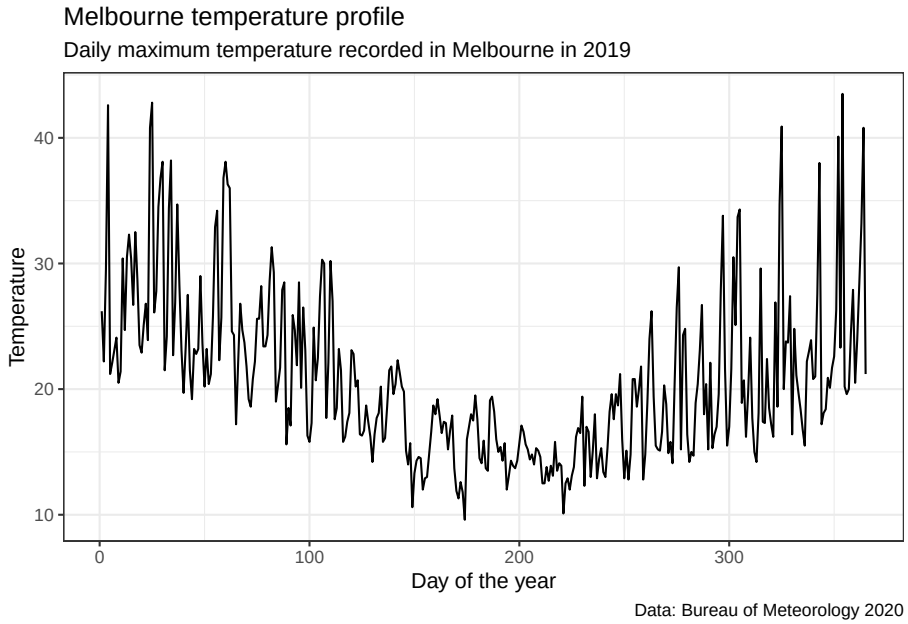
However we can expect that temperature follows a non-linear pattern throughout the year (in Australia it is hot in January-March, cold in June-August, then starts to warm up again). A linear model (e.g. a straight line) will be a very bad model for temperature — we need to introduce polynomials.

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)
```

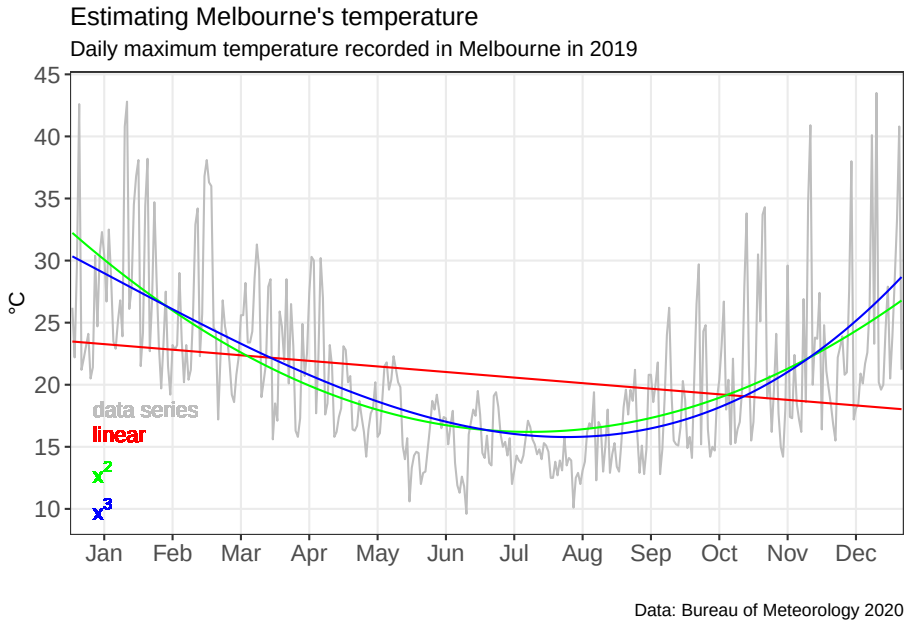
```
summary(poly2)
#Create a polynominal of order 3 to fit this data
poly3 <- lm(Max_temp ~ poly(Day_number,3,raw=TRUE),
         data=MEL_weather_2019)
summary(poly3)

#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
```

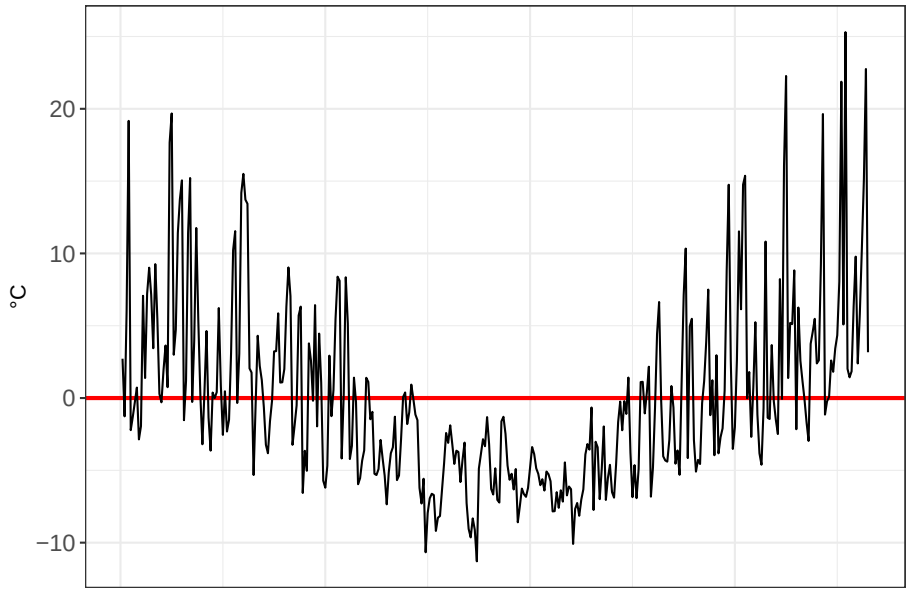


We can see in the chart above the polynomial models do much better at fitting the data. However, they are still highly variant.

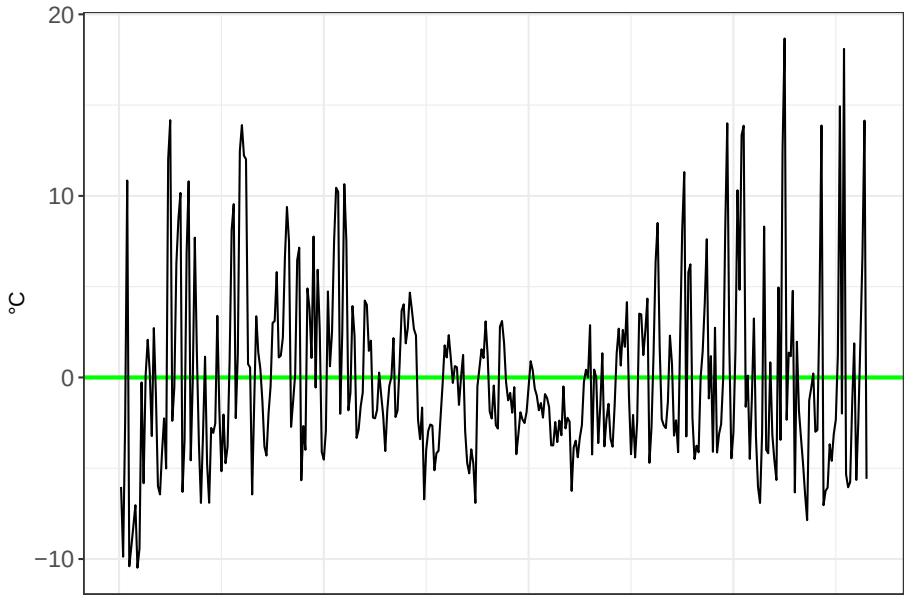
Just how variant are they? We can look at the residuals to find out. The residuals is the gap between the observed data point (i.e. the grey line) and our model.

```
#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")+
```

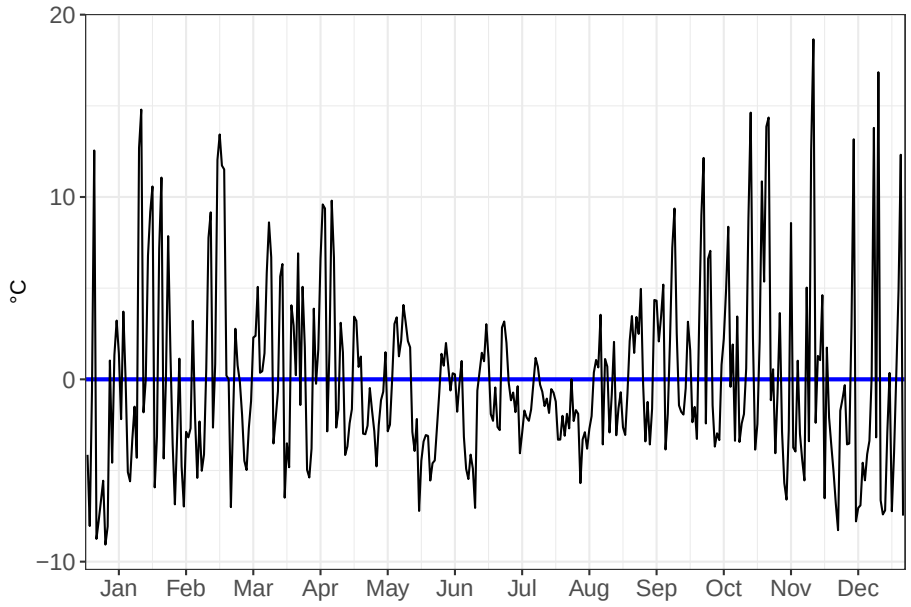
```
  theme_bw()+
  theme(axis.text=element_text(size=12))+
  theme(axis.ticks.x=element_blank(),
  axis.text.x=element_blank())
residuals_poly1_chart
```



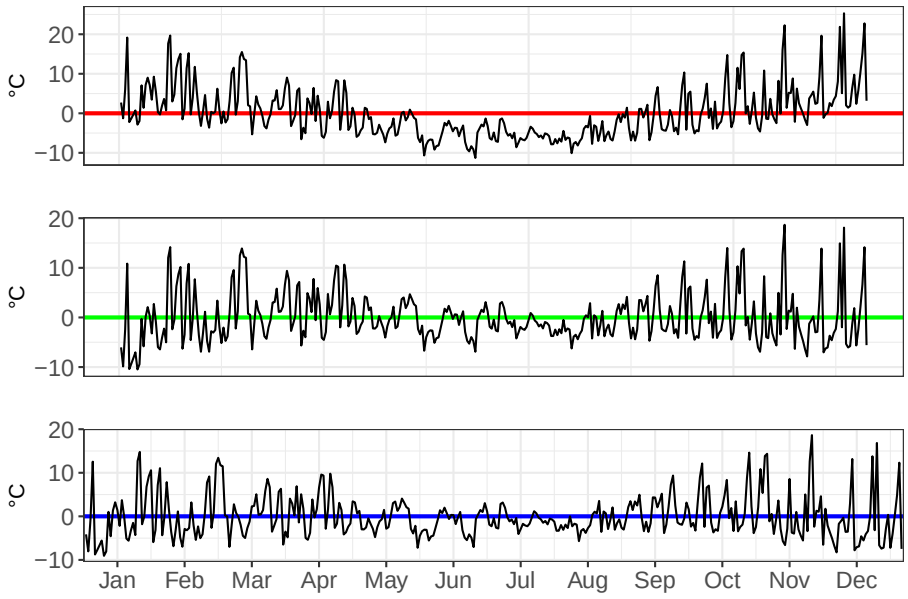

```
#Get the residuals for poly2
residuals_poly2 <- MEL_weather_2019%>%
  add_residuals(poly2)
residuals_poly2_chart <- ggplot(data=residuals_poly2,aes(x=Day_number, y=resid))+
  geom_ref_line(h=0,colour="green", size=1)+
  geom_line()+
  xlab("")+
  ylab("°C")+
  theme_bw()+
  theme(axis.text=element_text(size=12))+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

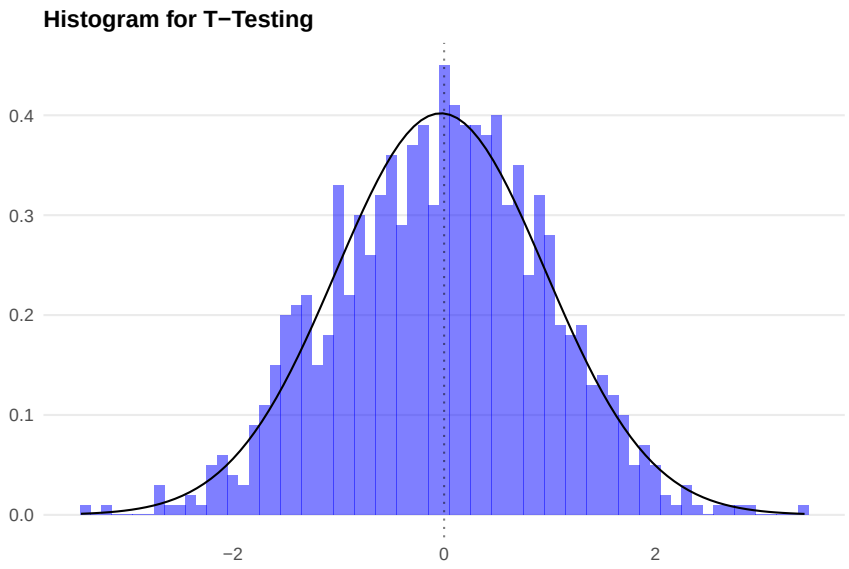
We'll start by creating a normally distributed random dataset using `rnorm`.

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")
```

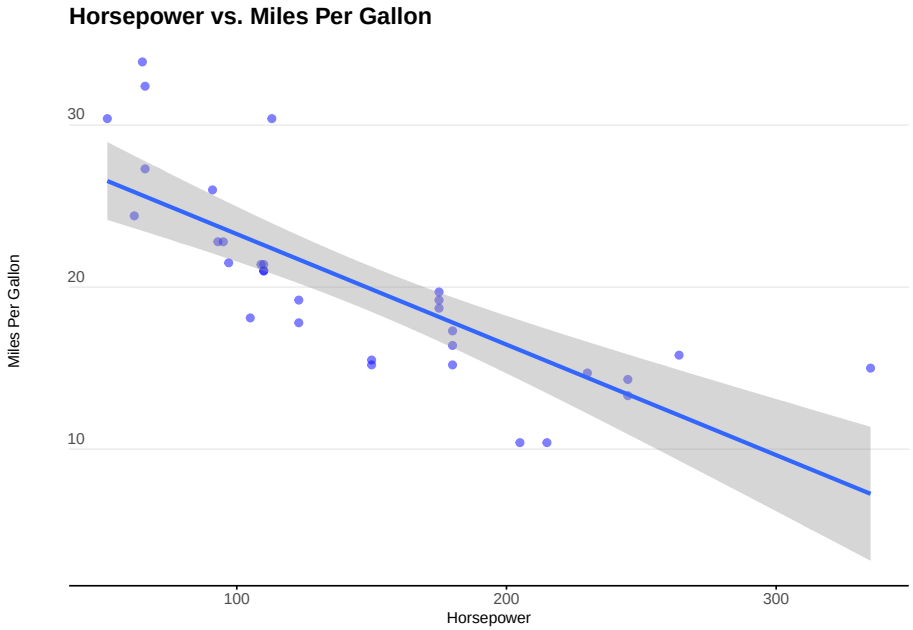
We see here the p-value is greater than 0.05, leading us to *fail to reject* the null hypothesis. In a sentence, we cannot say that the mean of dataset1$variable1 is different to 0.01.

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 = ggplot2::margin(0, 0, 25, 0)),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 8),
    axis.title.y = element_text(margin = ggplot2::margin(r = 3)),
    axis.text.y = element_text(vjust = -0.5, margin = ggplot2::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.

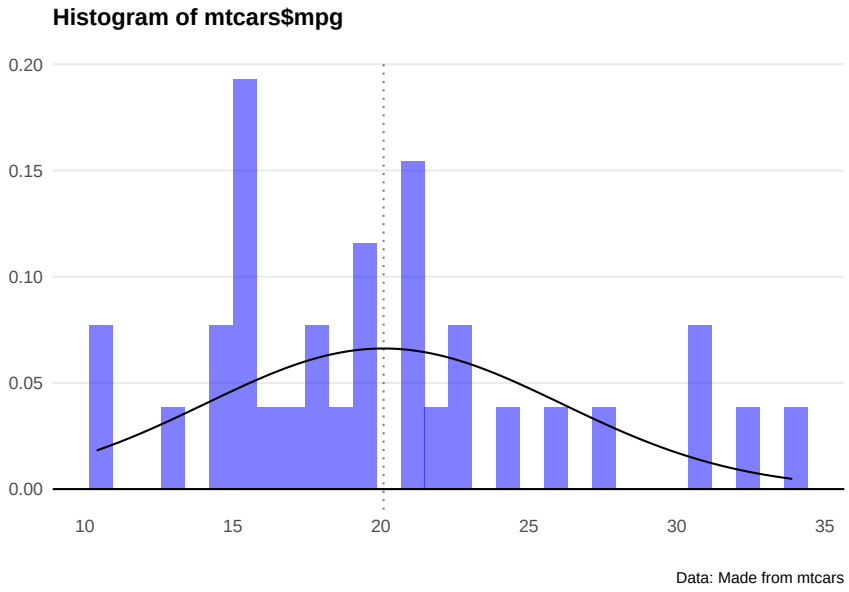
We specify the method as 'pearson' for the Pearson correlation coefficient.

```
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.

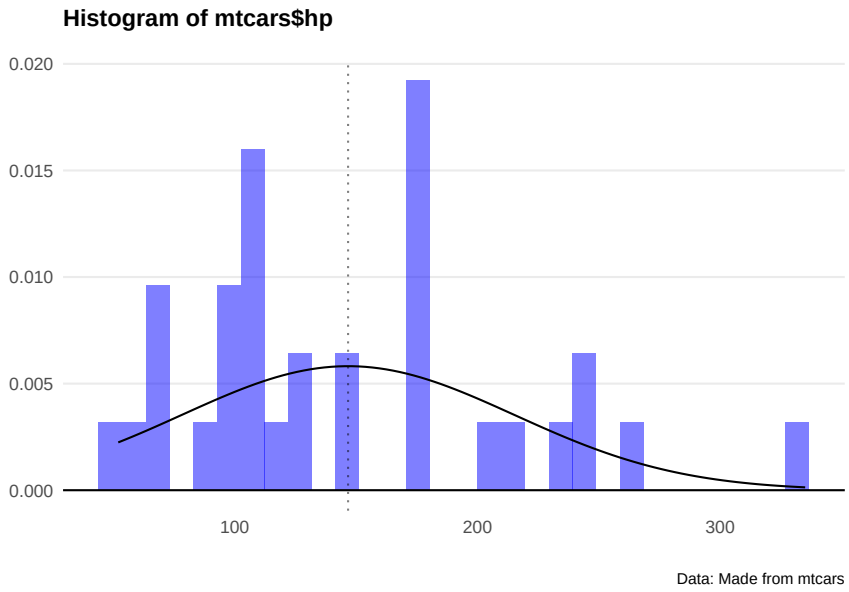
```
ggplot()+
  geom_histogram(aes(x=mtcars$mpg,y=..density..),fill="blue",alpha=0.5) +
  stat_function(fun = dnorm,
                args = list(mean = mean(mtcars$mpg), sd = sd(mtcars$mpg)))+
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = mean(mtcars$mpg), linetype="dotted",alpha=0.5)+
  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

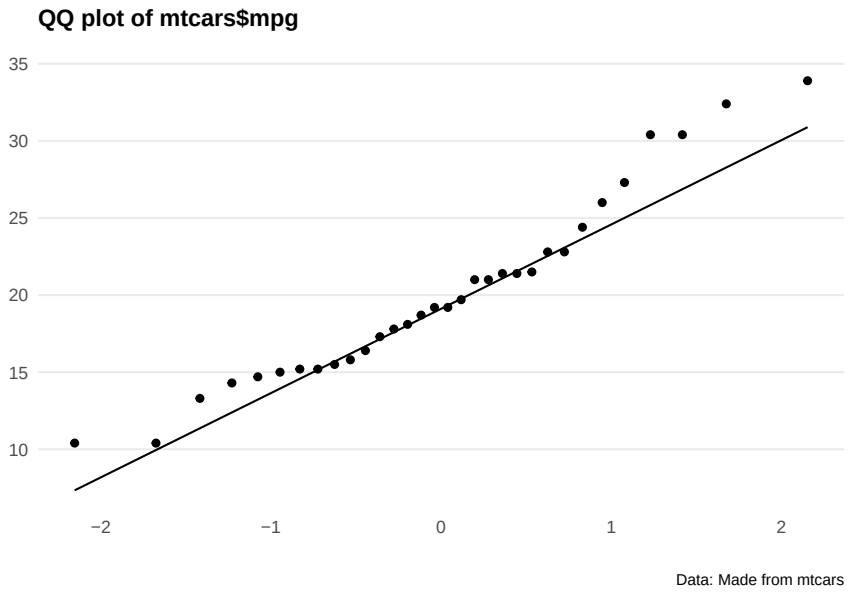


Crikey... they don't look very normal at all.

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

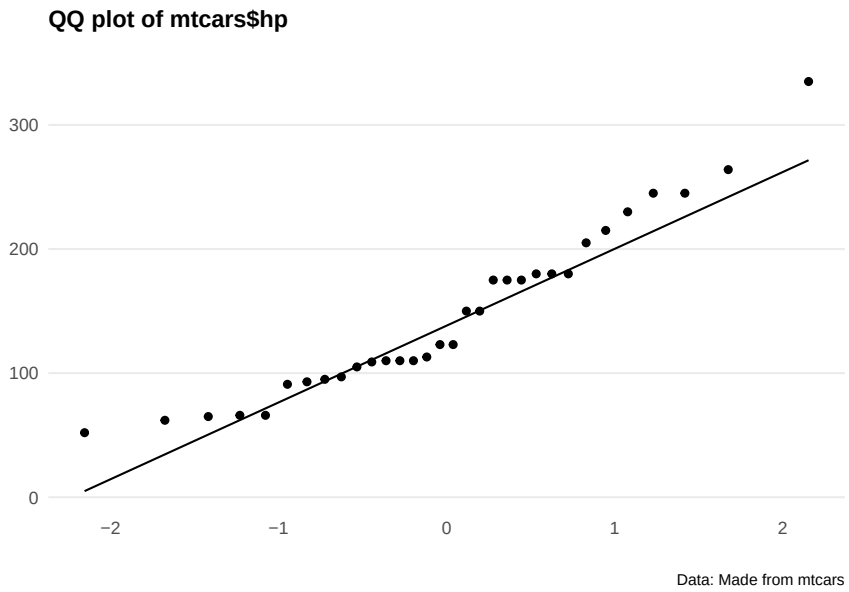
```
ggplot(mtcars, aes(sample = mpg)) +
  geom_qq()+
  geom_qq_line()+
            labs(title="QQ plot of mtcars$mpg",
                         caption = "Data: Made from mtcars",
                         x="",
                         y="") +
            theme_minimal() +
            theme(panel.spacing.x = 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))
```

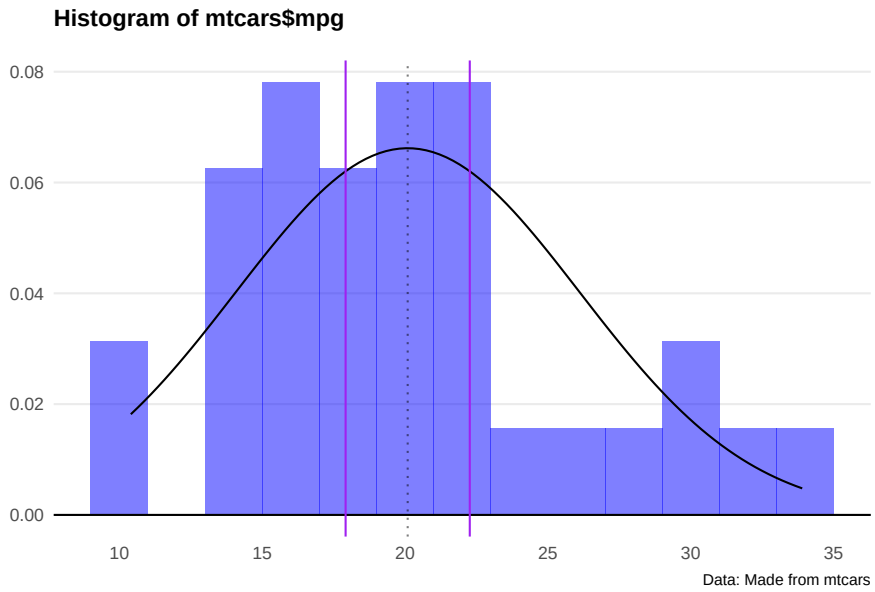
For the lazy folks among us - there's also this quick and dirty way of doing it.

```
# 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 = ggplot2::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")
  )
```

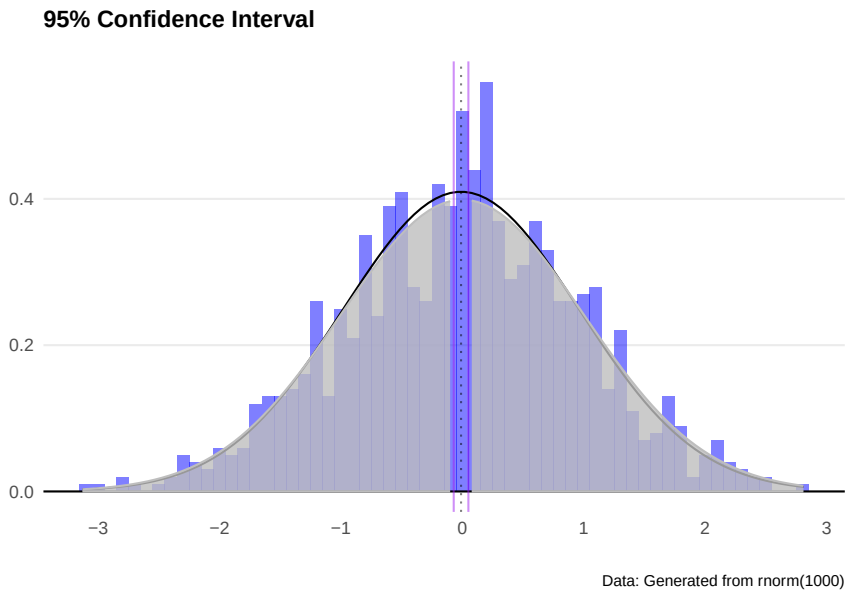


Two things we note here: Firstly, the distribution doesn't look *that* normal. Secondly, the 95% confidence interval looks narrow as a result.

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 = ggplot2::margin(0, 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.

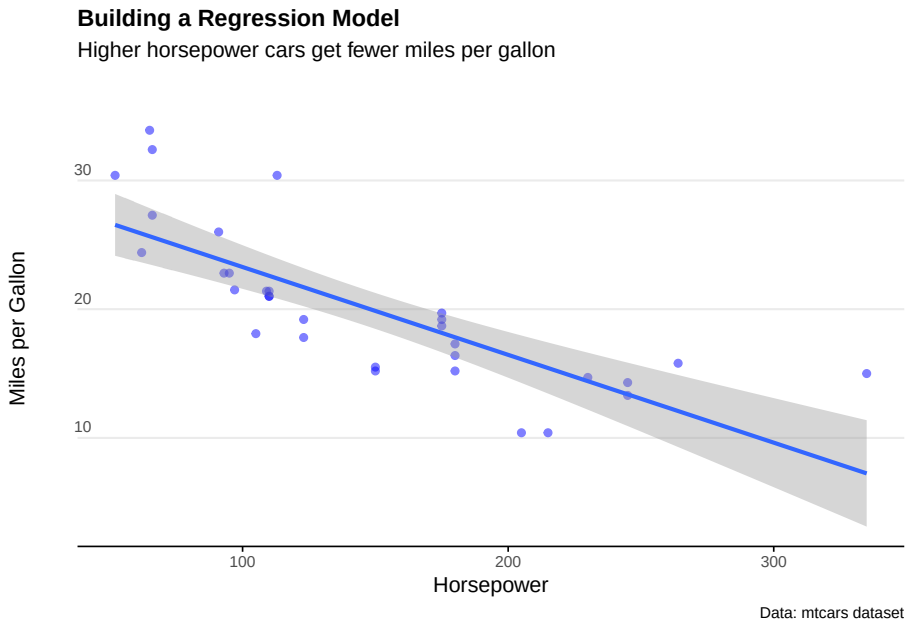
We'll create a linear model of mpg on the y-axis and horsepower on the x-axis.

```
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 = ggplot2::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 = ggplot2::margin(r = 3)),
    axis.text.y = element_text(vjust = -0.5, margin = ggplot2::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
library(readrba)     # Get aus econ data
```

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.

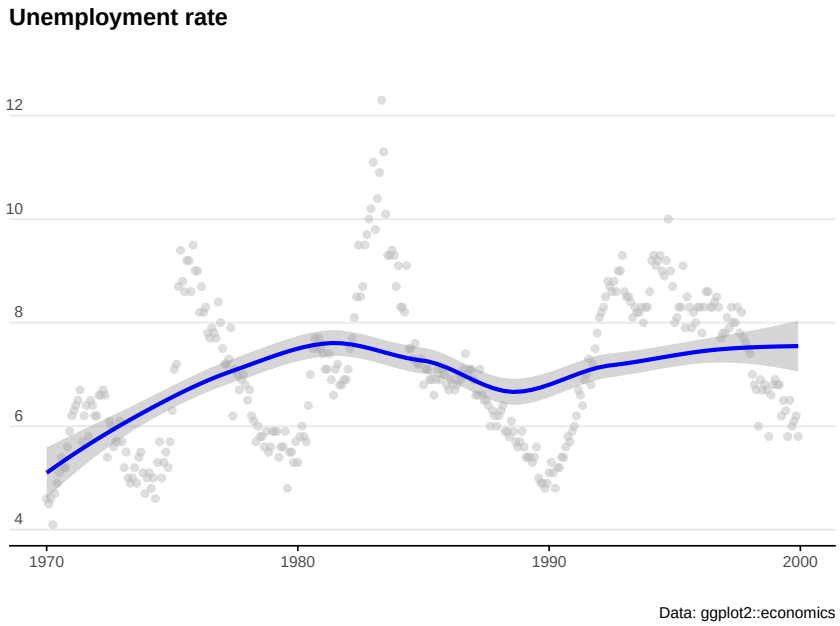
```
econ_data <- economics %>% dplyr::select(c("date", "uempmed"))
econ_data <- econ_data %>% dplyr::filter((date >= as.Date("1970-01-01")
                                          & date <= as.Date("1999-12-31")))
```

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

```
aus_unemp_rate <- readrba::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 = ggplot2::margin(0, 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 = ggplot2::margin(t = 0, r = 3, b = 0, l = 0)),
    axis.text.y = element_text(vjust = -0.5, margin = ggplot2::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"
  )
```



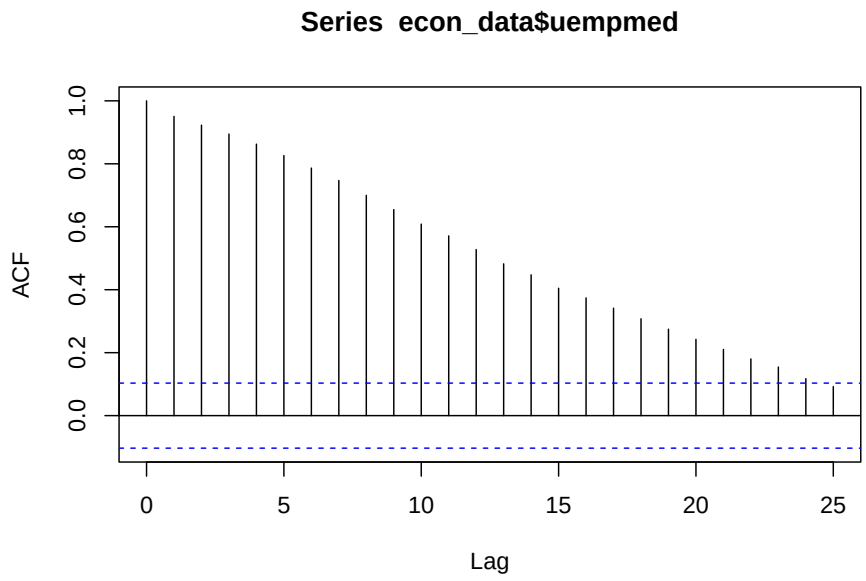
ARIMA

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.

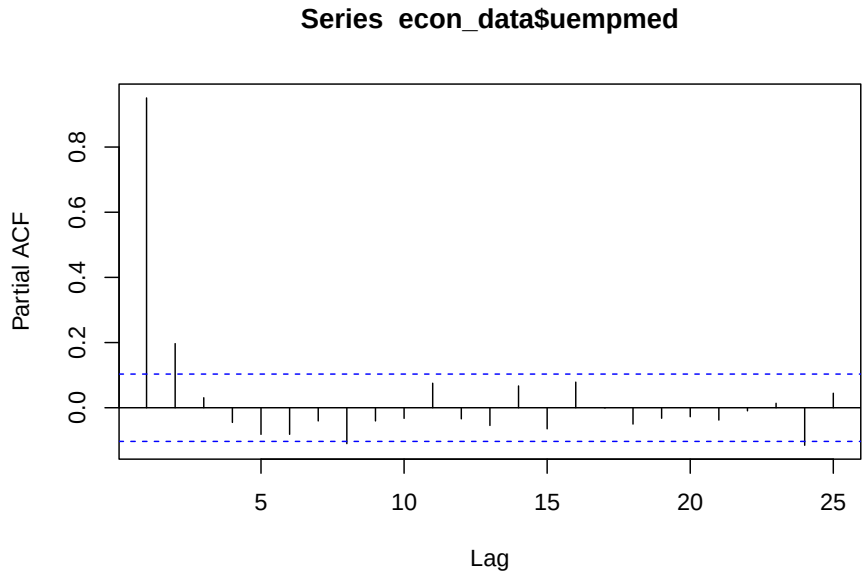
When working with ARIMAs, we need to 'difference' our series to make it stationary.

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.

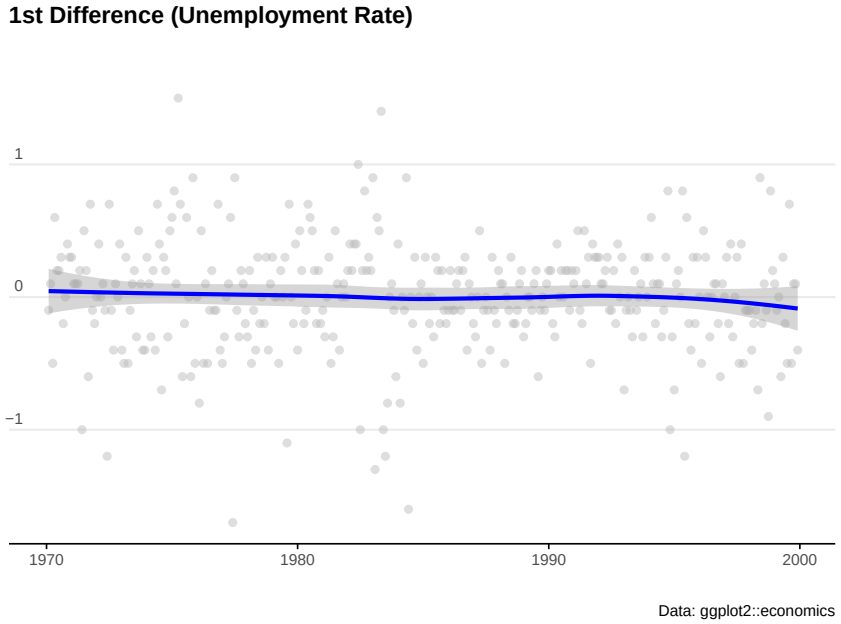
```
# Test for stationarity
aTSA::adf.test(econ_data$uempmed)

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



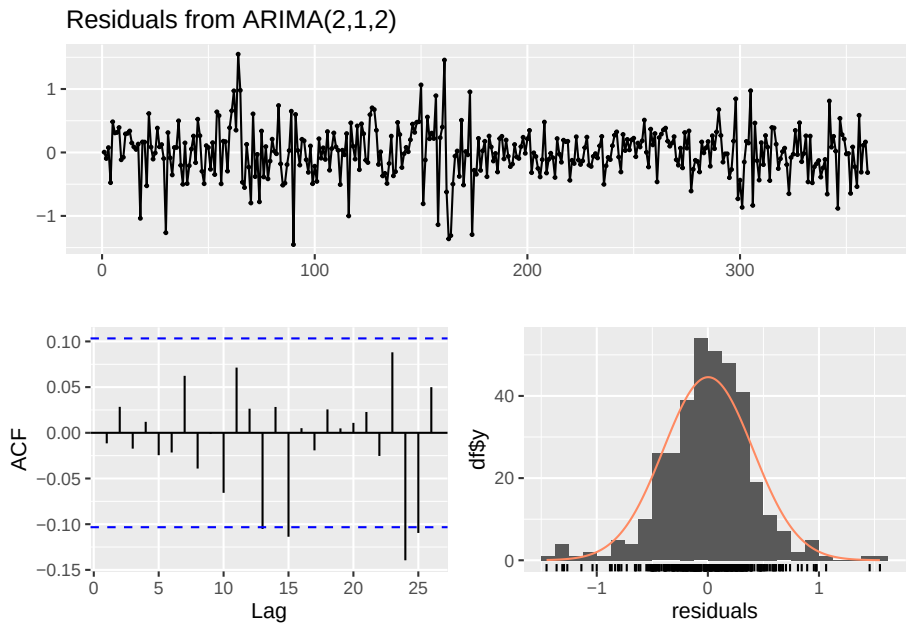

```
# Identify partial auto correlation
pacf(econ_data$uempmed)
```



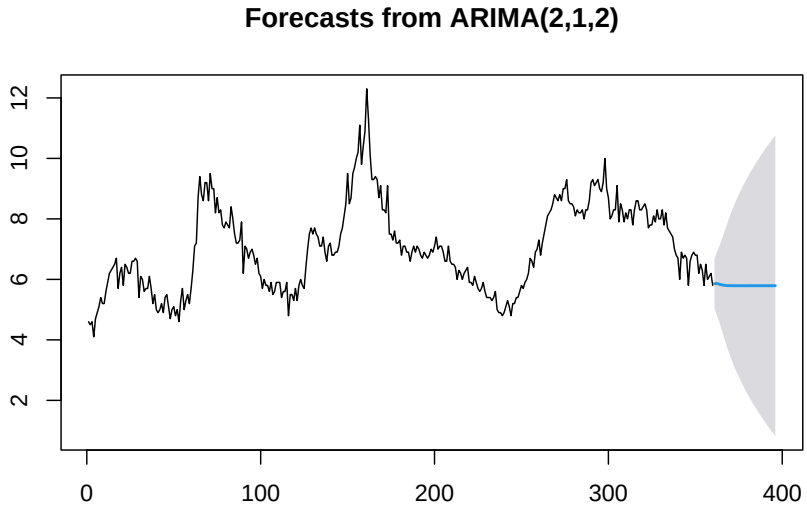
```r
# Take the first differences of the series
econ_data <- econ_data %>% mutate(diff = uempmed - lag(uempmed))

# 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 = ggplot2::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 = ggplot2::margin(t = 0, r = 3, b = 0, l = 0)),
    axis.text.y = element_text(vjust = -0.5, margin = ggplot2::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)
```



Machine learning

There probably isn't a hotter term this decade than 'machine learning'. But the principles of getting machines to be able to perform an operation based on an unknown input has been around for the best part of 100 years. 'Figure it out' is now an instruction you can tell a computer... and sure enough with some careful programming (and a truckload of data to conduct trial and error experiments on) - we can get pretty close.

I used these handy resources when writing this guide:

- The Caret Package - Max Kuhn
- Caret Package - Selva Prabhakaran
- Multivariate Adaptive Regression Splines

We'll start our ML journey by loading some packages.

```
if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
pacman::p_load(
  ggplot2, caret, skimr, RANN, randomForest, gbm, xgboost,
  caretEnsemble, C50, caTools
)
```

We know that a model is just any function (of one or more variables) that helps to explain observations. The most basic models to explain a data set are `mean`, `min`, and `max`. These functions don't rely on any variables outside of the variable of interest itself.

Moving to the slightly more abstract, we recall that a linear regression fits a straight line through a data set that minimizes the variance (i.e. the errors) between each point and the model line.

To fit a linear regression, we use the `lm()` function in the following format:
`linear_model <- lm(y ~ x, dataset)`

To make predictions using mod on the original data, we can call the predict() function:

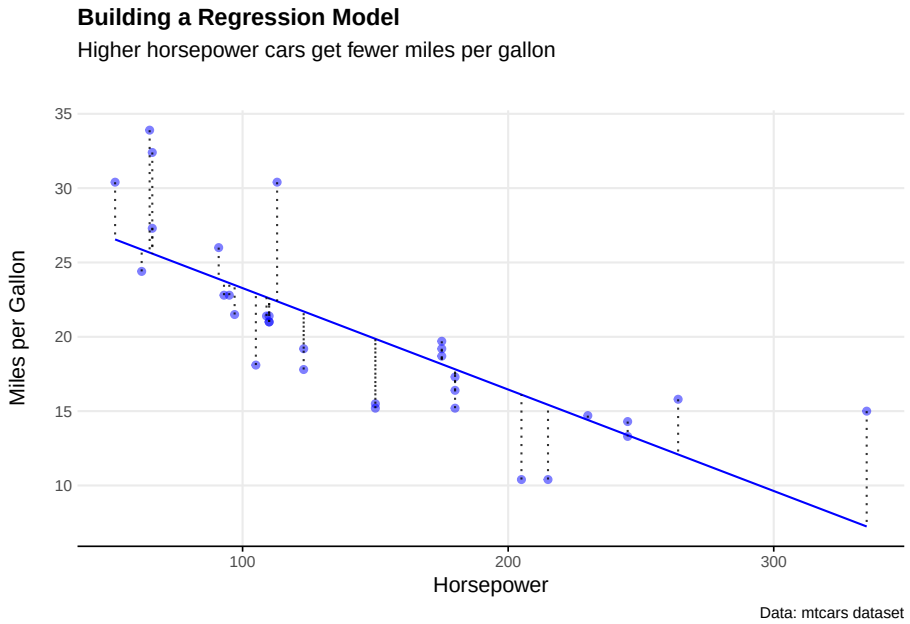
```
predict_data <- predict(linear_model, dataset)
```

```
# Let's use the mtcars dataset as an example

# Fit a model to the mtcars data
data(mtcars)

# Build a linear model to explain mpg using hp
mtcars_model <- lm(mpg ~ hp, mtcars)

# Run the model over the data
mtcars_model_estimates <- predict(mtcars_model,mtcars)

# Bind the original figures for mpg with the predicted figures for mpg
mtcars_model_outputs <- cbind(Actual_mpg=mtcars$mpg,
                              Actual_hp=mtcars$hp,
                              Predicted_mpg=mtcars_model_estimates,
                              Residuals=resid(mtcars_model),
                              Fitted=fitted(mtcars_model))

# Ensure the outputs are a df
mtcars_model_outputs <- as.data.frame(mtcars_model_outputs)
```

We can see how well this model estimated the dataset by looking at the residuals on a plot.

```
ggplot(mtcars_model_outputs) +
  geom_line(aes(x = Actual_hp, y = Predicted_mpg), color = "blue") +
  geom_point(aes(x = Actual_hp, y = Actual_mpg), color = "blue", alpha = 0.5) +
  geom_segment(aes(x = Actual_hp, y = Actual_mpg, xend = Actual_hp, yend = Fitted),
               color = "black", alpha = 0.8, linetype = "dotted") +
```

```
  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 = ggplot2::margin(b = 25)),
    plot.caption = element_text(size = 8),
    axis.text = element_text(size = 8),
    axis.title.y = element_text(margin = ggplot2::margin(r = 3)),
    axis.text.y = element_text(margin = ggplot2::margin(l = 10)),
    axis.line.x = element_line(color = "black", size = 0.4),
    axis.ticks.x = element_line(color = "black", size = 0.4),
    panel.grid.minor = element_blank()
  )
```



To quantify the model fit in a single number, we can use the Root Mean Square Error (RMSE).

```r
# Manually calculate RMSE
error <- mtcars_model_outputs$Actual_mpg - mtcars_model_outputs$Fitted
mtcars_rmse_manual <- sqrt(mean(error^2))

# Alternative: Using Residuals column directly
mtcars_rmse_residuals <- sqrt(mean(mtcars_model_outputs$Residuals^2))

# Print results
print(mtcars_rmse_manual)
print(mtcars_rmse_residuals)
```

So in very rough terms, we can say that the model predicts miles per gallon for a given level of horsepower with around 4 mpg of error.

Random forest

True 'machine learning' takes this process a step further by creating *many* models using *many* parameters to find the best explanatory variables. There's a myriad of ways to achieve this. One useful approach is Random Forest models. These algorithms combine decision trees (discussed below) to make predictions.

In this section, we will use the `Boston` dataset from the `MASS` package to predict house prices using machine learning models. The dataset contains information on housing prices in the Boston area, with various predictors such as crime rate, number of rooms, and property tax rates. The target variable `medv` represents the median value of owner-occupied homes in $1000s.

We will first split the dataset into training and testing sets, train a Random Forest model, evaluate its performance, and then compare it with a linear regression model using `tidymodels`.

```r
# Load necessary libraries
library(MASS)
library(caret)

# Load the Boston housing dataset
data("Boston")

# Splitting the dataset into training (80%) and testing (20%) sets
set.seed(123)
train_index <- createDataPartition(Boston$medv, p=0.8, list=FALSE)
train_data <- Boston[train_index, ]
test_data <- Boston[-train_index, ]
```

```r
# Display the first few rows of each dataset to confirm the split
head(train_data)
head(test_data)
```

We will now train a Random Forest model using the `randomForest` package. The algorithm builds multiple decision trees and averages their predictions to improve accuracy and reduce overfitting.

```r
# Train a Random Forest model with 100 trees
model <- randomForest::randomForest(medv ~ ., data=train_data, ntree=100)

# Make predictions on the test set
predictions <- predict(model, test_data)
```

We use Root Mean Square Error (RMSE) to measure the model's performance as we did in the previous section.

```r
# Calculate RMSE
rmse <- sqrt(mean((predictions - test_data$medv)^2))

# Print RMSE result
print(paste("RMSE:", round(rmse, 2)))
```

To compare performance of our `randomForest` model, we train a simple linear regression model using the `tidymodels` framework. Linear regression attempts to fit a straight line that best represents the relationship between predictors and the target variable.

```r
# Load necessary library
library(tidymodels)

# Set a seed for reproducibility
set.seed(123)

# Define a linear regression model
ml_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# Create a workflow that includes the model and the formula
tidymodels_workflow <- workflow() %>%
  add_model(ml_model) %>%
  add_formula(medv ~ .)
```

```
# Train the linear regression model
trained_model <- tidymodels_workflow %>% fit(data = train_data)
```

After training the model, we can use it to make predictions on the `test_data`. The `predict()` function returns predicted values, which are then combined with the actual test data for comparison.

The `metrics()` function calculates various performance metrics, such as RMSE, MAE, and R^2, to evaluate the model's predictive accuracy. The `truth` argument specifies the actual values (`medv`), and the `estimate` argument specifies the predicted values (`.pred`).

```
# Make predictions using the trained model
predictions <- predict(trained_model, test_data) %>% bind_cols(test_data)

# Evaluate model performance using metrics function
metrics(predictions, truth = medv, estimate = .pred)
```

As we see, our `randomForest` model performed much better, with a RMSE of 2.96 compared to 4.58.

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.

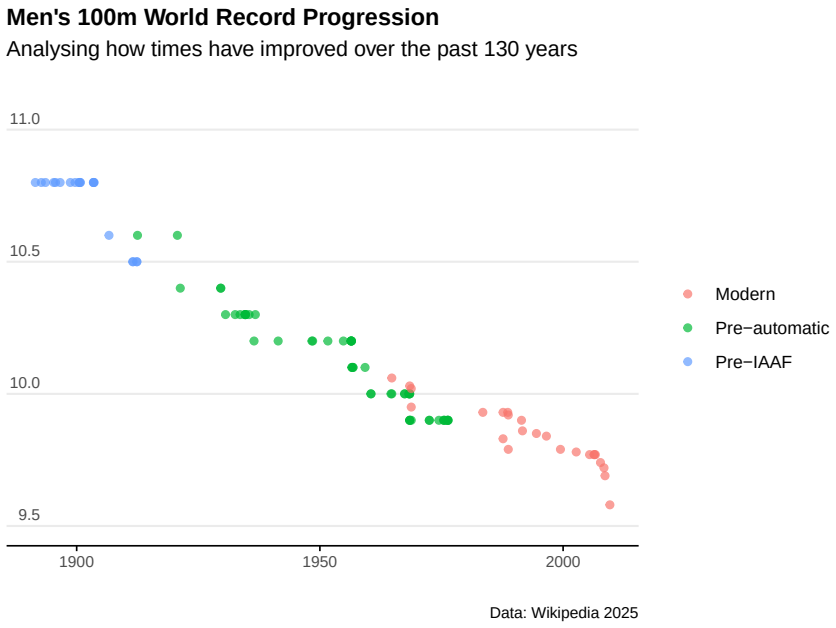
```
wr100 <- rbind(
    pre_iaaf %>% dplyr::select(time, athlete, nationality, date) %>%
    mutate(era = "Pre-IAAF"),
    iaaf_76 %>% dplyr::select(time, athlete, nationality, date) %>%
    mutate(era = "Pre-automatic"),
    iaaf %>% dplyr::select(time, athlete, nationality, date) %>%
    mutate(era = "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 = ggplot2::margin(l = 20, r = -20)),
```

```
    plot.subtitle = element_text(margin = ggplot2::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

Lats and longs

Geocoding converts addresses into coordinates (latitude and longitude) and vice versa. This helps turn basic address data into spatial points, making it useful for mapping and analyzing location-based patterns.

For example, geocoding can help define catchment areas around an asset or visualize spatial data on a map. To do this, we need access to a large database of addresses and coordinates. Google Maps is a popular choice, offering an API with a freemium model for access.

This guide by Oleksandr Titorchuk also provides a great overview of the geocoding process in R.

Spatial data

Let's get started by loading up some packages.

```
#Loads the required required packages
pacman::p_load(ggmap,
               tmaptools,
               RCurl,
               jsonlite,
               tidyverse,
               leaflet,
               writexl,
               readr,
               readxl,
               sf,
               mapview,
               rgdal)
```

```
##
##   There are binary versions available but the source versions are later:
##           binary source needs_compilation
## sf        1.0-15 1.0-19              TRUE
## stars      0.5-5  0.6-8             FALSE
## tmaptools  3.1-1    3.2             FALSE
```

Next, we'll introduce some address data.

Approved speed camera locations in Victoria, Australia are publicly available through the government website. We can download the dataset as a spreadsheet and import it into R as a csv.

```
#Read in spreadsheet
url <- 'https://raw.githubusercontent.com/charlescoverdale/bookdata/main/mobile-camera-

#Note: We need to import as a csv rather than xlsx for url functionality
camera_address <- read.csv(url, header=TRUE)

#Fix the column labels
camera_address <- janitor::row_to_names(camera_address,1)

#Convert the suburb column to title case
camera_address$SUBURB <- str_to_title(camera_address$SUBURB)

#Concatenate the two fields into a single address field
camera_address$ADDRESS <- paste(camera_address$LOCATION,
                                camera_address$SUBURB,
                                sep=", ")

#Add in Australia to the address field just to idiot proof the df
camera_address$ADDRESS <- paste(camera_address$ADDRESS, ", Australia", sep="")

#Preview the data
head(camera_address)
```

```
##           LOCATION            SUBURB Reason Code Audit Date
## 2        Abey Road        Cobblebank         BCD     May-22
## 3      Adam Street     Golden Square         ACD     Apr-22
## 4        Agar Road       Coronet Bay          BC          T
## 5    Airport Drive Melbourne Airport        ABCD     Apr-22
## 6 Aitken Boulevard     Roxburgh Park         ABC     Apr-22
## 7 Aitken Boulevard       Craigieburn        ABCD     Apr-22
##                                    ADDRESS
## 2            Abey Road, Cobblebank, Australia
## 3       Adam Street, Golden Square, Australia
```

```
## 4           Agar Road, Coronet Bay, Australia
## 5 Airport Drive, Melbourne Airport, Australia
## 6  Aitken Boulevard, Roxburgh Park, Australia
## 7    Aitken Boulevard, Craigieburn, Australia
```

This dataset does not have street numbers - only street names. This makes intuitive sense (as most speed cameras aren't placed outside residential houses, but rather along main roads).

We can see that the street and suburb names are in different columns. To run a geocode, we want a single field of address data. This makes it easier for the database we're fuzzy matching against to select the best possible point. Let's append the street name and the suburb name into a single field.

We know all these addresses are in Australia, so we can add the word 'Australia' to the end of each entry in the newly created address field to help the geocoder find the location.

It makes sense that all these addressed should have a postcode in the format 3XXX (as the data set is for the State of Victoria). However, let's hold off on that assumption for now.

Google API

Now that we have a useful and highly descriptive address field, we're ready to run the geocode.

We can geocode the lats and lons using Google's API through ther `ggmap` package.

- We must register a API key (by creating an account in the google developer suite).
- You can ping the API 2,500 times a day. Lucky for us this dataset is only 1,800 rows long!

Uncomment the chunk below (using Ctrl+Shift+C) and enter your unique key from google to run the code.

```
# #Input the google API key
# register_google(key = "PASTE YOUR UNIQUE KEY HERE")
#
# #Run the geocode function from ggmap package
# camera_ggmap <- ggmap::geocode(location = camera_address$ADDRESS,
#                            output = "more",
#                            source = "google")
```

```
#
# #We'll bind the newly created address columns to the original df
# camera_ggmap <- cbind(camera_address, camera_ggmap)
#
# #Print the results
# head(camera_ggmap)
#
# #Write the data to a df
# readr::write_csv(camera_ggmap,"C:/Data/camera_geocoded.csv")
```

We'll load up the output this chunk generates and continue.

```
url <- 'https://raw.githubusercontent.com/charlescoverdale/bookdata/main/camera_geocode

#Note: We need to import as a csv rather than xlsx for url functionality
camera_ggmap <- read.csv(url, header=TRUE)
```

Geocode analysis

The raw dataset our geocode produced looks good! Although... it could probably use some cleaning.

Let's rename this dataset so we don't loose the original df (and therefore have to run the google query again). This is simply a best practice step to build in some redundancy.

```
camera_geocoded <- camera_ggmap
```

```
#Rename df
camera_geocoded_clean <- camera_geocoded

#Rename the API generated address field
camera_geocoded_clean <- camera_geocoded_clean %>% rename(address_long=address)

#Select only the columns we need
camera_geocoded_clean <- camera_geocoded_clean %>%
                         dplyr::select("LOCATION",
                                "SUBURB",
                                "ADDRESS",
                                "lon",
                                "lat",
                                "address_long"
                                )
```

```
#Make all the column names the same format
camera_geocoded_clean <- janitor::clean_names(camera_geocoded_clean)

#We still need to convert address_long to title case
camera_geocoded_clean$address_long <- str_to_title(camera_geocoded_clean$address_long)
```

If we want to go one step further, we can create spatial points from this list of coordinates. This is a good step for eyeballing the data. We see most of it is in Victoria (as expected!)... but it has picked up a couple of points in Sydney and WA.

These are worth investigating separately for correction or exclusion.

Uncomment the mapview function below to see these points on a leaflet map.

```
#Convert data frame to sf object
camera_points <- sf::st_as_sf(x = camera_geocoded_clean,
                              coords = c("lon", "lat"),
                              crs = "+proj=longlat
                                     +datum=WGS84
                                     +ellps=WGS84
                                     +towgs84=0,0,0")

#Plot an interactive map
# mapview(camera_points)
```

We can export these points as a shapefile using the `rgdal` package.Let's now have a look at our edge case datapoints.

There's a couple of ways to do this... but one of the simplest is to extract the postcode as a separate field. We can then simply sort by postcodes that do not start with the number 3.

```
camera_points$postcode <- str_sub(camera_points$address_long,
                                  start= -16,
                                  end= -12)

camera_points$postcode <- as.numeric(camera_points$postcode)

outliers <- camera_points %>% filter (postcode <3000 | postcode >= 4000)

print(outliers)
```

```
## Simple feature collection with 7 features and 5 fields
## Geometry type: POINT
```

```
## Dimension:     XY
## Bounding box:  xmin: 115.9555 ymin: -33.90941 xmax: 152.7034 ymax: -25.53849
## Geodetic CRS:  +proj=longlat
##                                        +datum=WGS84
##                                        +ellps=WGS84
##                                        +towgs84=0,0,0
##                     location     suburb
## 1               Epping Road     Epping
## 2               High Street     Epping
## 3           Kensington Road Kensington
## 4          Maryborough Road      Ascot
## 5 Maryborough-Ballarat Road     Talbot
## 6  Mooroopna-Murchison Road  Murchison
## 7             Plumpton Road   Plumpton
##                                          address
## 1                 Epping Road, Epping, Australia
## 2                 High Street, Epping, Australia
## 3         Kensington Road, Kensington, Australia
## 4             Maryborough Road, Ascot, Australia
## 5   Maryborough-Ballarat Road, Talbot, Australia
## 6 Mooroopna-Murchison Road, Murchison, Australia
## 7             Plumpton Road, Plumpton, Australia
##                                      address_long                    geometry
## 1         Epping Rd, Epping Nsw 2121, Australia POINT (151.0906 -33.77076)
## 2           High St, Epping Nsw 2121, Australia POINT (151.0836 -33.77611)
## 3 Kensington Rd, Kensington Nsw 2033, Australia POINT (151.2211 -33.90941)
## 4               Maryborough Qld 4650, Australia POINT (152.7034 -25.53849)
## 5               Maryborough Qld 4650, Australia POINT (152.7034 -25.53849)
## 6                  Murchison Wa 6630, Australia POINT (115.9555 -26.89259)
## 7     Plumpton Rd, Plumpton Nsw 2761, Australia POINT (150.8439 -33.74823)
##   postcode
## 1     2121
## 2     2121
## 3     2033
## 4     4650
## 5     4650
## 6     6630
## 7     2761
```

Easy enough. We see there's 7 rows that don't have a postcode starting in a 3000 postcode.

Four of these are in NSW, two in QLD, and one in WA. It looks like they are real (e.g. the streets and suburbs do exist in that state)... so for now let's just exclude them from our dataset. We'll do this by filtering by postcode.

Uncomment the mapview function below to see these points on a leaflet map.

```
camera_points_vic <- camera_points %>% filter (postcode>"3000" & postcode<"4000")

head(camera_points_vic)
```

```
## Simple feature collection with 6 features and 5 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 144.2738 ymin: -38.43479 xmax: 145.4467 ymax: -36.77596
## Geodetic CRS:  +proj=longlat
##                                       +datum=WGS84
##                                       +ellps=WGS84
##                                       +towgs84=0,0,0
##           location            suburb
## 1        Abey Road        Cobblebank
## 2      Adam Street     Golden Square
## 3        Agar Road       Coronet Bay
## 4    Airport Drive Melbourne Airport
## 5 Aitken Boulevard     Roxburgh Park
## 6 Aitken Boulevard       Craigieburn
##                                       address
## 1            Abey Road, Cobblebank, Australia
## 2       Adam Street, Golden Square, Australia
## 3           Agar Road, Coronet Bay, Australia
## 4 Airport Drive, Melbourne Airport, Australia
## 5  Aitken Boulevard, Roxburgh Park, Australia
## 6    Aitken Boulevard, Craigieburn, Australia
##                                         address_long                    geometry
## 1           Abey Rd, Cobblebank Vic 3338, Australia POINT (144.5879 -37.70471)
## 2        Adam St, Golden Square Vic 3555, Australia POINT (144.2738 -36.77596)
## 3          Agar Rd, Coronet Bay Vic 3984, Australia POINT (145.4467 -38.43479)
## 4 Airport Dr, Melbourne Airport Vic 3045, Australia POINT (144.8661 -37.69291)
## 5    Aitken Blvd, Roxburgh Park Vic 3064, Australia  POINT (144.915 -37.62426)
## 6      Aitken Blvd, Craigieburn Vic 3064, Australia   POINT (144.91 -37.59446)
##   postcode
## 1     3338
## 2     3555
## 3     3984
## 4     3045
## 5     3064
## 6     3064
```

```
#Plot an interactive map
# mapview(camera_points_vic)
```

There we go! A clean, geocoded dataset of speed camera locations in Victoria.

Drivetime analysis

A 'drive time' describes how far you can drive (i.e in a car on a public road) in a certain amount of time.

- Running a drive time analysis is useful to identify demographic catchments around a point (e.g. a school, hospital, road, or precinct.
- This can assist in defining the 'catchment' of users for a particular infrastructure asset.
- These polygons can then be overlayed with ABS census data (e.g. SA2) and spliced in with census variables (age, income, housing, SES status, etc).
- There's various companies that own drive time data. Most of these are map makers (e.g. google, ESRI, and Tom Tom).

To get started, let's install and load packages.

```
#Loads the required required packages
pacman::p_load(ggmap,
               tmaptools,
               RCurl,
               jsonlite,
               tidyverse,
               leaflet,
               writexl,
               readr,
               readxl,
               sf,
               mapview,
               rgdal,
               osrm)
```

```
##
```

```
##   There are binary versions available but the source versions are later:
##           binary source needs_compilation
## sf        1.0-15 1.0-19              TRUE
## stars      0.5-5  0.6-8             FALSE
## tmaptools  3.1-1    3.2             FALSE
```

OSRM isochrones

Useful links for further reading: Source 1, Source 2

The OSRM package (Github) pulls from OpenStreetMap to find travel times based on location.

The downside is that the polygons it generates are pretty chunky... i.e. it doesn't take into account major roads and streets as the key tributaries/arteries of a city area. We can get around this a bit by dialing up the 'res' (i.e. the resolution) in the `osrmIsochrone` function... but it's only a partial solution.

Uncomment the leaflet function below to view an interactive map.

```
# Create a dataframe with the latitude and longitude
locations <- tibble::tribble(
  ~place,     ~lon,     ~lat,
  "Melbourne", 144.9631, -37.8136
)

# Run it through the osrm package to calculate isochrones
iso <- osrmIsochrone(
  loc = c(locations$lon, locations$lat),
  breaks = seq(from = 0, to = 30, by = 5),
  res = 50
)

# Create an interactive map
# leaflet() %>%
#   setView(mean(locations$lon), mean(locations$lat), zoom = 7) %>%
#   addProviderTiles("CartoDB.Positron", group = "Greyscale") %>%
#   addMarkers(lng = locations$lon, lat = locations$lat) %>%
#   addPolygons(
#     fill = TRUE, stroke = TRUE, color = "black",
#     weight = 0.5, fillOpacity = 0.2,
#     data = iso,
#     group = "Drive Time"
#   )
```

Raster data

Raster data (sometimes referred to as gridded data) is a type of spatial data that is stored in a grid rather than a polygon. Imagine a chessboard of individuals squares covering the Australian landmass compared to 8 different shapes covering each state and territory.

The nice thing about gridded data is that all the cells in the grid are the same size - making calculations much easier. The tricky part is those squares are overlayed on a (very roughly) spherical earth. We'll have to think about mapping projections along the way.

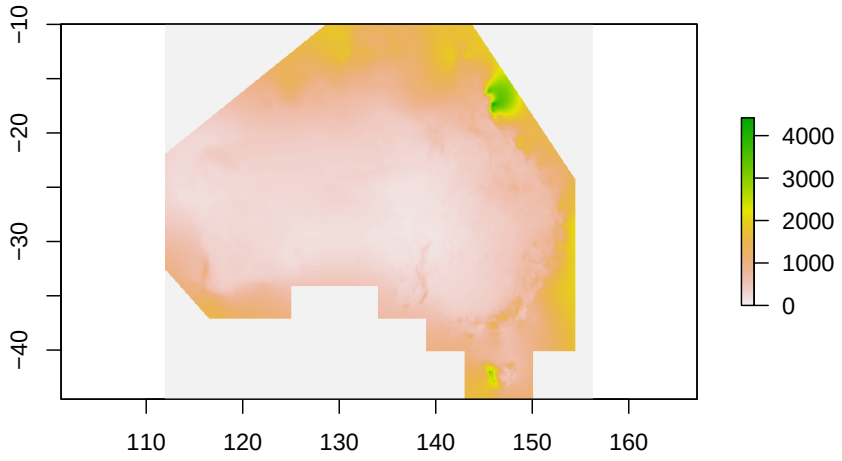
Let's get started. First up we need to load some spatial and data crunching packages.

```
# Core tidyverse packages (includes ggplot2, dplyr, tidyr, broom, etc.)
library(tidyverse)

# Spatial data handling
library(sf)          # Modern replacement for sp + rgdal
library(raster)      # Working with raster data
library(ncdf4)       # NetCDF file handling
library(leaflet)     # Interactive maps

# Mapping & Visualization
library(ggspatial)  # Adds scale bars, north arrows, etc.
#library(tmap)       # Thematic mapping
library(ggmap)       # Google Maps integration
library(rasterVis)  # Raster visualization
library(viridis)     # Color scales for ggplot2 and rasterVis
library(RColorBrewer) # Color palettes

# Other Utilities
library(absmapsdata) # Australian Bureau of Statistics map data
```

Import raster

Next, we want to import annual rainfall data from github (original source available from the Bureau of Meterology)

```
rainfall <- raster("https://raw.github.com/charlescoverdale/ERF/master/rainan.txt")

plot(rainfall)
```



Straight away we see this is for the whole of Australia (and then some). We're only interested in what's going on in QLD... so let's crop the data down to scale. For this we'll need to import a shapefile for QLD.

The easiest way to do this is using the absmapsdata package - importing a shapefile of Australia then filtering for only Queensland.

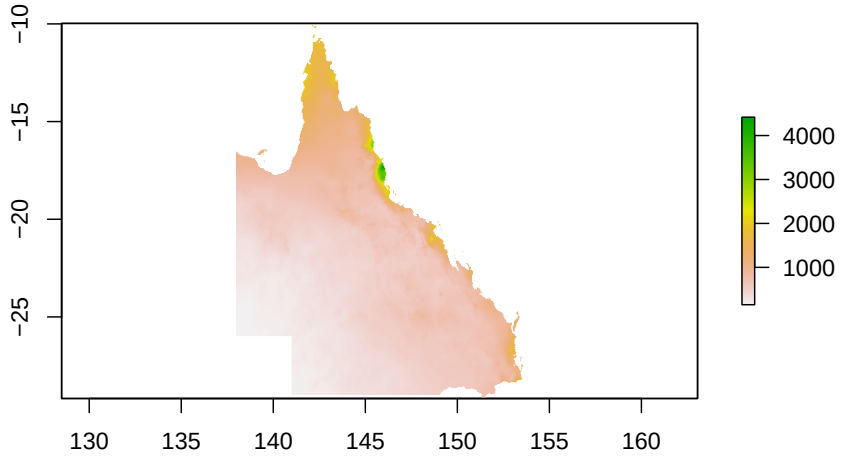
```
# Import a polygon for the state of Queensland
QLD_shape <- absmapsdata::state2016 %>%
             filter(state_name_2016=="Queensland")
```

Choosing geographies

We have raster data for the entirety of Australia (and then some as it's pulled from one of the BOMs satellites). This is a bit messy to work with - so let's

crop the rainfall data from the entire Australian continent to just Queensland.

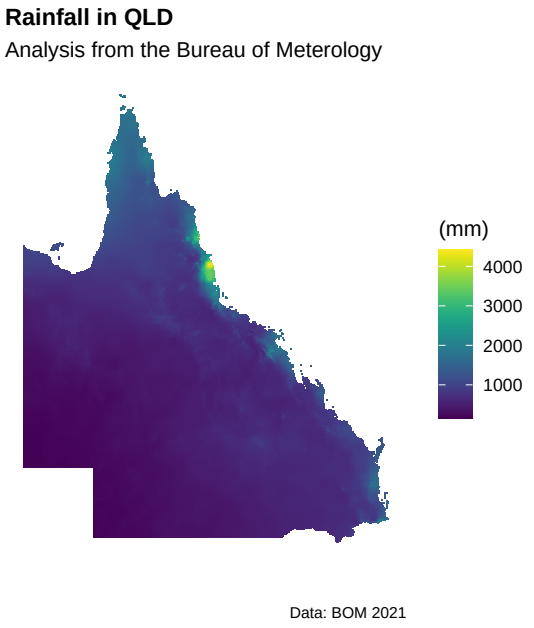
```
#Crop data
r2 <- crop(rainfall,extent(QLD_shape))
r3 <- mask(r2,QLD_shape)
plot(r3)
```



]Next up, let's transform the cropped raster into a data frame that we can use in the ggplot package.

```
r3_df <- as.data.frame(r3,xy=TRUE)

r3_df <- r3_df %>%
         filter(rainan!="NA")

ggplot() +
  geom_tile(data=r3_df, aes(x=x, y=y, fill=rainan)) +
  scale_fill_viridis() +
  coord_equal() +
  theme(legend.position="bottom") +
  theme(legend.key.width=unit(1.2, "cm"))+
  labs(title="Rainfall in QLD",
               subtitle = "Analysis from the Bureau of Meterology",
               caption = "Data: BOM 2021",
               x="",
```

```
              y="",
              fill="(mm)") +
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 = "bottom")+
          theme(plot.title=element_text(face="bold",size=12))+
          theme(plot.subtitle=element_text(size=11))+
          theme(plot.caption=element_text(size=8))
```

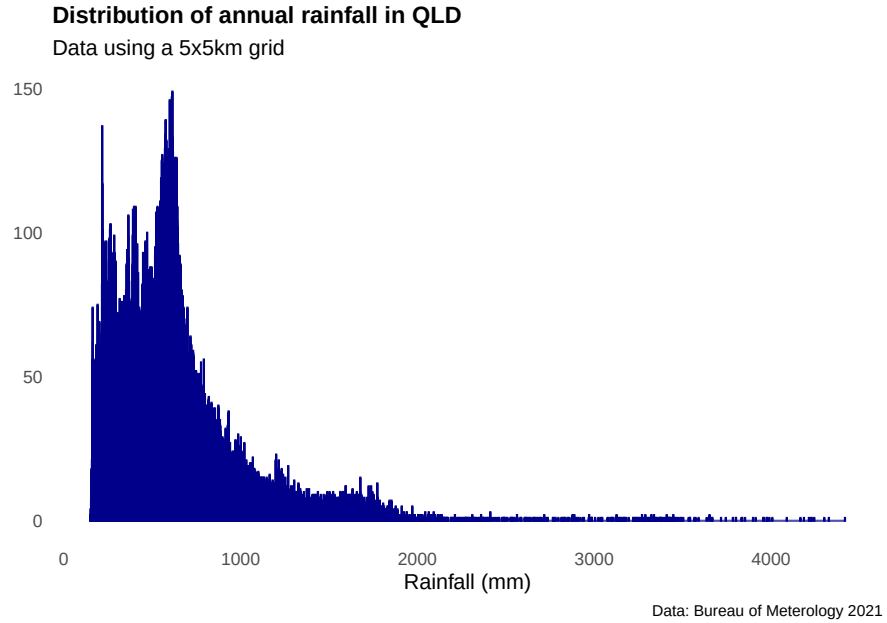


Excellent, we've got a working map of rainfall in Queensland using the ggplot package. We'll tidy up the map also and add a title and some better colours.

Analysis with raster data

For our first piece of data analysis, we're going to look at areas with less than 600mm of annual rainfall. How many of our data points will have less than 600mm of rain? Let's take a look at the data distribution and find out.

```
ggplot(r3_df) +
    geom_histogram(aes(rainan),binwidth=1,col="darkblue")+
    labs(title="Distribution of annual rainfall in QLD",
                subtitle = "Data using a 5x5km grid",
                caption = "Data: Bureau of Meterology 2021",
                x="Rainfall (mm)",
                y="",
                fill="(mm)") +
  theme_minimal() +
            theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())+
            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))
```

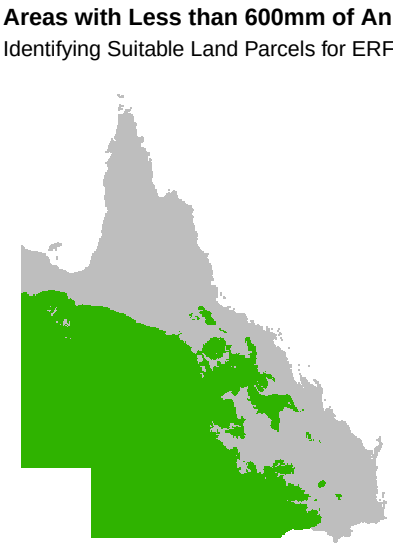


Interesting. The data is heavily right tailed skewed (the mean will be much higher than the median)... and most of the data looks to be between 0-1000mm (this makes sense).

Let's create a 'flag' column of 0's and 1's that shows when a data point is less than 600mm.

```
# Define flag column and colors
r3_df <- r3_df %>%
  mutate(flag_600mm = as.factor(ifelse(rainan <= 600, 1, 0)))
```

```
flag_colours <- c("grey", "#2FB300")

# Plot
ggplot(r3_df, aes(x = x, y = y, fill = flag_600mm)) +
  geom_tile() +
  scale_fill_manual(values = flag_colours) +
  coord_equal() +
  theme_minimal() +
  labs(
    title = "Areas with Less than 600mm of Annual Rainfall in QLD",
    subtitle = "Identifying Suitable Land Parcels for ERF Plantings",
    caption = "Data: Bureau of Meteorology 2021",
    x = "",
    y = "",
    fill = "(mm)"
  ) +
  theme(
    axis.ticks = element_blank(),
    axis.text = element_blank(),
    panel.grid = element_blank(),
    legend.position = "none",
    plot.title = element_text(face = "bold", size = 12),
    plot.subtitle = element_text(size = 11),
    plot.caption = element_text(size = 8)
  )
```



Making an interactive map

The static map above is good… but an interactive visual (ideally with place names below) is better. The `leaflet` package is exceptional here.

```
# This code creates an interactive leaflet map. It is commented out to allow the html version of

# leaflet() %>%
#   addTiles() %>%
#   addRasterImage(r5, opacity = 0.5) %>%
#   addLegend(pal = colorNumeric("viridis", values(r5)), values = values(r5))
```

Australian economic data

Australia has exceptional financial and economic institutions. Three of these institutions release periodic data useful for economic analysis:

- Australian Bureau of Statistics
- Reserve Bank of Australia
- Treasury

As usual, there are catches. Most of this data is in inconsistent formats (the reasons for which continue to baffle me). What's more, it's currently not possible to ping databases or API's for access to this data... it is mainly accessed through spreadsheets.

The scripts below run through some of the main ways to import, clean, and analyse Australian macroeconomic data in R.

Some of the key packages we'll use are readabs and readrba.

To get started, let's install and load packages.

```
# Loads the required packages
pacman::p_load(
  ggmap, tmaptools, RCurl, jsonlite, tidyverse,
  leaflet, writexl, readr, readxl, readabs, readrba, lubridate,
  zoo, scales
)
```

```
##
##   There are binary versions available but the source versions are later:
##           binary source needs_compilation
## sf        1.0-15 1.0-19              TRUE
## stars      0.5-5  0.6-8             FALSE
## tmaptools  3.1-1    3.2             FALSE
```

GDP

To get GDP data from the ABS, we'll use the read_abs function from the `readrba` package.

```
#For simplicity, we keep the download function seperate to the analysis
all_gdp <- readabs::read_abs("5206.0", tables=2)
```

```
#Select the seasonally adjusted data and filter for data and value columns
gdp_level <- all_gdp %>%
              filter(series == "Gross domestic product: Chain volume measures ;",
                     !is.na(value)) %>%
              filter(series_type =="Seasonally Adjusted") %>%
              dplyr::select(date,value) %>%
              dplyr::rename(quarterly_output=value)

gdp_level <- gdp_level %>%
             mutate(quarterly_growth_rate =
                    ((quarterly_output / lag(quarterly_output,1)-1))*100) %>%
             mutate(annual_gdp =
                    rollapply(quarterly_output,
                              4,
                              sum,
                              na.rm=TRUE,
                              fill = NA,
                              align = "right")) %>%
             mutate(annual_gdp_trillions=annual_gdp/1000000)%>%
             mutate(annual_growth_rate =
                   ((annual_gdp / lag(annual_gdp, 4) - 1))*100)%>%
             mutate(Quarter_of_year =
                    lubridate::quarter(date,
                                       with_year = FALSE,
                                       fiscal_start = 1))

#Set a baseline value
gdp_level$baseline_value <- gdp_level$quarterly_output[
                            which(gdp_level$date =="2022-03-01")]


gdp_level <- gdp_level %>%
             mutate(baseline_comparison =
                   (quarterly_output/baseline_value)*100)

tail(gdp_level)
```

```
## # A tibble: 0 x 9
## # i 9 variables: date <date>, quarterly_output <dbl>,
## #   quarterly_growth_rate <dbl>, annual_gdp <dbl>, annual_gdp_trillions <dbl>,
## #   annual_growth_rate <dbl>, Quarter_of_year <int>, baseline_value <dbl>,
## #   baseline_comparison <dbl>
```

Now we can plot the GDP data for Australia.

```
plot_gdp <- ggplot(data=gdp_level)+
            geom_line((aes(x=date, y=annual_gdp_trillions)), col="blue") +

            labs(title = "Australian GDP ($AUD)",
                 subtitle = "Annualised figures",
                 caption = "Data: Australian Bureau of Statistics",
                 y = "",
                 x = " ")+

            scale_y_continuous(breaks = c(0,0.5,1.0,1.5,2.0,2.5),
                               labels = label_number(suffix = " trillion"))+

            scale_x_date(date_breaks = "10 years", date_labels="%Y")+

            theme_minimal() +
            theme(legend.position="bottom")+

            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=8))+
            theme(panel.grid.minor = element_blank())+
            theme(panel.grid.major.x = element_blank()) +

            theme(axis.title.y =
            element_text(margin = ggplot2::margin(t = 0, r = 0, b = 0, l = 0)))+

            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = ggplot2::margin(l = 20, r = -45)))+

            theme(axis.line.x = element_line(colour ="black", size=0.4))+
            theme(axis.ticks.x = element_line(colour ="black", size=0.4))


plot_gdp
```

Australian GDP ($AUD)
Annualised figures

Data: Australian Bureau of Statistics

Unemployment rate

Download the data

```
#Download the time sreies
all_unemployment <- readabs::read_abs("6202.0", tables=1)
```

Clean and analyse the data

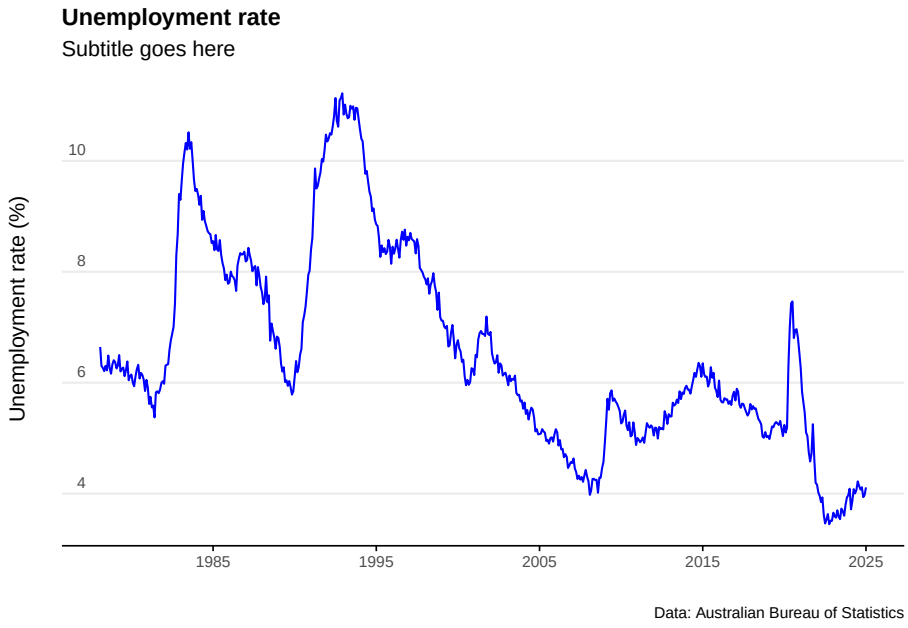
```
unemployment_rate <- all_unemployment %>%
              filter(series == "Unemployment rate ;  Persons ;",!is.na(value)) %>%
              filter(table_title=="Table 1. Labour force status by Sex, Australia - Tr
              filter(series_type =="Seasonally Adjusted") %>%
              mutate(mean_unemployment_rate=mean(value)) %>%
              mutate(percentile_25=quantile(value,0.25))%>%
              mutate(percentile_75=quantile(value,0.75)) %>%
              dplyr::select(date,value,mean_unemployment_rate,percentile_25,percentile


tail(unemployment_rate)
```

```
## # A tibble: 6 x 5
```

```
##   date       value mean_unemployment_rate percentile_25 percentile_75
##   <date>     <dbl>                  <dbl>         <dbl>         <dbl>
## 1 2024-08-01  4.14                   6.57          5.27          7.92
## 2 2024-09-01  4.07                   6.57          5.27          7.92
## 3 2024-10-01  4.11                   6.57          5.27          7.92
## 4 2024-11-01  3.94                   6.57          5.27          7.92
## 5 2024-12-01  3.98                   6.57          5.27          7.92
## 6 2025-01-01  4.11                   6.57          5.27          7.92
```

Plot the data

```
plot_unemployment_rate <- ggplot(data=unemployment_rate)+
                          geom_line(aes(x = date, y = value), col = "blue") +

            labs(title = "Unemployment rate",
                 subtitle = "Subtitle goes here",
                 caption = "Data: Australian Bureau of Statistics",
                 y = "Unemployment rate (%)",
                 x = " ")+

            scale_y_continuous(labels = scales::comma)+
            scale_x_date(date_breaks = "10 years", date_labels="%Y")+

            theme_minimal() +
            theme(legend.position="bottom")+

            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=8))+
            theme(panel.grid.minor = element_blank())+
            theme(panel.grid.major.x = element_blank()) +

            theme(axis.title.y =
            element_text(margin = ggplot2::margin(t = 0, r = 0, b = 0, l = 0)))+

            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = ggplot2::margin(l = 20, r = -15)))+

            theme(axis.line.x = element_line(colour ="black", size=0.4))+
            theme(axis.ticks.x = element_line(colour ="black", size=0.4))

plot_unemployment_rate
```



Inflation (CPI)

Download the data

```
all_CPI <- readabs::read_abs("6401.0", tables = c(1, 2))
```

Clean and analyse the data

```
Australia_CPI <- all_CPI %>%
                 filter(series == "Percentage Change from Corresponding Quarter of Prev
                 mutate(mean_CPI=mean(value)) %>%
                 mutate(percentile_25=quantile(value,0.25))%>%
                 mutate(percentile_75=quantile(value,0.75)) %>%
                 dplyr::select(date, value,mean_CPI,percentile_25,percentile_75)

tail(Australia_CPI)
```

```
## # A tibble: 6 x 5
##   date       value mean_CPI percentile_25 percentile_75
##   <date>     <dbl>    <dbl>         <dbl>         <dbl>
## 1 2023-09-01   5.4     4.95          1.92          7.28
## 2 2023-12-01   4.1     4.95          1.92          7.28
```

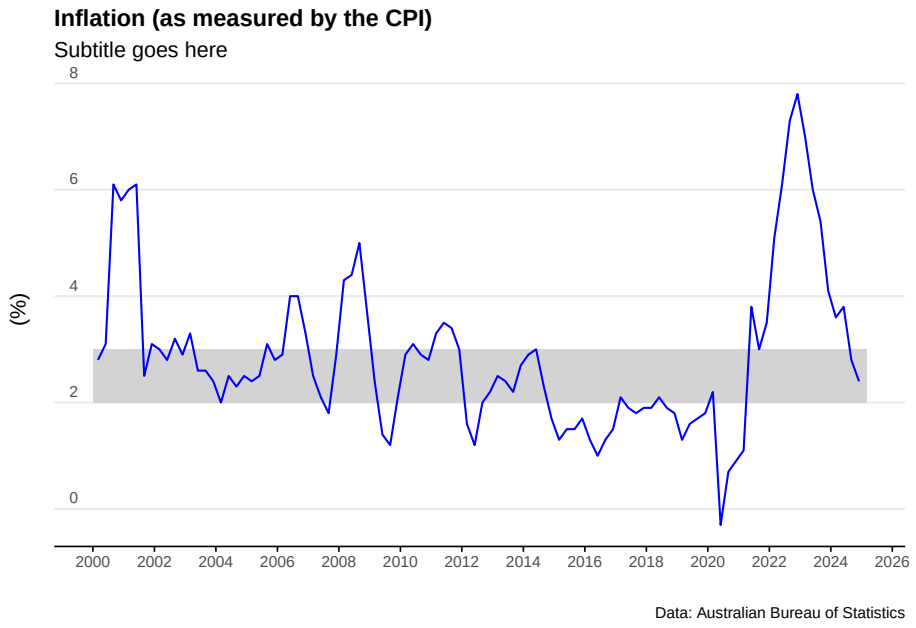
```
## 3 2024-03-01   3.6     4.95          1.92          7.28
## 4 2024-06-01   3.8     4.95          1.92          7.28
## 5 2024-09-01   2.8     4.95          1.92          7.28
## 6 2024-12-01   2.4     4.95          1.92          7.28
```

```
#Can add in the below line to filter
#filter(date>"2010-01-01") %>%
```

Plot the data

```
plot_CPI <- ggplot(data = Australia_CPI %>%
                   filter(date > as.Date("2000-01-01"))) +
  geom_rect(aes(xmin = as.Date("2000-01-01"),
                xmax = as.Date("2025-03-01"),
                ymin = 2,
                ymax = 3),
            alpha = 0.1,  # Adjusted alpha for better visibility
            fill = "lightgrey") +
  geom_line(aes(x = date, y = value), col = "blue") +

  scale_x_date(date_breaks = "2 years", date_labels = "%Y") +

  labs(title = "Inflation (as measured by the CPI)",
       subtitle = "Subtitle goes here",
       caption = "Data: Australian Bureau of Statistics",
       y = "(%)",
       x = "") +

  scale_y_continuous(labels = scales::comma) +

  theme_minimal() +
  theme(legend.position = "bottom") +

  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 = 8)) +
  theme(panel.grid.minor = element_blank()) +
  theme(panel.grid.major.x = element_blank()) +

  theme(axis.title.y = element_text(margin = ggplot2::margin(t = 0, r = 0, b = 0, l = 0))) +
  theme(axis.text.y = element_text(vjust = -0.5,
                                   margin = ggplot2::margin(l = 20, r = -15))) +
```

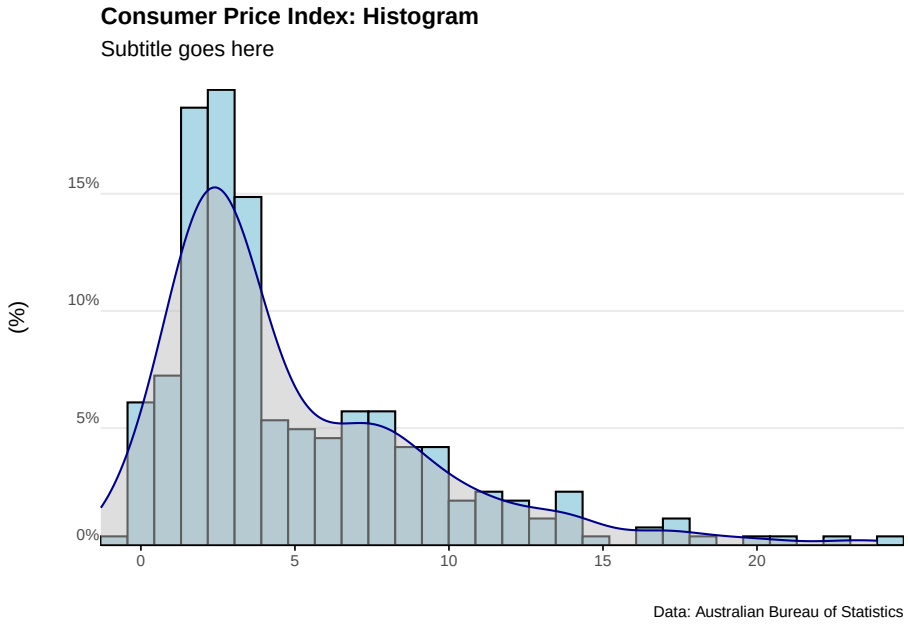
```
  theme(axis.line.x = element_line(colour = "black", size = 0.4)) +
  theme(axis.ticks.x = element_line(colour = "black", size = 0.4))

plot_CPI
```



Plot a histogram of the data

```
plot_CPI_hist <- ggplot(Australia_CPI, aes(x = value)) +
  geom_histogram(aes(y = ..density..),
                 colour = "black", fill = "lightblue") +
  geom_density(alpha = .5, fill = "grey", colour = "darkblue") +

  scale_x_continuous(expand = c(0, 0)) +  # Remove extra space on the x-axis

  labs(title = "Consumer Price Index: Histogram",
       subtitle = "Subtitle goes here",
       caption = "Data: Australian Bureau of Statistics",
       y = "(%)",
       x = "") +

  scale_y_continuous(labels = scales::percent, expand = c(0, 0)) +  # Ensure no space

  theme_minimal() +
  theme(
```

```
    legend.position = "bottom",
    plot.title = element_text(face = "bold", size = 12),
    plot.subtitle = element_text(size = 11, margin = ggplot2::margin(b = 15)),
    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 = ggplot2::margin(t = 0, r = 0, b = 20, l = 0)),
    axis.text.y = element_text(vjust = -0.5, margin = ggplot2::margin(l = 20, r = -2)),
    axis.line.x = element_line(colour = "black", size = 0.4),
    axis.ticks.x = element_line(colour = "black", size = 0.4)
  )

plot_CPI_hist
```



Wage Price Index

Download the data

```
all_wpi <- readabs::read_abs("6345.0")
```

Clean and analyse the data

```
Australia_WPI <- all_wpi %>%
                 filter(series == "Percentage Change From Corresponding Quarter of Prev
                                  !is.na(value)) %>%
                 filter(series_type=="Seasonally Adjusted") %>%
                 mutate(mean_WPI=mean(value)) %>%
                 dplyr::select(date, value,mean_WPI)

tail(Australia_WPI)
```

```
## # A tibble: 6 x 3
##   date       value mean_WPI
##   <date>     <dbl>    <dbl>
## 1 2023-09-01   4       3.12
## 2 2023-12-01   4.2     3.12
## 3 2024-03-01   4       3.12
## 4 2024-06-01   4.1     3.12
## 5 2024-09-01   3.6     3.12
## 6 2024-12-01   3.2     3.12
```

Plot the data

```
plot_WPI <- ggplot(data=Australia_WPI)+
                    geom_line(aes(x = date, y = value), col = "blue") +

             labs(title = "Wage Price Index",
                           subtitle = "Subtitle goes here",
                           caption = "Data: Australian Bureau of Statistics",
                           y = "(%)",
                           x = " ")+

             scale_y_continuous(labels = scales::comma)+
             scale_x_date(date_breaks = "5 years", date_labels="%Y")+

             theme_minimal() +
             theme(legend.position="bottom")+

             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=8))+
             theme(panel.grid.minor = element_blank())+
             theme(panel.grid.major.x = element_blank()) +
```

```
            theme(axis.title.y =
            element_text(margin = ggplot2::margin(t = 0, r = 0, b = 0, l = 0)))+

            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = ggplot2::margin(l = 20, r = -15)))+

            theme(axis.line.x = element_line(colour ="black", size=0.4))+
            theme(axis.ticks.x = element_line(colour ="black", size=0.4))


plot_WPI
```

AUD exchange rate

God knows why - but there are super quirky names for the official exchange rate tables

Download the data

```
exchange_rate_all<- readrba::read_rba(table_no = (c("ex_daily_8386",
                                                    "ex_daily_8790",
                                                    "ex_daily_9194",
                                                    "ex_daily_9598",
```

```
                                                    "ex_daily_9902",
                                                    "ex_daily_0306",
                                                    "ex_daily_0709",
                                                    "ex_daily_1013",
                                                    "ex_daily_1417",
                                                    "ex_daily_18cur")),
                                        cur_hist = "historical")
```

Clean and analyse the data

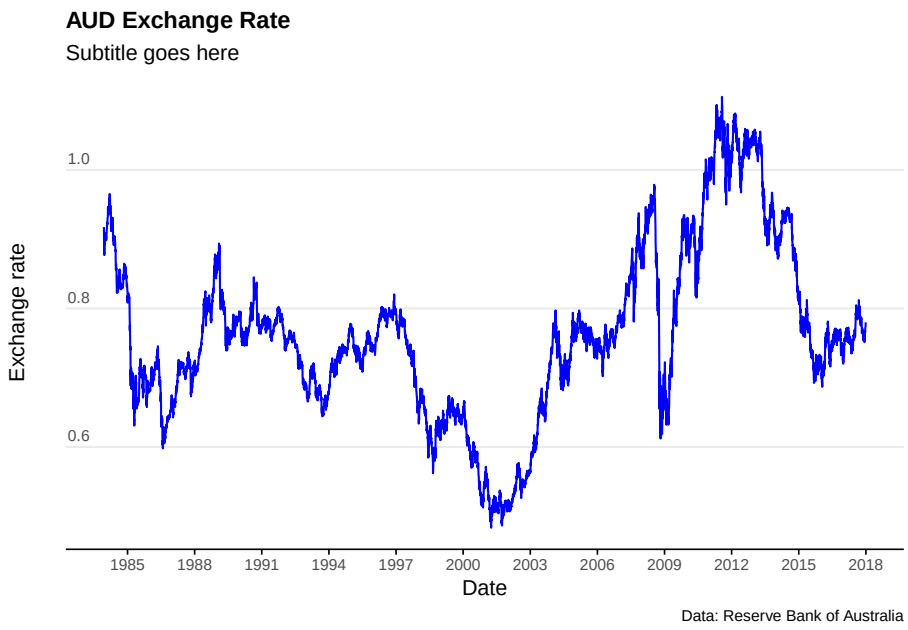
```
exchange_rate_AUD <-   exchange_rate_all %>%
                       filter(series=="A$1=USD") %>%
                       dplyr::select(date, value)


tail(exchange_rate_AUD)
```

```
## # A tibble: 6 x 2
##   date       value
##   <date>     <dbl>
## 1 2017-12-20 0.766
## 2 2017-12-21 0.767
## 3 2017-12-22 0.771
## 4 2017-12-27 0.774
## 5 2017-12-28 0.779
## 6 2017-12-29 0.78
```

Plot the data

```
plot_exchange_rate_AUD <- ggplot(data = exchange_rate_AUD) +
  geom_line(aes(x = date, y = value), col = "blue") +

  labs(
    title = "AUD Exchange Rate",
    subtitle = "Subtitle goes here",
    caption = "Data: Reserve Bank of Australia",
    y = "Exchange rate",  # Adding a Y-axis label
    x = "Date"
  ) +

  scale_y_continuous(labels = scales::comma) +
  scale_x_date(date_breaks = "3 years", date_labels = "%Y") +

  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(margin = ggplot2::margin(t = 0, r = 0, b = 0, l = 0)),
    axis.text.y = element_text(vjust = -0.5, margin = ggplot2::margin(l = 20, r = -15)),
    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"
  )

plot_exchange_rate_AUD
```



Australian election data

Elections tend to create fascinating data sets. They are spatial in nature, comparable over time (i.e. the number of electorates roughly stays the same) - and more importantly they are **consequential** for all Australians.

Australia's compulsory voting system is a remarkable feature of our Federation. Every three-ish years we all turn out at over 7,000 polling booths our local schools, churches, and community centres to cast a ballot and pick up an obligatory election day sausage. The byproduct is a fascinating longitudinal and spatial data set.

The following code explores different R packages, election data sets, and statistical processes aimed at exploring and modelling federal elections in Australia.

One word of warning: I use the term electorates, divisions, and seats interchangeably throughout this chapter.

Let's start by loading up some packages

```
library(ggparliament)  # If working with parliamentary data
library(tidyverse)      # Includes dplyr, ggplot2, tidyr, purrr, and more
library(ggthemes)
library(readxl)         # Reading Excel files
library(sf)             # Spatial data handling
library(DT)
library(eechidna)
library(absmapsdata)
```

Some phenomenal Australian economists and statisticians have put together a handy election package called eechidna. It includes three main data sets for the most recent Australia federal election (2022).

- **fp22:** first preference votes for candidates at each electorate
- **tpp22:** two party preferred votes for candidates at each electorate
- **tcp22:** two candidate preferred votes for candidates at each electorate

They've also gone to the trouble of aggregating some census data to the electorate level. This can be found with the **abs2022** function.

```
data(fp22)
data(tpp22)
data(tcp22)
data(abs2022)

# Show the first few rows
head(tpp22)
```

```
## # A tibble: 6 x 9
##   DivisionNm DivisionID StateAb LNP_Votes LNP_Percent ALP_Votes ALP_Percent
##   <chr>           <dbl> <chr>       <dbl>       <dbl>     <dbl>       <dbl>
## 1 CANBERRA          101 ACT         25424        27.5     66898        72.5
## 2 FENNER            102 ACT         31315        34.3     59966        65.7
## 3 BANKS             103 NSW         48969        53.2     43076        46.8
## 4 BARTON            104 NSW         31569        34.5     60054        65.5
## 5 BENNELONG         105 NSW         48847        49.0     50801        51.0
## 6 BEROWRA           106 NSW         55771        59.8     37535        40.2
## # i 2 more variables: TotalVotes <dbl>, Swing <dbl>
```

```
head(tcp22)
```

```
## # A tibble: 6 x 13
##   StateAb DivisionID DivisionNm BallotPosition CandidateID Surname GivenNm
##   <chr>        <dbl> <chr>               <dbl>       <dbl> <chr>   <chr>
## 1 ACT            318 BEAN                    3       36231 SMITH   DAVID
## 2 ACT            318 BEAN                    6       37198 HIATT   JANE
## 3 ACT            101 CANBERRA                5       36241 HOLLO   TIM
## 4 ACT            101 CANBERRA                6       36228 PAYNE   ALICIA
## 5 ACT            102 FENNER                  1       36234 LEIGH   ANDREW
## 6 ACT            102 FENNER                  2       37203 KUSTER  NATHAN
## # i 6 more variables: PartyAb <chr>, PartyNm <chr>, Elected <chr>,
## #   HistoricElected <chr>, OrdinaryVotes <dbl>, Percent <dbl>
```

Election maps

As noted in the introduction, elections are ***spatial*** in nature.

Not only does geography largely determine policy decisions, we see that many electorates vote for the same party (or even the same candidate) for decades. How electorate boundaries are drawn is a long story (see here, here, and here).

The summary version is the AEC carves up the population by state and territory, uses a wacky formula to decide how many seats each state and territory should be allocated, then draws maps to try and get a *roughly* equal number of people in each electorate.

Oh... and did I mention for reasons that aren't worth explaining that Tasmania has to have at least 5 seats? Our Federation is a funny thing. Anyhow, at time of writing this is how the breakdown of seats looks.

State/Territory	**Number of members of the House of Representatives (2022)**
New South Wales	47
Victoria	39
Queensland	30
Western Australia	15
South Australia	10
Tasmania	5
Australian Capital Territory	3
Northern Territory	2
TOTAL	151

Note: The NT doesn't have the population to justify it's second seat . The AEC scheduled to dissolve it after the 2019 election but Parliament intervened in late 2020 and a bill was passed to make sure both seats were kept (creating 151 nationally).

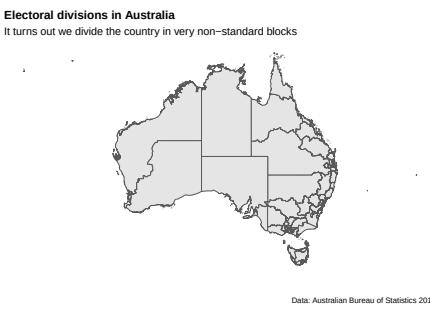
How variant are these 151 electorates in size? Massive.

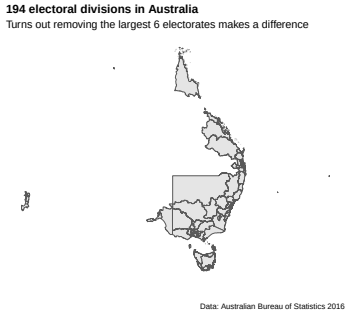
Durack in Western Australia (1.63 million square kilometres) is by far the largest and the smallest is Grayndler in New South Wales (32 square kilometres).

Let's make a map to make things easier.

```
CED_map <- absmapsdata::ced2021 %>%
           ggplot()+
           geom_sf()+
           labs(title="Electoral divisions in Australia",
               subtitle = "It turns out we divide the country in very non-standard blocks",
               caption = "Data: Australian Bureau of Statistics 2016",
               x="",
               y="") +
           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))


CED_map_remove_6 <- ced2021 %>%
                    dplyr::filter(!ced_code_2021 %in% c(506,701,404,511,321,317)) %>%
                    ggplot()+
          geom_sf()+
          labs(title="194 electoral divisions in Australia",
              subtitle = "Turns out removing the largest 6 electorates makes a differe
              caption = "Data: Australian Bureau of Statistics 2016",
              x="",
              y="") +
          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))

CED_map
CED_map_remove_6
```





Next let's look at what party/candidate is currently the sitting member for each electorate. To do this on a map we're going to need to join our tcp22 data and the ced2021 spatial data.

In the first data set, the electorate column in called 'DivisionNm' and in the second 'ced_name_2021'.

We see the data in our DivisionNm variable is in UPPERCASE while our ced_name_2021 variable is in Titlecase. Let's change the first variable to Titlecase to match.

We can then then join the two dataframes using the left_join function.

```
#Pull in the electorate shapefiles from the absmapsdata package
electorates <- ced2021

#Make the DivisionNm Titlecase
tcp22$DivisionNm=str_to_title(tcp22$DivisionNm)

tcp22_edit <- tcp22 %>% distinct() %>% filter(Elected == "Y")

#Make the column names the same
electorates <- dplyr::rename(electorates, DivisionNm = ced_name_2021)

ced_map_data <- left_join(tcp22_edit, electorates, by = "DivisionNm")

ced_map_data <- as.data.frame(ced_map_data)

head(ced_map_data, n=151)
```

Analysis

Let's start by answering a simple question: who won the election? For this we'll need to use the two-candidate preferred data set (to make sure we capture all the minor parties that won seats).

Note in the table above, the PartyNm variable is a mess. Some candidates noted their party by it's abbreviation rather than full name, others put in a state specific prefix For this analysis, the PartyAb variable is cleaner to use.

```
who_won <- tcp22 %>%
  filter(Elected == "Y") %>%
  group_by(PartyAb) %>%
  tally() %>%
  arrange(desc(n))

head(who_won, n=10)
```

```
## # A tibble: 7 x 2
##   PartyAb     n
##   <chr>   <int>
## 1 ALP        77
## 2 LP         48
## 3 IND        10
## 4 NP         10
```

```
## 5 GRN          4
## 6 KAP          1
## 7 XEN          1
```

Next up let's see which candidates won with the smallest percentage of first preference votes. Australia's preferential voting system normally makes these numbers quite interesting.

```
who_won_least_votes_prop <- fp22 %>%
  filter(Elected == "Y") %>%
  arrange(Percent) %>%
  mutate(candidate_full_name = paste0(GivenNm, " ", Surname, " (", CandidateID, ")")) %
  dplyr::select(candidate_full_name, PartyNm, DivisionNm, Percent)

head(who_won_least_votes_prop,n=10)
```

```
## # A tibble: 10 x 4
##    candidate_full_name            PartyNm                DivisionNm   Percent
##    <chr>                          <chr>                  <chr>           <dbl>
##  1 KYLEA JANE TINK (37452)        INDEPENDENT            NORTH SYDNEY     25.2
##  2 SAM BIRRELL (36061)            NATIONAL PARTY         NICHOLLS         26.1
##  3 STEPHEN BATES (37338)          QUEENSLAND GREENS      BRISBANE         27.2
##  4 MICHELLE ANANDA-RAJAH (36433)  AUSTRALIAN LABOR PARTY HIGGINS          28.5
##  5 JUSTINE ELLIOT (36802)         AUSTRALIAN LABOR PARTY RICHMOND         28.8
##  6 BRIAN MITCHELL (37276)         AUSTRALIAN LABOR PARTY LYONS            29.0
##  7 KATE CHANEY (36589)            INDEPENDENT            CURTIN           29.5
##  8 DAI LE (36240)                 INDEPENDENT            FOWLER           29.5
##  9 ELIZABETH WATSON-BROWN (37370) QUEENSLAND GREENS      RYAN             30.2
## 10 REBEKHA SHARKIE (37710)        CENTRE ALLIANCE        MAYO             31.4
```

This is really something.

The relationship we're seeing here seems to be these are *many* the seats that are won with barely 30% of the first preference vote.

The electorate I grew up in is listed here (Richmond) - let's look at how the votes were allocated.

```
Richmond_fp <- fp22 %>%
  filter(DivisionNm == "RICHMOND") %>%
  arrange(-Percent) %>%
  mutate(candidate_full_name = paste0(GivenNm, " ", Surname, " (", CandidateID, ")")) %
  dplyr::select(candidate_full_name, PartyNm, DivisionNm, Percent, OrdinaryVotes)

head(Richmond_fp,n=10)
```

```
## # A tibble: 10 x 5
##    candidate_full_name            PartyNm       DivisionNm Percent OrdinaryVotes
##    <chr>                          <chr>         <chr>        <dbl>         <dbl>
##  1 JUSTINE ELLIOT (36802)         AUSTRALIAN L~ RICHMOND     28.8          28733
##  2 MANDY NOLAN (36361)            THE GREENS    RICHMOND     25.3          25216
##  3 KIMBERLY HONE (36351)          NATIONAL PAR~ RICHMOND     23.4          23299
##  4 GARY BIGGS (36648)             LIBERAL DEMO~ RICHMOND      7.7           7681
##  5 TRACEY BELL-HENSELIN (37831)   ONE NATION    RICHMOND      4.08          4073
##  6 ROBERT JAMES MARKS (37084)     UNITED AUSTR~ RICHMOND      2.93          2922
##  7 DAVID WARTH (37813)            INDEPENDENT   RICHMOND      2.35          2341
##  8 MONICA SHEPHERD (36408)        INFORMED MED~ RICHMOND      2.28          2271
##  9 NATHAN JONES (37721)           INDEPENDENT   RICHMOND      1.98          1974
## 10 TERRY PATRICK SHARPLES (37823) INDEPENDENT   RICHMOND      1.28          1274
```

Sure enough - the Greens certainly helped get the ALP across the line.

The interpretation that these seats are the 'most marginal' is incorrect under a preferential voting system (e.g. imagine if ALP win 30% and the Greens win 30% - that is a pretty safe 10% margin assuming traditional preference flows).

But - let's investigate which seats are the most marginal.

```
who_won_smallest_margin <- tcp22 %>%
  filter(Elected == "Y") %>%
  mutate(percent_margin = 2*(Percent - 50), vote_margin = round(percent_margin * OrdinaryVotes /
  arrange(Percent) %>%
  mutate(candidate_full_name = paste0(GivenNm, " ", Surname, " (", CandidateID, ")")) %>%
  dplyr::select(candidate_full_name, PartyNm, DivisionNm, Percent, OrdinaryVotes, percent_margin,

head(who_won_smallest_margin, n=20)
```

```
## # A tibble: 20 x 7
##    candidate_full_name   PartyNm DivisionNm Percent OrdinaryVotes percent_margin
##    <chr>                 <chr>   <chr>        <dbl>         <dbl>          <dbl>
##  1 FIONA PHILLIPS (3677~ AUSTRA~ Gilmore       50.2         56039          0.340
##  2 MICHAEL SUKKAR (3671~ LIBERA~ Deakin        50.2         50322          0.380
##  3 JAMES STEVENS (37067) LIBERA~ Sturt         50.4         56813          0.900
##  4 IAN GOODENOUGH (3659~ LIBERA~ Moore         50.7         52958          1.32
##  5 KEITH WOLAHAN (36733) LIBERA~ Menzies       50.7         51198          1.36
##  6 BRIAN MITCHELL (3727~ AUSTRA~ Lyons         50.9         37341          1.84
##  7 MARION SCRYMGOUR (37~ A.L.P.  Lingiari      51.0         23339          1.90
##  8 JEROME LAXALE (36827) AUSTRA~ Bennelong     51.0         50801          1.96
##  9 KATE CHANEY (36589)   INDEPE~ Curtin        51.3         53847          2.52
## 10 BRIDGET KATHLEEN ARC~ LIBERA~ Bass          51.4         35288          2.86
## 11 AARON VIOLI (36711)   LIBERA~ Casey         51.5         51283          2.96
## 12 DAI LE (36240)        INDEPE~ Fowler        51.6         44348          3.26
```

```
## 13 PETER DUTTON (37493)  LIBERA~ Dickson       51.7         51196          3.40
## 14 MICHELLE ANANDA-RAJA~ AUSTRA~ Higgins       52.1         49726          4.12
## 15 GORDON REID (36801)   AUSTRA~ Robertson     52.3         50277          4.52
## 16 PAT CONAGHAN (36342)  NATION~ Cowper        52.3         58204          4.64
## 17 SAM LIM (37337)       AUSTRA~ Tangney       52.4         56331          4.76
## 18 SOPHIE SCAMPS (37450) INDEPE~ Mackellar     52.5         51973          5
## 19 ELIZABETH WATSON-BRO~ QUEENS~ Ryan          52.6         52286          5.3
## 20 ALAN TUDGE (36704)    LIBERA~ Aston         52.8         51840          5.62
## # i 1 more variable: vote_margin <dbl>
```

Crikey. We see Fiona Phillips in Gilmore got in with a 0.17% margin (just 380 votes!)

While we're at it, we better do the opposite and see who romped it in by the largest margin.

```
who_won_largest_margin <- tcp22 %>%
  filter(Elected == "Y") %>%
  mutate(percent_margin = 2*(Percent - 50), vote_margin = round(percent_margin * Ordina
  arrange(desc(Percent)) %>%
  mutate(candidate_full_name = paste0(GivenNm, " ", Surname, " (", CandidateID, ")")) %
  dplyr::select(candidate_full_name, PartyNm, DivisionNm, Percent, OrdinaryVotes, perce

 head(who_won_largest_margin, n=20)
```

```
## # A tibble: 20 x 7
##    candidate_full_name   PartyNm DivisionNm Percent OrdinaryVotes percent_margin
##    <chr>                 <chr>   <chr>        <dbl>         <dbl>           <dbl>
##  1 DAVID LITTLEPROUD (3~ LIBERA~ Maranoa       72.1         67153            44.2
##  2 ANDREW WILKIE (33553) INDEPE~ Clark         70.8         46668            41.6
##  3 DARREN CHESTER (3604~ NATION~ Gippsland     70.6         71205            41.1
##  4 ANNE WEBSTER (36060)  NATION~ Mallee        69.0         70523            38.0
##  5 SHARON CLAYDON (3680~ AUSTRA~ Newcastle     68.0         71807            36.0
##  6 MARK COULTON (36363)  NATION~ Parkes        67.8         60433            35.7
##  7 ANTHONY ALBANESE (36~ AUSTRA~ Grayndler     67.0         63413            34.1
##  8 JOSH WILSON (37314)   AUSTRA~ Fremantle     66.9         65585            33.8
##  9 MADELEINE KING (3718~ AUSTRA~ Brand         66.7         63829            33.4
## 10 TANYA PLIBERSEK (368~ AUSTRA~ Sydney        66.7         68770            33.4
## 11 TONY PASIN (37083)    LIBERA~ Barker        66.6         70483            33.2
## 12 DANIEL MULINO (36385) AUSTRA~ Fraser        66.5         61251            33
## 13 BARNABY JOYCE (36335) NATION~ New Engla~    66.4         64622            32.9
## 14 SUSSAN LEY (37032)    LIBERA~ Farrer        66.4         66739            32.7
## 15 AMANDA RISHWORTH (36~ AUSTRA~ Kingston      66.4         72564            32.7
## 16 ANDREW LEIGH (36234)  AUSTRA~ Fenner        65.7         59966            31.4
## 17 ANDREW GILES (36447)  AUSTRA~ Scullin       65.6         59761            31.2
```

```
## 18 LINDA BURNEY (36820)  AUSTRA~ Barton        65.5         60054          31.1
## 19 MATT KEOGH (37195)    AUSTRA~ Burt          65.2         59704          30.4
## 20 TONY BURKE (36809)    AUSTRA~ Watson        65.1         55810          30.2
## # i 1 more variable: vote_margin <dbl>
```

Wowza. That's really something. Some candidates won seats with a 40-45 percent margin - scooping up 70% of the two candidate preferred vote in the process!

We can also cut the seats by state for a look at where the 'strongholds' are across the country.

```
who_won <- tcp22 %>%
  filter(Elected == "Y") %>%
  group_by(PartyAb, StateAb) %>%
  tally() %>%
  arrange(desc(n))

who_won_by_state <- spread(who_won,StateAb, n) %>% arrange(desc(NSW))

head(who_won_by_state, n=10)
```

Trends

Now we've figured out how to work with election data - let's link it up to some AUstralian demographic data. The eechidna package includes a cleaned set of census data from 2022 that has already been adjusted from ASGS boundaries to Commonwealth Electoral Divisions.

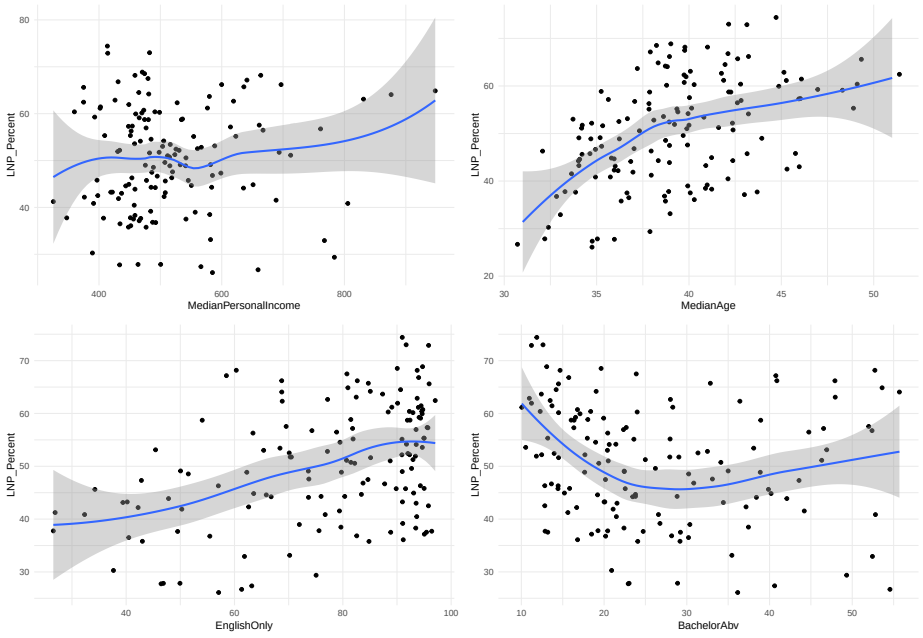
```
# Import the census data from the eechidna package
data(eechidna::abs2021)
head(abs2021)

# Join with two-party preferred voting data
data(tpp10)
election2022 <- left_join(abs2021, tpp10, by = "DivisionNm")
```

That's what we want to see. 151 rows of data (one for each electorate) and over 80 columns of census variables.

A starting exploratory exercise is too see which of these variables are correlated with voting for one party or another. There's some old narrative around LNP voters being rich, old, white, and somehow 'upper class' compared to the population at large. Let's pick a few variables that roughly match with this criteria

(Income, Age, English language speakers, and Bachelor educated) and chart it compared to LNP percentage of the vote.

```
# See relationship between personal income and Liberal/National support

ggplot(election2022, aes(x = MedianPersonalIncome, y = LNP_Percent)) +
  geom_point() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = MedianAge, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = EnglishOnly, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = BachelorAbv, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()
```



First impressions: Geez this data looks messy.

Second impression: Maybe there's a bit of a trend with age and income?

Let's build a regression model to run all the 80 odd census variables in the abs2022 data set against the LNP_percent variable.

```
# We can use colnames(election2022) to get a big list of all the variables available

# Now we build the model
election_model <- lm(LNP_Percent~
                     Population+
                     Area+
                     Age00_04+
                     Age05_14+
                     Age15_19+
                     Age20_24+
                     Age25_34+
                     Age35_44+
                     Age45_54+
                     Age55_64+
                     Age65_74+
                     Age75_84+
                     Age85plus+
                     Anglican+
                     AusCitizen+
                     AverageHouseholdSize+
                     BachelorAbv+Born_Asia+
                     Born_MidEast+Born_SE_Europe+
                     Born_UK+
                     BornElsewhere+
                     Buddhism+
                     Catholic+
                     Christianity+
                     Couple_NoChild_House+Couple_WChild_House+
                     CurrentlyStudying+DeFacto+
                     DiffAddress+
                     DipCert+
                     EnglishOnly+
                     FamilyRatio+
                     Finance+
                     HighSchool+
                     Indigenous+
                     InternetUse+
                     Islam+
                     Judaism+
                     Laborer+
```

```
                    LFParticipation+
                    Married+
                    MedianAge+
                    MedianFamilyIncome+
                    MedianHouseholdIncome+
                    MedianLoanPay+
                    MedianPersonalIncome+
                    MedianRent+
                    Mortgage+
                    NoReligion+
                    OneParent_House+
                    Owned+
                    Professional+
                    PublicHousing+
                    Renting+
                    SocialServ+
                    SP_House+
                    Tradesperson+
                    Unemployed+
                    Volunteer,
                    data=election2022)

summary(election_model)
```

```
##
## Call:
## lm(formula = LNP_Percent ~ Population + Area + Age00_04 + Age05_14 +
##     Age15_19 + Age20_24 + Age25_34 + Age35_44 + Age45_54 + Age55_64 +
##     Age65_74 + Age75_84 + Age85plus + Anglican + AusCitizen +
##     AverageHouseholdSize + BachelorAbv + Born_Asia + Born_MidEast +
##     Born_SE_Europe + Born_UK + BornElsewhere + Buddhism + Catholic +
##     Christianity + Couple_NoChild_House + Couple_WChild_House +
##     CurrentlyStudying + DeFacto + DiffAddress + DipCert + EnglishOnly +
##     FamilyRatio + Finance + HighSchool + Indigenous + InternetUse +
##     Islam + Judaism + Laborer + LFParticipation + Married + MedianAge +
##     MedianFamilyIncome + MedianHouseholdIncome + MedianLoanPay +
##     MedianPersonalIncome + MedianRent + Mortgage + NoReligion +
##     OneParent_House + Owned + Professional + PublicHousing +
##     Renting + SocialServ + SP_House + Tradesperson + Unemployed +
##     Volunteer, data = election2022)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.9358  -2.0714  -0.1947   2.1152   8.1823
##
```

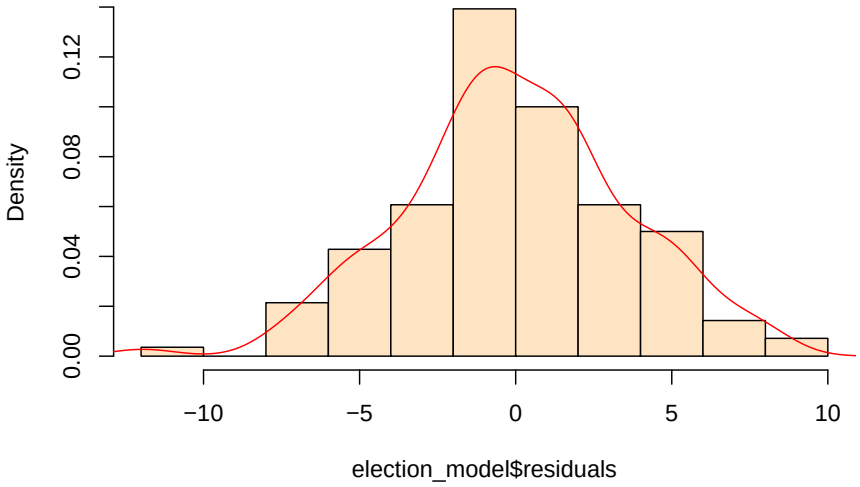
```
## Coefficients: (1 not defined because of singularities)
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             2.973e+04  1.408e+04   2.111 0.037912 *
## Population              8.911e-05  4.885e-05   1.824 0.071865 .
## Area                    7.945e-06  5.069e-06   1.567 0.120957
## Age00_04               -2.830e+02  1.402e+02  -2.018 0.046910 *
## Age05_14               -2.923e+02  1.403e+02  -2.084 0.040367 *
## Age15_19               -2.793e+02  1.401e+02  -1.993 0.049677 *
## Age20_24               -2.989e+02  1.402e+02  -2.132 0.036042 *
## Age25_34               -2.921e+02  1.404e+02  -2.080 0.040715 *
## Age35_44               -2.942e+02  1.401e+02  -2.100 0.038900 *
## Age45_54               -2.921e+02  1.404e+02  -2.080 0.040689 *
## Age55_64               -2.903e+02  1.400e+02  -2.073 0.041402 *
## Age65_74               -2.888e+02  1.401e+02  -2.062 0.042477 *
## Age75_84               -2.844e+02  1.404e+02  -2.025 0.046152 *
## Age85plus              -3.014e+02  1.398e+02  -2.156 0.034123 *
## Anglican                8.452e-01  4.518e-01   1.871 0.065057 .
## AusCitizen              4.673e-01  7.032e-01   0.664 0.508282
## AverageHouseholdSize   -2.556e+01  1.684e+01  -1.518 0.133076
## BachelorAbv            -3.068e+00  8.049e-01  -3.812 0.000270 ***
## Born_Asia               3.963e-01  3.771e-01   1.051 0.296516
## Born_MidEast            1.091e+00  1.029e+00   1.060 0.292396
## Born_SE_Europe         -1.628e+00  2.119e+00  -0.768 0.444585
## Born_UK                 1.487e-01  4.886e-01   0.304 0.761591
## BornElsewhere           7.110e-01  5.618e-01   1.266 0.209346
## Buddhism               -1.097e+00  7.070e-01  -1.551 0.124835
## Catholic               -6.257e-01  4.202e-01  -1.489 0.140356
## Christianity            5.736e-01  5.095e-01   1.126 0.263605
## Couple_NoChild_House   -1.581e+00  3.747e+00  -0.422 0.674192
## Couple_WChild_House    -3.264e-01  4.061e+00  -0.080 0.936150
## CurrentlyStudying      -4.117e-01  1.241e+00  -0.332 0.740988
## DeFacto                -6.600e+00  1.931e+00  -3.417 0.000997 ***
## DiffAddress             9.427e-01  3.174e-01   2.970 0.003935 **
## DipCert                -9.857e-01  6.871e-01  -1.435 0.155319
## EnglishOnly            -3.977e-01  4.103e-01  -0.969 0.335365
## FamilyRatio            -2.873e+01  4.547e+01  -0.632 0.529296
## Finance                 1.399e+00  8.717e-01   1.605 0.112372
## HighSchool              7.160e-01  4.161e-01   1.721 0.089173 .
## Indigenous              3.919e-01  4.572e-01   0.857 0.393876
## InternetUse                    NA         NA      NA       NA
## Islam                  -4.161e-01  5.624e-01  -0.740 0.461601
## Judaism                 4.377e-01  6.511e-01   0.672 0.503385
## Laborer                -8.258e-01  7.283e-01  -1.134 0.260251
## LFParticipation         6.822e-01  6.359e-01   1.073 0.286562
## Married                -5.852e+00  1.734e+00  -3.375 0.001141 **
## MedianAge              -9.926e-01  1.036e+00  -0.958 0.340782
```

```
## MedianFamilyIncome    -3.456e-02  2.524e-02  -1.369 0.174711
## MedianHouseholdIncome  6.410e-02  2.756e-02   2.326 0.022568 *
## MedianLoanPay         -1.889e-02  1.143e-02  -1.653 0.102290
## MedianPersonalIncome   2.957e-02  5.171e-02   0.572 0.569043
## MedianRent             6.781e-03  5.786e-02   0.117 0.906997
## Mortgage               2.021e-02  1.406e+00   0.014 0.988568
## NoReligion             6.649e-01  4.779e-01   1.391 0.167983
## OneParent_House       -6.243e+00  3.876e+00  -1.611 0.111150
## Owned                  1.168e-02  1.334e+00   0.009 0.993036
## Professional           1.314e+00  8.840e-01   1.487 0.141064
## PublicHousing         -8.154e-01  5.754e-01  -1.417 0.160350
## Renting               -2.602e-02  1.463e+00  -0.018 0.985859
## SocialServ            -5.254e-02  4.674e-01  -0.112 0.910776
## SP_House              -2.021e-01  8.735e-01  -0.231 0.817658
## Tradesperson          -1.299e-01  7.929e-01  -0.164 0.870301
## Unemployed            -4.229e+00  1.503e+00  -2.814 0.006160 **
## Volunteer              4.773e-01  6.417e-01   0.744 0.459120
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.729 on 80 degrees of freedom
##   (11 observations deleted due to missingness)
## Multiple R-squared:  0.8975, Adjusted R-squared:  0.822
## F-statistic: 11.88 on 59 and 80 DF,  p-value: < 2.2e-16
```

For the people that care about statistical fit and endogenous variables, you may have concerns (and rightly so) with the above approach.

It's pretty rough. Let's run a basic check to see if the residuals are normally distributed.

```
hist(election_model$residuals, col="bisque", freq=FALSE, main=NA)
lines(density(election_model$residuals), col="red")
```



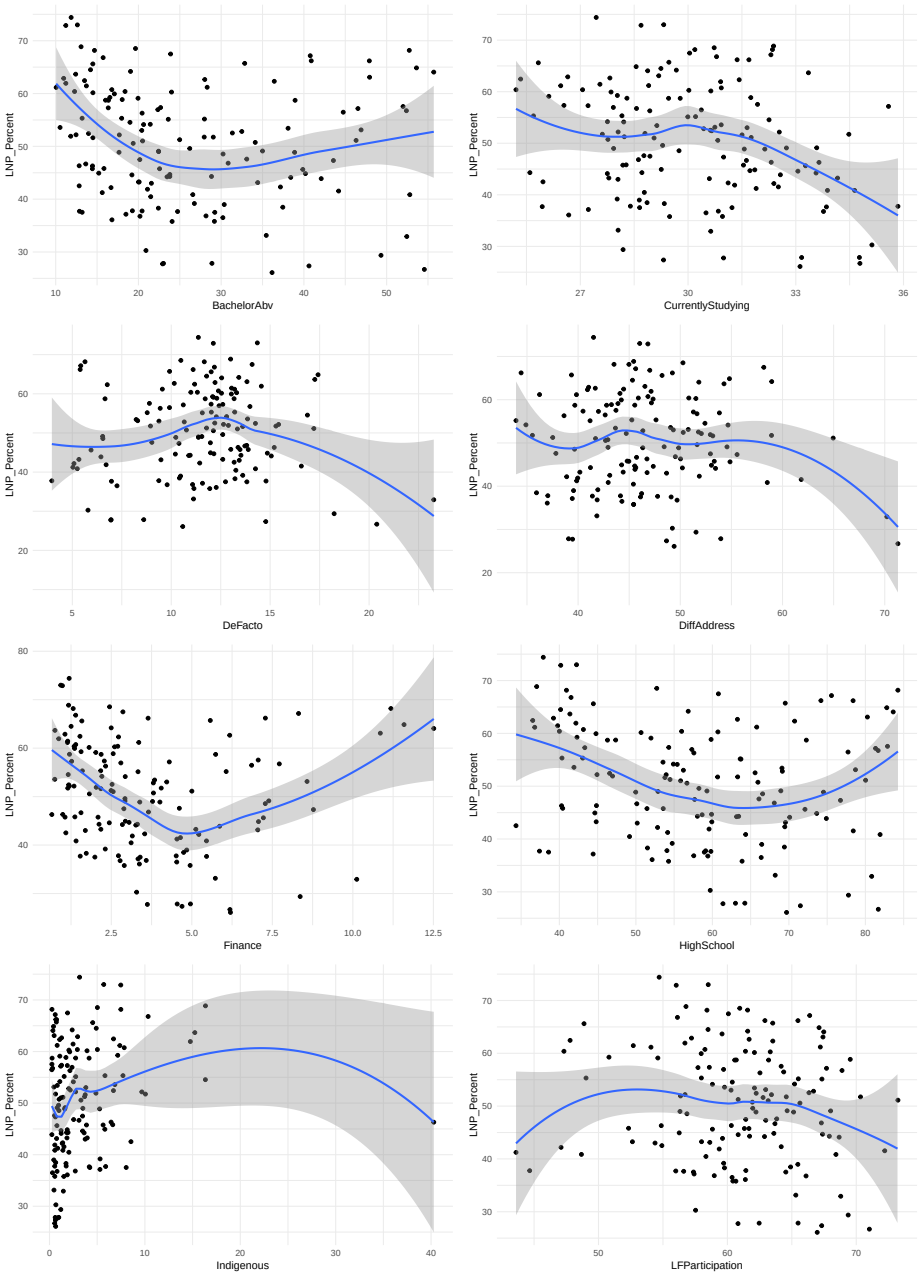
Hmm... that's actually not too bad. Onwards.

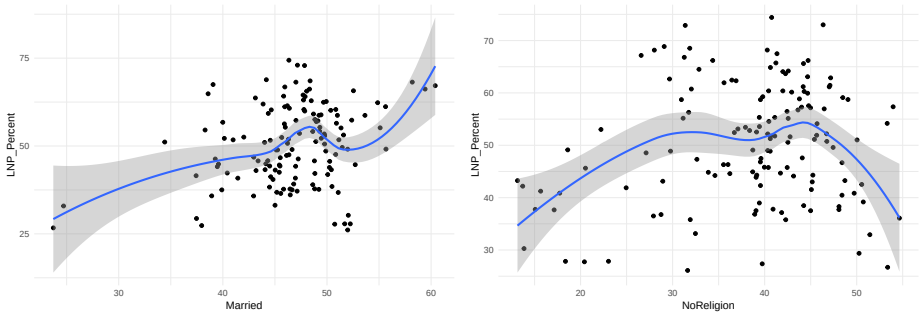
We see now that only a handful of these variables in the table above are statistically significant. Running an updated (and leaner) model gives:

```
election_model_lean <- lm(LNP_Percent~
                            BachelorAbv+
                            CurrentlyStudying+
                            DeFacto+
                            DiffAddress+
                            Finance+HighSchool+
                            Indigenous+
                            LFParticipation+
                            Married+
                            NoReligion,
                            data=election2022)

summary(election_model_lean)
```

```
ggplot(election2022, aes(x = BachelorAbv, y = LNP_Percent)) +
  geom_point() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = CurrentlyStudying, y = LNP_Percent)) +
```

```
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = DeFacto, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = DiffAddress, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = Finance, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = HighSchool, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = Indigenous, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = LFParticipation, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = Married, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()

ggplot(election2022, aes(x = NoReligion, y = LNP_Percent)) +
  geom_jitter() +
  geom_smooth() +
  theme_minimal()
```


LNP_Percent
BachelorAbv
CurrentlyStudying
DeFacto
DiffAddress
Finance
HighSchool
Indigenous
LFParticipation



One of the more novel issues with the above analysis is that it's done on the electorate level, however electorates vary dramatically in side (google the Modifiable Area Unit Problem if this piques your interest)

Instead of using one data point for a whole electorate (regardless of how big it is), we can fetch data from all 7,000 voting booths, the match it up with the corresponding local demographic data.

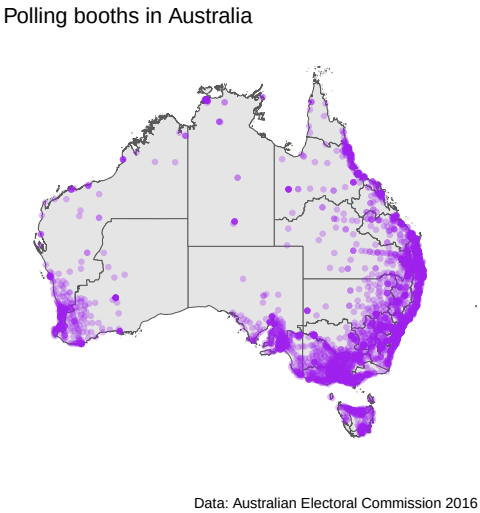
Booth data

The AEC maintains a handy spreadsheet of booth locations for recent federal elections. You can search for your local booth location (probably a school, church, or community center) in the table below.

```
## # A tibble: 10 x 15
##    State DivisionID DivisionNm PollingPlaceID PollingPlaceTypeID PollingPlaceNm
##    <chr>      <dbl> <chr>               <dbl>              <dbl> <chr>
##  1 ACT          101 Canberra             8829                  1 Barton
##  2 ACT          101 Canberra            64583                  5 Belconnen CANB~
##  3 ACT          101 Canberra            65504                  5 BLV Canberra P~
##  4 ACT          101 Canberra            11877                  1 Bonython
##  5 ACT          101 Canberra             8802                  1 Braddon (Canbe~
##  6 ACT          101 Canberra            11452                  1 Calwell
##  7 ACT          101 Canberra             8806                  1 Campbell
##  8 ACT          101 Canberra             8761                  1 Chapman
##  9 ACT          101 Canberra             8763                  1 Chisholm
## 10 ACT          101 Canberra             8808                  1 City (Canberra)
## # i 9 more variables: PremisesNm <chr>, PremisesAddress1 <chr>,
## #   PremisesAddress2 <chr>, PremisesAddress3 <chr>, PremisesSuburb <chr>,
## #   PremisesStateAb <chr>, PremisesPostCode <chr>, Latitude <dbl>,
## #   Longitude <dbl>
```

What do these booths look like on a map? Let's reuse the CED map above and plot a point for each booth location.

```
ggplot() +
  geom_sf(data = ced2021) +
  geom_point(data = booths, aes(x = Longitude, y = Latitude),
             colour = "purple", size = 1, alpha = 0.3, inherit.aes = FALSE) +
  labs(
    title = "Polling booths in Australia",
    subtitle = " ",
    caption = "Data: Australian Electoral Commission 2016",
    x = "",
    y = ""
  ) +
  theme_minimal() +
  theme(
    axis.ticks.x = element_blank(),
    axis.text.x = element_blank(),
    axis.ticks.y = element_blank(),
    axis.text.y = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    legend.position = "right",
    plot.title = element_text(size = 12),
    plot.subtitle = element_text(size = 11),
    plot.caption = element_text(size = 8)
  ) +
  xlim(c(112, 157)) +
  ylim(c(-44, -11))
```



Figuring out where a candidates votes come from *within* an electorate is funda-

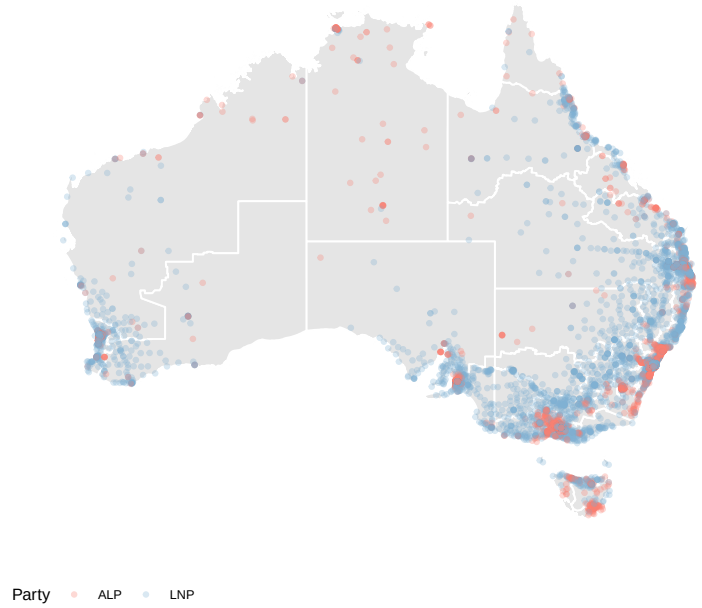
mental to developing a campaign strategy. Even in small electorates (e.g. Wentworth), there are pockets of right leaning and left leaning districts. Once you factor in preference flows, this multi-variate calculus becomes important to winning or maintaining a seat.

In the `eechidna`package, election results are provided at the resolution of polling place. Unfortunately, these are yet to be updated for elections after 2016.

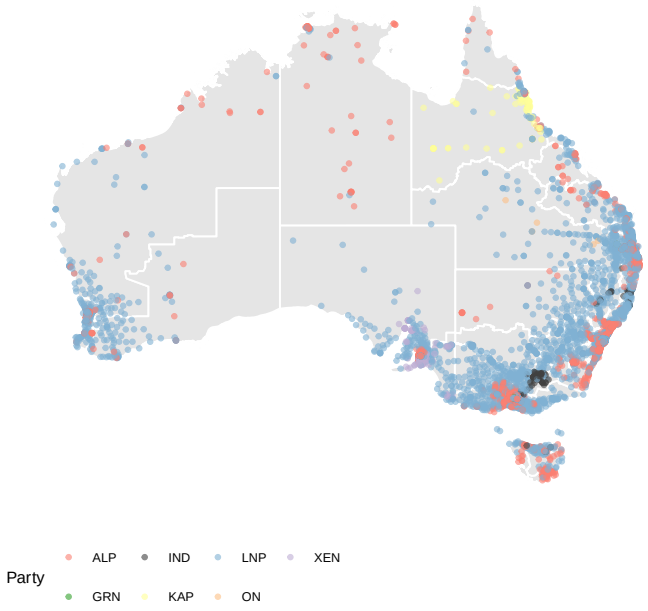
The data sets must be downloaded using the functions `firstpref_pollingbooth_download`, `twoparty_pollingbooth_download` or `twocand_pollingbooth_download` (depending on the vote type).

The two files need to be merged to be useful for analysis. Both have a unique ID for the polling place that can be used to match the records. The two party preferred vote, a measure of preference between only the Australian Labor Party (ALP) and the Liberal/National Coalition (LNP), is downloaded using `twoparty_pollingbooth_download`. The preferred party is the one with the higher percentage, and we use this to colour the points indicating polling places.

We see that within some big rural electorates (e.g. in Western NSW), there are pockets of ALP preference despite the seat going to the LNP. Note that this data set is on a tpp basis - so we can't see the booths that were won by minor parties (although it would be fascinating).We see that within some big rural electorates (e.g. in Western NSW), there are pockets of ALP preference despite the seat going to the LNP. Note that this data set is on a tpp basis - so we can't see the booths that were won by minor parties (although it would be fascinating).



The two candidate preferred vote (downloaded with `twocand_pollingbooth_download`) is a measure of preference between the two candidates who received the most votes through the division of preferences, where the winner has the higher percentage.



Informal votes

Over 700,000 people (around 5% of all votes cast) vote informally each election. Of these, over have 'no clear first preference', meaning their vote did not contribute to the campaign of any candidate.

I'll be honest, informal votes absolutely *fascinate* me. Not only are there 8 types of informal votes (you can read all about the Australian Electoral Commission's analysis here), but the rate of informal voting varies a tremendous amount by electorate.

Broadly, we can think of informal votes in two main buckets.

1. Protest votes
2. Stuff-ups

If we want to get particular about it, I like to subcategorise these buckets into:

1. Protest votes (i.e. a person that thinks they are voting against):
    - the democratic system,
    - their local selection of candidates on the ballot, or
    - the two most likely candidates for PM.
2. Stuff ups (people who):
    - filled in the form wrong but a clear preference was still made
    - stuffed up the form entirely and it didn't contribute towards the tally for any candidtate

The AEC works tirelessly to reduce stuff-ups on ballot papers (clear instructions and UI etc), but there isn't much of a solution for protest votes.

Bayesian for the common man

Note: Chapter under development

Enter Rev. Bayes

Bayes' Theorem is named after 18th-century British statistician and theologian, Thomas Bayes. The theorem describes how to update the probability of a hypothesis based on new evidence.

In traditional frequentist statistics, probability is interpreted as the long-run frequency of events in repeated trials (sampling). A p-value from a sampling distribution tells us: *What is the chance of seeing this result, given some hypothesis?*

Bayesian statistics, on the other hand, allows us to incorporate *prior* knowledge or beliefs into our analysis and update those beliefs as we gather more data. The central question of Bayes' Theorem therefore is: *How likely is the hypothesis to be true, given the data I've seen?*

This isn't semantics - it's a completely different way of seeing the world.

We can write Bayes' Theorem using probability notation.

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Causal inference

Note: Chapter under development

What causes what

Causal inference the process of determining cause-and-effect relationships. Simply finding an association between two variables might be suggestive of a causal effect... but it also might not.

The best book on this topic is Causal Inference: The Mixtape by the phenomenally talented Scott Cunningham.

Functions

Note: Chapter under development

Functions are set sof instructions that take inputs (arguments), processes them, and returns an output. Instead of writing the same code repeatedly, you can define a function once and use it whenever needed. This makes your scripts cleaner and easier to maintain.

I recommend Hadley Wickham's great chapter on functions in his book R for Data Science 2e.

Components

A function in R is defined using the `function` keyword. Here is a simple function that adds two numbers:

```
add_numbers <- function(a, b) {
  result <- a + b
  return(result)
}

# Using the function
add_numbers(3, 5)
```

In this example, we see a function consisting of:

1. A name (e.g., `add_numbers`)
2. Arguments (e.g., `a`, `b`)
3. A body that performs calculations or operations
4. A return statement (optional, but recommended)

We can also reference functions (from other packages) for use in our own custom functions. This data cleaning functions references `dplyr`, `stringr,` and `tidyr`.

```r
# Practical Example: Data Cleaning Function

library(dplyr)
library(stringr)
library(tidyr)

data_cleaner <- function(df) {
  df <- df %>%
    dplyr::mutate(across(where(is.character),
    stringr::str_trim)) %>% # Trim whitespace
    tidyr::drop_na() # Remove missing values
  return(df)
}

# Example
df <- data.frame(name = c("Alice    ", " Bob", NA), age = c(25, 30, 22))
data_cleaner(df)
```

Luhn example

It turns out credit card numbers aren't a random 16 digits, there's an algorithm (named after it's inventor Hans Peter Luhn) that determines the combinations possible.

It follows these steps:

1. Starting from the right, double every second digit.
2. If doubling a digit results in a number greater than 9, subtract 9 from it.
3. Sum all the digits.
4. If the total sum is a multiple of 10, the number is valid; otherwise, it is invalid.

That's a lot of info... instead let's write a function to check if a credit card number is valid:

```r
validate_credit_card <- function(card_number) {
  digits <- as.numeric(strsplit(as.character(card_number), "")[[1]])
  n <- length(digits)

  # Double every second digit from the right
  for (i in seq(n - 1, 1, by = -2)) {
    digits[i] <- digits[i] * 2
    if (digits[i] > 9) {
```

```
      digits[i] <- digits[i] - 9
    }
  }

  # Check if the sum is a multiple of 10
  return(sum(digits) %% 10 == 0)
}

# Example usage
validate_credit_card(4532015112830366) # Should return TRUE
validate_credit_card(1234567812345678) # Should return FALSE
```

Loops in functions

The real power of functions comes when we introduce loops. Loops allow us to repeat a calculation for different parameter values. Factorials are a great example where a loop is needed.

A *factorial* is a mathematical operation denoted by an exclamation mark (!), used to find the product of all positive integers less than or equal to a given number.

Let's write a looping function to calculate 10! (10x9x8x7x6x5x4x3x2x1)

```
## Calculating Factorial Using a Loop

factorial_loop <- function(n) {
  if (n < 0) {
    stop("Factorial is not defined for negative numbers.")
  }
  result <- 1
  for (i in 1:n) {
    result <- result * i
  }
  return(result)
}

# Example usage
factorial_loop(10) # Test 10!
```

We see factorials blow out in size *very* quickly. Try inputting 100! and see what happens.

Packages

Note: Chapter under development

Pack it up

The 'fundamental unit of shareable code' is a package. At its core, a package is about automation, convenience, and integrity. Rather than copy-pasting, or repeating analytical steps, a package guarantees the same functions are available time and time again.

The best book on this topic is R Packages (2e) by Hadley Wickham and Jenifer Bryan.