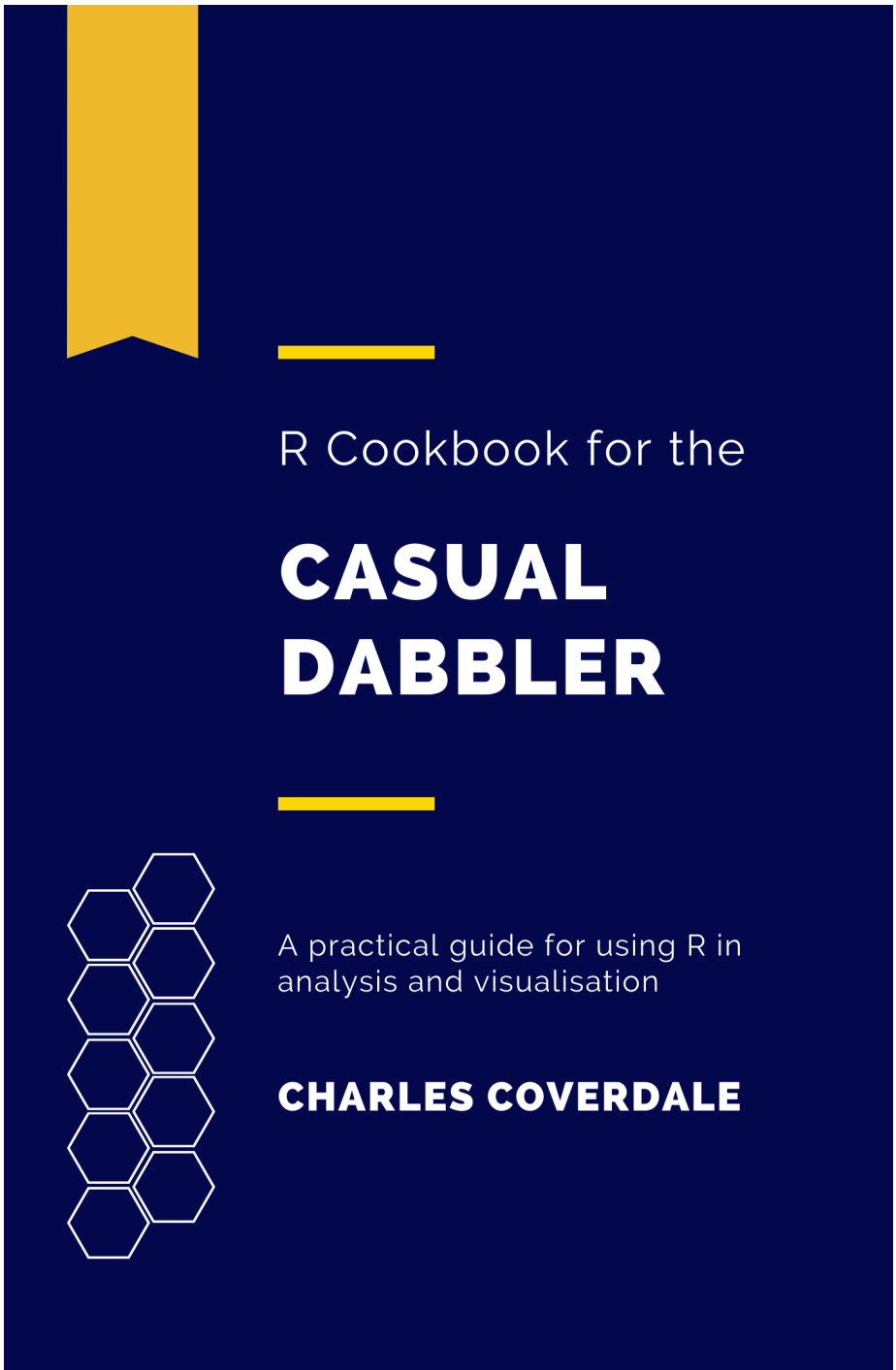
R cookbook for the casual dabbler

Charles Coverdale

2022-06-15

Introduction



G'day and welcome to *R cookbook for the casual dabbler.*

Some history: I use R a lot for work and for side projects. Over the years I've collated a bunch of useful scripts, from macroeconomic analysis to quick hacks for making map legends format properly.

Historically my code has been stored in random Rpubs documents, medium articles, and a bunch of .Rmd files on my hardrive. Occasionally I feel like doing things properly - and upload code to a repository on github.

It doesn't take a genius to realize this isn't a very sustainable solution - and it also isn't very useful for sharing code with others. It turns out 2-years of lockdown in Melbourne was enough incentive to sit down and collate my best and most useful code into a single place. In the spirit of open source, a book seemed like the most logical format. The following is a *very rough* book written in **markdown** - R's very own publishing language.

Usage

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

Most of this code will not be extraordinary to the seasoned R aficionado. The vast majority can be found elsewhere if you dig around on stackexchange or read some of Hadley's books.

However I find that in classic Pareto style ~20% of my code contributes to the **vast** majority of my work output. Having this on hand will hopefully be useful to both myself and others.

Additional resources

The R community is continually writing new books and package documentation with great worked examples. Some of my favourites (which all happen to be written in the R markdown language) are:

- Geocomputation with R
- R Markdown: The Definite Guide
- R Cookbook, 2nd Edition
- R for Data Science
- Data Science in Education using R
- Introduction to R: Walter and Eliza Hall Institute

- PhD lectures notes in environmental economics and data science (University of Oregon)

Limitations

I'll be honest with you - there's bound to be bugs galore in this. If you find one (along with spelling errors etc) please email me at charlesfcoverdale@gmail.com with the subject line 'R cookbook for the casual dabbler.'

About the author

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

Making beautiful maps in R

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 professional 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 (note: they can be fiddly). 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 quick-reference guide is to get you from 'I can produce a map in R' to something more like 'I can conduct spatial analysis and produce a visual which is ready to send without any further work.'

Getting started

First up, we need to load a bunch of packages.

```
#Loads the required required packages
library(plyr)
library(dplyr)
library(vctrs)
library(tidyr)
library(ggplot2)
library(tmap)
library(ggmap)
library(dplyr)
library(sf)
library(ggspatial)
library(rlang)
library(broom)
```

```
library(tidyverse)
library(readxl)
library(purrr)
library(Census2016)
library(absmapsdata)
library(officer)
library(bookdown)
```

The absmapsdata package is particularly important, as it contains most ASGS shapefiles (the ones found on the ABS website).

The main ASGS structures and their names in this package are:

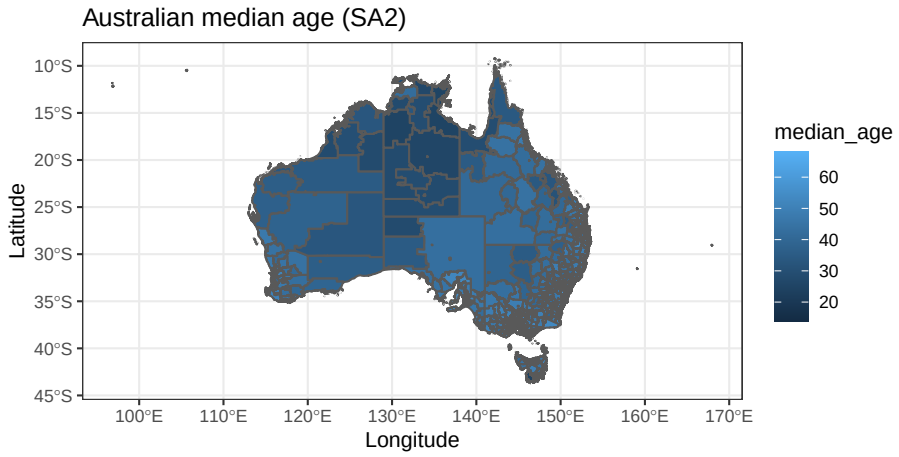
SA1 2016: sa12016 SA2 2016: sa22016 SA3 2016: sa32016 SA4 2016: sa42016 Greater Capital Cities 2016: gcc2016 Remoteness Areas 2016: ra2016 State 2016: state2016

Making your first map

To get a basic demographic map up and running, we will splice together the ABS SA2 shapefile and some data from the 2016 Australian Census. There is 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(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! This looks 'okay'... but it can be much better.

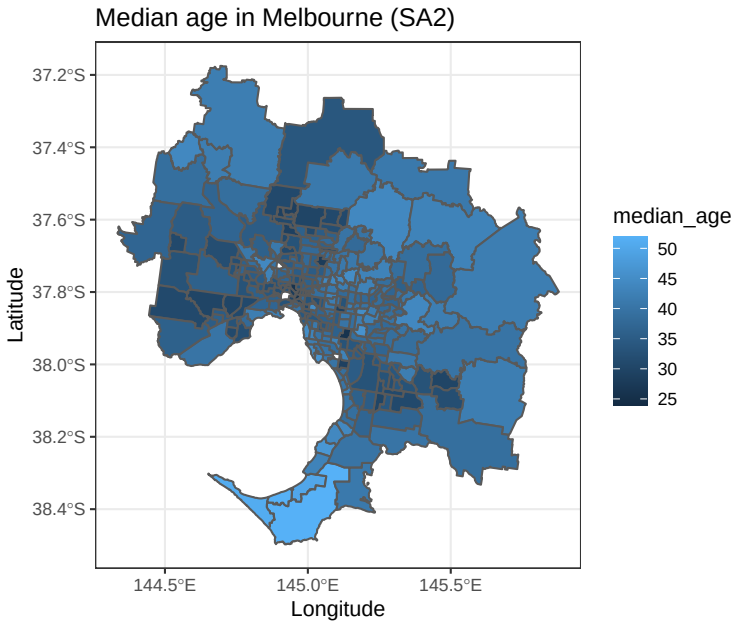
From okay to good

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.

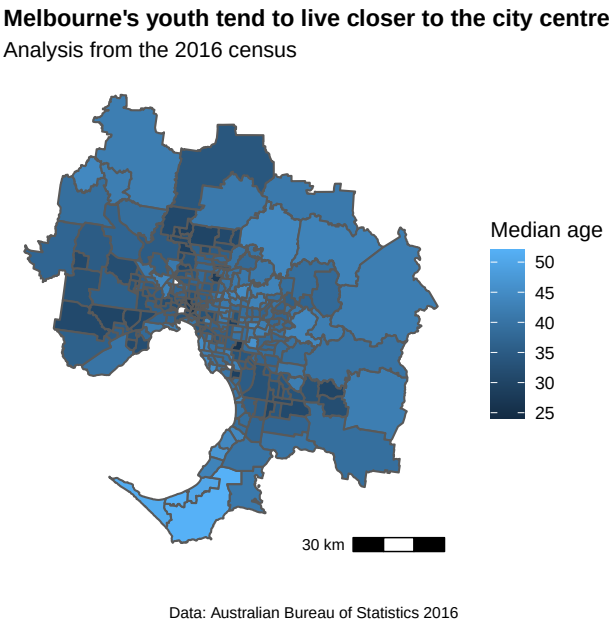
From good to great

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
```

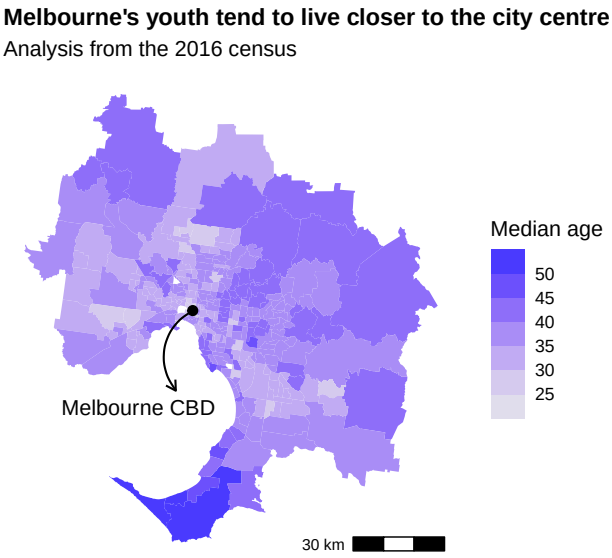


From great to fantastic

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
```



There we go! 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 ggsave function.

Basic modelling in R

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(dplyr)
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).

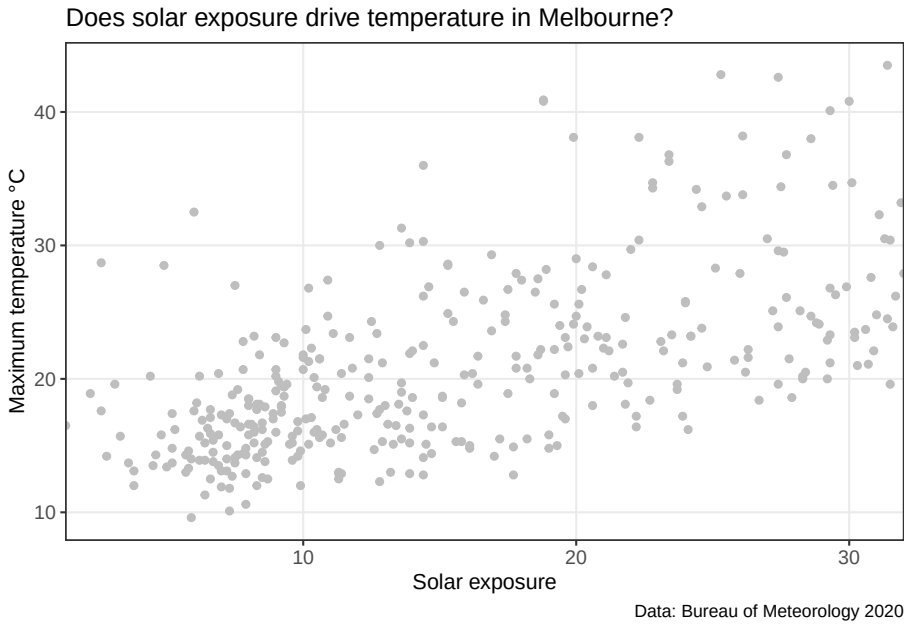
```
#Add a proper date column
MEL_weather_2019 <- MEL_weather_2019 %>%
  mutate(Date = make_date(Year, Month, Day))
```

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)
```

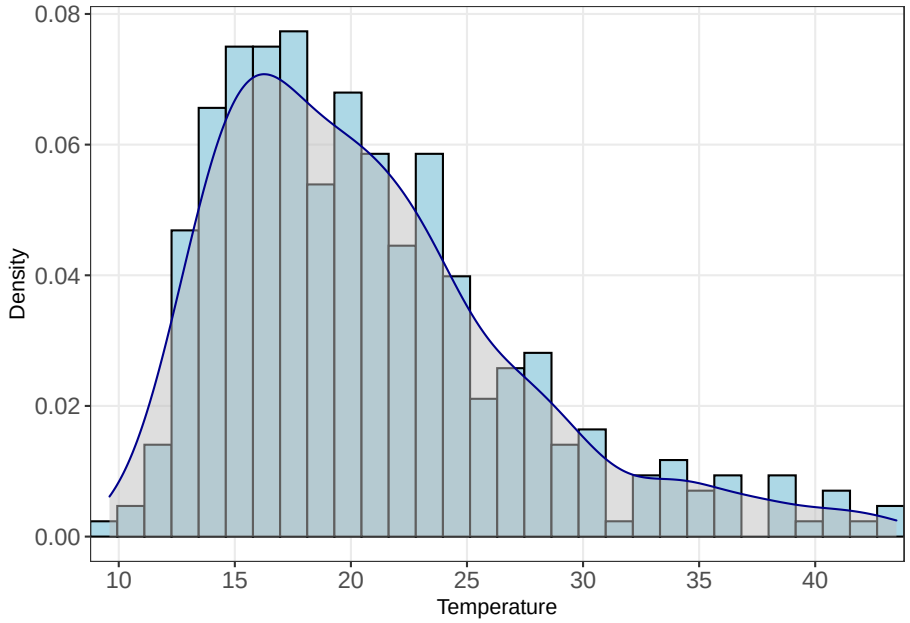
We're aiming to investigate if other weather variables can predict maximum temperatures. Solar exposure seems like a plausible place to start. We start by plotting the two variables to if there is a trend.

```
#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

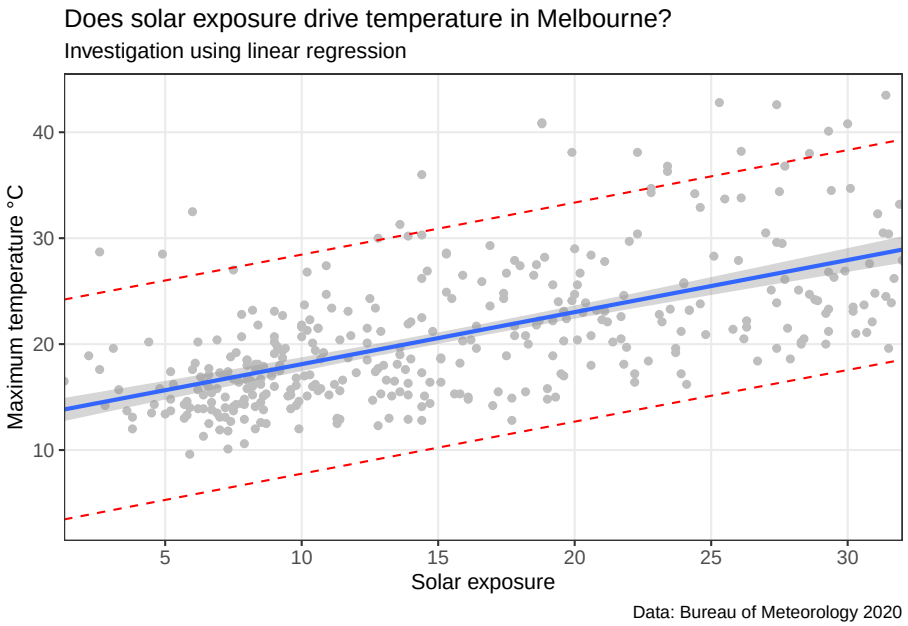
We can use this lm function to predict values of temperature based on the level of solar exposure. We can then compare this to the actual temperature record, and see how well the model fits the data set.

```
#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
```



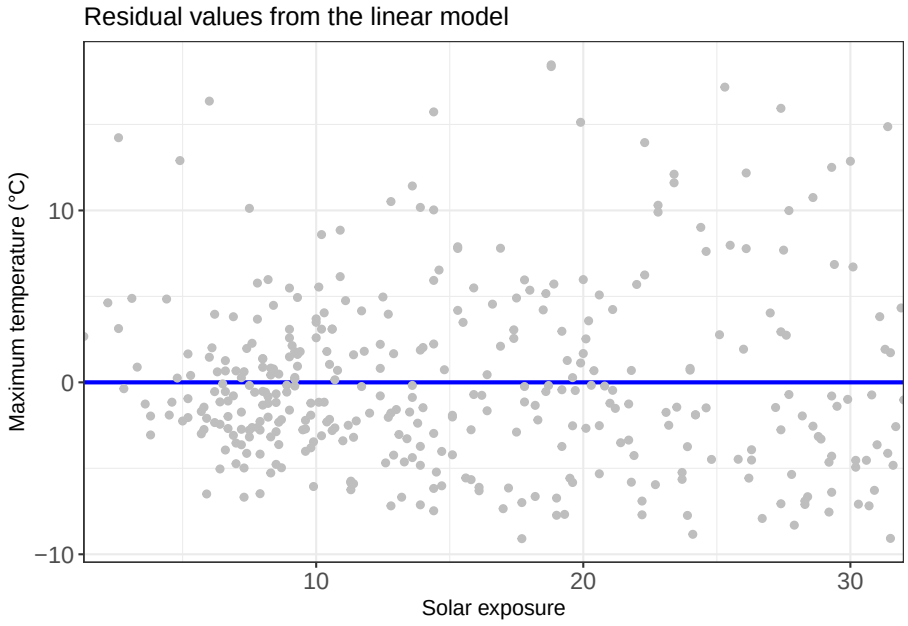
This chart includes the model (blue line), confidence interval (grey band around the blue line), and a prediction interval (red dotted line). A prediction interval reflects the uncertainty around a single value (put simple: what is the reasonable upper and lower bound that this data point could be estimated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

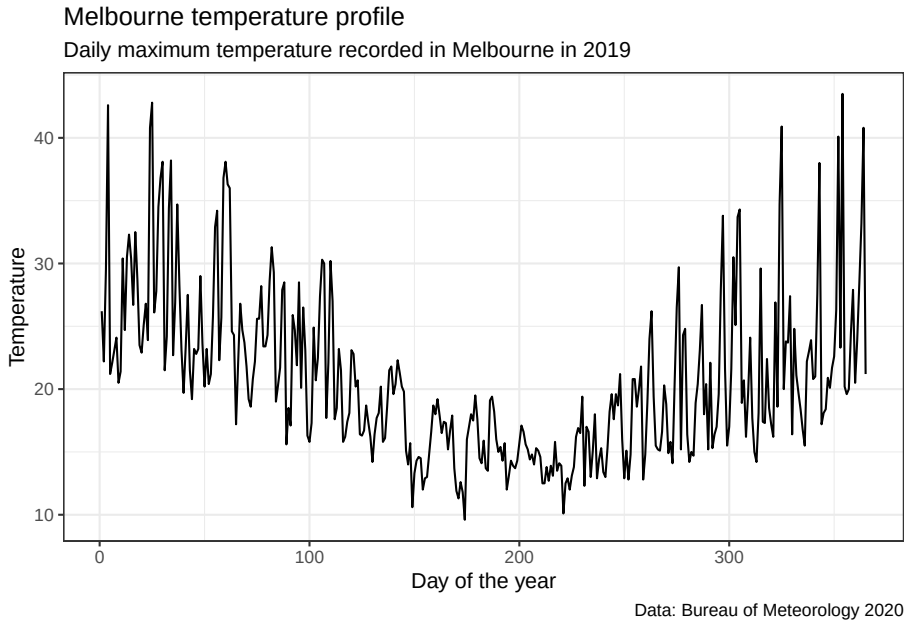
However we can expect that temperature follows a non-linear pattern throughout the year (in Australia it is hot in January-March, cold in June-August, then starts to warm up again). A linear model (e.g. a straight line) will be a very bad model for temperature — we need to introduce polynomials.

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)
```

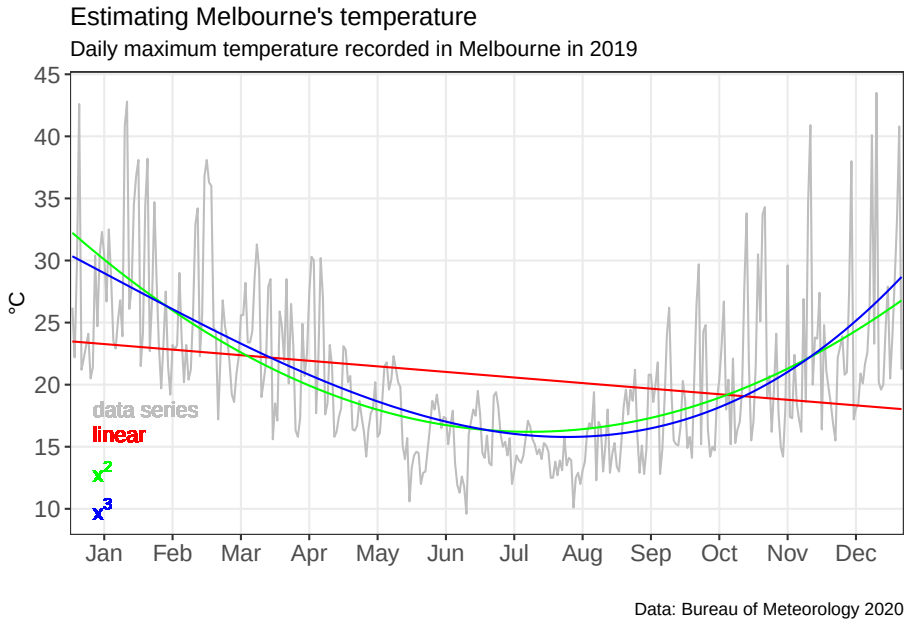
```
summary(poly2)
#Create a polynominal of order 3 to fit this data
poly3 <- lm(Max_temp ~ poly(Day_number,3,raw=TRUE),
         data=MEL_weather_2019)
summary(poly3)

#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
```

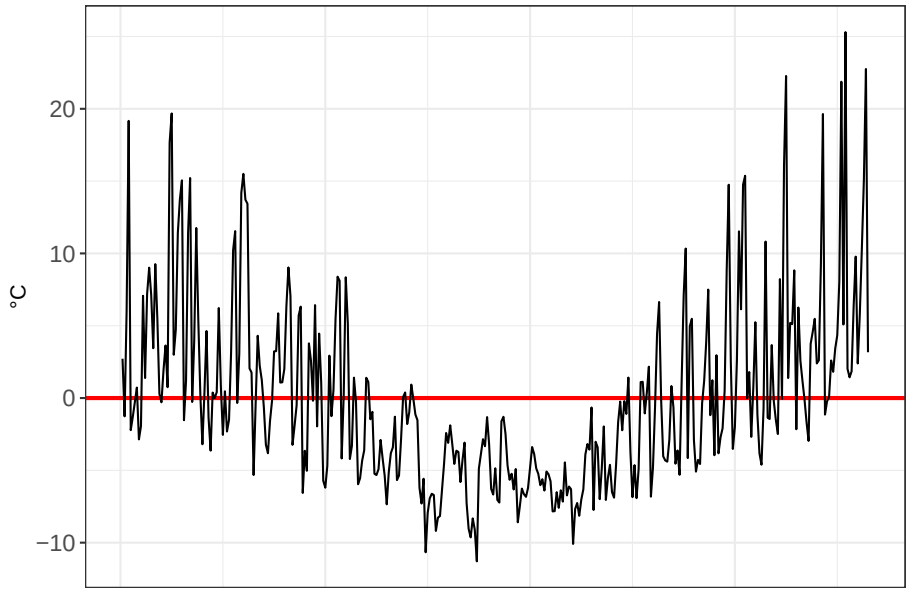


We can see in the chart above the polynomial models do much better at fitting the data. However, they are still highly variant.

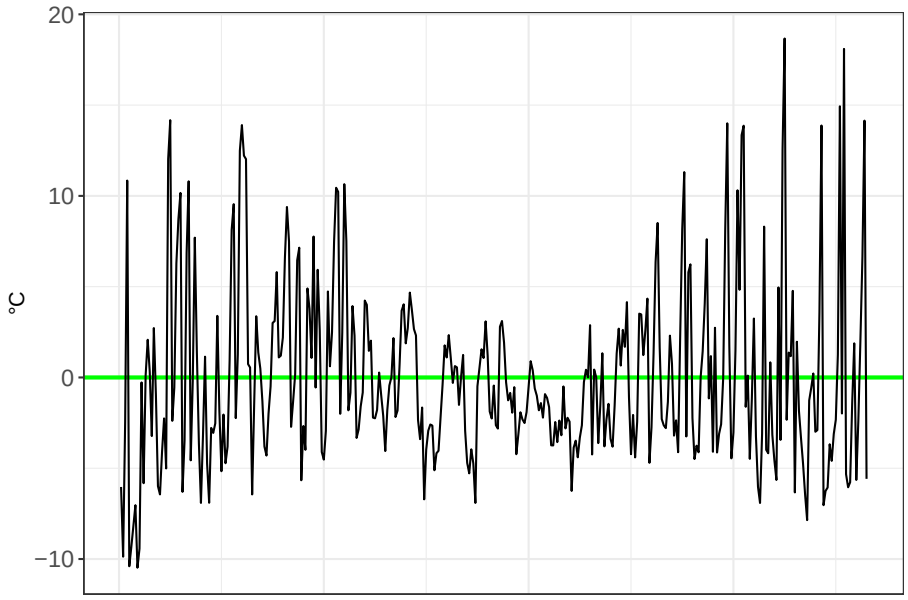
Just how variant are they? We can look at the residuals to find out. The residuals is the gap between the observed data point (i.e. the grey line) and our model.

```
#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")+
```

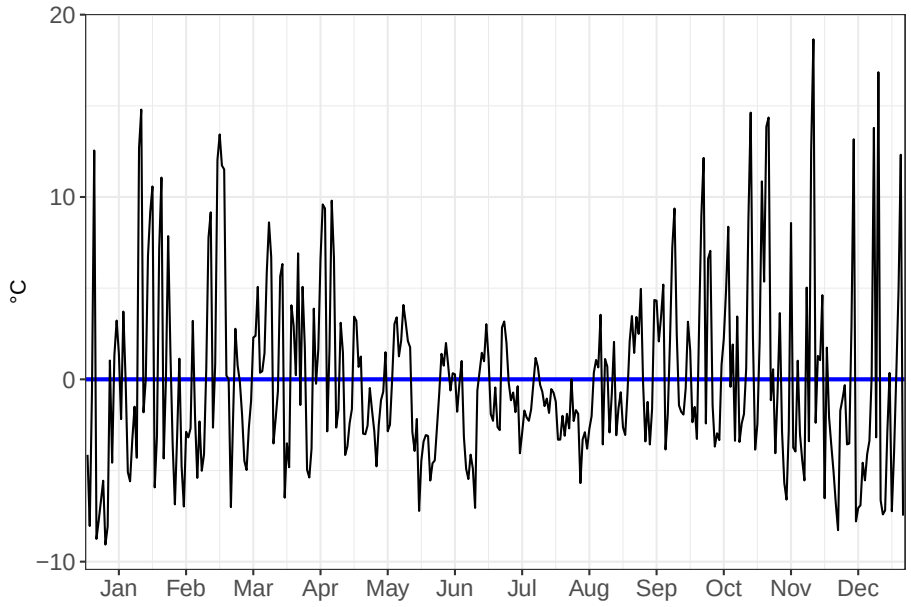
```
  theme_bw()+
  theme(axis.text=element_text(size=12))+
  theme(axis.ticks.x=element_blank(),
  axis.text.x=element_blank())
residuals_poly1_chart
```



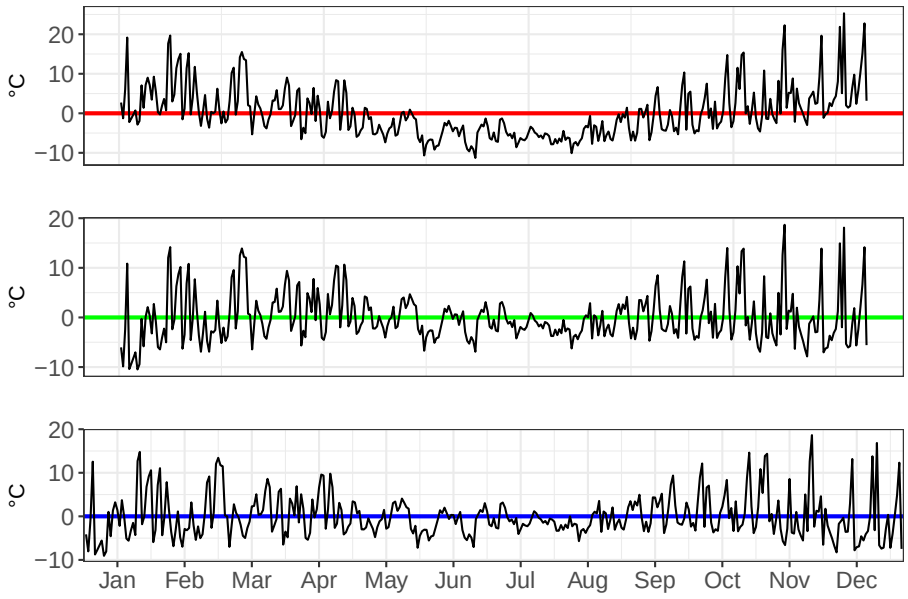

```
#Get the residuals for poly2
residuals_poly2 <- MEL_weather_2019%>%
  add_residuals(poly2)
residuals_poly2_chart <- ggplot(data=residuals_poly2,aes(x=Day_number, y=resid))+
  geom_ref_line(h=0,colour="green", size=1)+
  geom_line()+
  xlab("")+
  ylab("°C")+
  theme_bw()+
  theme(axis.text=element_text(size=12))+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

Working with raster data in R

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.

Getting started

First up we need to load some spatial and data crunching packages.

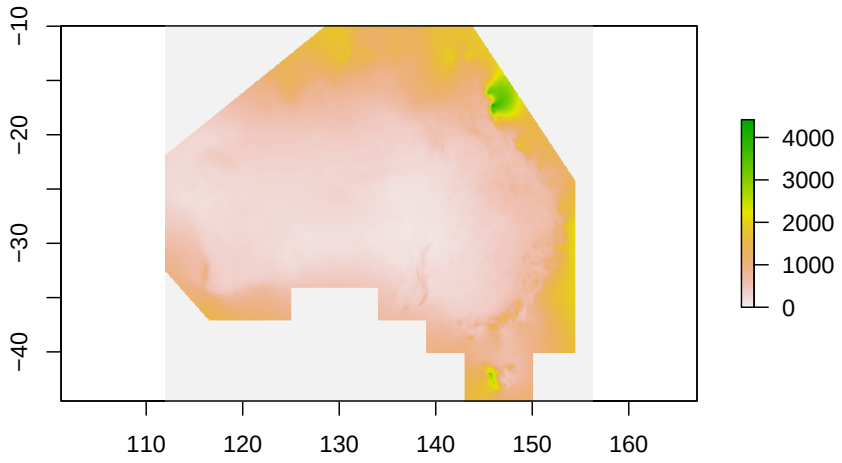
```
#Load packages
library(ncdf4)
library(raster)
library(rgdal)
library(ggplot2)
library(sp)
library(rgdal)
library(absmapsdata)
library(dplyr)
library(rasterVis)
library(RColorBrewer)
library(ggplot2)
library(viridis)
library(sf)
library(plyr)
library(vctrs)
library(tidyr)
library(tmap)
library(ggmap)
library(dplyr)
```

```
library(ggspatial)
library(rlang)
library(broom)
library(tidyverse)
```

Import data

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.

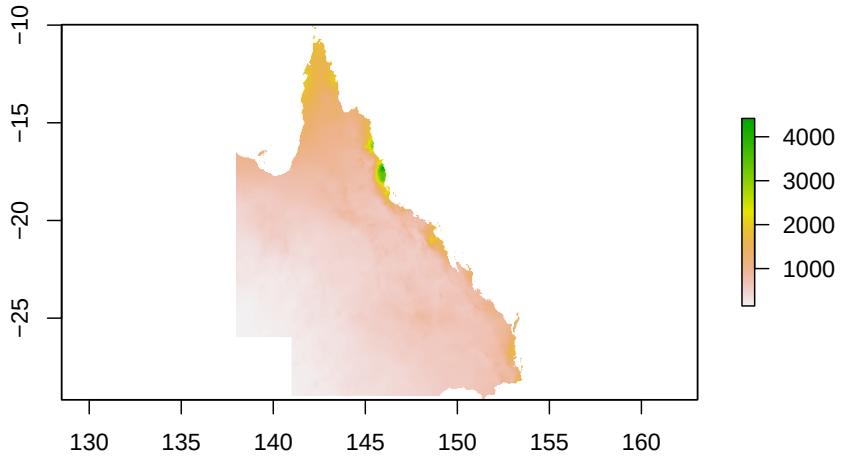
The easiest way to do this is using the absmapsdata package - importing a shapefile of Australia then filtering for only Queensland.

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

Data Wrangling

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.

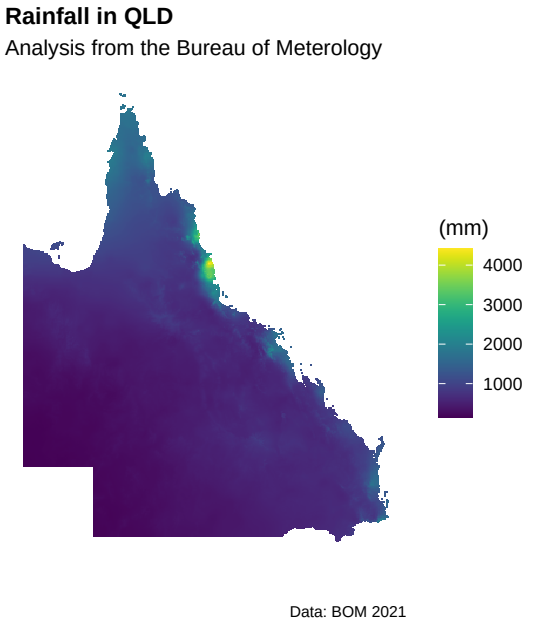
```
#Crop data
r2 <- crop(rainfall,extent(QLD_shape))
r3 <- mask(r2,QLD_shape)
plot(r3)
```



Great. That looks like it worked well. Next up, let's transform the cropped raster (i.e. gridded data) into a data frame (df) 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))
```



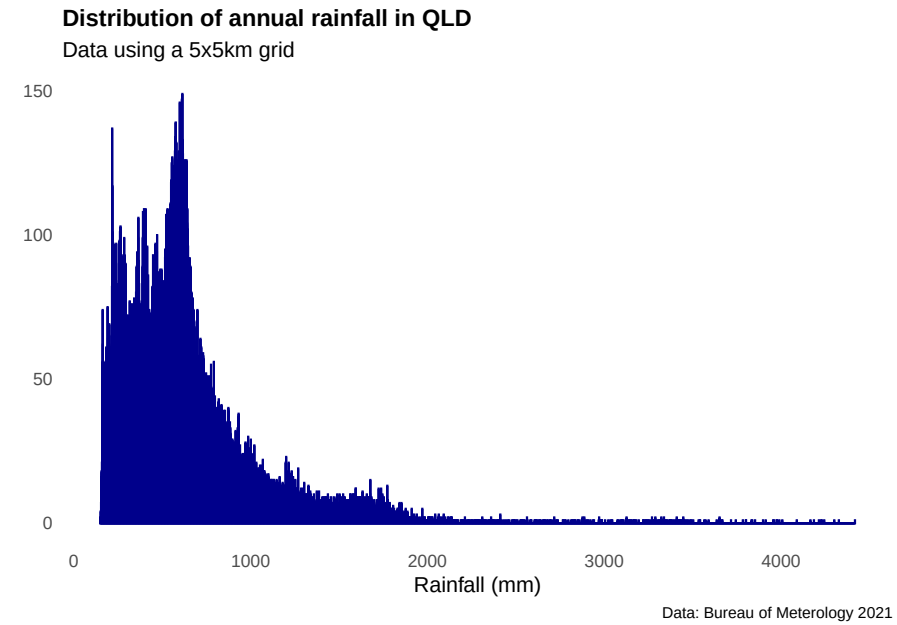
Excellent, we've got a working map of rainfall in Queensland using the ggplot

package. We'll tidy up the map also and add a title and some better colours.

Working 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.

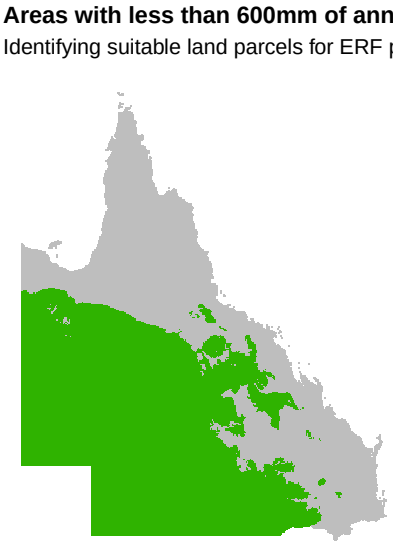
```
ggplot(r3_df) +
    geom_histogram(aes(rainan),binwidth=1,col="darkblue")+
    labs(title="Distribution of annual rainfall in QLD",
                subtitle = "Data using a 5x5km grid",
                caption = "Data: Bureau of Meterology 2021",
                x="Rainfall (mm)",
              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

```
r3_df <- r3_df %>%
          mutate(flag_600mm = ifelse(rainan<=600,1,0))

flagcolours<-(c("grey", "#2FB300"))


ggplot() +
  geom_tile(data=r3_df, aes(x=x, y=y, fill=as.factor(flag_600mm))) +
  scale_fill_manual(values=flagcolours)+
  coord_equal() +
  theme(legend.position="bottom") +
  theme(legend.key.width=unit(1.2, "cm"))+
  labs(title="Areas with less than 600mm of annual rainfall in QLD",
               subtitle = "Identifying suitable land parcels for ERF plantings",
               caption = "Data: Bureau of Meterology 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 = "none")+
            theme(plot.title=element_text(face="bold",size=12))+
            theme(plot.subtitle=element_text(size=11))+
            theme(plot.caption=element_text(size=8))
```




```
#ggsave("C:/06 R_code/ERF_600_mm_rainfall.png",units="cm",dpi=200, device="png")
```

Making an interactive map

The static map above is good... but an interactive visual (ideally with place names below) is better.

To to this we'll need to convert our data frame back to a raster layer, then plot it using the tmap package.

```
# r4_df <- subset(r3_df,select= -c(rainan))
#
# coordinates(r4_df) <- ~ x + y
# gridded(r4_df) <- TRUE
# r5 <- raster(r4_df)
#
# tmap_mode("view")
#       tm_shape(r5)+
#         tm_raster(style="cont",alpha=0.5)
```

Election data in R

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.

Getting started

Let's load up some packages

```
#Load packages
library(ggparliament)
library(eechidna)
library(dplyr)
library(ggplot2)
library(readxl)
library(tidyr)
library(tidyverse)
library(purrr)
library(knitr)
library(broom)
library(absmapsdata)
library(sf)
library(tmap)
```

```
library(rmarkdown)
library(bookdown)
```

Some phenomenal Australia 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 (2019).

- **fp19:** first preference votes for candidates at each electorate
- **tpp19:** two party preferred votes for candidates at each electorate
- **tcp19:** 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 **abs2016** function.

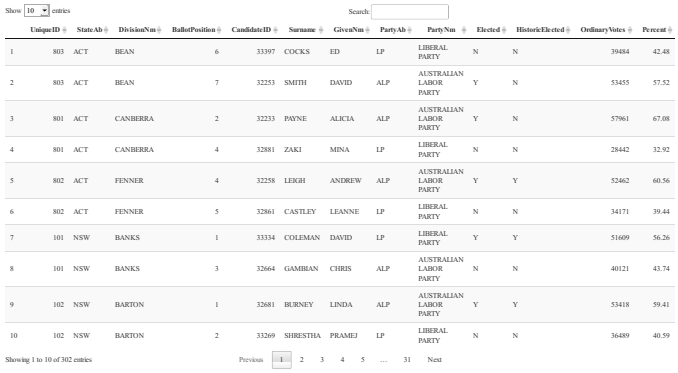
```
data(fp19)
data(tpp19)
data(tcp19)
data(abs2016)

# Show the first few rows
#head(tpp16) %>% kable("simple")
#head(tcp16) %>% kable("simple")
DT::datatable(tpp19)
```

Show 10 entries Search:

	UniqueID	DivisionNm	StateAb	LNP_Votes	LNP_Percent	ALP_Votes	ALP_Percent	TotalVotes	Swing
1	801	CANBERRA	ACT	28442	32.92	57961	67.08	86403	-4.14
2	802	FENNER	ACT	34171	39.44	52462	60.56	86633	1.28
3	101	BANKS	NSW	51609	56.26	40121	43.74	91730	4.82
4	102	BARTON	NSW	36489	40.59	53418	59.41	89907	-1.11
5	103	BENNELONG	NSW	54809	56.91	41496	43.09	96305	-2.81
6	104	BEROWRA	NSW	61675	65.65	32272	34.35	93947	-0.8
7	105	BLAXLAND	NSW	28509	35.28	52299	64.72	80808	4.76
8	106	BRADFIELD	NSW	63997	66.56	32146	33.44	96143	-4.48
9	107	CALARE	NSW	66006	63.29	38282	36.71	104288	1.48
10	108	CHIFLEY	NSW	34245	37.63	56761	62.37	91006	6.82

Showing 1 to 10 of 151 entries Previous 1 2 3 4 5 … 16 Next

```
DT::datatable(tcp19)
```



Show 10 entries Search:

	UniqueID	StateAb	DivisionNm	BallotPosition	CandidateID	Surname	GivenNm	PartyAb	PartyNm	Elected	HistoricElected	OrdinaryVotes	Percent
1	803	ACT	BEAN	6	33397	COCKS	ED	LP	LIBERAL PARTY	N	N	39484	42.48
2	803	ACT	BEAN	7	32253	SMITH	DAVID	ALP	AUSTRALIAN LABOR PARTY	Y	N	53455	57.52
3	801	ACT	CANBERRA	2	32233	PAYNE	ALICIA	ALP	AUSTRALIAN LABOR PARTY	Y	N	57961	67.08
4	801	ACT	CANBERRA	4	32881	ZAKI	MINA	LP	LIBERAL PARTY	N	N	28442	32.92
5	802	ACT	FENNER	4	32258	LEIGH	ANDREW	ALP	AUSTRALIAN LABOR PARTY	Y	Y	52462	60.56
6	802	ACT	FENNER	5	32861	CASTLEY	LEANNE	LP	LIBERAL PARTY	N	N	34171	39.44
7	101	NSW	BANKS	1	33334	COLEMAN	DAVID	LP	LIBERAL PARTY	Y	Y	51609	56.26
8	101	NSW	BANKS	3	32664	GAMBIAN	CHRIS	ALP	AUSTRALIAN LABOR PARTY	N	N	40121	43.74
9	102	NSW	BARTON	1	32681	BURNEY	LINDA	ALP	AUSTRALIAN LABOR PARTY	Y	Y	53418	59.41
10	102	NSW	BARTON	2	33269	SHRESTHA	PRAMEJ	LP	LIBERAL PARTY	N	N	36489	40.59

Showing 1 to 10 of 302 entries Previous 1 2 3 4 5 … 31 Next

Working with 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**
New South Wales	47
Victoria	39
Queensland	30
Western Australia	15
South Australia	10
Tasmania	5

State/Territory	Number of members of the House of Representatives
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).

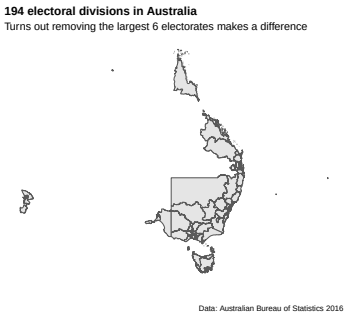
How variant are these 151 electorates in size?

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 <- ced2018 %>%
           ggplot()+
           geom_sf()+
           labs(title="Electoral divisions in Australia",
               subtitle = "It turns out we divide the country in very non-standard blo
               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 <- ced2018 %>%
                    dplyr::filter(!ced_code_2018 %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 differ
               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
```



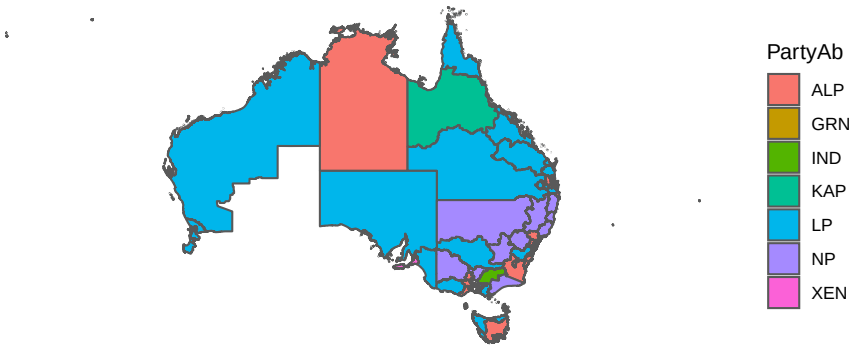
Next let's look at what party/candidate is currently the sitting member for each electorate. To do this on a map we're going to need to join our tcp19 data and the ced2018 spatial data.

In the first data set, the electorate column in called 'DivisionNm' and in the second 'ced_name_2018.'

We see the data in our DivisionNm variable is in UPPERCASE while our ced_name_2018 variable is in Titlecase. Let's change the first variable to Titlecase. We can then make the column names the same, and run our left_join function.

```
#Pull in the electorate shapefiles from the absmapsdata package
electorates <- ced2018

#Make the DivisionNm Titlecase
tcp19$DivisionNm=str_to_title(tcp19$DivisionNm)

tcp19_edit <- tcp19 %>% distinct() %>% filter(Elected == "Y")

#Make the column names the same
electorates <- dplyr::rename(electorates, DivisionNm = ced_name_2018)

ced_map_data <- left_join(tcp19_edit, electorates, by = "DivisionNm")
```

```
ced_map_data <- as.data.frame(ced_map_data)

head(ced_map_data)
str(ced_map_data)

ggplot()+
  geom_sf(data=ced_map_data,aes(geometry = geometry,fill=PartyAb)) +
  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))+
            scale_color_manual("PartyAb", values=c("LP" ="#80b1d3",
                                                   "NP" = "#006400",
                                                   "ALP"= "#fb8072",
                                                   "GVIC" = "#33a02c",
                                                   "XEN" = "#beaed4",
                                                   "ON" = "#fdc086",
                                                   "KAP" = "#ffff99",
                                                   "IND" = "grey25"))
```



Answering election questions

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).

```
who_won <- tcp19 %>%
  filter(Elected == "Y") %>%
  group_by(PartyNm) %>%
  tally() %>%
  arrange(desc(n))

# inspect
who_won %>% kable("simple")
```

PartyNm	n
AUSTRALIAN LABOR PARTY	68
LIBERAL PARTY	67
NATIONAL PARTY	10
INDEPENDENT	3
CENTRE ALLIANCE	1
KATTER'S AUSTRALIAN PARTY (KAP)	1
THE GREENS	1

Next up let's see which candidates won with the smallest percentage of votes

```
who_won_least_votes_prop <- fp16 %>%
  filter(Elected == "Y") %>%
  arrange(Percent) %>%
  mutate(candidate_full_name = paste0(GivenNm, " ", Surname, " (", CandidateID, ")")) %>%
  dplyr::select(candidate_full_name, PartyNm, DivisionNm, Percent)

who_won_least_votes_prop %>% head %>% kable("simple")
```

candidate_full_name	PartyNm	DivisionNm	Percent
MICHAEL DANBY (28267)	AUSTRALIAN LABOR PARTY	MELBOURNE PORTS	27.00
CATHY O'TOOLE (28997)	AUSTRALIAN LABOR PARTY	HERBERT	30.45
JUSTINE ELLIOT (28987)	AUSTRALIAN LABOR PARTY	RICHMOND	31.05
TERRI BUTLER (28921)	AUSTRALIAN LABOR PARTY	GRIFFITH	33.18
STEVE GEORGANAS (29071)	AUSTRALIAN LABOR PARTY	HINDMARSH	34.02
CATHY MCGOWAN (23288)	INDEPENDENT	INDI	34.76

This is really something. The relationship we're seeing here seems to be these are the seats in which the ALP relies heavily on preference flows from the Greens or Independents to win. The electorate I grew up in is listed here (Richmond) - let's look at how the votes were allocated.

```
Richmond_fp <- fp16 %>%
  filter(DivisionNm == "RICHMOND") %>%
  arrange(-Percent) %>%
  mutate(candidate_full_name = paste0(GivenNm, " ", Surname, " (", CandidateID, ")")) %
  dplyr::select(candidate_full_name, PartyNm, DivisionNm, Percent, OrdinaryVotes)

Richmond_fp %>% knitr::kable("simple")
```

candidate_full_name	PartyNm	DivisionNm	Percent
MATTHEW FRASER (29295)	NATIONAL PARTY	RICHMOND	37.61
JUSTINE ELLIOT (28987)	AUSTRALIAN LABOR PARTY	RICHMOND	31.05
DAWN WALKER (28783)	THE GREENS	RICHMOND	20.44
NEIL GORDON SMITH (28349)	ONE NATION	RICHMOND	6.26
ANGELA POLLARD (29290)	ANIMAL JUSTICE PARTY	RICHMOND	3.14
RUSSELL KILARNEY (28785)	CHRISTIAN DEMOCRATIC PARTY	RICHMOND	1.51

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

The interpretation that these seats are the most marginal is incorrect (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 <- tcp16 %>%
  filter(Elected == "Y") %>%
  mutate(percent_margin = 2*(Percent - 50), vote_margin = round(percent_margin * Ordina
  arrange(Percent) %>%
  mutate(candidate_full_name = paste0(GivenNm, " ", Surname, " (", CandidateID, ")")) %
  dplyr::select(candidate_full_name, PartyNm, DivisionNm, Percent, OrdinaryVotes, perce

# have a look
who_won_smallest_margin %>%
 head %>%
 knitr::kable("simple")
```

candidate_full_name	PartyNm	DivisionNm	Percent	Ord
CATHY O'TOOLE (28997)	AUSTRALIAN LABOR PARTY	HERBERT	50.02	

candidate_full_name	PartyNm	DivisionNm	Percent	OrdinaryVotes
STEVE GEORGANAS (29071)	AUSTRALIAN LABOR PARTY	HINDMARSH	50.58	49586
MICHELLE LANDRY (28034)	LIBERAL PARTY	CAPRICORNIA	50.63	44633
BERT VAN MANEN (28039)	LIBERAL PARTY	FORDE	50.63	42486
ANNE ALY (28727)	AUSTRALIAN LABOR PARTY	COWAN	50.68	41301
ANN SUDMALIS (28668)	LIBERAL PARTY	GILMORE	50.73	52336

Crikey. We see Cathy O'Toole got in with a 0.04% margin (just 35 votes!)

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

```
who_won_largest_margin <- tcp16 %>%
  filter(Elected == "Y") %>%
  mutate(percent_margin = 2*(Percent - 50), vote_margin = round(percent_margin * OrdinaryVotes /
  arrange(desc(Percent)) %>%
  mutate(candidate_full_name = paste0(GivenNm, " ", Surname, " (", CandidateID, ")")) %>%
  dplyr::select(candidate_full_name, PartyNm, DivisionNm, Percent, OrdinaryVotes, percent_margin,

# Look at the data
 who_won_largest_margin %>%
 head %>%
 knitr::kable("simple")
```

candidate_full_name	PartyNm	DivisionNm	Percent	OrdinaryVotes
ANDREW BROAD (28415)	NATIONAL PARTY	MALLEE	71.32	62383
PAUL FLETCHER (28565)	LIBERAL PARTY	BRADFIELD	71.04	66513
JULIE BISHOP (28746)	LIBERAL PARTY	CURTIN	70.70	60631
SUSSAN LEY (28699)	LIBERAL PARTY	FARRER	70.53	68114
JASON CLARE (28931)	AUSTRALIAN LABOR PARTY	BLAXLAND	69.48	55507
BRENDAN O'CONNOR (28274)	AUSTRALIAN LABOR PARTY	GORTON	69.45	68135

Wowza. That's really something. Some candidates won seats with a 30-40 percent margin - scooping up 70% of the two candidate preferred vote in the process!

```
who_won <- tcp16 %>%
  filter(Elected == "Y") %>%
  group_by(PartyNm, StateAb) %>%
  tally() %>%
  arrange(desc(n))
```

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

#View data set
who_won_by_state %>%
knitr::kable("simple")
```

Demographic analysis of voting trends

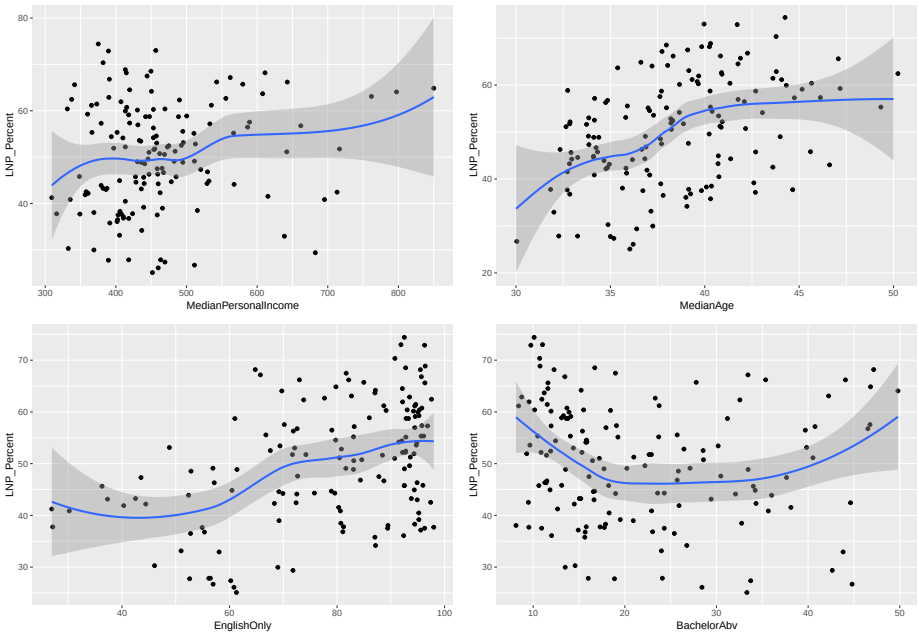
Now we've figured out how to work with election data - let's link it up to some AUstralian demographic data. The eechidna package includes a cleaned set of census data from 2016 that has already been adjusted from ASGS boundaries to Commonwealth Electoral Divisions.

```
# Import the census data from the eechidna package
data(eechidna::abs2016)
head(abs2016)

# Join with two-party preferred voting data
data(tpp10)
election2016 <- left_join(abs2016, tpp10, by = "DivisionNm")
```

That's what we want to see. 150 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(election2016, aes(x = MedianPersonalIncome, y = LNP_Percent)) + geom_point() + g
ggplot(election2016, aes(x = MedianAge, y = LNP_Percent)) + geom_jitter() + geom_smoothl
ggplot(election2016, aes(x = EnglishOnly, y = LNP_Percent)) + geom_jitter() + geom_smoo
ggplot(election2016, aes(x = BachelorAbv, y = LNP_Percent)) + geom_jitter() + geom_smoo
```



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 abs2016 data set against the LNP_percent variable.

```
# We can use colnames(election2016) 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+
```

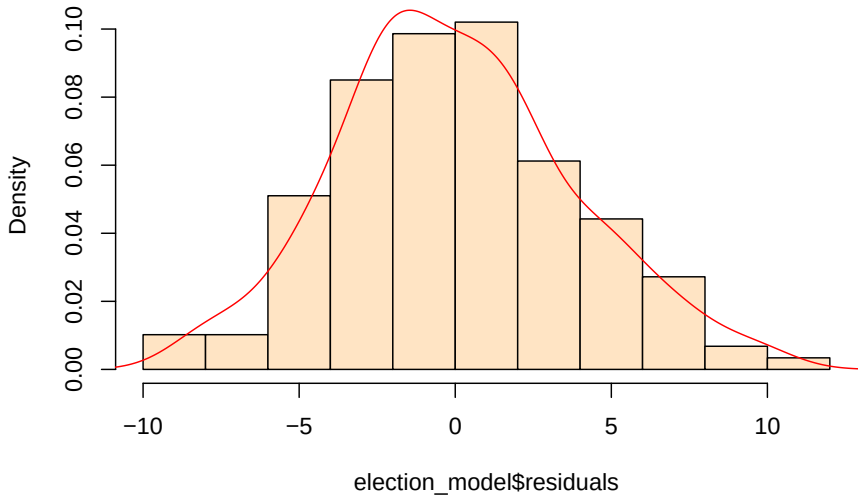
```
                      BachelorAbv+Born_Asia+
                      Born_MidEast+Born_SE_Europe+
                      Born_UK+
                      BornElsewhere+
                      Buddhism+
                      Catholic+
                      Christianity+
                      Couple_NoChild_House+Couple_WChild_House+
                      CurrentlyStudying+DeFacto+
                      DiffAddress+
                      DipCert+
                      EnglishOnly+
                      FamilyRatio+
                      Finance+
                      HighSchool+
                      Indigenous+
                      InternetAccess+
                      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=election2016)

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 + InternetAccess +
##     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 = election2016)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.6197 -2.5288 -0.2903  2.2118 10.0752
##
## Coefficients: (1 not defined because of singularities)
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -7.602e+03  1.198e+04  -0.634  0.52753
## Population             4.224e-05  5.045e-05   0.837  0.40468
## Area                   6.359e-06  5.436e-06   1.170  0.24527
## Age00_04               7.063e+01  1.193e+02   0.592  0.55554
## Age05_14               7.280e+01  1.194e+02   0.609  0.54380
## Age15_19               7.215e+01  1.193e+02   0.605  0.54686
## Age20_24               6.383e+01  1.198e+02   0.533  0.59559
## Age25_34               7.265e+01  1.196e+02   0.607  0.54522
## Age35_44               6.830e+01  1.195e+02   0.572  0.56912
## Age45_54               6.964e+01  1.196e+02   0.582  0.56192
## Age55_64               7.224e+01  1.194e+02   0.605  0.54677
## Age65_74               7.915e+01  1.197e+02   0.661  0.51017
## Age75_84               7.583e+01  1.193e+02   0.635  0.52682
## Age85plus              6.974e+01  1.197e+02   0.582  0.56178
## Anglican               2.912e-01  3.954e-01   0.737  0.46338
## AusCitizen             1.679e-01  7.769e-01   0.216  0.82938
## AverageHouseholdSize  -2.683e+01  1.673e+01  -1.604  0.11241
## BachelorAbv           -3.045e+00  1.003e+00  -3.035  0.00318 **
## Born_Asia             -3.451e-01  3.842e-01  -0.898  0.37151
## Born_MidEast           8.345e-01  1.256e+00   0.664  0.50832
## Born_SE_Europe        -2.007e+00  1.479e+00  -1.357  0.17825
## Born_UK                6.477e-02  4.615e-01   0.140  0.88872
## BornElsewhere          6.542e-01  6.265e-01   1.044  0.29933
```

```
## Buddhism               6.375e-01  8.253e-01   0.772  0.44196
## Catholic              -3.970e-01  3.754e-01  -1.058  0.29322
## Christianity           1.112e+00  6.283e-01   1.769  0.08038 .
## Couple_NoChild_House   3.345e+00  3.078e+00   1.087  0.28021
## Couple_WChild_House    3.762e+00  3.156e+00   1.192  0.23661
## CurrentlyStudying      2.597e+00  1.303e+00   1.993  0.04939 *
## DeFacto               -7.035e+00  2.617e+00  -2.688  0.00862 **
## DiffAddress            9.320e-01  3.877e-01   2.404  0.01836 *
## DipCert               -8.790e-01  7.296e-01  -1.205  0.23163
## EnglishOnly           -1.316e-01  4.717e-01  -0.279  0.78095
## FamilyRatio            1.962e+01  4.994e+01   0.393  0.69543
## Finance                1.528e+00  9.060e-01   1.687  0.09523 .
## HighSchool             9.371e-01  4.683e-01   2.001  0.04854 *
## Indigenous             1.054e+00  4.781e-01   2.205  0.03013 *
## InternetAccess        -9.364e-01  9.367e-01  -1.000  0.32028
## InternetUse                   NA         NA      NA       NA
## Islam                  2.894e-01  6.407e-01   0.452  0.65258
## Judaism                7.306e-01  7.474e-01   0.978  0.33105
## Laborer               -2.925e-02  7.905e-01  -0.037  0.97057
## LFParticipation        2.926e+00  8.800e-01   3.325  0.00130 **
## Married               -4.168e+00  1.890e+00  -2.205  0.03011 *
## MedianAge             -7.214e-01  1.071e+00  -0.674  0.50218
## MedianFamilyIncome     2.146e-02  3.339e-02   0.643  0.52204
## MedianHouseholdIncome  2.879e-02  3.037e-02   0.948  0.34584
## MedianLoanPay         -9.533e-03  1.336e-02  -0.713  0.47757
## MedianPersonalIncome  -2.286e-02  5.907e-02  -0.387  0.69973
## MedianRent            -4.070e-02  5.306e-02  -0.767  0.44514
## Mortgage               1.923e+00  1.741e+00   1.104  0.27264
## NoReligion             1.285e+00  6.365e-01   2.019  0.04658 *
## OneParent_House        2.164e-01  2.987e+00   0.072  0.94241
## Owned                  1.488e+00  1.591e+00   0.935  0.35229
## Professional           7.449e-01  1.010e+00   0.737  0.46300
## PublicHousing         -5.464e-01  6.280e-01  -0.870  0.38670
## Renting                2.068e+00  1.736e+00   1.192  0.23664
## SocialServ            -3.356e-01  6.232e-01  -0.539  0.59155
## SP_House              -1.034e+00  8.907e-01  -1.161  0.24895
## Tradesperson           6.347e-01  8.058e-01   0.788  0.43305
## Unemployed             2.815e-01  1.176e+00   0.239  0.81149
## Volunteer              7.346e-01  6.179e-01   1.189  0.23772
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.933 on 86 degrees of freedom
##   (3 observations deleted due to missingness)
## Multiple R-squared:  0.8911, Adjusted R-squared:  0.8151
## F-statistic: 11.73 on 60 and 86 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")
```

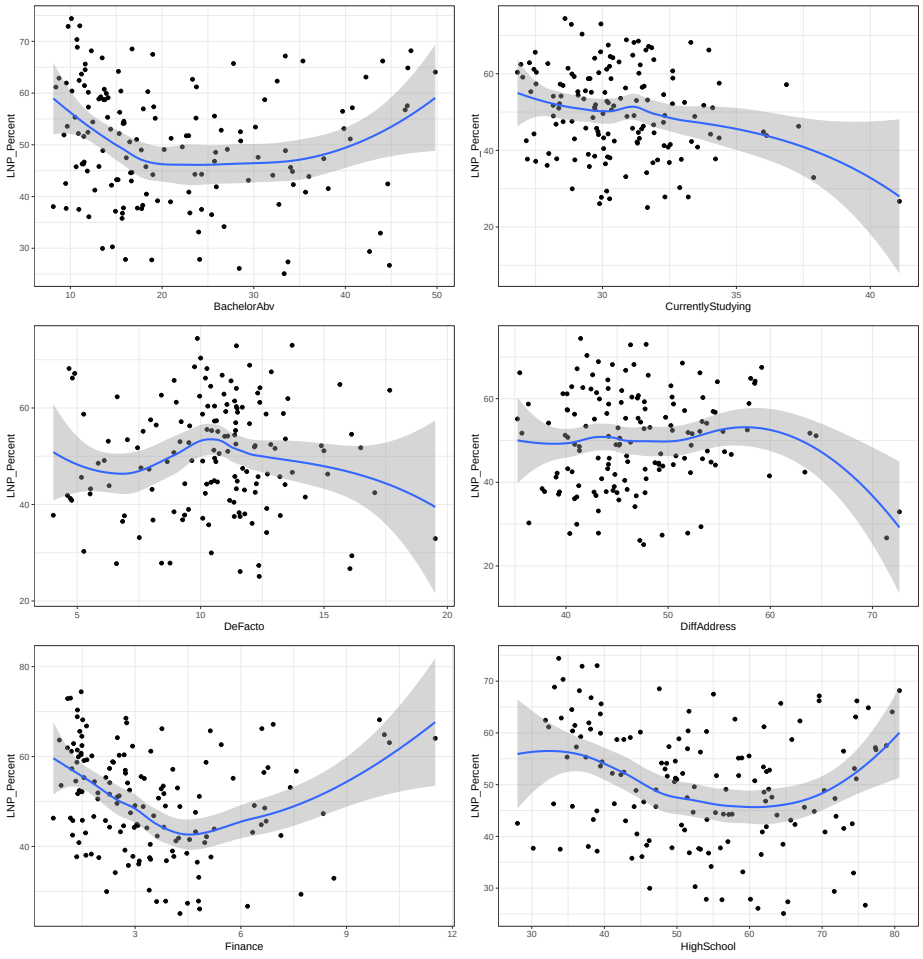


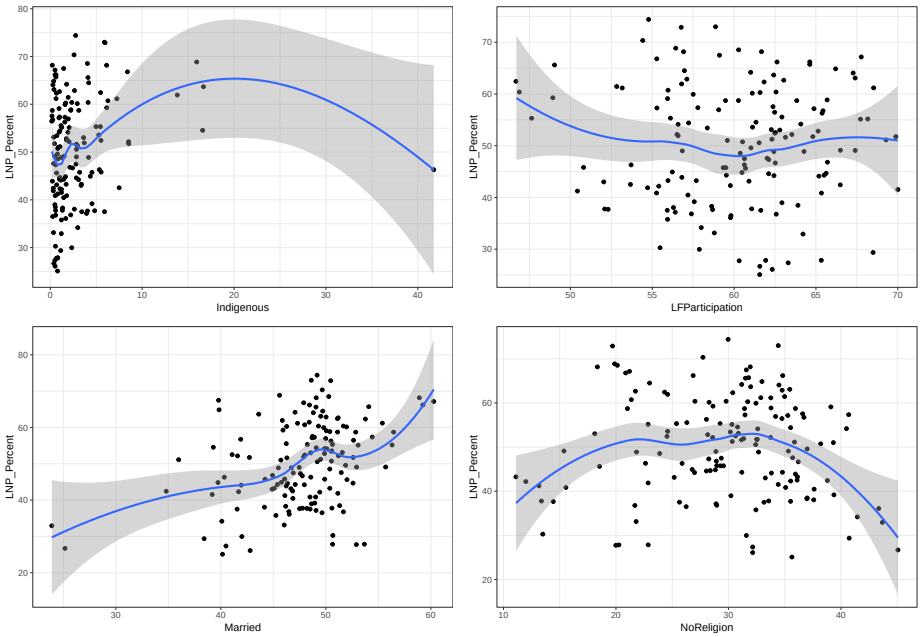
Hmm... that's actually not too bad. Onwards.

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=election2016)

summary(election_model_lean)
```

```
ggplot(election2016, aes(x = BachelorAbv, y = LNP_Percent)) + geom_point() + geom_smoot
ggplot(election2016, aes(x = CurrentlyStudying, y = LNP_Percent)) + geom_jitter() + geo
ggplot(election2016, aes(x = DeFacto, y = LNP_Percent)) + geom_jitter() + geom_smooth()
ggplot(election2016, aes(x = DiffAddress, y = LNP_Percent)) + geom_jitter() + geom_smoo
ggplot(election2016, aes(x = Finance, y = LNP_Percent)) + geom_jitter() + geom_smooth()
ggplot(election2016, aes(x = HighSchool, y = LNP_Percent)) + geom_jitter() + geom_smoot
ggplot(election2016, aes(x = Indigenous, y = LNP_Percent)) + geom_jitter() + geom_smoot
ggplot(election2016, aes(x = LFParticipation, y = LNP_Percent)) + geom_jitter() + geom_
ggplot(election2016, aes(x = Married, y = LNP_Percent)) + geom_jitter() + geom_smooth()
ggplot(election2016, aes(x = NoReligion, y = LNP_Percent)) + geom_jitter() + geom_smoot
```





My main gripe with the above is that electorates are very different in size. Therefore trying to conclude any statistical relationship on an electorate level is prone to errors. Adding more data isn't always the best method to solve what's formally known as the Modifiable Area Unit Problem... but in this case it's worth a try.

So here goes, let's run the analysis above, this time using all 7,000 voting booths (and their local demographic data) as the data set rather than just the 150 electorates.

Mapping booths

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.

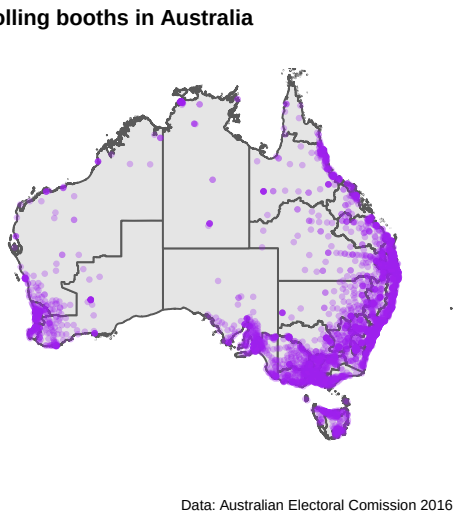
Show 10 entries Search:

	State	DivisionID	DivisionNm	PollingPlaceID	PollingPlaceTypeID	PollingPlaceNm	PremisesNm	PremisesAddress1	PremisesAddress2	PremisesAddress3	PremisesSuburb	PremisesStateAb	PremisesPostCode	Latitude	Longitude
1	ACT	101	Canberra	8629	1	Barton	Telopea Park School	New South Wales Cres			BARTON	ACT	2600	-35.3151	149.135
2	ACT	101	Canberra	64583	5	Belconnen CANBERRA PPVC	Belconnen Community Centre	26 Chandler St			BELCONNEN	ACT	2900	-35.2189038	149.070109
3	ACT	101	Canberra	65504	5	BLV Canberra PPVC	BLV Canberra	50 Marcus Clarke St			CANBERRA CITY	ACT	2601	-35.2790544	149.1262791
4	ACT	101	Canberra	11877	1	Bonython	Bonython Primary School	Hurtle Ave			BONYTHON	ACT	2905	-35.4318	149.083
5	ACT	101	Canberra	8802	1	Braddon (Canberra)	Ainslie School	Donaldson St			BRADDON	ACT	2601	-35.27363	149.14001
6	ACT	101	Canberra	11452	1	Calwell	Calwell High School	Casey Cres			CALWELL	ACT	2905	-35.44067	149.1176
7	ACT	101	Canberra	8806	1	Campbell	Campbell Primary School	Chauvel St			CAMPBELL	ACT	2601	-35.2903	149.156
8	ACT	101	Canberra	8761	1	Chapman	Chapman Primary School	Perry Dr			CHAPMAN	ACT	2611	-35.3564	149.042
9	ACT	101	Canberra	8763	1	Chifley	Caroline Chisholm School	Hambidge Cres			CHIFLEY	ACT	2905	-35.4187478	149.1225386
10	ACT	101	Canberra	8808	1	City (Canberra)	Pilgrim House	69 Northbourne Ave			CANBERRA CITY	ACT	2601	-35.2767018	149.1290814

Showing 1 to 10 of 8,329 entries Previous 1 2 3 4 5 ... 833 Next

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=ced2018)+
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 Comission 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)) +
xlim(c(112,157)) + ylim(c(-44,-11))
```



Exploring booth level data

Figuring out where a candidates votes come from ***within*** an electorate is fundamental 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. They nmust 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).

```
## Error in `geom_map()`:
## ! `map` must have the columns `x`, `y`, and `id`
```

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.

```
## Error in `geom_map()`:
## ! `map` must have the columns `x`, `y`, and `id`
```

Donkeys, dicks, and other informalities

We're about to go off the deep end into a certain type of election data.

In the 2016 Australian Federal Election, over 720,915 people (5.5% of all votes cast) voted informally. Of these, over half (377,585) had '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

This is the good bit:

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. What's interesting is you can track the 'vibe' of how consequential an election is by the proportion of protest votes.

Let's pull some informal voting data from the AEC website.

Charts

Getting started

There's exceptional resources online for using the `ggplot2` package 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 in packages
library(ggridges)
library(ggplot2)
library(ggrepel)
library(viridis)
library(readxl)
library(hrbrthemes)
library(dplyr)
library(stringr)
library(reshape)
library(tidyr)
library(lubridate)
library(gapminder)
library(ggalt)
library(purrr)
library(scales)
library(purrr)
library(patchwork)
#library(bbplot)
```

Make the data tidy

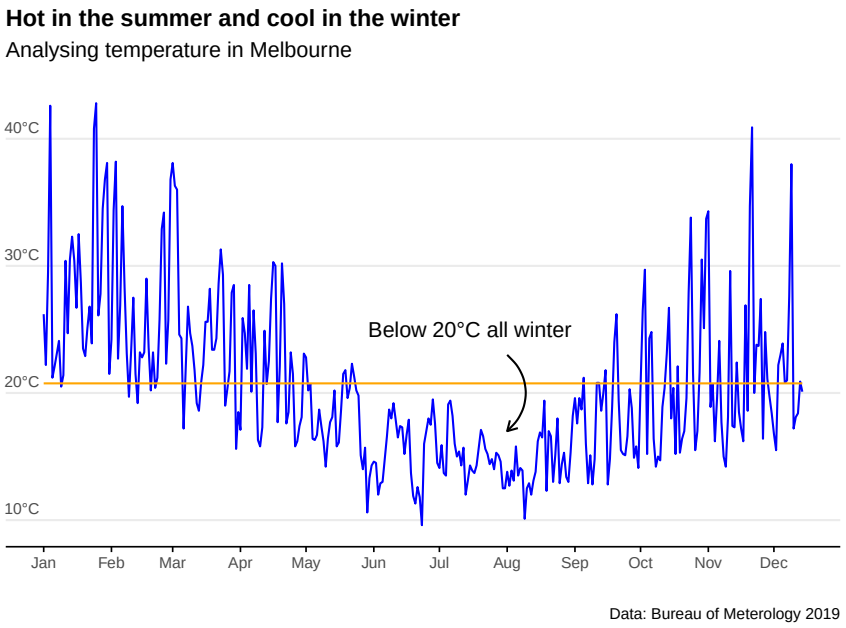
Before making a chart ensure the data is "tidy" - meaning there is a new row for every changed variable. It also doesn't hurt to remove NA's for consistency (particularly in time series).

```
#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)+
                          geom_line(aes(x = Date, y = Temp), col = "blue")+
                          geom_line(aes(x = Date, y = Mean), col = "orange")+
                          labs(title="Hot in the summer and cool in the winter",
                          subtitle = "Analysing temperature in Melbourne",
                          caption = "Data: Bureau of Meterology 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(label = unit_format(unit="\u00b0C", sep="")) +

                          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 = margin(t = 0, r = 0, b = 0, l = 0)))+

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

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

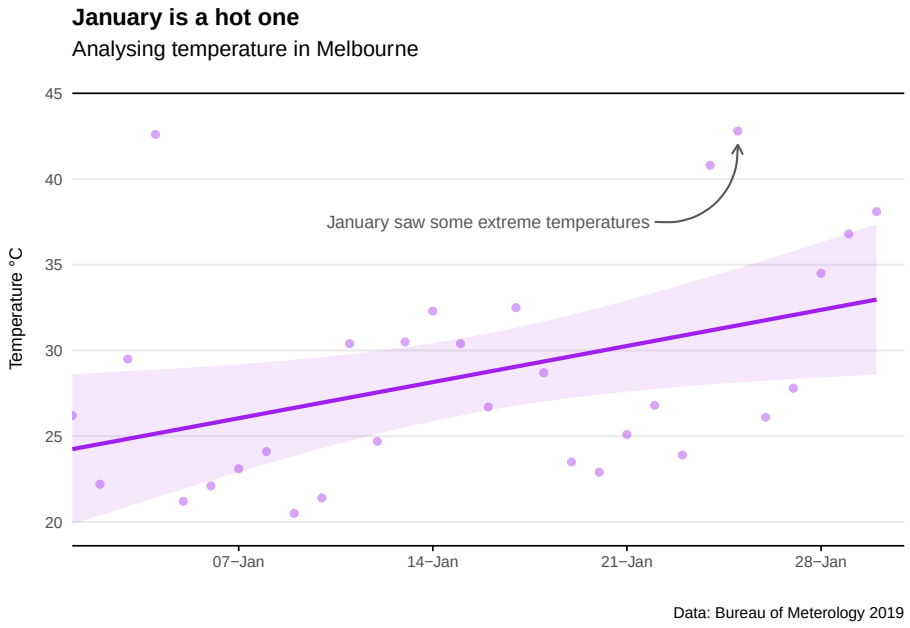
```
plot_MEL_temp
```



Scatter and trend plot

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

ggplot(MEL_temp_Jan)+
  geom_point(aes(x = Date, y = Temp), col = "purple",alpha=0.4)+
  geom_smooth(aes(x = Date, y = Temp), col = "purple",fill="purple", alpha=0.1,method =
  #scale_colour_manual(values = c("purple"),labels="Trend") +
  #scale_fill_manual(values = c("purple"),labels="Confidence interval")+
  labs(title="January is a hot one",
  subtitle = "Analysing temperature in Melbourne",
  caption = "Data: Bureau of Meterology 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)) +

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=8))+
theme(axis.title.y = element_text(size=9,margin=margin(t = 0, r = 10, b = 0, l = 0)))+
theme(panel.grid.minor = element_blank())+
theme(panel.grid.major.x = element_blank()) +

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

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")
```



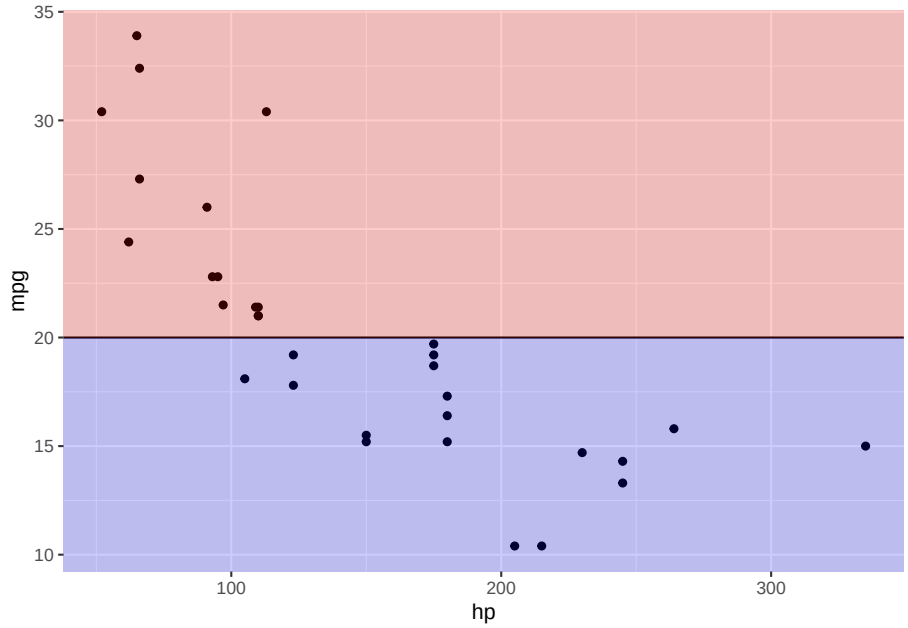
Shading areas on plots

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.

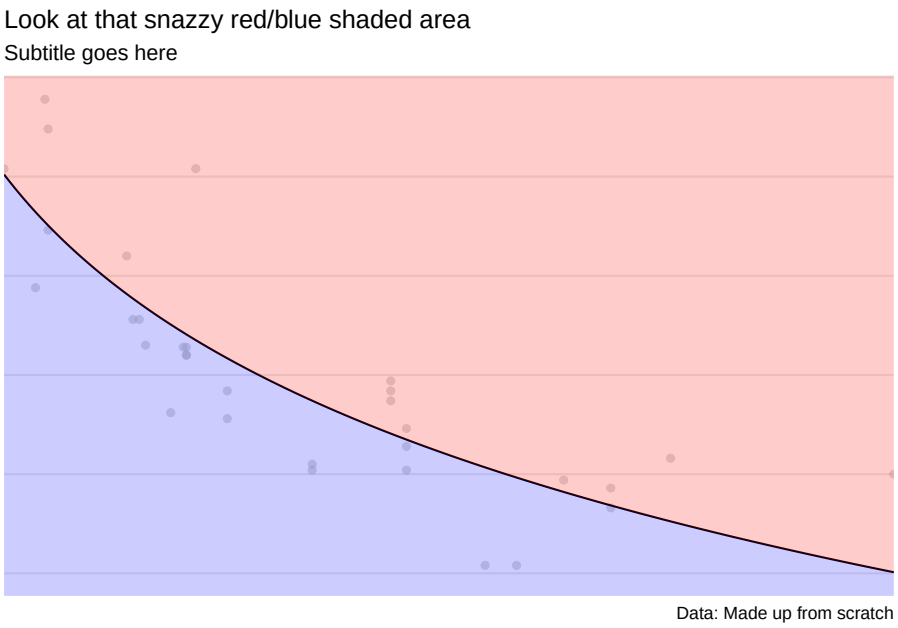
```
#Example 1
#Set a custom limit for the y-axis threshold
threshold <- 20

ggplot() +
  geom_point(data = mtcars,
             aes(x = hp, y = mpg)) +

  #Add an intercept line
  geom_hline(yintercept = threshold) +

  # Shade area under y_lim
  geom_rect(aes(xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = threshold),
            alpha = 1/5,
            fill = "blue") +

  # Shade area above y_lim
```

```
  geom_rect(aes(xmin = -Inf, xmax = Inf, ymin = threshold, ymax = Inf),
            alpha = 1/5,
            fill = "red")
```




```
#Example 2
#Now to add colour either way of a model line

#Define the model
model <- lm(mpg ~ log(hp), data = mtcars)

# Get the predictions for plotting. Here, df_line, is a data frame with new
# coordinates that will be used for plotting the trend line and further for
# building the polygons for shading.
min_x <- min(mtcars$hp)
max_x <- max(mtcars$hp)

df_line <- data.frame(hp = seq(from = min_x, to = max_x, by = 1))
df_line$mpg <- predict(model, newdata = df_line)

  p <- ggplot() +
  geom_point(data = mtcars,
             aes(x = hp, y = mpg),color="grey",alpha=0.5) +

  geom_line(data = df_line,
```

```
            aes(x = hp, y = mpg),col="black")

#Define two polygons (one above, and one below the line)
  df_poly_under <- df_line %>%
  tibble::add_row(hp = c(max_x, min_x),
                  mpg = c(-Inf, -Inf))

  df_poly_above <- df_line %>%
  tibble::add_row(hp = c(max_x, min_x),
                  mpg = c(Inf, Inf))

#Plot the data, line, and the shades above and below the line
  p +

  geom_polygon(data = df_poly_under,
               aes(x = hp, y = mpg),
               fill = "blue",
               alpha = 1/5) +

  geom_polygon(data = df_poly_above,
               aes(x = hp, y = mpg),
               fill = "red",
               alpha = 1/5)+

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

  theme_minimal() +

    labs(title="Look at that snazzy red/blue shaded area",
                          subtitle = "Subtitle goes here",
                          caption = "Data: Made up from scratch",
                          x="",
                          y="") +

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

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

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



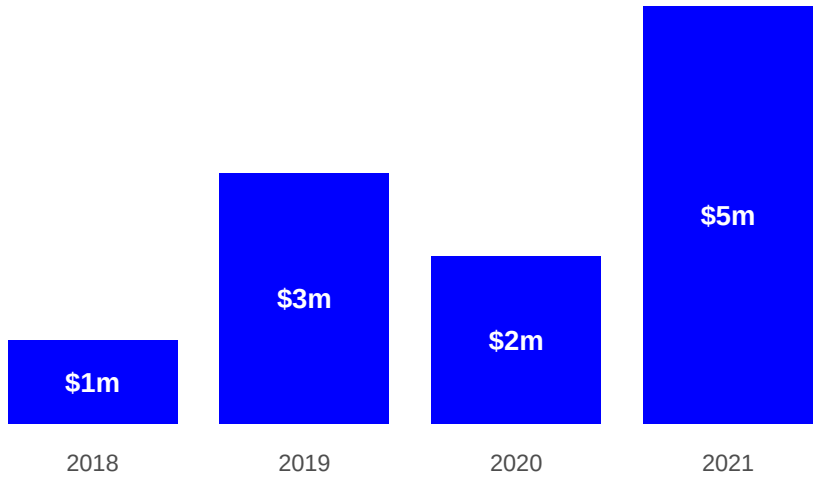
Bar chart (numercial)

```
Year = c("2018", "2019", "2020", "2021")

Value = (c(1000000,3000000, 2000000, 5000000))

bar_data_single <- (cbind(Year, Value))

bar_data_single <- as.data.frame(bar_data_single)

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


ggplot(bar_data_single, aes(x = Year, y = Value, label=Value)) +
  geom_bar(stat='identity',fill="blue",width=0.8)+
  geom_text(size = 5,
            col="white",fontface="bold",
            position = position_stack(vjust = 0.5),
            label=scales::dollar(Value,scale=1/1e6,suffix="m"))+

  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))+
  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())
```

Bar chart example

Subtitle goes here



Data: Made up from scratch

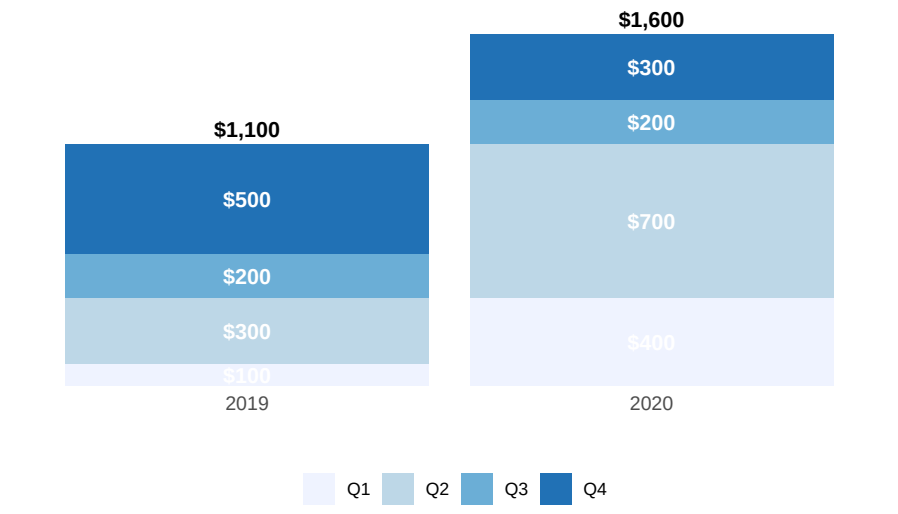
```
# ggsave(plot=last_plot(),
#        width=10,
#        height=10,
#        units="cm",
#        dpi = 600,
```

```
#        filename = "/Users/charlescoverdale/Desktop/test.png")
```

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())
```



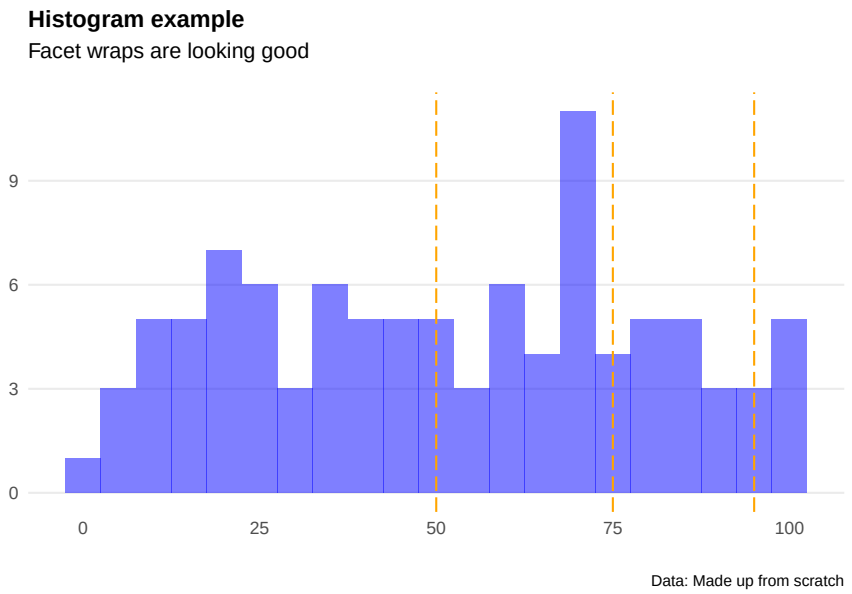
Histogram

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="orange") +

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


            theme_minimal() +

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

            theme(legend.position="none")+

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

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

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

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

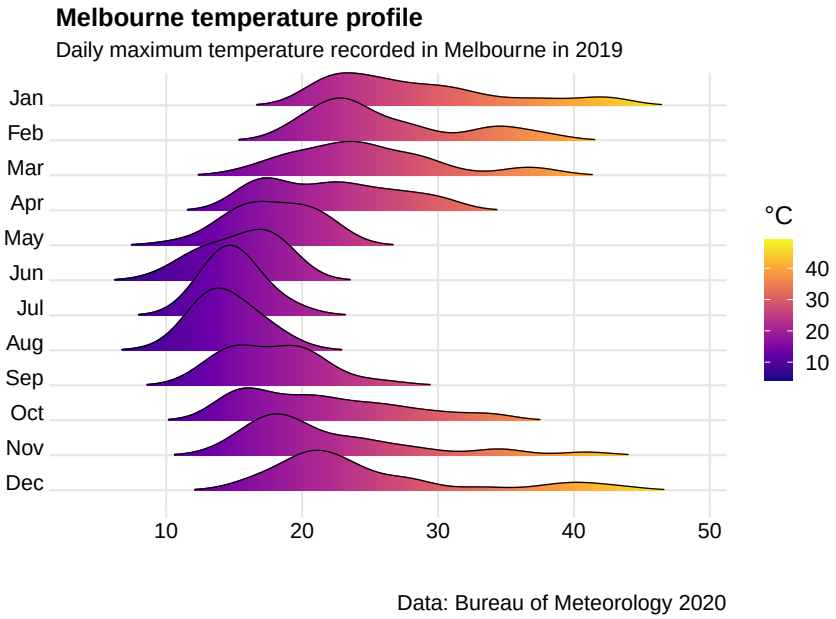


Ridge chart

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

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

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

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

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

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

# Plot
ggplot(MEL_temp_daily,
                                aes(x = Temp, y = Month, fill = stat(x))) +
                        geom_density_ridges_gradient(scale =2,
```

```
                                                size=0.3,
                                                rel_min_height = 0.01,
                                                gradient_lwd = 1.) +
                        scale_y_discrete(limits = unique(rev(MEL_temp_daily$Month)))+
                        scale_fill_viridis_c(name = "°C", option = "C") +
                        labs(title = 'Melbourne temperature profile',
                        subtitle = 'Daily maximum temperature recorded in Melbourne in 2019',
                        caption = "Data: Bureau of Meteorology 2020") +
                        xlab(" ")+
                        ylab(" ")+
                        theme_ridges(font_size = 13, grid = TRUE)
```



BBC style: Bar charts (categorical)

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

```
# bars <- ggplot(bar_df, aes(x = country, y = lifeExp)) +
#   geom_bar(stat="identity",
#            position="identity",
#            fill=ifelse(bar_df$country == "Mauritius", "#1380A1", "#dddddd")) +
#   geom_hline(yintercept = 0, size = 1, colour="#333333") +
#   bbc_style() +
#   labs(title="Reunion is highest",
#        subtitle = "Highest African life expectancy, 2007")
#
# bars <- bars + coord_flip()
# bars <- bars + coord_flip() +
#   theme(panel.grid.major.x = element_line(color="#cbcbcb"),
#         panel.grid.major.y=element_blank())
# bars <- bars + scale_y_continuous(limits=c(0,85),
#                     breaks = seq(0, 80, by = 20),
#                     labels = c("0","20", "40", "60", "80 years"))
#
# labelled.bars <- bars +
#   geom_label(aes(x = country, y = lifeExp, label = round(lifeExp, 0)),
#              hjust = 1,
#              vjust = 0.5,
#              colour = "white",
#              fill = NA,
#              label.size = NA,
#              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.

```
# #Prepare data
# dumbbell_df <- gapminder %>%
#   filter(year == 1967 | year == 2007) %>%
#   select(country, year, lifeExp) %>%
#   spread(year, 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))
#                 size = 3,
#                 colour_x = "#FAAB18",
#                 colour_xend = "#1380A1") +
#   bbc_style() +
#   labs(title="We're living longer",
#        subtitle="Biggest life expectancy rise, 1967-2007")
```

Facet wraps

Handy rather than showing multiple lines on the same chart.

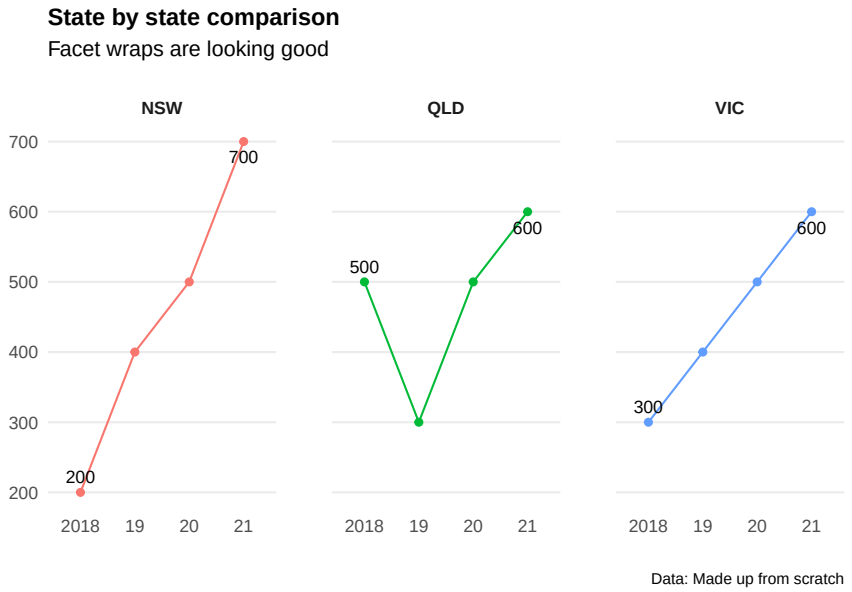
Top tips: `facet_wrap()`dataframes need to be in long form in order to be manipulated easily.

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

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

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

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

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

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

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

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

facet_data_long <- facet_data_long %>%
```

```
                    dplyr::mutate(start_label =
                           if_else(Year == min(Year),
                           as.integer(Value), NA_integer_))

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



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

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

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

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

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


base_chart +

            scale_x_discrete(
            breaks = seq(2018, 2021, 1),
```

```
            labels = c("2018", "19", "20", "21"))+

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

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


            theme_minimal() +

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

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

            theme(legend.position="none")+

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

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

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

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

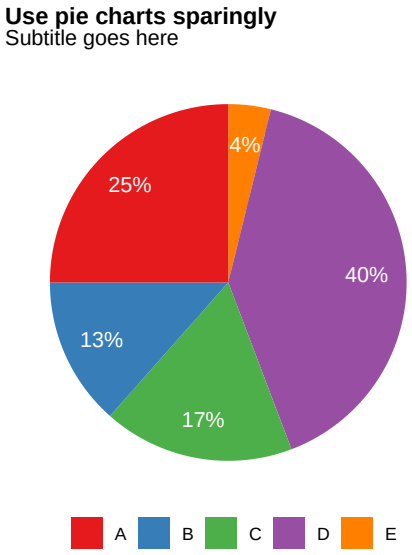


Pie chart

These should be used sparingly... but they are handy for showing proportions when the proportion of the whole is paramount (e.g. 45%) - rather than the proportion in relation to another data point (e.g. 16% one year vs 18% the next).

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

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

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

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

scale_fill_brewer(palette="Set1")+

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

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

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

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

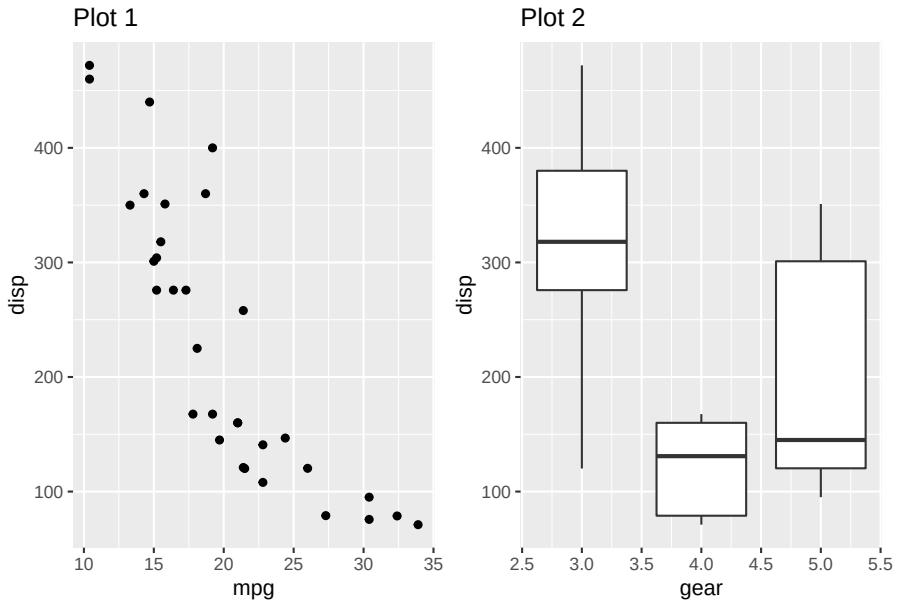


```
  # ggsave(plot=last_plot(),
  #       width=10,
  #       height=10,
  #       units="cm",
  #       dpi = 600,
  #       filename = "/Users/charlescoverdale/Desktop/pietest.png")
```

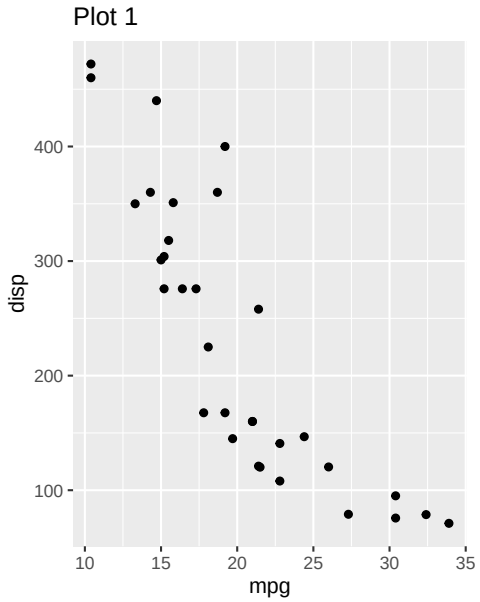
Patchwork

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

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

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

#Example of side by side charts
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")
```

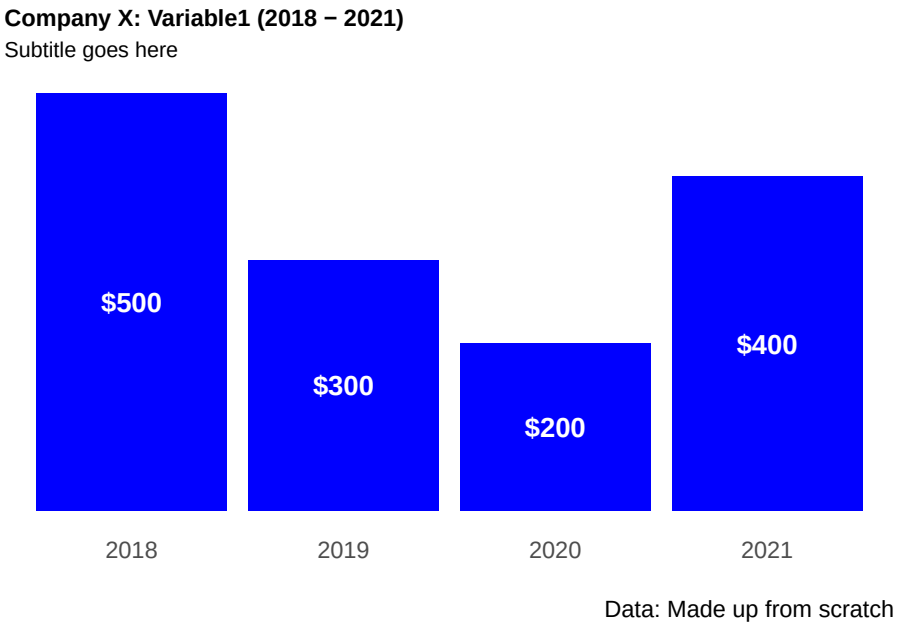
Automating chart creation

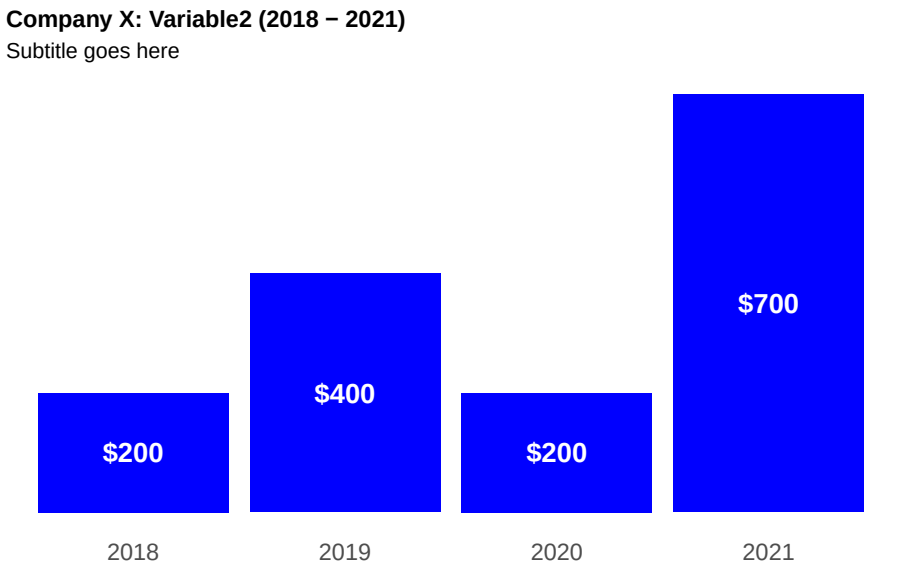
Let's say we have a dataframe of multiple variables. We want to produce simple charts of the same style for each variable (including formatting and titles etc). Sure we can change the `aes(x=)`and `aes(y=)` variables in ggplot2 manually for each column - but this is time intensive especially for large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

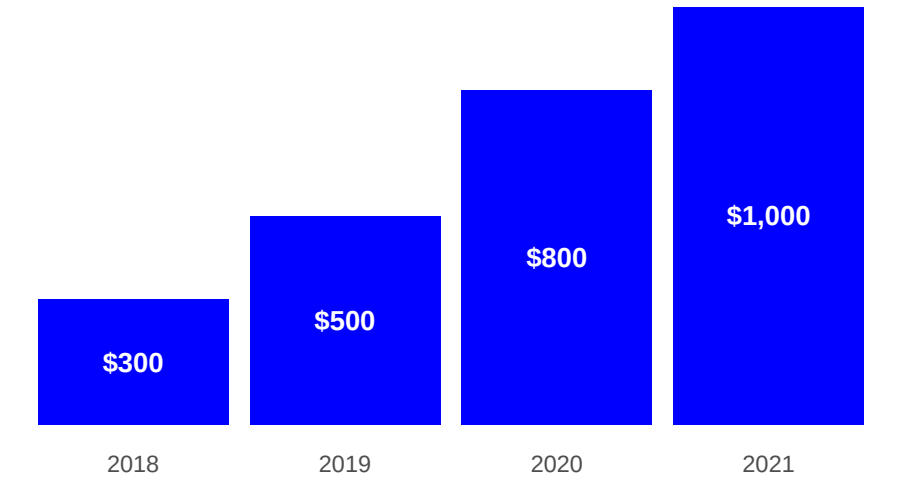
```
#Create a data set
Year = c("2018", "2019", "2020", "2021")

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
```








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

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 < 0.05) indicates strong evidence ***against*** the null hypothesis, so we ***reject*** it.
- A large p-value (> 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 in packages
library(ggridges)
library(ggplot2)
library(forecast)
library(ggrepel)
library(viridis)
library(readxl)
library(hrbrthemes)
library(dplyr)
```

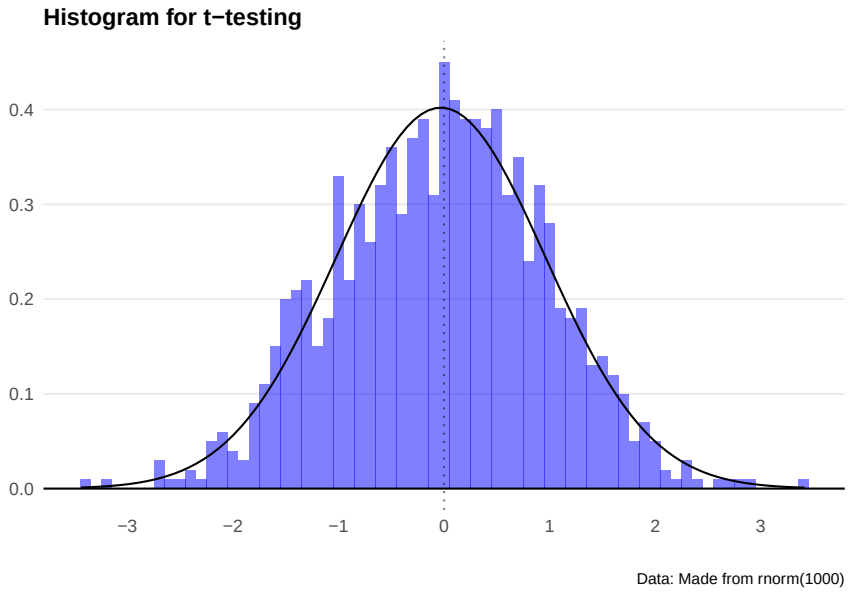
```
library(stringr)
library(reshape)
library(tidyr)
library(lubridate)
library(gapminder)
library(ggalt)
library(purrr)
library(scales)
library(purrr)
library(aTSA)
library(readrba)
```

T-testing our first hypothesis

We'll start simple. Let's create a random dataset - with the caveat that it will be normally distributed. By default the `rnorm` function will generate a dataset that has a mean of 0 and a standard deviation of 1.

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

ggplot()+
  geom_histogram(aes(x=dataset1$variable1,y=..density..),binwidth=0.1,fill="blue",alpha
  stat_function(fun = dnorm,
                args = list(mean = mean(dataset1$variable1), sd = sd(dataset1$variable
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = 0, linetype="dotted",alpha=0.5)+

            labs(title="Histogram for t-testing",
                          caption = "Data: Made from rnorm(1000)",
                          x="",
                          y="") +

            theme_minimal() +

            scale_x_continuous(breaks = seq(-3, 3, by = 1))+

            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"))
```



We know by default that the mean of dataset1 will be approximately zero... but let's check anyway.

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

Great, 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).

So quite simply, we can test the null hypothesis that the mean for dataset1$variable1 is 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.

Going further, let's test the hypothesis that the mean is 0.01.

```
# 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.

Tailed tests

In the previous level we were cheating slightly, in that we didn't specify the 'tail' for the test, or our confidence level. By default the t.test function assumes tests are two tailed, and the desired confidence level is 0.95 (i.e. 95%). However, in some cases we might want to have a hypothesis that says one variable is greater than or less than another variable (rather than just different from each other). This is where we use tails.

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

Correlation (and working with normal distributions)

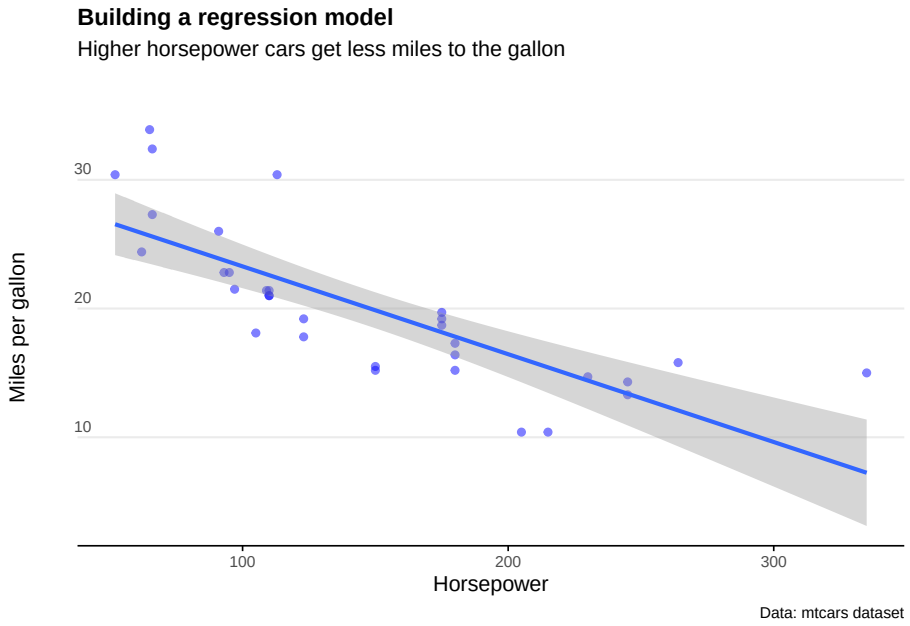
A correlation coefficient measures the direction and strength of the correlation between two variables.

The tricky thing is - if variables aren't normally distributed, none of our correlation theory works very well.

In the example below, 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.

```
ggplot(mtcars)+
      geom_point(aes(x = hp, y = mpg), col = "blue", alpha=0.5)+
      geom_smooth(aes(x = hp, y = mpg),method='lm')+
      labs(title="Building a regression model",
      subtitle = "Higher horsepower cars get less miles to the gallon",
      caption = "Data: mtcars dataset",
```

```
      x="Horsepower",
      y="Miles per gallon") +
      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 = 3, b = 0, l = 0)))+
      theme(axis.text.y = element_text(vjust = -0.5,
                        margin = ggplot2::margin(l = 20, r = -10)))+
      theme(plot.subtitle = element_text(margin=ggplot2::margin(0,0,25,0))) +
      theme(axis.line.x =
            element_line(colour ="black",size=0.4))+
      theme(axis.ticks.x = element_line(colour ="black",size=0.4))+
      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 = 3, b = 0, l = 0)))+
      theme(axis.text.y = element_text(vjust = -0.5,
                        margin = ggplot2::margin(l = 20, r = -10)))+
      theme(plot.subtitle = element_text(margin=ggplot2::margin(0,0,25,0))) +
      theme(axis.line.x =
            element_line(colour ="black",size=0.4))+

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



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

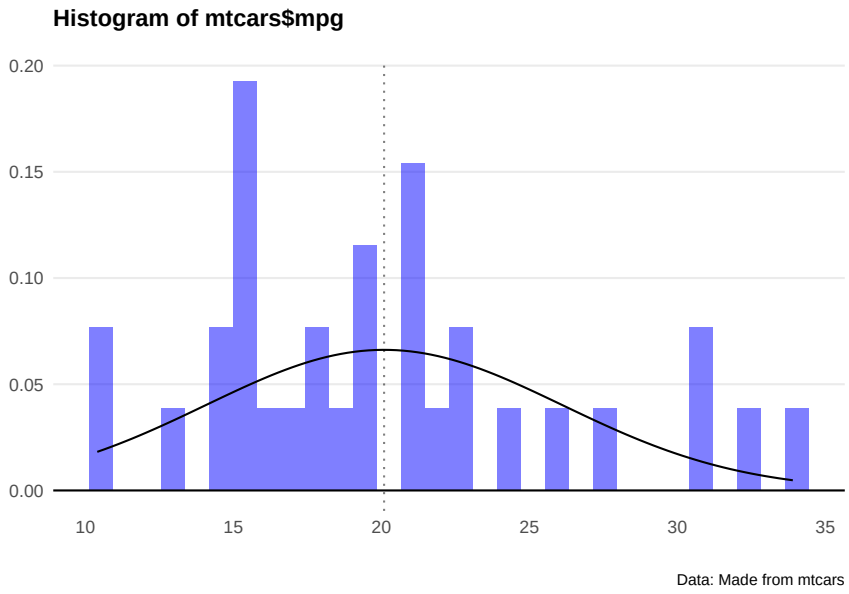
We specify the method as 'pearson' for the Pearson correlation coefficient. But we remember this method only works well if both our variables are normally distributed.

That's worth checking - let's plot a histogram for both hp and mpg.

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

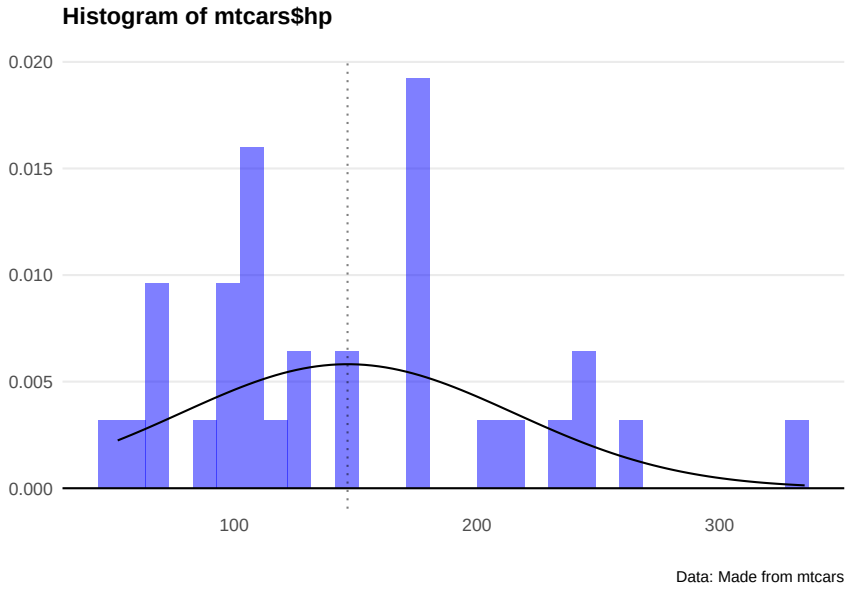
```
ggplot()+
  geom_histogram(aes(x=mtcars$mpg,y=..density..),fill="blue",alpha=0.5) +
  stat_function(fun = dnorm,
                args = list(mean = mean(mtcars$mpg), sd = sd(mtcars$mpg)))+
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = mean(mtcars$mpg), linetype="dotted",alpha=0.5)+
  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

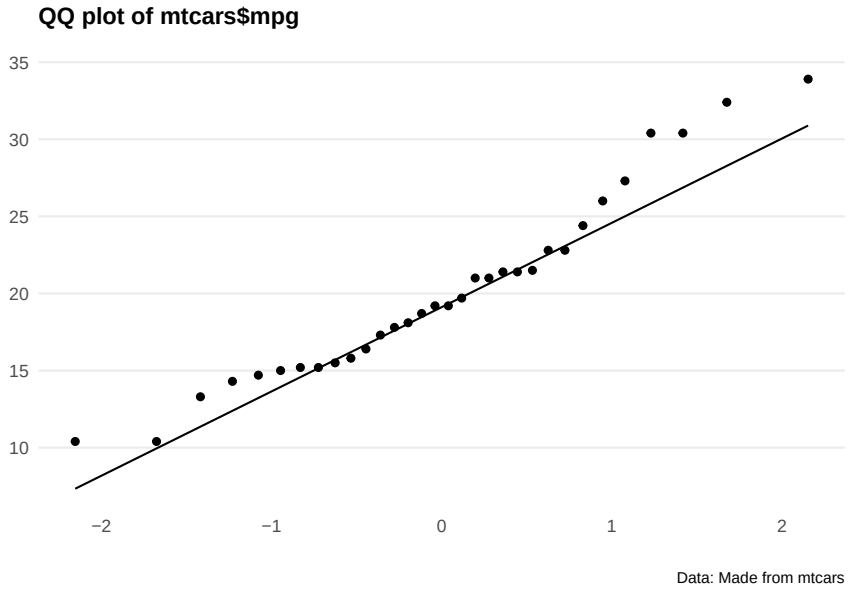


Crikey... they don't look very normal at all.

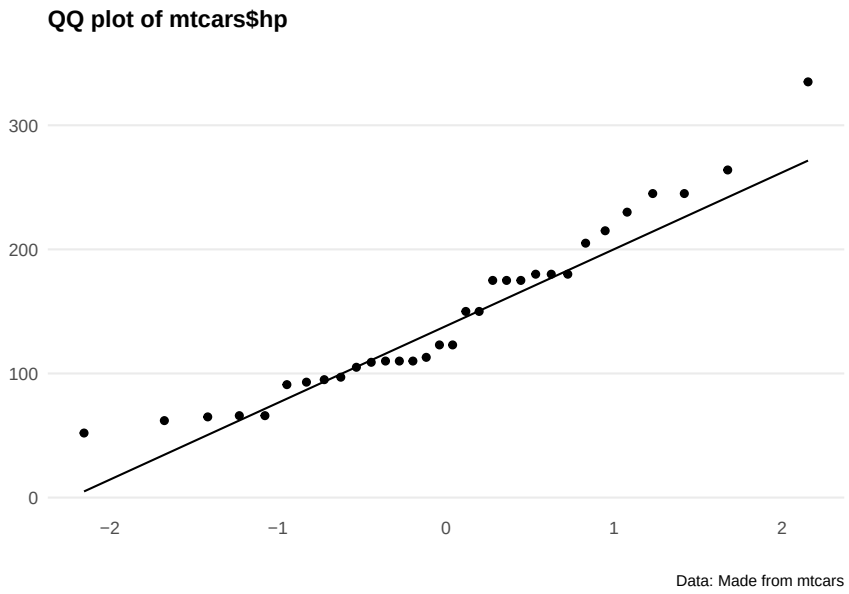
Let's plot QQ plots of our variables and see what's going on.

```
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 (mean)

Firstly, we can calculate the confidence interval on a single variable. Essentially this measures the variance of the normal distribution, and gives us an idea of how 'clustered' the values are to the mean.

There are roughly 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))
```

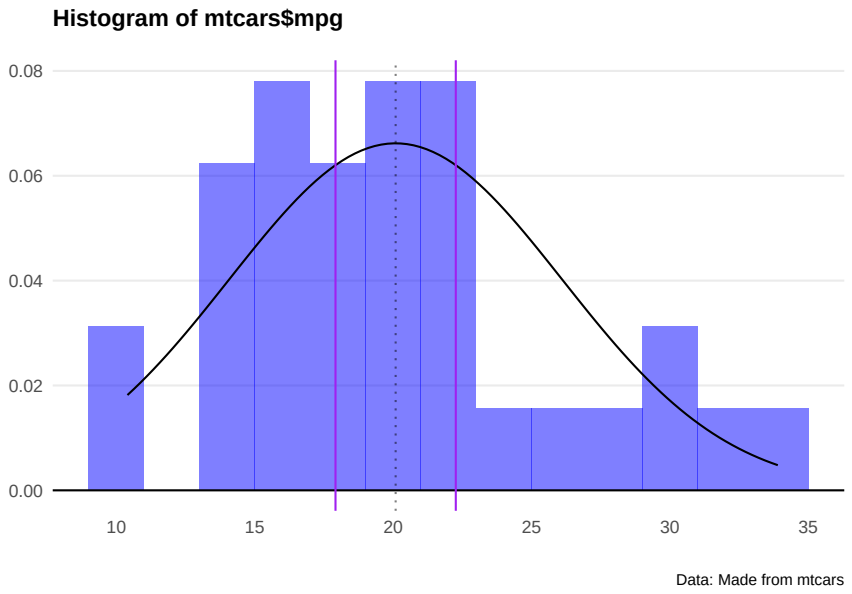
For the lazy folks among us - there's also this quick and dirty way of doing it.

```
# 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()+
  geom_histogram(aes(x=mtcars$mpg,y=..density..),binwidth=2,fill="blue",alpha=0.5) +
  stat_function(fun = dnorm,
                args = list(mean = mean(mtcars$mpg), sd = sd(mtcars$mpg)))+
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = mean(mtcars$mpg), linetype="dotted",alpha=0.5)+
  geom_vline(xintercept = lower.bound,col="purple")+
  geom_vline(xintercept = upper.bound, col="purple")+
            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"))
```



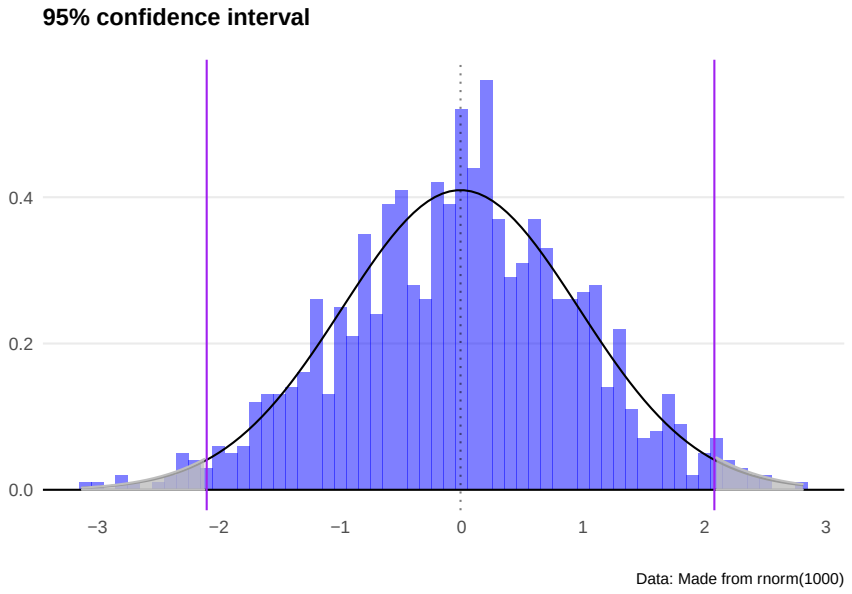
Two things we note here: First, the distribution doesn't look ***that*** normal. Second, geez that 95% confidence interval looks narrow as a result.

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)
print(df2.mean)

df2.n <- length(dataset2$variable1)
df2.sd <- sd(dataset2$variable1)
df2.se <- mpg.sd/sqrt(mpg.n)
print(df2.se)
```

```
alpha = 0.05
degrees.freedom = df2.n - 1
t.score = qt(p=alpha/2, df=degrees.freedom,lower.tail=F)
print(t.score)

df2.error <- t.score * df2.se

lower.bound.df2 <- df2.mean - df2.error
upper.bound.df2 <- df2.mean + df2.error
print(c(lower.bound.df2,upper.bound.df2))

#Make a function to colour the tails
upper_tail <- function(x) {
              y <- dnorm(x,mean=0,sd=1)
              y[x < upper.bound.df2  | x > 1000] <- NA
              return(y) }

lower_tail <- function(x) {
              y <- dnorm(x,mean=0,sd=1)
              y[x < -1000  | x > lower.bound.df2] <- NA
              return(y) }

#Plot the distributions
ggplot()+
  geom_histogram(aes(x=dataset2$variable1,y=..density..),binwidth=0.1,fill="blue",alpha=0.5) +
  stat_function(fun = dnorm,
                args = list(mean = mean(dataset2$variable1), sd = sd(dataset2$variable1)))+
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = mean(dataset2$variable1), linetype="dotted",alpha=0.5)+
  geom_vline(xintercept = lower.bound.df2,col="purple")+
  geom_vline(xintercept = upper.bound.df2, col="purple")+
  stat_function(fun = upper_tail, geom = "area", fill = "grey",col="grey", alpha = 0.8) +
  stat_function(fun = lower_tail, geom = "area", fill = "grey",col="grey", alpha = 0.8) +

            labs(title="95% confidence interval",
                         caption = "Data: Made from rnorm(1000)",
                         x="",
                         y="") +

            theme_minimal() +

            scale_x_continuous(breaks = seq(-3, 3, by = 1))+

            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"))
```



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 (model)

Secondly, we can calculate the confidence internal around a linear model. Somewhat similar to above, this process shows us how confident we can be about any single point in the linear estimate (e.g. if it has an enormous confidence interval

attached to it... the linear estimate of the value at that point is probably a bit dodgy).

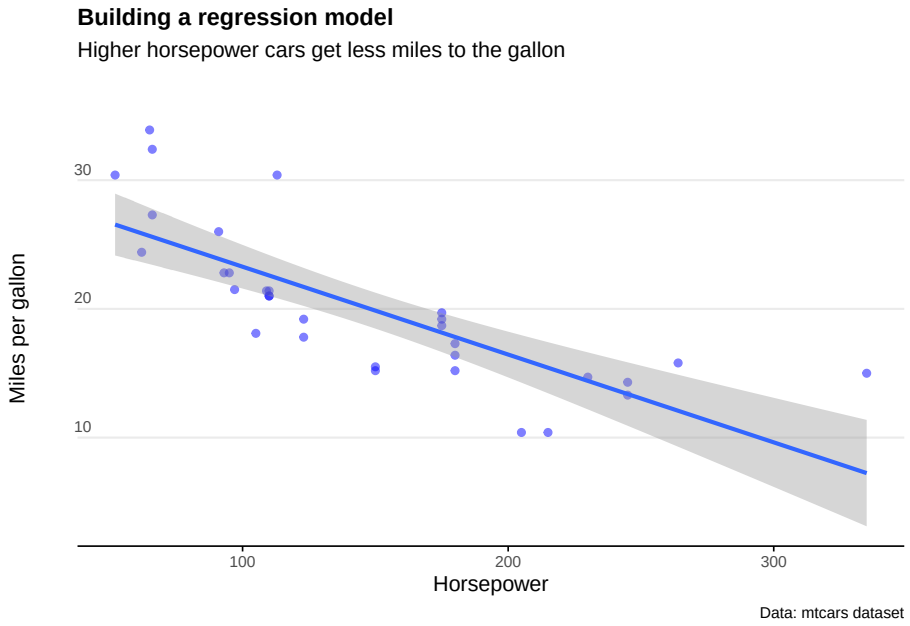
```
mtcars.lm <- lm(mpg ~ hp, data = mtcars)
summary(mtcars.lm)

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

It's great to have the raw data - but there's an even easier way of plotting the confidence interval on a chart: we use the `geom_smooth()`function.

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

```
ggplot(mtcars)+
      geom_point(aes(x = hp, y = mpg), col = "blue", alpha=0.5)+
      geom_smooth(aes(x = hp, y = mpg),method='lm',level=0.95)+
      labs(title="Building a regression model",
      subtitle = "Higher horsepower cars get less miles to the gallon",
      caption = "Data: mtcars dataset",
      x="Horsepower",
      y="Miles per gallon") +
      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 = 3, b = 0, l = 0)))+
      theme(axis.text.y = element_text(vjust = -0.5,
                        margin = ggplot2::margin(l = 20, r = -10)))+
      theme(plot.subtitle = element_text(margin=ggplot2::margin(0,0,25,0))) +
      theme(axis.line.x =
            element_line(colour ="black",size=0.4))+
      theme(axis.ticks.x = element_line(colour ="black",size=0.4))+
      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 = 3, b = 0, l = 0)))+
      theme(axis.text.y = element_text(vjust = -0.5,
                                       margin = ggplot2::margin(l = 20, r = -10)))+
      theme(plot.subtitle = element_text(margin=ggplot2::margin(0,0,25,0))) +
      theme(axis.line.x =
            element_line(colour ="black",size=0.4))+

      theme(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

Background

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 do a confidence interval (as we learned in the previous chapter) - but what about forecasting for multiple periods into the future.

That's where we need to build some models.

```
# Load in packages
library(ggridges)
library(ggplot2)
library(forecast)
library(ggrepel)
library(viridis)
library(readxl)
library(hrbrthemes)
library(dplyr)
library(stringr)
library(reshape)
library(tidyr)
library(lubridate)
library(gapminder)
library(ggalt)
library(purrr)
library(scales)
library(purrr)
library(aTSA)
library(readrba)
```

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

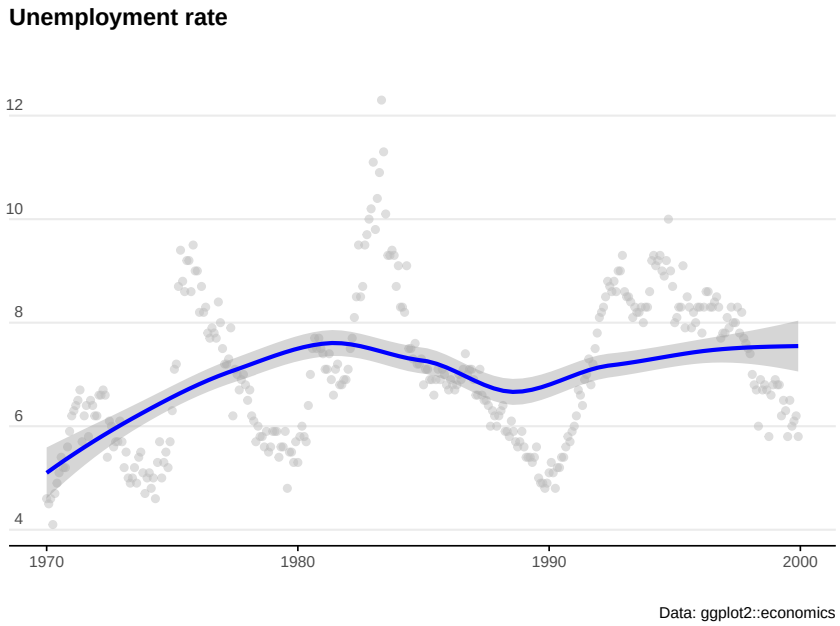
```
econ_data <- economics %>% dplyr::select(c("date", "uempmed"))
econ_data <- econ_data %>% dplyr::filter((date >= as.Date("1970-01-01")
                                          & date <= as.Date("1999-12-31")))
```

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

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

Let's plot the 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(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 = 3, b = 0, l = 0)))+

      theme(axis.text.y = element_text(vjust = -0.5,
                        margin = ggplot2::margin(l = 20, r = -10)))+

      theme(plot.title = element_text(margin=ggplot2::margin(0,0,25,0))) +

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

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



ARIMA models

AutoRegressive Integrated Moving Average (ARIMA) models are a handy tool to have in the toolbox. An auto regressive model is one 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.

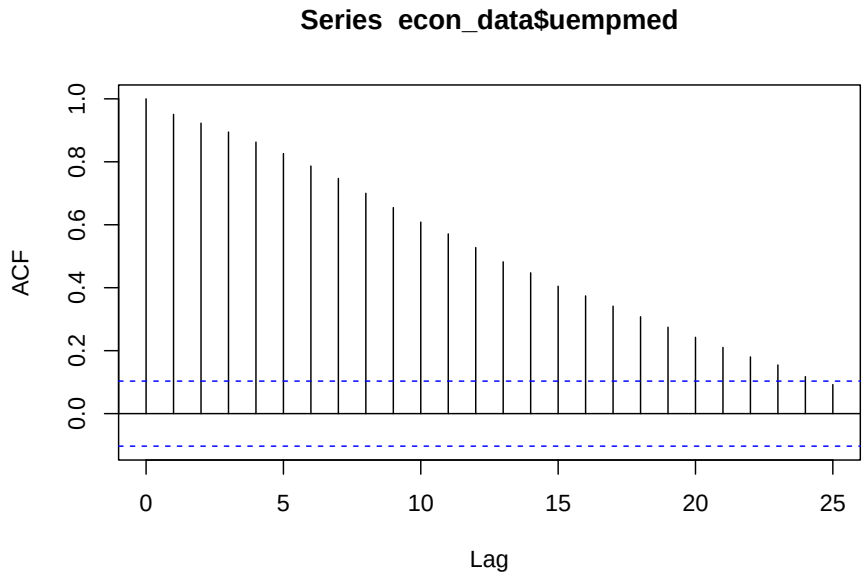
First order of business, we may need to 'difference' our series to make it stationary. Let's check if it is stationary using the augmented Dickey-Fuller test. The null hypothesis assumes that the series is non-stationary. A series is said to be stationary when its mean, variance, and autocovariance don't change much over time.

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

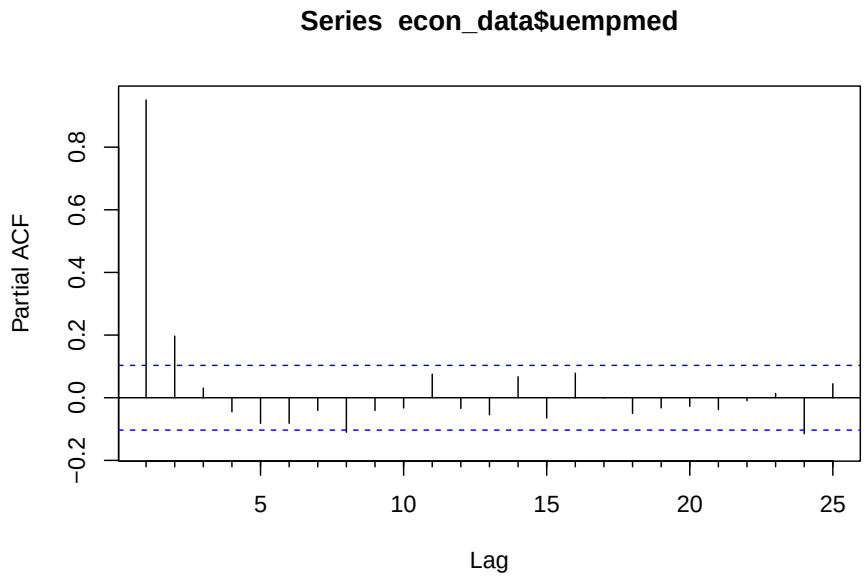
```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag    ADF p.value
## [1,]   0 -0.448   0.515
```

```
## [2,]   1 -0.279   0.564
## [3,]   2 -0.252   0.571
## [4,]   3 -0.218   0.581
## [5,]   4 -0.321   0.552
## [6,]   5 -0.399   0.529
## Type 2: with drift no trend
##      lag   ADF p.value
## [1,]   0 -3.05  0.0337
## [2,]   1 -2.54  0.1158
## [3,]   2 -2.43  0.1572
## [4,]   3 -2.58  0.0986
## [5,]   4 -2.68  0.0826
## [6,]   5 -2.82  0.0597
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -2.94   0.179
## [2,]   1 -2.33   0.438
## [3,]   2 -2.20   0.491
## [4,]   3 -2.33   0.439
## [5,]   4 -2.49   0.368
## [6,]   5 -2.69   0.285
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

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




```
# Identify patial auto correlation
Pacf(econ_data$uempmed)
```

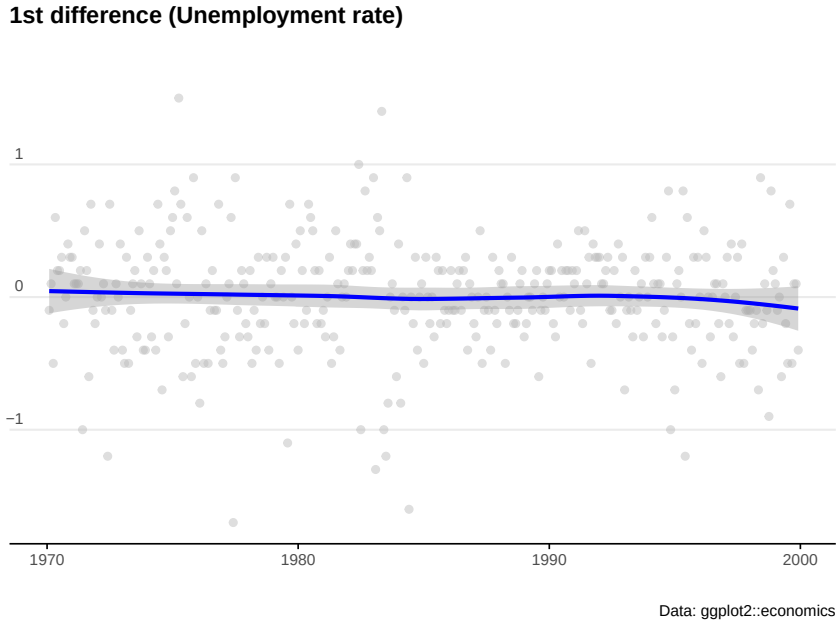


```r
# Take the first differences of the series
econ_data <- econ_data %>% mutate(diff = uempmed-lag(uempmed))

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

      labs(title="1st difference (Unemployment rate)",
      caption = "Data: ggplot2::economics",
      x="",
      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 = 3, b = 0, l = 0)))+

      theme(axis.text.y = element_text(vjust = -0.5,
                        margin = ggplot2::margin(l = 20, r = -10)))+

      theme(plot.title = element_text(margin=ggplot2::margin(0,0,25,0))) +

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

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

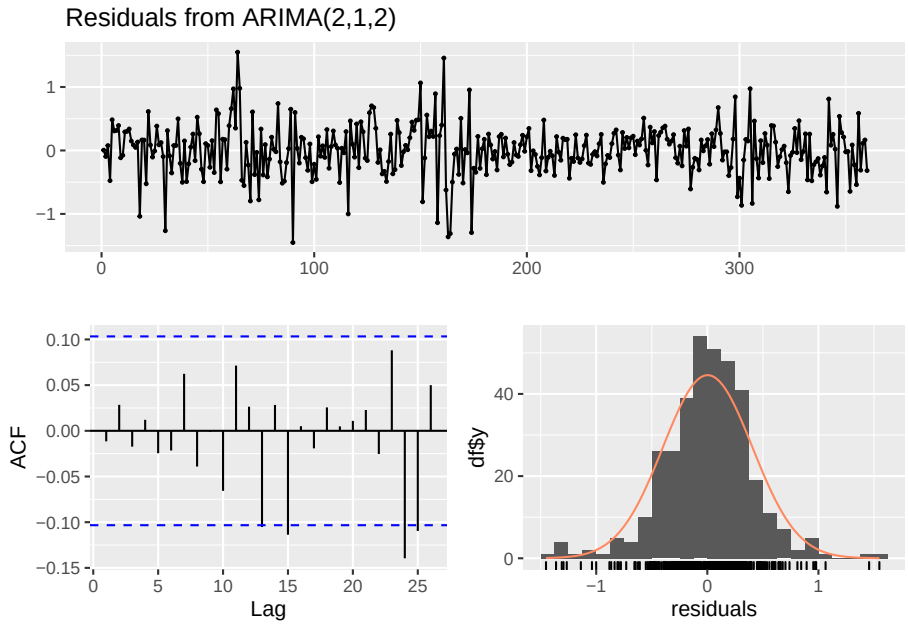
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```
## Warning: Removed 1 rows containing non-finite values (stat_smooth).
```

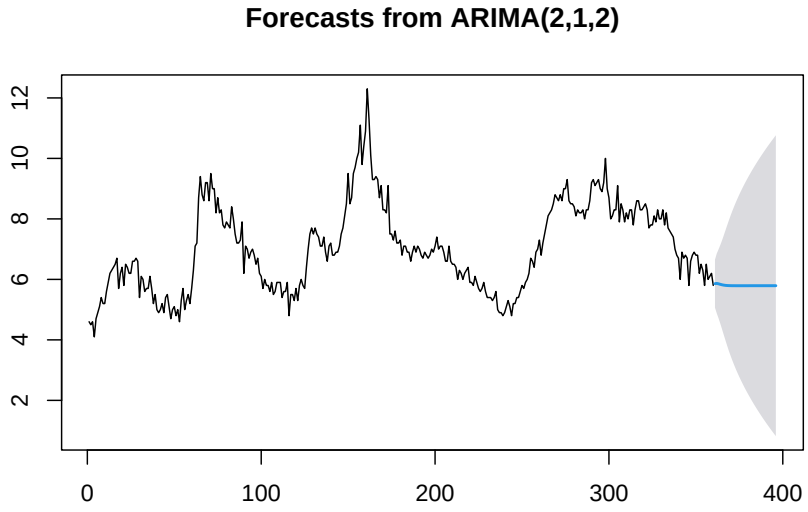
```
## Warning: Removed 1 rows containing missing values (geom_point).
```




```
ARIMA_model = forecast::auto.arima(econ_data$uempmed)

ARIMA_model
summary(ARIMA_model)
checkresiduals(ARIMA_model)
```




```
# Forecast for the next 10 time units
ARIMA_forecast <- forecast::forecast(ARIMA_model, newdata=econ_data$uempmed,h = 36,leve

# Plot forecasts
plot((ARIMA_forecast))
```


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

Web-scraping

Introduction

Getting content off websites can be a nightmare. The worst case resort 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 tutorial by Grant R. McDermott in his *Data Science for Economists*series.

First up, let's load some packages.

```
# Load and install the packages
pacman::p_load(tidyverse, dplyr, rvest, lubridate, janitor, data.table, hrbrthemes)
```

Anatomy of a webpage

Let's introduce some terminology: ***server side***

This describes information being embeded directly in the webpage's HTML. When dealing with server side webpages, finding the correct CSS (or Xpath) "selectors" becomes the hardest part of the task.

Iterating through dynamic webpages (e.g. "Next page" and "Show More" tabs) is also tricky - but we'll get there.

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("https://en.wikipedia.org/wiki/Men%27s_100_metres_world_reco
m100
```

```
## {html_document}
## <html class="client-nojs" lang="en" dir="ltr">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject  ...
```

. . . 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 is named in the HTML code: **.wikitable :nth-child(1)**

Side note: The names of tables in HTML webpages changes all the time. Therefore this code is considered unstable in terms of setting and forgetting.

```
pre_iaaf <-
  m100 %>%
  #Select table
  rvest::html_element(".wikitable :nth-child(1)") %>%
  #Convert to data frame
  rvest::html_table()

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, ~
## # ... with 11 more rows
```

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

```
pre_iaaf <- pre_iaaf %>%
            clean_names() %>%
            dplyr::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
## # ... with 11 more rows
```

Let's also scrape the data for the more recent running records. That's the tables called 'Pre-automatic timing (1912–1976),' and 'Modern Era (1977 onwards).'

For the second table:

```
iaaf_76 <- m100 %>%
           html_element("h3+ .wikitable") %>%
           rvest::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]
## # ... with 44 more rows
```

And now for the third table:

```
iaaf <- m100 %>%
        html_element("p+ .wikitable") %>%
        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
## # ... with 14 more rows, and 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"),
```

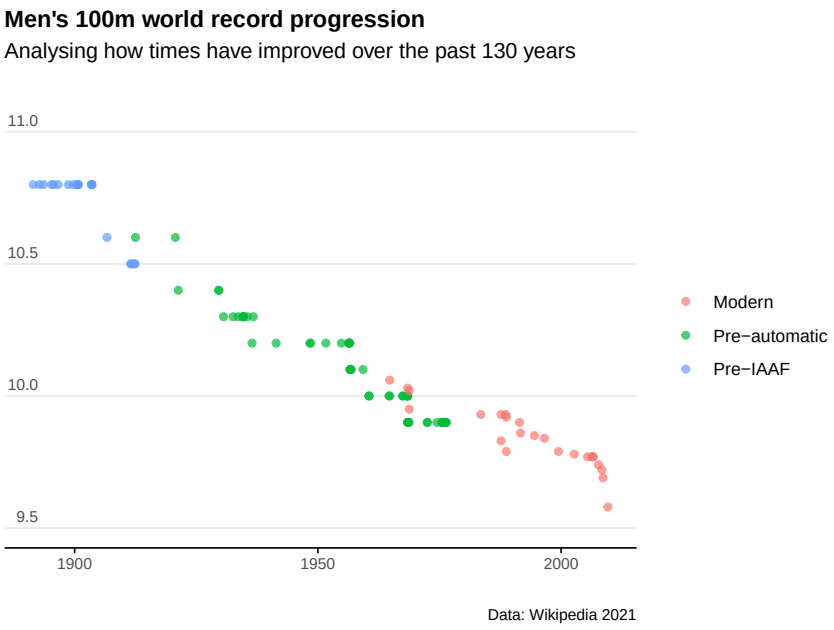
```
    iaaf %>% dplyr::select(time, athlete, nationality, date) %>%
    mutate(era = "Modern")
    )

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
## # ... with 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 2021",
  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)))+

  theme(plot.subtitle = element_text(margin=ggplot2::margin(0,0,25,0))) +

  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(axis.text=element_text(size=8))+
theme(panel.grid.minor = element_blank())+
theme(panel.grid.major.x = element_blank()) +

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



Text-mining

Power with words

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 (mostly because it expires over time). These can be called with the R package called `gutenbergr`

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

```
library(gutenbergr)

bible <- gutenberg_download(30)

bible_tidy <- bible %>%
```

```
  unnest_tokens(word, text) %>%
  anti_join(stop_words)

#Find the most common words

bible_tidy %>%
  count(word, sort=TRUE)
```

```
## # A tibble: 12,595 x 2
##    word       n
##    <chr>  <int>
##  1 lord    7830
##  2 thou    5474
##  3 thy     4600
##  4 god     4446
##  5 ye      3982
##  6 thee    3827
##  7 001     2783
##  8 002     2721
##  9 israel  2565
## 10 003     2560
## # ... with 12,585 more rows
```

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.

```
#Download sentiment list
nrcjoy <- get_sentiments("nrc") %>%
  filter(sentiment == "joy")

#Join bible words with sentiment list
bible_tidy %>%
  inner_join(nrcjoy) %>%
  count(word, sort=TRUE)
```

```
## # A tibble: 258 x 2
##    word         n
##    <chr>    <int>
##  1 god       4446
##  2 art        494
##  3 peace      429
##  4 found      402
##  5 glory      402
##  6 daughter   324
##  7 pray       313
##  8 love       310
##  9 blessed    302
## 10 mighty     284
## # ... with 248 more rows
```

Economic indicators

Overview

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 required packages
pacman::p_load(ggmap, ggplot2, dplyr, tmaptools, RCurl, jsonlite, tidyverse, leaflet, writexl, re
```

Gross Domestic Product

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 <- read_abs("5206.0")
```

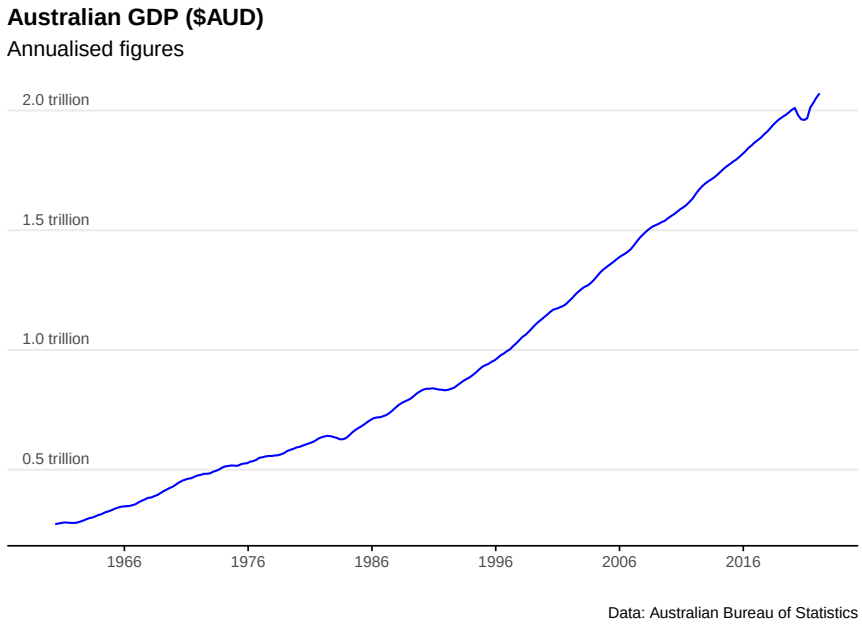
```
#Select the seasonally adjusted data and filter for data and value columns
gdp_level <- all_gdp %>%
              filter(series == "Gross domestic product: Chain volume measures ;",
                     !is.na(value)) %>%
              filter(series_type =="Seasonally Adjusted") %>%
              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: 6 x 9
##   date       quarterly_output quarterly_growth_rate annual_gdp annual_gdp_trill~
##   <date>                <dbl>                 <dbl>      <dbl>             <dbl>
## 1 2020-12-01           501644                  3.28     1960142              1.96
## 2 2021-03-01           510590                  1.78     1967236              1.97
```

```
## 3 2021-06-01           514784                 0.821    2012745              2.01
## 4 2021-09-01           505413                -1.82     2032431              2.03
## 5 2021-12-01           523725                 3.62     2054512              2.05
## 6 2022-03-01           527676                 0.754    2071598              2.07
## # ... with 4 more variables: 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 = margin(t = 0, r = 0, b = 0, l = 0)))+

            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = 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
```



Unemployment rate

Download the data

```
#Download the time sreies
all_unemployment <- read_abs("6202.0")
```

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 2021-11-01  4.60                   6.74          5.47          8.01
## 2 2021-12-01  4.16                   6.74          5.47          8.01
## 3 2022-01-01  4.18                   6.74          5.47          8.01
## 4 2022-02-01  4.02                   6.74          5.47          8.01
## 5 2022-03-01  3.93                   6.74          5.47          8.01
## 6 2022-04-01  3.85                   6.74          5.47          8.01
```

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 = margin(t = 0, r = 0, b = 0, l = 0)))+

            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = 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 <- read_abs("6401.0")
```

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 2020-12-01   0.9     4.94           1.9           7.3
## 2 2021-03-01   1.1     4.94           1.9           7.3
```

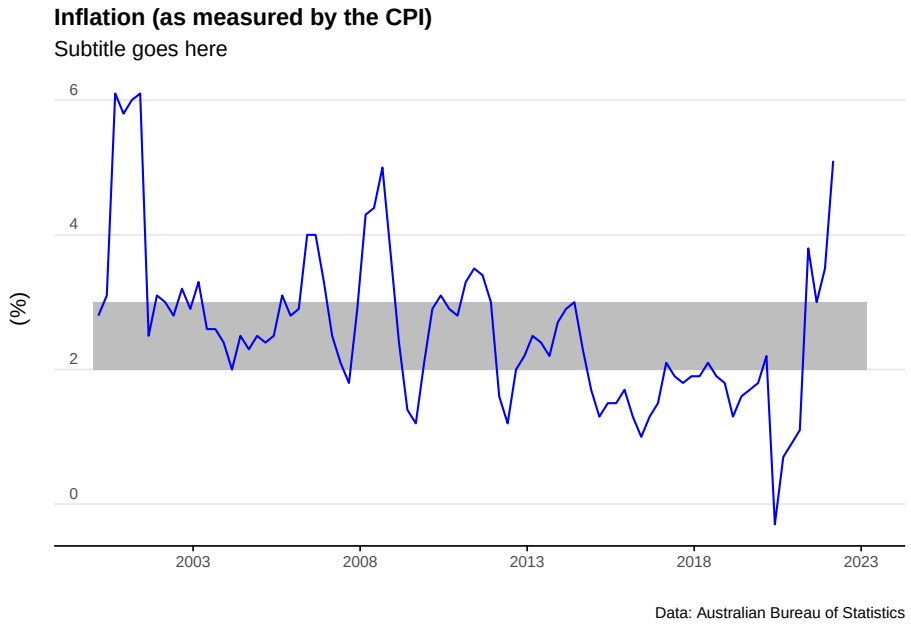
```
## 3 2021-06-01   3.8     4.94           1.9           7.3
## 4 2021-09-01   3       4.94           1.9           7.3
## 5 2021-12-01   3.5     4.94           1.9           7.3
## 6 2022-03-01   5.1     4.94           1.9           7.3
```

```
#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("2023-03-01"),
                                 ymin=2,
                                 ymax=3),
                                 alpha=0.01,
                                 fill="grey")+
                   geom_line(aes(x = date, y = value), col = "blue") +

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

            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)+
            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 = margin(t = 0, r = 0, b = 0, l = 0)))+
```

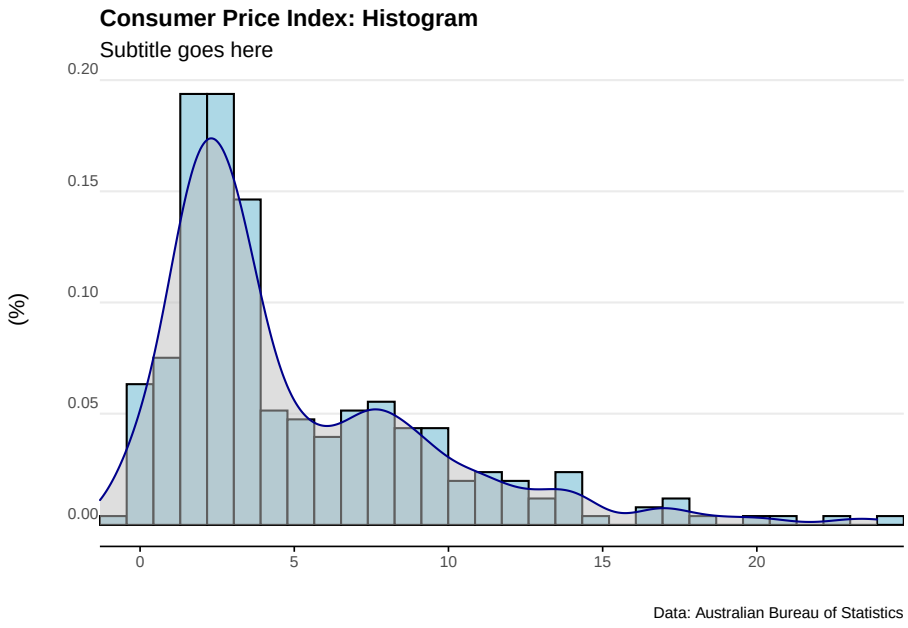
```
            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = margin(l = 20, r = -15)))+

            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))+

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

```
            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 = margin(t = 0, r = 0, b = 20, l = 0)))+

            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = margin(l = 20, r = -2)))+

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


plot_CPI_hist
```



Wage Price Index

Download the data

```
all_wpi <- 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 2020-12-01   1.4     3.06
## 2 2021-03-01   1.5     3.06
## 3 2021-06-01   1.7     3.06
## 4 2021-09-01   2.2     3.06
## 5 2021-12-01   2.3     3.06
## 6 2022-03-01   2.4     3.06
```

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 = margin(t = 0, r = 0, b = 0, l = 0)))+

            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = 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
```

RBA cash rate

Download the data

```
cash_rate_all<- readrba::read_rba(table_no = "F13")
head(cash_rate_all)
```

```
## # A tibble: 6 x 11
##   date       series          value frequency series_type units source pub_date
##   <date>     <chr>           <dbl> <chr>     <chr>       <chr> <chr>  <date>
## 1 1990-01-31 Australia Targ~  17   Monthly   Original    Per ~ RBA    2022-06-02
## 2 1990-02-28 Australia Targ~  16.5 Monthly   Original    Per ~ RBA    2022-06-02
## 3 1990-03-31 Australia Targ~  16.5 Monthly   Original    Per ~ RBA    2022-06-02
## 4 1990-04-30 Australia Targ~  15   Monthly   Original    Per ~ RBA    2022-06-02
## 5 1990-05-31 Australia Targ~  15   Monthly   Original    Per ~ RBA    2022-06-02
## 6 1990-06-30 Australia Targ~  15   Monthly   Original    Per ~ RBA    2022-06-02
## # ... with 3 more variables: series_id <chr>, description <chr>,
## #   table_title <chr>
```

Clean and analyse the data

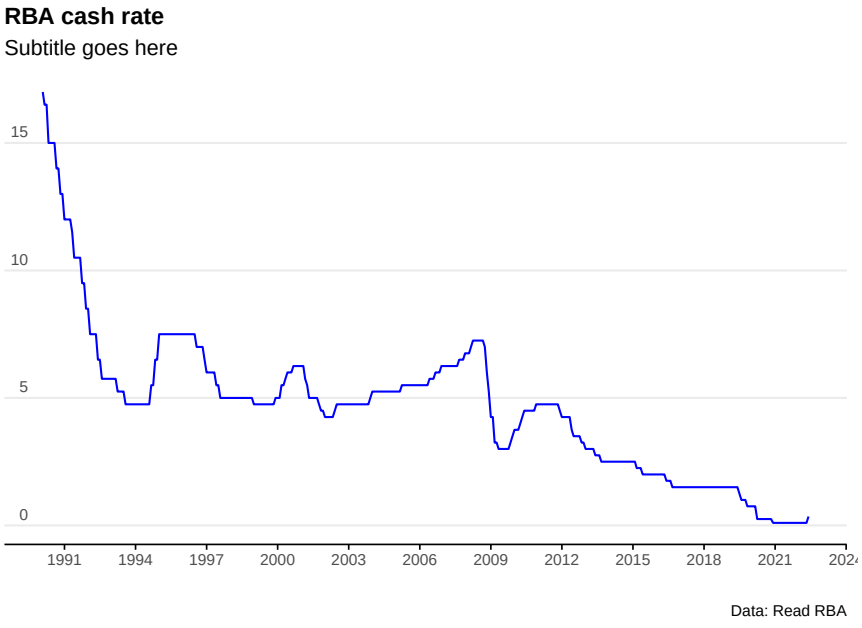
```
cash_rate_Australia <- cash_rate_all %>%
                       filter(series=="Australia Target Cash Rate") %>%
                       dplyr::select(date, value)

tail(cash_rate_Australia)
```

```
## # A tibble: 6 x 2
##   date       value
##   <date>     <dbl>
## 1 2021-12-31  0.1
## 2 2022-01-31  0.1
## 3 2022-02-28  0.1
## 4 2022-03-31  0.1
## 5 2022-04-30  0.1
## 6 2022-05-31  0.35
```

Plot the data

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

            labs(title = "RBA cash rate",
                           subtitle = "Subtitle goes here",
                           caption = "Data: Read RBA",
                           y = " ",
                           x = " ")+
```

```
            scale_y_continuous(labels = scales::comma)+
            scale_x_date(date_breaks = "3 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 = margin(t = 0, r = 0, b = 0, l = 0)))+

            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = 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_cash_rate
```



AUD exchange rate

God knows why - but 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 2022-06-06 0.720
## 2 2022-06-07 0.718
## 3 2022-06-08 0.720
## 4 2022-06-09 0.718
## 5 2022-06-10 0.712
## 6 2022-06-14 0.697
```

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 = " ",
                            x = " ")+

            scale_y_continuous(labels = scales::comma)+
            scale_x_date(date_breaks = "3 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 = margin(t = 0, r = 0, b = 0, l = 0)))+

            theme(axis.text.y = element_text(vjust = -0.5,
                                             margin = 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_exchange_rate_AUD
```

AUD exchange rate

Subtitle goes here

Data: Reserve Bank of Australia

Geocoding in R

About geocoding

Geocoding allows us to find coordinates for a data point (or vice versa). For instance, if we have the street address of an asset/object we can use geocoding to find the latitude and longitude. Similarly, if we have the lat and lon, we can find the street address. This allows us to turn 'boring' address data into a *spatial data point*.

This makes geocoding immensely useful when trying to find the 'catchments' around a particular asset/object. It's also handy in visualising location based data on a map.

To find the lat and long for an address - we have to find someone with a massive database of addresses and coordinates. Unsurprisingly, Google (through their Google Maps team) has this data.

We can ping Google's data base through an API (they charge a 'freemium' model) to find the data we need.

Further reading

- Breaking Down Geocoding in R: A Complete Guide | by Oleksandr Titorchuk | Towards Data Science

Introducing the dataset

Approved speed camera locations in Victoria, Australia are publicly available through the government website. We can download the dataset as a spreadsheet.

Note: 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). However, it will cause some niche issues in the geocoding process.

Data cleaning

First, we start by installing and loading up some packages.

```
#Loads the required required packages
pacman::p_load(ggmap, tmaptools, RCurl, jsonlite, tidyverse, leaflet, writexl, readr, r

#Make sure the HTML works
webshot::install_phantomjs()
```

Next, we import the data as a CSV.

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. Let's combine 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.

We think all these addresses are in the state of Victoria... however let's not at the state in at this stage (we can see why later).

```
#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
```

Running the geocode

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

We can geocode the lats and lons using the Google API through `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)
```

From geocode to shapefile

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.

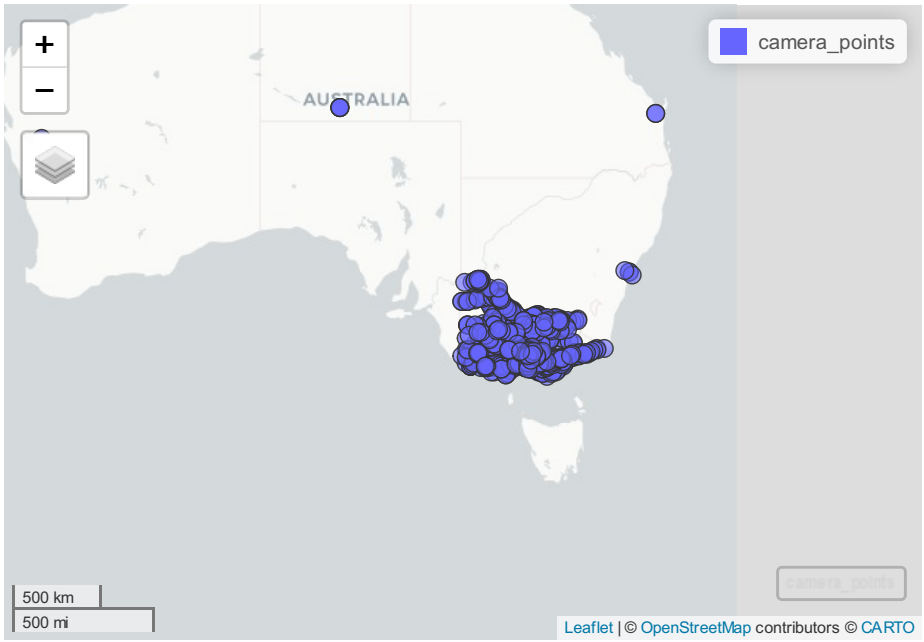
```
camera_geocoded <- camera_ggmap
```

```
#Rename the API generated address field
camera_geocoded_clean <- camera_geocoded %>% dplyr::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 seperately for correction or exclusion.

```
#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.

```
#Write the points to a shapefile for use in QGIS
#sf::st_write(camera_points, "C:/Data/camera_points.shp")
```

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)
```

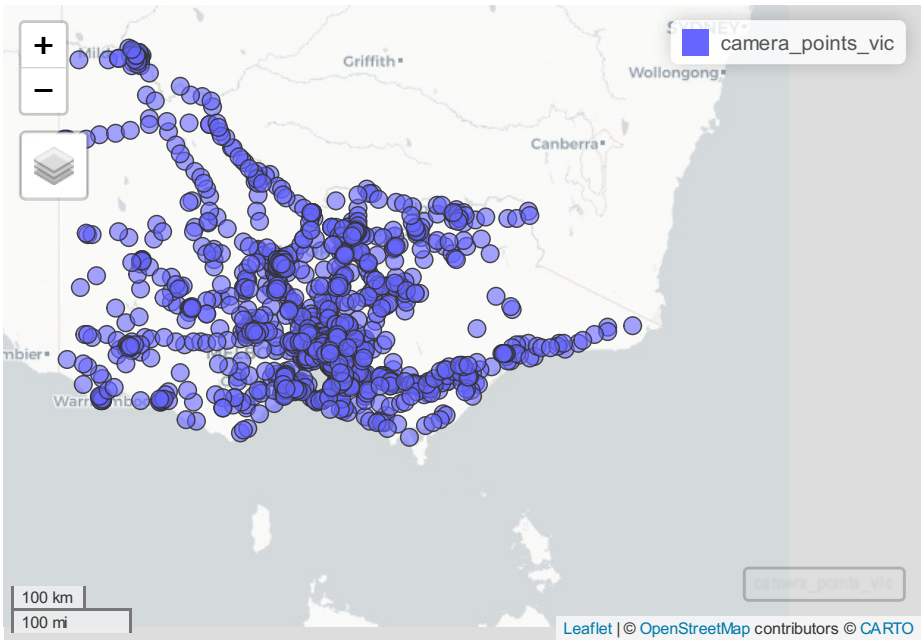
```
## Simple feature collection with 7 features and 5 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 115.9555 ymin: -33.90941 xmax: 152.7034 ymax: -25.53849
## CRS:           +proj=longlat
##                                      +datum=WGS84
##                                      +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.

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

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




```
#Write the points to a shapefile for use in QGIS
#sf::st_write(camera_points_vic, "C:/Data/camera_points_vic.shp")
```

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

Drive time analysis

Background on drive times

- 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,
```

Method 1: OSRM package

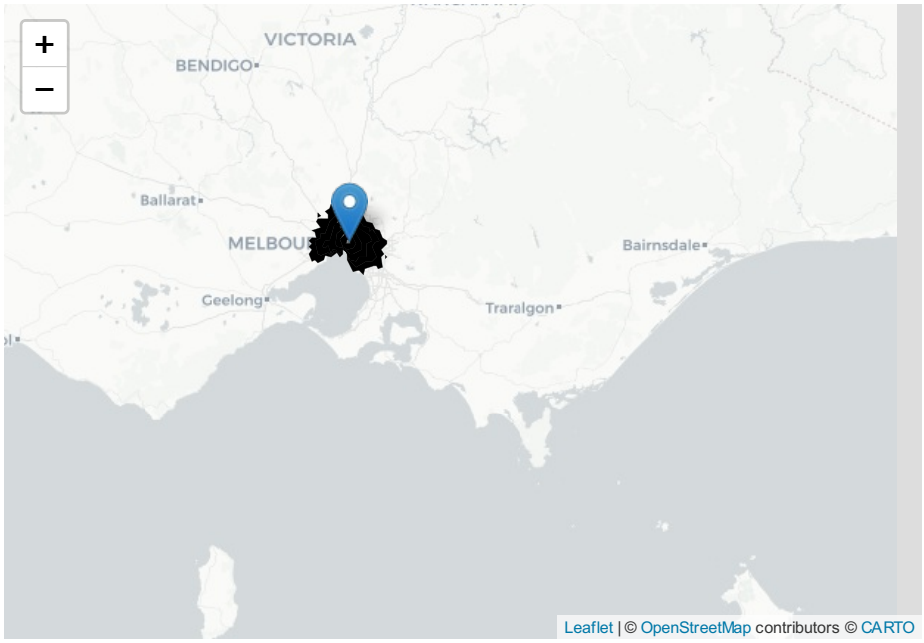
Useful links for further reading: Source 1, Source 2

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.

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

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

#iso@data$drive_times <- factor(paste(iso@data$min, "to", iso@data$max, "mins"))

#factpal <- colorFactor("RdYlBu", iso@data$drive_times)

leaflet() %>%
  setView(mean(locations$lon), mean(locations$lat), zoom = 7) %>%

  addProviderTiles("CartoDB.Positron", group="Greyscale") %>%

  addMarkers(lng = locations$lon, locations$lat) %>%

  addPolygons(fill=TRUE, stroke=TRUE, color = "black",
              #fillColor = ~factpal(iso@data$drive_times),
              weight=0.5, fillOpacity=0.2,
              data = iso, #popup = iso@data$drive_times,
              group = "Drive Time") #%>%
```




```
  # addLegend("bottomright", pal = factpal,
  #                          values = iso@data$drive_time,
  #                          title = "Drive Time")
```

We can export these polygons as shapefiles for use in QGIS or other spatial programs.

```
#Write the points to a shapefile for use in QGIS
#sf::st_write(iso, "C:/Data/iso_polygons.shp")
```