

CT5136: Data Visualisation. GGPLOT2 Code Notes

Conor Hayes

2020-03-19

Contents

1	Visualisation with ggplot2	5
2	R Studio	7
2.1	Getting RStudio to run on your computer	7
3	Introducing ggplot2 : Scatterplot	11
3.1	Basic ggplot2 layers	11
3.2	Scatterplot	11
3.3	Assignment 1	23
4	Visualising Amounts	25
4.1	Bar Plots	25
4.2	Dot plots	82
5	Pies and Proportions	93
5.1	The Pie Chart	93
5.2	Bar charts	97
5.3	Waffle chart	113
5.4	Don't forget Tables	114
6	Comparing Proportions	117
6.1	Stacked Bars in Series	117
6.2	The Stacked Area graph	121
6.3	Grouped Bar Chart	124
6.4	Overlapping Area Plot	126
6.5	Proportional Stacked Bar again	129
7	Hierarchical Proportions	135
7.1	Tree maps	135
8	Visualising Time Series Data	141
8.1	Data	141
8.2	Line Graphs	152
8.3	Missing Data	153

8.4 Line graphs emphasise trend	160
8.5 Line graph with a fill	161
8.6 Visualising Multiple Time Series	163
8.7 Using a hidden second y-axis to label series	167
8.8 A faceted approach to multiple time series	171
8.9 A faceted approach with fill	172
8.10 Bar Graphs for emphasizing and comparing individual values	174
8.11 A faceted approach	175
8.12 Stacked Bar Graphs	178
8.13 100% or Proportional Stacked Bar Graph For Time Series	180
8.14 Proportional Stacked Area Graph For Time Series	182
8.15 Steam Graphs	184
8.16 Interactive Steam graph	186
8.17 Handling overplotting with time series data	186
8.18 Using faceting for multiple time series	192
9 Tutorials: Creating a Bubble Plot	197
9.1 Objective: Make bubble plot to examine Life expectancy vs GDP per capita	197
9.2 Loading Data	200
9.3 Load required packages and the data	200
9.4 Create a subset of the data to plot	240
9.5 Create the basic plot	240
9.6 Identify the features you need to fix/adjust with the basic plot	241
9.7 Fix the missing data messages	242
9.8 Add the chart title	242
9.9 Format the axis labels	243
9.10 Axis scale	243
9.11 Colour and overlap	245
9.12 Refining Point Area	249
9.13 Labelling Data points	250
10 Visualising Trends	259
10.1 Representing Trends	259
10.2 Moving Average	261
10.3 Loess Smoother	263
10.4 4 Irish Tech Stocks	265
11 UK Election Polls Case Study	269
11.1 Read in the data	269
11.2 Transform the data to long format	377
11.3 Calculate daily mean poll scores	377
11.4 Make a basic plot	378
11.5 Add a smoothing line	378
11.6 Further steps	380
11.7 Customise colour allocation	380

CONTENTS	5
11.8 Customise the x-axis	381
11.9 Customise the y-axis	382
11.10Customise the legend	383
11.11Customise the gridlines	384
11.12Add vertical lines with labels to represent events	385
11.13Turn ‘clipping’ off	387
11.14Label the final points on the right handside	388
11.15Create a hidden second y-axis	388
11.16Colour the right y-axis text	390
11.17The full code	391
12 Tutorial: Bubble Charts over time (#bubbleseries)	397
12.1 Bubble plot	433
12.2 A Faceted approach	439
12.3 Observations	442
13 Time Data as Ordinal Values	445
13.1 Scottish Independence	445
13.2 Read the data	445
13.3 Convert to long format	447
14 Slope Charts	459
14.1 Reading in the two data sets	459
14.2 Merging the two data sets	468
14.3 Selecting the data	481
14.4 Building a basic slope chart	490
14.5 Comparing Rankings	497
14.6 Colouring Slope lines	500
14.7 Foregrounding a subset of slope lines	503
14.8 Labelling issues	507
14.9 Visualising all councils	507
15 Parallel Sets	515
15.1 Parallel plot 1: Visualising Departure times	515
15.2 Parallel plot 2: adding an extra ‘axis’	519
15.3 Parallel plot 3: reordering the ‘axes’	521
15.4 Alluvial plot	523
16 Time Series Heat Maps	525
16.1 Read the data	525
16.2 Convert epi_week to year and week	528
16.3 Calculate yearly means	531
16.4 Make an initial plot	532
16.5 Order the rows	534
16.6 Plot heatmap with ordered rows	541
16.7 Clustering the rows	543

16.8 Plot the heat map with clustered rows	544
17 Reference Materials	547
17.1 Colour Table	548
17.2 Shapes	549

Chapter 1

Visualisation with ggplot2

Chapter 2

R Studio

In this chapter, we'll focus on how you can install RStudio on your computer. If you already have RStudio installed and if you're an experienced R user already, feel free to skip this chapter.

2.1 Getting RStudio to run on your computer

As a prerequisite for this guide, you need to have RStudio and a few essential R packages installed.

You have to follow these steps to get your RStudio version set.

1. Download RStudio on the RStudio Website.
2. If you do not have R installed yet, you will have to install the latest R Version before you can use RStudio.
3. You can get the latest R version here.
4. You have to choose the right R version depending on the operating system you use (i.e., Windows or Mac).

Once RStudio is running, open the Console on the bottom left corner of your screen.

We will now install the tidyverse packages, which are a large bundle of packages which make it easy to manipulate and visualize data in R. Functions included in the tidyverse have become very popular in the R community in recent years, and are used by many researchers, programmers and data scientists. If you want to learn more about the tidyverse, you can click on this link: <https://www.tidyverse.org/>.

5. To install these packages, we use the `install.packages()` function in R.

```
install.packages("tidyverse")
```

```

Console | Terminal x
~/1920-CT5136-Dataviz/Test/ ↵

R version 3.5.2 (2018-12-20) -- "Eggshell Igloo"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> install.packages("tidyverse")

```

Figure 2.1: The R Studio Console

In RStudio, you simply have to type in the code displayed above into the the Console (usually, this is the bottom-left pane in your RStudio window) next to the little arrow (>) in the last line. Then hit Enter

We will have to install and load a few other packages throughout the module, for which you can use the `install.packages()` the same way as you did before.

2.1.1 Running R or R Markdown

You can write or run your code in an R window on RStudio or in an R markdown file. There are multiple tutorials on the web. I suggest that you get used to writing your code in a R markdown file, as it will allow you to produce short reports interspersed with graphics.

The figure below illustrates the steps in Studio to create, write and render (knit) an R markdown document.

I find it useful to create an R project in which I keep my R and R mark down files for a particular piece of work. In the lecture video in week 2, I show quickly how to do this.

2.1.2 Getting Help

Most R packages come with extensive documentation showing you how to use functions. In RStudio, it is easy to access the documentation: in the Console, simply put a question mark in front of the function you want to use, and then hit Enter. If a documentation file for the function exists, a page should open in the Help (bottom-right corner) pane of your RStudio window. This page will then show the arguments required to use the function, and what the function does. As an example, you can write `?mean` into your Console. If you then hit Enter, the documentation for the `mean()` function should be displayed in the Help pane.

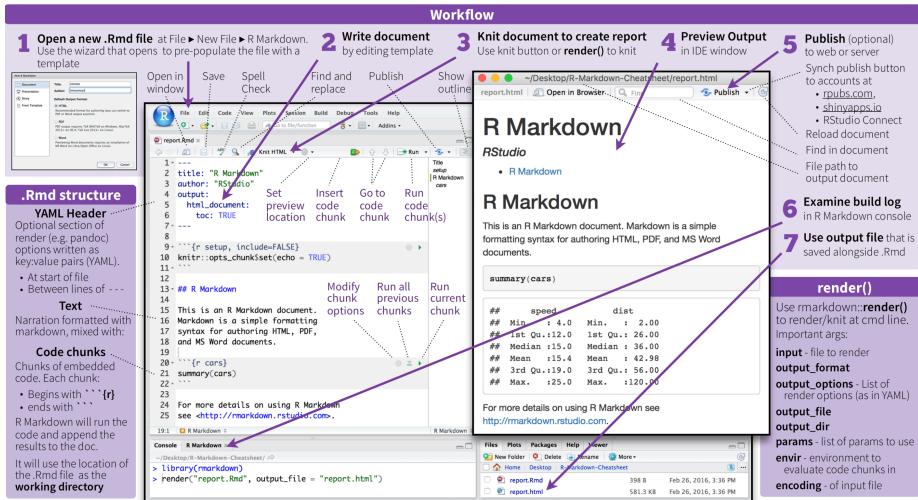


Figure 2.2: The steps to creating and running R code in a R Markdown file

If you get stuck somewhere, the best way is to start with a Web Search. If you get an error message and you have no idea what to do with it, try a web search first. The chances are that someone else will have puzzled over the same error.

If you are still stuck, post a message to the module discussion board with as much detail as possible.

Chapter 3

Introducing ggplot2 : Scatterplot

3.1 Basic ggplot2 layers

Every ggplot2 plot has 3 key components - Data - Aesthetic Mappings between variables in the data and visual properties - Geometrics Layer (geom) that describes what visual elements are used to visually render the data.

In terms of the other layers, ggplot2 uses sensible default settings, which we will learn to override and customize.

3.2 Scatterplot

A scatterplot is simple graph in which the values of two variables are plotted along two axes. The resulting visualisation may reveal a correlation between the variables. In this session you will learn how to put together a a scatter plot like the one shown here.

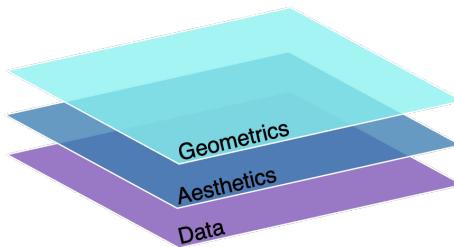


Figure 3.1: The RStudio console

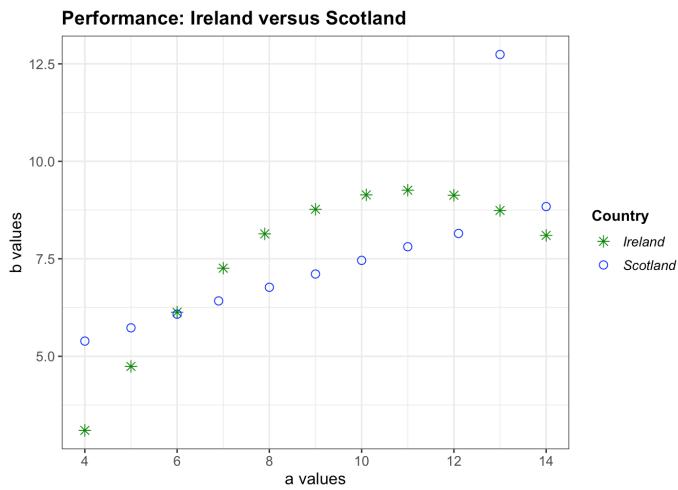


Figure 3.2: The scatterplot we will draw today

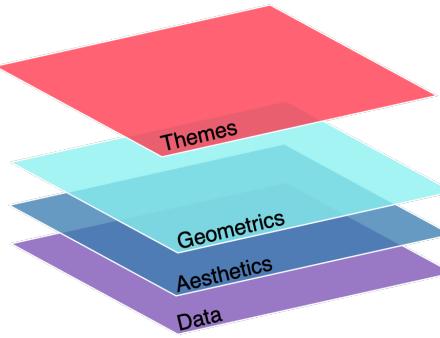


Figure 3.3: The scatter plot we will draw will have 4 layers

A scatterplot probably the easiest graph to draw in ggplot. There will be a short exercise at the end that will demonstrate another positive aspect of visualisation that I haven't touched upon yet.

The code here should be inserted into your own R Markdown document. I strongly advise that you do not just cut and paste the code. Attempting to write the code yourself will help you become familiar with writing code in R.

The scatterplot we will draw uses 4 layers from the grammar of graphics — the obligatory first 3 layers of data, aesthetics and geometrics plus the themes layer. The themes layer enables us to tweak and customise the plot so it looks exactly the way we want it.

3.2.1 Reading the data

To start with we read in data into the data layer. The data I have provided is in .csv format (comma separated values). We can use the the `read.csv` function from the R base library (or `read_csv` which is part of the Tidyverse `readr` package). Both functions make a good guess at the data types of the data in the csv file. We can check this afterwards and correct any mis-typed data

```
data1 <- read.csv("../data/week2_data_2cat.csv")
# str gives the type of the data container (a data.frame), the variables and types
str(data1)

## 'data.frame':    22 obs. of  3 variables:
## $ var_a : num  10.1 7.9 13.0 9.11 14.6 ...
## $ var_b : num  9.14 8.14 8.74 8.77 9.26 ...
## $ country: Factor w/ 2 levels "ireland","scotland": 1 1 1 1 1 1 1 1 1 1 ...
# head shows the first few rows of the data
head(data1)

##   var_a var_b country
## 1 10.1  9.14 ireland
## 2  7.9  8.14 ireland
## 3 13.0  8.74 ireland
## 4  9.0  8.77 ireland
## 5 11.0  9.26 ireland
## 6 14.0  8.10 ireland
```

The data consists of three variables. As the `str` function shows `var_a` and `var_b` are continuous variables and `country` is a factor.

3.2.2 Defining the aesthetic mapping

The aesthetic mapping for the first version of our scatter plot will map the values of `var_a` to the scale of the x axis and the values of `var_b` to the scale of the y-axis. We will ignore `country` for now. The `aes()` function is used to specify the X and Y axes.

```
# we first tell R that we will use functions from the ggplot2 package
library(ggplot2)

# the ggplot function takes the data and aesthetic mapping function aes as arguments
g <- ggplot(data1, aes(x=var_a, y=var_b))

g
```

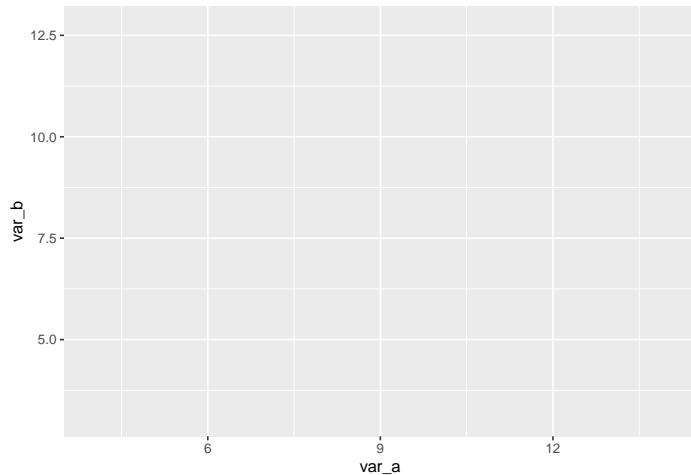


Figure 3.4: blank graph

3.2.3 Adding a geometrics layer

A blank ggplot is drawn. Even though the x and y are specified, there are no points or lines in it. This is because, we have only created the first two layers - data and aesthetic mapping. We now need to tell ggplot the types of geometrics we want to use to visually represent the values of our variables. In ggplot these are known as **geoms** and every geom is prefixed by **geom_**

For a list of the possible geoms see [The Geoms Layer](#)

We will use **geom_point** which will place a point on the graph at each (**var_a**, **var_b**)

Note how we add the **geom_point** using the **+** operator.

```
g <- ggplot(data1, aes(x=var_a, y=var_b)) +
  geom_point()

g
```

3.2.4 Adding an additional aesthetic mapping

We have a very basic scatterplot, where each point represents a **var_a** and **var_b** value. We will now add another aesthetic mapping to handle the **country** factor. This variable has two values **green** and **blue**. We will map these to two colours. At first we will allow ggplot to assign the colours using its defaults. Then we will override the colours assigned.

```
g <- ggplot(data1, aes(x=var_a, y=var_b, colour=country)) +
  geom_point()
```

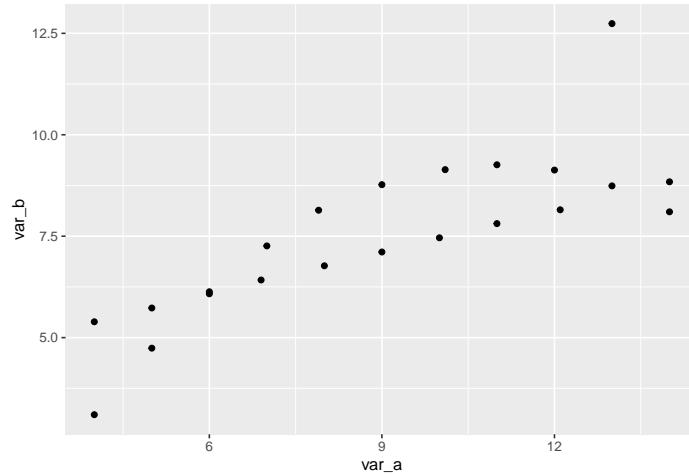


Figure 3.5: a very basic scatter plot

g

3.2.5 Changing the default colours

The colour aesthetic mapping visually groups the instances with the same level value. For easier interpretation, we will make the points with the level `ireland` green and those with the level `scotland` blue. To do this, we have to manually assign the colours by specifying a vector of colours, in this case, of size 2. We do this by passing the vector of colours to the `scale_colour_manual` function. In general, the functions of types `scale_`x`_manual` enable you to override the default mappings ggplot makes to the visual scale of the aesthetic. You can see the range of colours available in Section 17.1

```
g <- ggplot(data1, aes(x=var_a, y=var_b, colour=country)) +
  geom_point(size=2.5) # Note that I've increased the point size
  # This function allows you to manually override the default colour mappings defined by the aes
  scale_colour_manual(values = c("green4","blue"))
```

g

We've created a basic scatterplot. We can see that the distribution of blue values is quite different from green values. At one value, two points overlap ($x = 6$). It would be better if we could make the partially obscured green point a bit more visible. There are a few ways to achieve this. We will look at changing the shape and changing the alpha value.

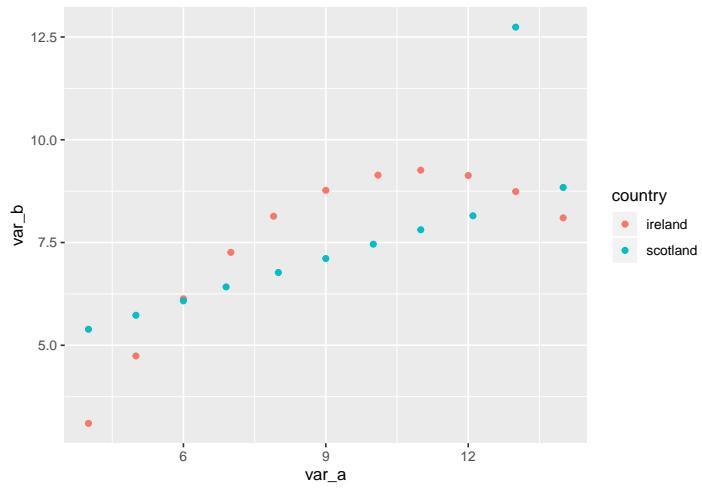


Figure 3.6: scatterplot with points coloured by country value

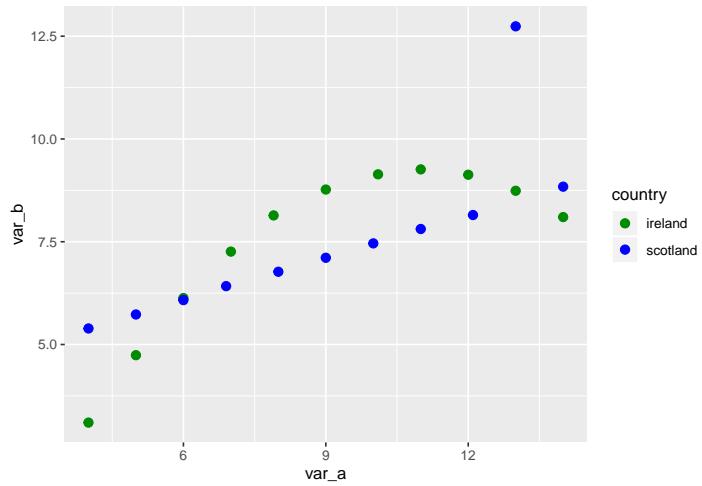


Figure 3.7: scatterplot: manually assigning colours

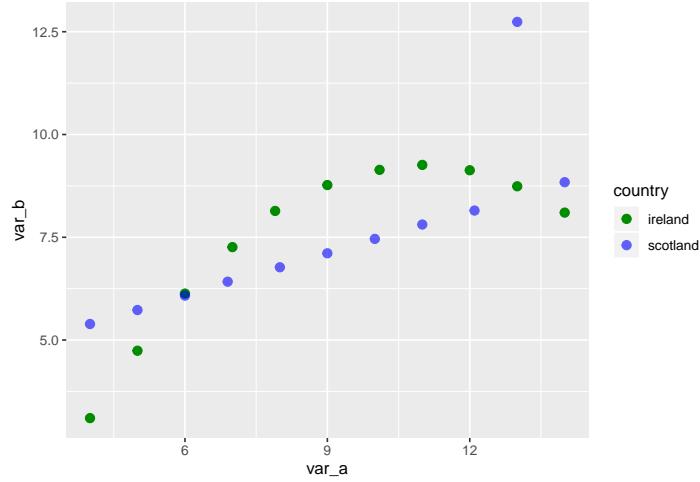


Figure 3.8: Setting alpha as an aesthetic

3.2.6 Changing alpha values

The alpha value controls the transparency of the geometric. Alpha values are in the range (0,1) where 0 means that the point is completely transparent or invisible. In this case, we want to lower the alpha value of the blue points. To manually set the transparency values of two sets of points to different values, we have to treat alpha as an aesthetic that is mapped to the variable that separates the points. Here the points with the `country` value of `blue` should have a lower alpha than points with `country` value `green`. Therefore, we map the alpha scale to the values of the variable `country`. We then override the default mappings using a `scale_alpha_manual` function, just as we did when manually setting the colour of the points. In Figure 3.8, the green point at $x = 6$ is now much more visible than in Figure 3.12

```
g <- ggplot(data1, aes(x=var_a, y=var_b, colour=country, alpha= country)) +
  geom_point(size=2.5) +
  scale_colour_manual(values = c("green4", "blue")) +
  scale_alpha_manual(values = c(1, 0.6))

g
```

3.2.7 Changing Geom Shape

Another approach would be to change the shape of the points so that, even when they overlap on the graph, we can still perceive two different points.

Any `geom_point` can be specified by a number of different shapes, shown in Figure 3.9. All shapes have a `colour` attribute which can be set. Shapes 21 to

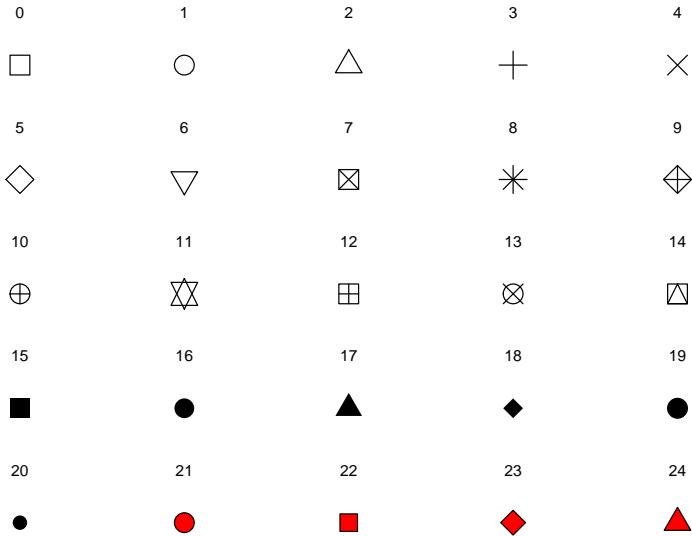


Figure 3.9: The shapes available to ggplot

24 also have a `fill` property, which means that the `colour` property controls their outline and the `fill` property controls the colour within the outline.

To distinguish blue from green shapes even more strongly we map the factor `country` to the shape aesthetic with the `aes` function, just as we did with the colour and alpha aesthetics. Then using the `scale_shape_manual` function we override the default aesthetic mapping (from the values of `country` to shape types) and specify the shapes we want to represent the values of the `country` variable.

In Figure 3.10 we set the blue points to be an unfilled circle (shape 1) and green points to be an asterix (shape 8). We could also combine this with an adjustment to alpha if an overlapped point still not clear.

```
g <- ggplot(data1, aes(x=var_a, y=var_b, colour=country, shape=country)) +
  geom_point(size=2.5) +
  scale_colour_manual(values = c("green4","blue")) +
  scale_shape_manual(values = c(8, 1))

g
```

3.2.8 Adjusting the themes layer

Note that we are still dealing with the first 3 layers. For this visualisation, we are using the defaults supplied by ggplot for the other layers. We will now make some changes to the themes layer. This is the non-data layer and is concerned

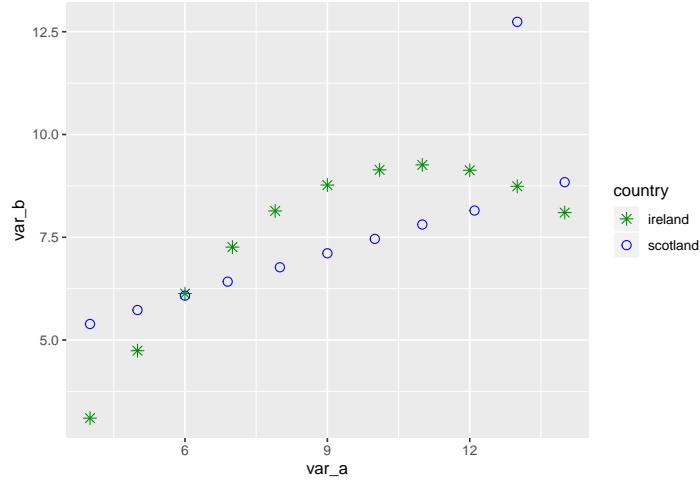


Figure 3.10: Setting shape as an aesthetic

with making the graph clearer and better looking. We will do the following :

3.2.8.1 add a title

use the `ggtitle` function

3.2.8.2 change the labels on the x and y axis

use `xlab` and `ylab` functions

3.2.8.3 place more breaks in the x axis

as the x axis uses a continuous scale, we make adjustments to its default layout using the `scale_x_continuous` function. In this case we set the `breaks()` attribute using a vector of custom values

3.2.8.4 Change the background

we can change the grey background by specifying the background colour as white in the `theme` function. However, as the default grid-lines are also white, we lose sight of them against the white background. We can specify the colour of the grid-lines as `lightgrey` as I have done below.

3.2.8.5 Change the size of the axis text

The default size for the axis text is often too small in my opinion. You can change it within the `theme` function using `axis.text = element_text(size = 10)`

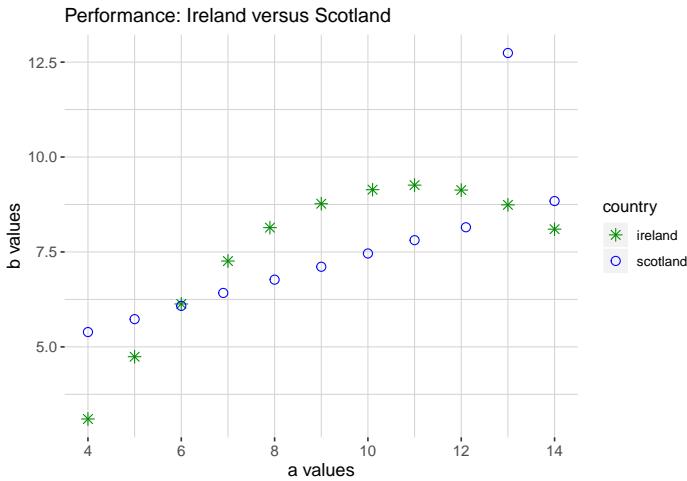


Figure 3.11: Manually adjusting the theme

3.2.8.6 Change the size of the axis title

Again I find that the default size of the axis title is often too small. In this case, I change it to size 12.

However, if I am in a hurry, I will use one of the pre-rolled themes supplied with ggplot, which I do in Figure 3.12

```

g <- ggplot(data1, aes(x=var_a, y=var_b, colour=country, shape=country)) +
  geom_point(size=2.5) +
  scale_colour_manual(values = c("green4","blue")) +
  scale_shape_manual(values = c(8, 1)) +
  ggtitle("Performance: Ireland versus Scotland") +
  xlab("a values") + # new label for x axis
  ylab ("b values") + # new label for x axis
  scale_x_continuous(breaks= c(4,6,8,10,12,14)) +
  theme(panel.background = element_rect(fill = "white"),
        panel.grid.major = element_line(size = 0.25, linetype = 'solid',
                                         colour = "lightgrey"),
        panel.grid.minor = element_line(size = 0.1, linetype = 'solid',
                                         colour = "lightgrey"),
        axis.text = element_text(size =10),
        axis.title = element_text(size =12))
g

g <- ggplot(data1, aes(x=var_a, y=var_b, colour=country, shape=country)) +
  geom_point(size=2.5) +
  scale_colour_manual(values = c("green4","blue"))

```

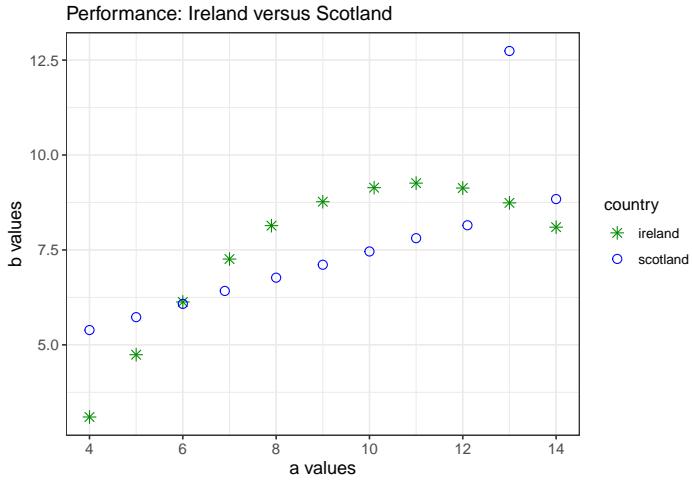


Figure 3.12: Using a pre-rolled theme

```

scale_shape_manual(values = c(8, 1)) +
  ggtitle("Performance: Ireland versus Scotland") +
  xlab("a values") + # new label for x axis
  ylab ("b values") + # new label for x axis
  scale_x_continuous(breaks= c(4,6,8,10,12,14)) +
  theme_bw() + # prerolled theme
  theme (axis.text = element_text(size =10),
         axis.title = element_text(size =12)) # I still want to override the default axis text size

g

```

3.2.8.7 Final touches : Adjust the legend Title

The final adjustment I want to make is to the legend. The legend title is the name of the variable that is assigned to the aesthetic illustrated by the legend. In this case, we mapped the `country` variable to the colour and shape aesthetics, and ggplot automatically creates a legend to illustrate this mapping. I want the title of the legend to be capitalised — Country rather than country. There are a few ways to do this in ggplot. The way I most commonly use is to use `scale_x` to define the title using the `name` attribute of the `scale_x` function I have used to define the aesthetic scale used. In the example, we have used two such functions `scale_colour_manual` and `scale_shape_manual`. In order to change the title of the legend, we have to set the `name` attribute in both these functions (See what happens if you set the `name` value in just one of these functions).

3.2.8.8 Adjusting the legend labels

By default the labels within the legend are the values of the variable that has been mapped to the aesthetic(s) being illustrated by the legend. In our example, the `country` factor has two levels : `ireland` and `scotland`. Whilst these two values are immediately understandable, in other situations the value might be an abbreviation or a code and you may want to overwrite the resulting label in the legend to something more intelligible. In this case, I am just going to capitalise the labels to `Ireland` and `Scotland`. To do this, I will set the value of the `labels` attribute in the `scale_x` functions to a vector of custom labels. Remember that the `scale_x` functions are the functions where we explicitly map the values of a variable to particular aesthetics. The legend is a visual explanation of such a mapping, and the values of the `name` and `labels` attributes of the `scale_x` functions we use are reproduced in the legend of the resulting plot.

3.2.8.9 Adjusting the legend label sizes

I think the legend label font size is a little small. A fairly minor issue - but I will change it to illustrate the control you have over every visual element. Changing font type or size is always handled by the themes layer (and the `theme` function). Here you can see that within the `theme` function, I explicitly specify the values of the `legend.title` and `legend.text` attributes

```
g <- ggplot(data1, aes(x=var_a, y=var_b, colour=country, shape=country)) +
  geom_point(size=2.5) +
  scale_colour_manual(values = c("green4", "blue"), name = "Country", labels=c("Ireland",
    "Scotland")) +
  scale_shape_manual(values = c(8, 1), name = "Country", labels=c("Ireland", "Scotland"))
  ggttitle("Performance: Ireland versus Scotland") +
  xlab("a values") + # new label for x axis
  ylab ("b values") + # new label for x axis
  scale_x_continuous(breaks= c(4,6,8,10,12,14)) +
  theme_bw() + # prerolled theme
  theme (axis.text = element_text(size =10),
         axis.title = element_text(size =12),

         legend.title = element_text(face="bold",
                                      size=11),
         legend.text = element_text(face="italic",
                                    size=10),
         plot.title = element_text(face="bold",
                                   size=14))

g
```

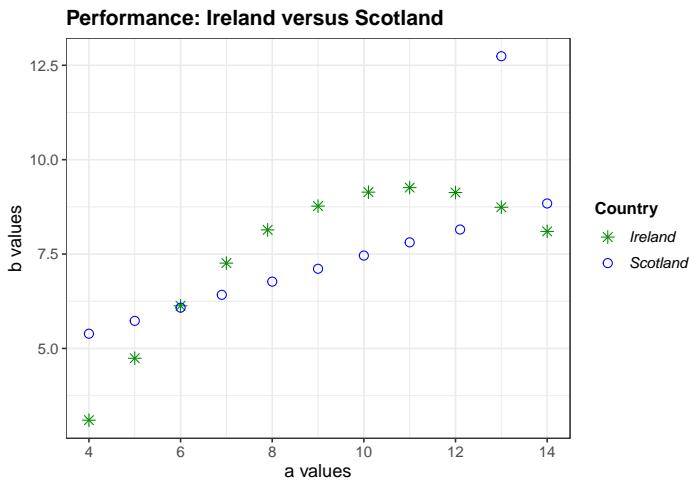


Figure 3.13: Adjusting Legend Title and Labels

3.2.9 Saving the plot to file

To save the plot, you can use `ggsave()`. It defaults to saving the last plot that you displayed, using the size of the current graphics device. It also guesses the type of graphics device from the extension. There are several options to specify the image size, dpi, output directory. However, the defaults will save the file to the current working directory using the size of the current graphics device. In the example, below I am saving the last rendered plot as a png file to my current working directory.

```
ggsave("ireland_scotland_scatter.png")
```

3.3 Assignment 1

On Blackboard, I have provided a link to another dataset `week2_data_4cat.csv`. This dataset is similar to today's data set except that I've added some extra rows of data. The country factor now has 4 levels : ireland, scotland, wales, england.

There are two parts to this exercise:

3.3.1 Part 1

Using the techniques, you've learned today I want you to create a scatterplot visualisation that clearly presents the data points highlighted according to the values of the country variable. There will be some overlapping points so you will need to use shape and alpha to make the points as clear as possible. You should

use appropriate colours that are visually seperable and meaningful. You may also adjust - theme values to make the plot as clear and attractive as possible.

3.3.2 Part 2

For each set of instances associated with each country value, calculate the mean, standard deviation of the `var_a` and `var_b` variables and their correlation. There are many ways in R to calculate the statistics of subsets of data. The example code below shows one way. The sources for this code are from here and here

Comment on the statistics calculated and their relationship to the visualisation you've produced. What extra information, if any, does the visualisation give you.

In the code section below, I've provided the code that can calculate the statistics. This uses the very handy `dplyr` library to calculate the stats per group, where each group is defined by a `country` value. The output is a dataframe, which can be used as input to several of the table formating packages in R.

3.3.2.1 Making a Table

As part of this assignment, I want you to produce a formatted table of values using an R package. There are multiple examples on the Web. While tables are not strictly part of visualisation, you should know how to produce a clear table. A good table presents information better than a bad visualisation. Furthermore, you should become used to using the Web as a source of information for this topic. Every visualisation or data formatting problem I have ever encountered has been answered previously on a Website such as StackOverflow.

For the information on formatting tables, you could look at the following two sites:

https://rpubs.com/tf_peterson/kableDemo

https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html#installation

```
library(dplyr)
data1_stats<- data1 %>%
  group_by(country) %>%
  summarize(cor_ab=cor(var_a,var_b), mean_a = mean(var_a), mean_b = mean(var_b), sd_a =
```

Chapter 4

Visualising Amounts

We will look at how to represent variation with category values with the following charts

- Bar charts
- Diverging Bar charts
- Cleveland Dot plots
- Lollipop plots
- Pareto Charts

4.1 Bar Plots

We started this week's lecture by looking at a table of film revenue figures, which I downloaded from scannan.ie

- Irish Box Office Top 10, Weekend: October 25th to 27th, 2019

```
ire_box_office <- read.csv("../data/ire-movie-october25-2019.csv")  
  
# quick display of variable abd variable types  
str(ire_box_office)  
  
## 'data.frame': 10 obs. of 5 variables:  
## $ rank : int 1 2 3 4 5 6 7 8 9 10  
## $ title : Factor w/ 10 levels "Abominable","Countdown",...: 5 9 6 8 1 7 10 2 4 3  
## $ title_short : Factor w/ 10 levels "Abominable","Addams Family",...: 6 2 7 9 1 8 10 3 5 4  
## $ weekend_gross: int 374216 254192 163910 153624 74161 66844 50149 38760 36500 35663  
## $ gross : int 5124763 254192 511974 196706 74161 206742 201983 38760 140812 2848074
```

We'll put this data into tabular form first. I want to show the following fields: *rank*, *film title*, *weekend gross* (in euro). The easiest way to create a formatted

euro amount is to use the `dollar_format` function using the `scales` library. Yes, `dollar_format` to create a euro symbol!

Using, an adaption of this function to insert the euro sign, I create another column in the data `weekend_euro`

```
library(scales)
ire_box_office$weekend_euro <- dollar_format(prefix = "\u20ac", big.mark = ",") (ire_box...
```

I will use the kable library to format the table. First, Using `dplyr` I select the columns I want to show into a new data_frame.

```
library(dplyr)

boxoffice_ire_display <- ire_box_office %>%
  select(rank, title, weekend_euro,) %>%
  rename(Rank = rank,
        Title = title,
        `Weekend gross` = weekend_euro)
```

I use the kableExtra library to create a nice looking table. There is a good tutorial [here] (https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html) on using Kable and kableExtra. It contains the word ‘awesome’ in its title so you know it must be good.

```
library(knitr)
library(kableExtra)

boxoffice_ire_display%>%
  kable(align = c('c', 'l', 'c'), caption = 'Highest grossing movies in Ireland for the year 2014') %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"), full_width = FALSE)
```

Data where we have ranked amounts is commonly visualized with vertical bars of different lengths. For each movie, we draw a bar that starts at zero and extends all the way to the value we wish to represent. This type of visualization is called a *bar plot* or *bar chart*.

In ggplot terms, the aesthetic that is used is the *length* of a column. As such the geometric we use is `geom_col`.

You can create a basic bar plot here using the first 3 layers of grammar of graphics. As you can see, there is still some work to do before this plot is presentable. We will now step through what is required

I have used the `theme_set` function to set the overall theme to be `theme_classic`. I could have done this by simply adding `+ theme_classic()` to the plot. This clears away all default gridlines, gives white background to the plot.

Table 4.1: Highest grossing movies in Ireland for the weekend of October 25-27, 2019. Data source: Scannain.ie

Rank	Title	Weekend gross
1	Joker	€374,216
2	The Addams Family	€254,192
3	Maleficent: Mistress Of Evil	€163,910
4	Terminator: Dark Fate	€153,624
5	Abominable	€74,161
6	Shaun The Sheep Movie: Farmageddon	€66,844
7	Zombieland: Double Tap	€50,149
8	Countdown	€38,760
9	Hocus Pocus	€36,500
10	Downton Abbey	€35,663

It makes sense to have some gridlines to help establish the values associated with each bar. I just use the major gridlines for the *y axis*, which can be defined by the `panel.grid.major.y` parameter within the `theme` function.

I'd like to be able to see the gridlines behind each bar - for the sake of visual continuity. I set the `alpha` value of the `geom_col` geometric to be 0.8. As you may recall, the alpha value defines the transparency of a visual object.

```
library(ggplot2)

theme_set(theme_classic())

ggplot(ire_box_office, (aes(x= title_short, y=weekend_gross))) +
  geom_col(alpha=0.85) +
  theme(panel.grid.major.y = element_line(size = 0.2, linetype = 'solid', colour = "lightgrey"))
```

4.1.1 Ordering the bars

We generally want to order the bars by their value on the y-axis.

We use the `reorder` function to reorder the bars by their euro value rather than by the default alphabetical ordering which is used by R to order a factor. An alternative approach would be to explicitly set the order of the levels in the factor; In this case, we would set the order of film titles in the `title_short` factor.

The `reorder` function sorts in ascending order by default. To sort in descending order, just make the make the variable you wish to order negative

```
library(ggplot2)
```

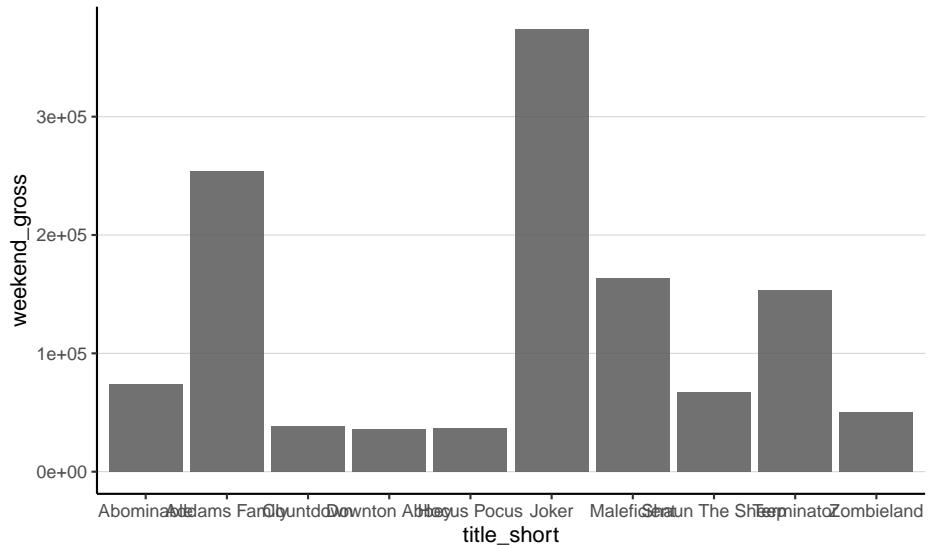


Figure 4.1: A basic bar plot

```
theme_set(theme_classic())

ggplot(ire_box_office, (aes(x= reorder(title_short, -weekend_gross) , y=weekend_gross))
  geom_col(alpha=0.85) +
  theme(panel.grid.major.y = element_line(size = 0.2, linetype = 'solid', colour = "lightgrey"))
```

One problem we commonly encounter with vertical bars is that the labels identifying each bar take up a lot of horizontal space. To save horizontal space, we could place the bars closer together and rotate the labels.

This is non-data related adjustment and we do this in the `theme` function by setting the `angle` and `vjust` and `hjust` values of the `axis.text.x` parameter.

Note that I have removed the *x-axis* title and the axis line on both the x and y axes. Again, this was done in the `theme` function. Setting the value of any parameter in the `theme` function to `element_blank` will make the corresponding visual feature disappear.

The `width` parameter in the `geom_col` function allows you to specify the bar width - and in this case, I have set the width to 0.7 of the default. Try changing it to see the effect it has.

```
library(ggplot2)

theme_set(theme_classic())
```

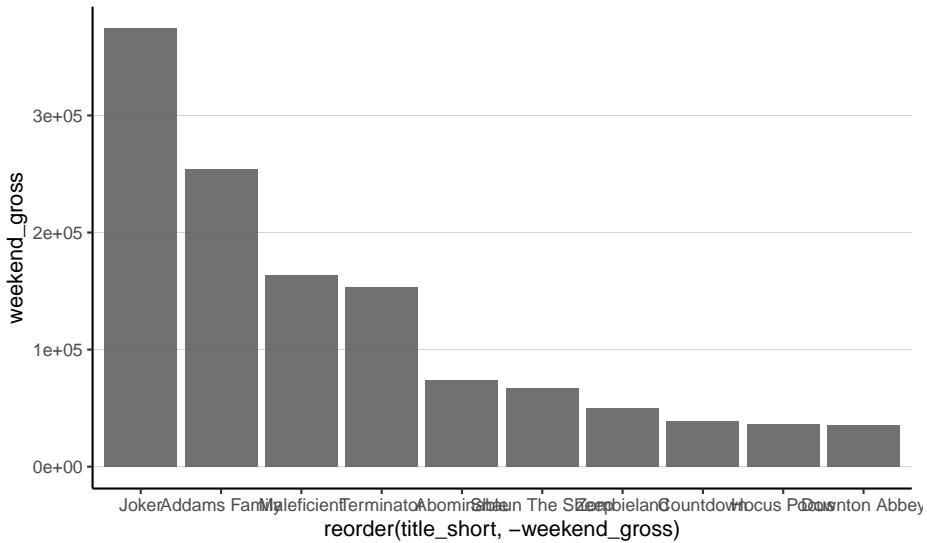


Figure 4.2: Bar plot with bars correctly ordered

```
ggplot(ire_box_office, (aes(x= reorder(title_short, -weekend_gross) , y=weekend_gross))) +
  geom_col(alpha=0.85, width = 0.7) +
  theme(
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1),
    axis.title.x = element_blank(),
    plot.margin = margin(3, 6, 3, 3),
    panel.grid.major.y = element_line(size = 0.2, linetype = 'solid', colour = "lightgrey")
  )
```

We still need to adjust the *y-axis*. The scale uses scientific notation, which in this case is not appropriate. It also has a title that needs reformatting. We can solve these issues by customising the *x-axis* scale. The function to use for this is `scale_x_continuous`.

Within this function, I specify the axis breaks I want displayed with a vector of evenly spaced values. I also specify how I want these values to be labelled. This overrides the default breaks and labels imposed by ggplot.

Often an axis becomes crowded and ugly with large numbers, hence the use of

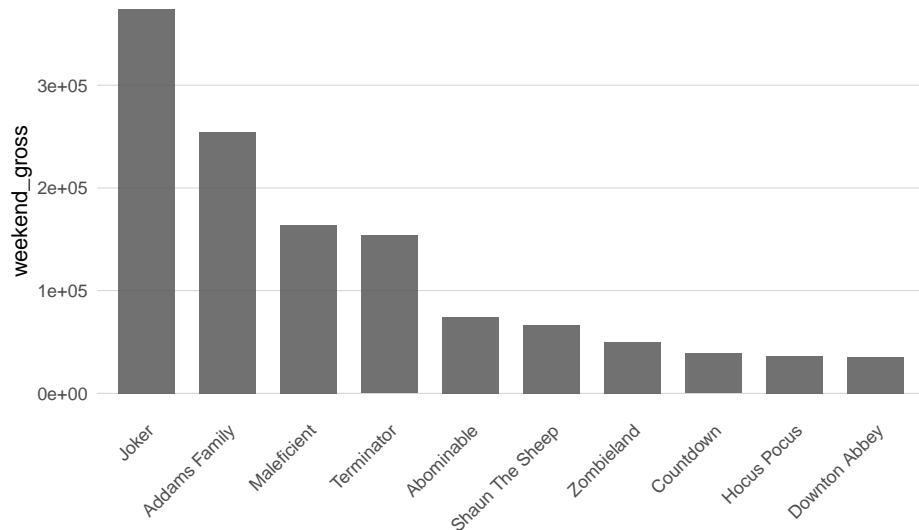


Figure 4.3: A bar graph with the x-axis labels rotated by 45 degrees

the more compact scientific notation. When you don't want to use this type of notation, you can specify the axis units to be in millions or thousands as I do here. I also specify the name or label I want for the y axis.

In this plot, I use a suggestion on bar plots from Edward Tufte - made in the spirit of reducing ink. I've made the gridlines white. Now gridlines are part of the overall background panel in a ggplot. If I bring the overall panel to the fore using `panel.on top = TRUE` in the `theme` function, the background panel also comes to the fore and obscures the plot. To deal with this issue, you set `panel.background` to be blank.

Using white gridlines eliminates the gridlines on the plot that do not intersect with the bars, and removes the ticks. It also has the effect of breaking bars into visible units based on the break intervals. This allows for visual quantification of the difference between bars. For example it's easy to see that the lowest ranking films are less all less than one unit high (< 50k), and that *Maleficent* and *Terminator* are both just over 3 units high (> 150k)

```
library(ggplot2)

theme_set(theme_classic())

ggplot(ire_box_office, (aes(x= reorder(title_short, -weekend_gross) , y=weekend_gross))
  geom_col( width=0.7) +
  scale_y_continuous(limits = c(0, 3.8e5),
```

```

breaks = c( 5e4, 1.0e5, 1.5e5, 2.0e5, 2.5e5, 3e5, 3.5e5 ),
labels = c("50", "100", "150", "200", "250", "300", "350" ),
name = "weekend gross (x €1,000)" +

ggtitle("weekend gross (x €1,000) - Oct 25-27 2019") +

theme(
  axis.title.y = element_blank(),
  axis.line.y = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1),
  axis.title.x = element_blank(),
  plot.title = element_text(vjust = -10, hjust = 0.25, size = 11),
  plot.margin = margin(6, 6, 3, 3),
  panel.background = element_blank(),
  panel.grid.major.y = element_line(size = 0.4, linetype = 'solid', colour = "white"),
  panel.on top = TRUE
)

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

```

```
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
```

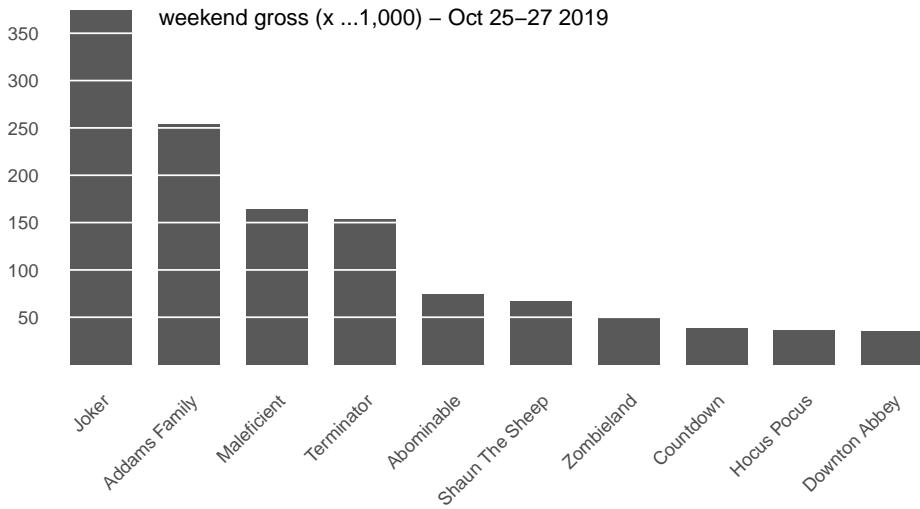


Figure 4.4: Bar graph with formatted y-axis

```

## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
```

The gridlines or ('Tufte' lines that are drawn here) help in guiding your eye to the from the top of each bar to the corresponding horizontal location on the y axis. As presented, the plot gives us the ranking of the movies and an approximation of the gross revenues each generated. We also get a sense of the comparative differences in revenue for these movies.

If we really needed more precision in presenting the actual revenue values per movie, we could insert the value over the bar and drop the *y-axis* and gridlines entirely. However, the visualisation does lose something in terms of being able to see at a glance the comparative difference in revenue values per movie.

We use the `geom_text` geometric to realise this. This demonstrates that an additional geom level aesthetic mapping is also possible. Here we specify that the `label` property of `geom_text` is to be associated with the value of `weekend_euro`, the variable I created earlier to hold formatted euro values of the `weekend_gross` values in the dataset. The `geom_text` elements will be placed at the *x*, *y* locations specified by the plot level aesthetic mapping, and the value of each `geom_text` point will be taken from `weekend_euro`.

I specify the size of the text and make a slight vertical adjustment with `vjust` so that the text hovers over the bar instead of just inside it.

To remove the *y-axis*, I have removed the `scale_y_continuous` function and in the `theme` function, I have set `axis.title.y` and `axis.text.y` to `element_blank`.

I have added a title using `ggtitle` and moved it with the plot area, using the `plot.title` parameter in `theme` so as to make clear what the numbers represent.

```
library(ggplot2)

theme_set(theme_classic())

ggplot(ire_box_office, (aes(x= reorder(title_short, -weekend_gross) , y=weekend_gross)
  geom_col(width=0.7) +
  geom_text(aes(label=weekend_euro), size =2.5, vjust=-0.4) +
  ggtitle("weekend gross - Oct 25-27 2019 ") +
  theme(
    axis.title.y = element_blank(),
    axis.text.y = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1),
    axis.title.x = element_blank(),
    plot.margin = margin(6, 6, 3, 3),
    plot.title = element_text(vjust = -10, hjust = 0.2, size = 11)
  )
  ## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
  ## conversion failure on '€374,216' in 'mbcsToSbcs': dot substituted for <e2>
  ## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
  ## conversion failure on '€374,216' in 'mbcsToSbcs': dot substituted for <82>
  ## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
  ## conversion failure on '€374,216' in 'mbcsToSbcs': dot substituted for <ac>
```

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€254,192' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€254,192' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€254,192' in 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€163,910' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€163,910' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€163,910' in 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€153,624' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€153,624' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€153,624' in 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€74,161' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€74,161' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€74,161' in 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€66,844' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€66,844' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€66,844' in 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€50,149' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€50,149' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€50,149' in 'mbcsToSbcs': dot substituted for <ac>
```

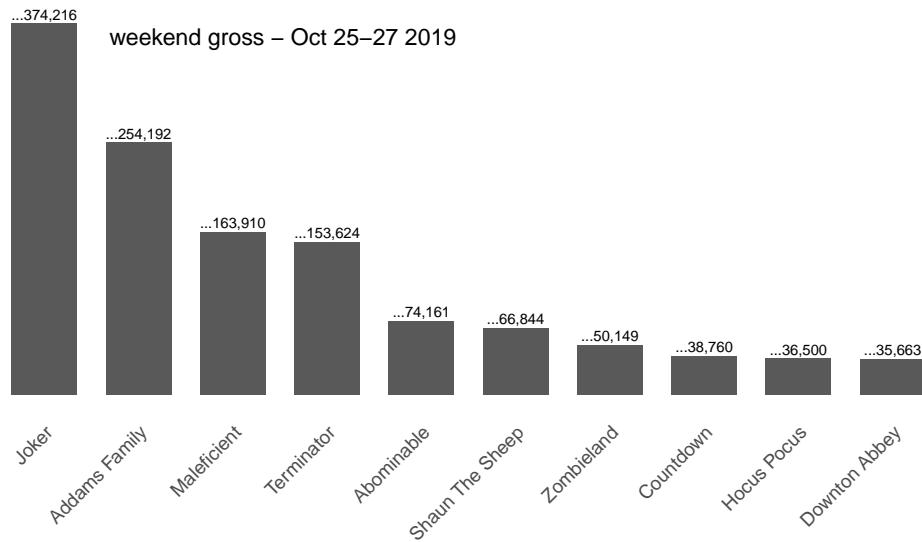


Figure 4.5: A bar graph with value labels over the bars

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€38,760' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€38,760' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€38,760' in 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€38,760' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€38,760' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€38,760' in 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€36,500' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€36,500' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€36,500' in 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€36,500' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€36,500' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€36,500' in 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€35,663' in 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€35,663' in 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on '€35,663' in 'mbcsToSbcs': dot substituted for <ac>
```

While the film titles are now readable, there is a disadvantage in that the plot is vertically longer than it needs to be, and the reader has to slightly tilt their

reading angle to read the titles on the x-axis.

An alternative is to swap the *x* and the *y axis*, so that the bars run horizontally. This is often done where there are more than, say, 6 categories to visualise.

Adding The `coord_flip` function will swap the axes. As the film titles are now on the y axis and the revenue values are on the x axis, I have to also reinstate the *x-axis* title and remove the *y -axis* title.

I've put in gridlines - this time, major gridlines for the x axis. This is done within the `theme` function by specifying values for the `panel.grid.major.x` parameter.

```
library(ggplot2)

theme_set(theme_classic())

ggplot(ire_box_office, (aes(x= reorder(title_short, weekend_gross) , y=weekend_gross))) +
  geom_col( width=0.8) +
  scale_y_continuous(limits = c(0, 3.8e5),
                     expand = c(0, 0),
                     breaks = c( 5e4, 1.0e5, 1.5e5, 2.0e5, 2.5e5,3e5,3.5e5 ),
                     labels = c("50", "100", "150", "200", "250", "300", "350" ),
                     name = "weekend gross (x €1,000)") +
  ggtile("weekend gross (x €1,000) - Oct 25-27 2019") +
  coord_flip(clip = "off") +
  theme(
    axis.title = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.title.y = element_blank(),
    plot.title = element_text(size = 11),
    plot.margin = margin(3, 6, 3, 3),
    panel.background = element_blank(),
    panel.grid.major.x = element_line(size = 0.4, linetype = 'solid',colour = "white"),
    panel.on top = TRUE
  )
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
```

```

## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

```

```

## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

```

It could be argued that without the visible gridline that the scale on the axis might be hard to reconcile to the white break lines in the bars. An approach to address this is to put the x-axis at the top of the figure. Now this is not commonly done - and it is not completely straightforward to do this in ggplot either. However with ordered bars and the *invisible* ‘Tuft’ gridlines, I think it is permissible.

To accomplish this ggplot, I specify the axis as a duplicate axis `sec.axis = dup_axis()`. As the name suggests, this creates a second duplicate axis (on top). In the `theme` function, I remove the axis from the bottom of the plot by setting the `axis.title.x.bottom` and `axis.text.x.bottom` attributes to `element_blank()`

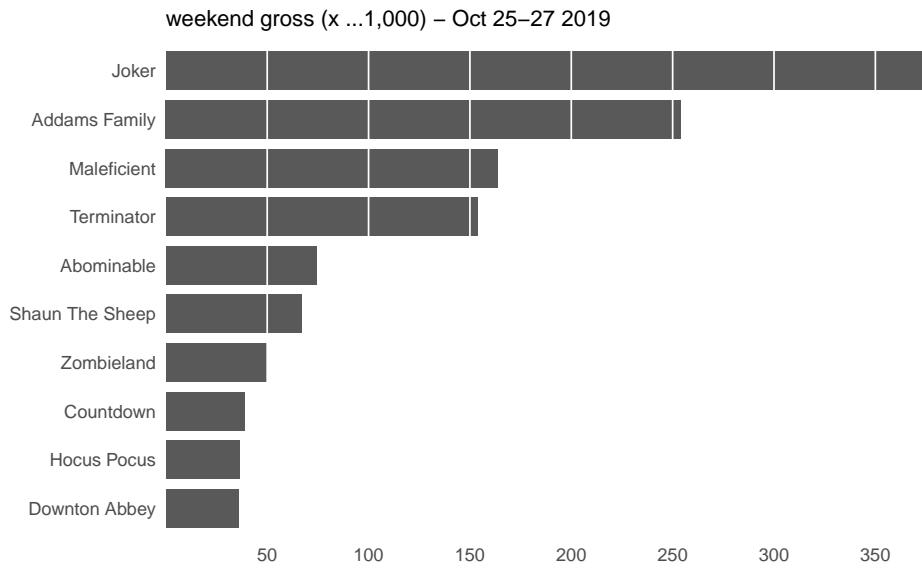


Figure 4.6: bar graph with labels on the y-axis

```

library(ggplot2)

theme_set(theme_classic())

ggplot(ire_box_office, (aes(x= reorder(title_short, weekend_gross) , y=weekend_gross)) +
  geom_col( width=0.8) + 

  scale_y_continuous(sec.axis = dup_axis(), limits = c(0, 3.8e5),
                     expand = c(0, 0),
                     breaks = c( 5e4,  1.0e5,  1.5e5,  2.0e5,  2.5e5, 3e5, 3.5e5 ),
                     labels = c("50", "100", "150", "200", "250", "300", "350" ),
                     name = "weekend gross (x €1,000)") +
  ggtitle("weekend gross (x €1,000) - Oct 25-27 2019 ") + 

  coord_flip(clip = "off") +
  theme(
    axis.title = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.title.y = element_blank(),
  )

```

```

axis.title.x.bottom = element_blank(),
axis.text.x.bottom = element_blank(),
plot.title = element_text(size = 11),
plot.margin = margin(3, 6, 3, 3),
panel.background = element_blank(),
panel.grid.major.x = element_line(size = 0.4, linetype = 'solid', colour = "white"),
panel.on top = TRUE)

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

```

```

## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in

```

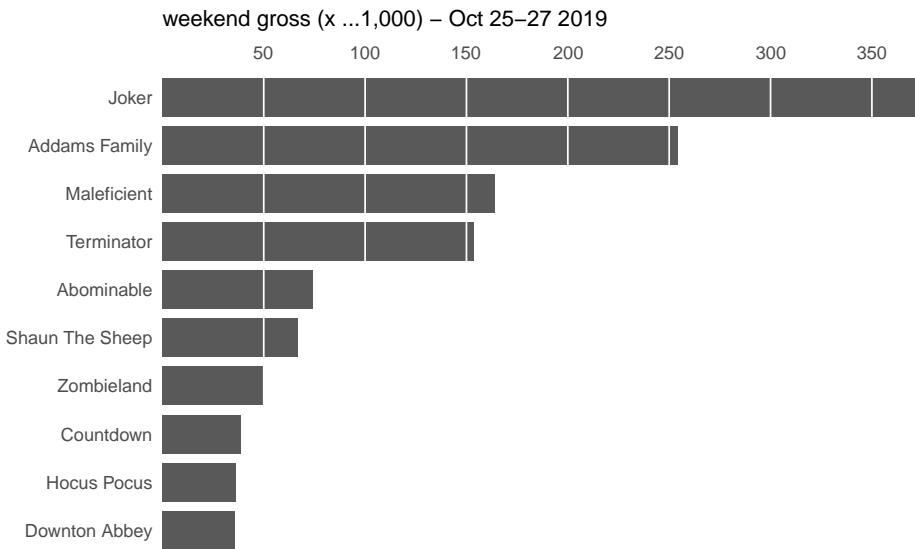


Figure 4.7: bar graph with labels on the y-axis and x-axis labels on top

```
## 'mbcsToSbcs': dot substituted for <ac>
```

After swapping the axes, we obtain a compact figure in which all visual elements, including all text, are horizontally oriented. As a result, the figure is much easier to read.

By default, bar graphs use dark grey bars. However we can customise the colour fill of the bars using the `fill` property

```
library(ggplot2)

theme_set(theme_classic())

ggplot(ire_box_office, (aes(x= reorder(title_short, weekend_gross) , y=weekend_gross))) +
  geom_col( fill="#5069be", width=0.8) +
  
  scale_y_continuous(sec.axis = dup_axis(), limits = c(0, 3.8e5),
                     expand = c(0, 0),
                     breaks = c( 5e4,  1.0e5,  1.5e5,  2.0e5,  2.5e5, 3e5, 3.5e5 ),
                     labels = c("50", "100", "150", "200", "250", "300", "350" ),
                     name = "weekend gross (x €1,000)") +
  ggttitle("weekend gross (x €1,000) - Oct 25–27 2019") +
  
  coord_flip(clip = "off") +
```

```

theme(
  axis.title = element_blank(),
  axis.line.y = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  axis.title.y = element_blank(),
  axis.title.x.bottom = element_blank(),
  axis.text.x.bottom = element_blank(),
  plot.title = element_text(size = 11),
  plot.margin = margin(3, 6, 3, 3),
  panel.background = element_blank(),
  panel.grid.major.x = element_line(size = 0.4, linetype = 'solid', colour = "white"),
  panel.on top = TRUE)

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019 ' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019 ' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019 ' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019 ' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019 ' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019 ' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019 ' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019 ' in
## 'mbcsToSbcs': dot substituted for <82>

```

```

## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
## Warning in grid.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in

```

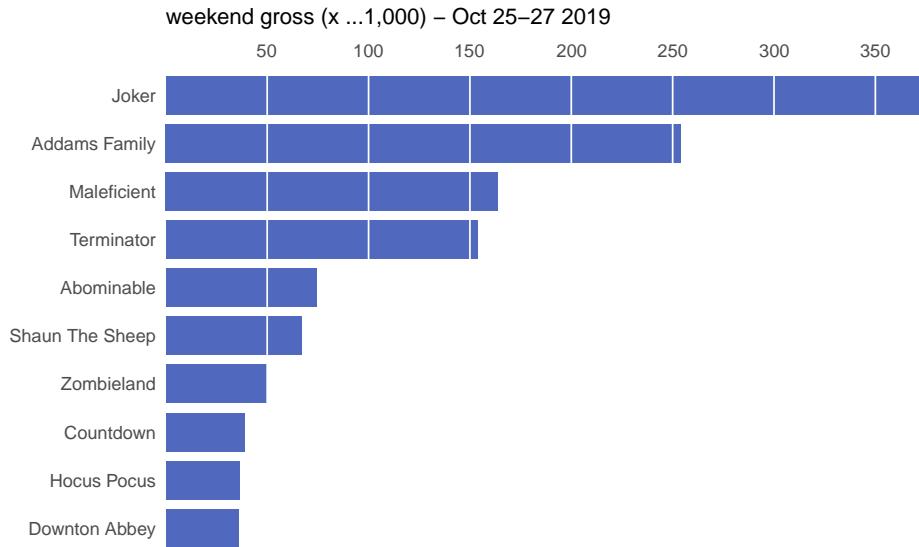


Figure 4.8: bar graph coloured

```
## 'mbcsToSbcs': dot substituted for <e2>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
```

And here is the same plot with the x-axis on the bottom and standard grey gridlines

```

ggtile("weekend gross (x €1,000) - Oct 25-27 2019") +
  coord_flip(clip = "off") +
  theme(
    axis.title = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.title.y = element_blank(),
    plot.title = element_text(size = 11),
    plot.margin = margin(3, 6, 3, 3),
    panel.grid.major.x = element_line(size = 0.2, linetype = 'solid', colour = "lightgrey"))

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

```

```

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>

## Warning in grid.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <e2>

```

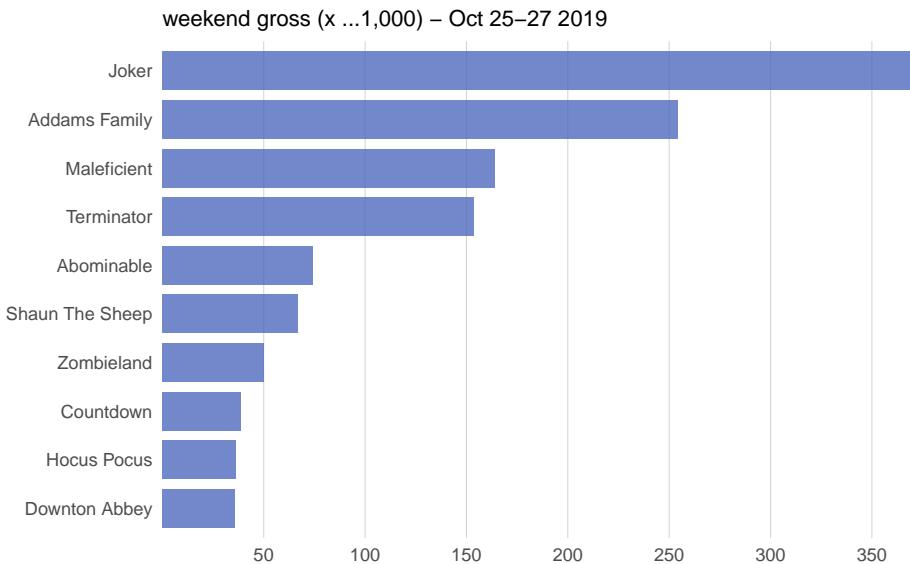


Figure 4.9: bar graph coloured

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <82>
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000) - Oct 25-27 2019' in
## 'mbcsToSbcs': dot substituted for <ac>
```

Here it is with the title removed and the axis title reinstated.

```
library(ggplot2)

theme_set(theme_classic())

ggplot(ire_box_office, (aes(x= reorder(title_short, weekend_gross) , y=weekend_gross))) +
  geom_col(fill="#5069be",alpha = 0.85, width=0.8) +
  scale_y_continuous(limits = c(0, 3.8e5),
                     expand = c(0, 0),
                     breaks = c( 5e4, 1.0e5, 1.5e5, 2.0e5, 2.5e5, 3e5, 3.5e5 ),
                     labels = c("50", "100", "150", "200", "250", "300", "350" ),
                     name = "weekend gross (x €1,000)") +
  coord_flip(clip = "off") +
  theme(
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
```

```

axis.line.x = element_blank(),
axis.ticks.x = element_blank(),
axis.title.y = element_blank(),
plot.margin = margin(3, 6, 3, 3),
panel.grid.major.x = element_line(size = 0.2, linetype = 'solid', colour = "lightgreen")

)

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <82>

```

```
## substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <ac>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <82>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <ac>

## Warning in grid.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <e2>

## Warning in grid.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
## substituted for <82>

## Warning in grid.graphics(C_text, as.graphicsAnnot(x$label), x$x,
## x$y, : conversion failure on 'weekend gross (x €1,000)' in 'mbcsToSbcs': dot
```

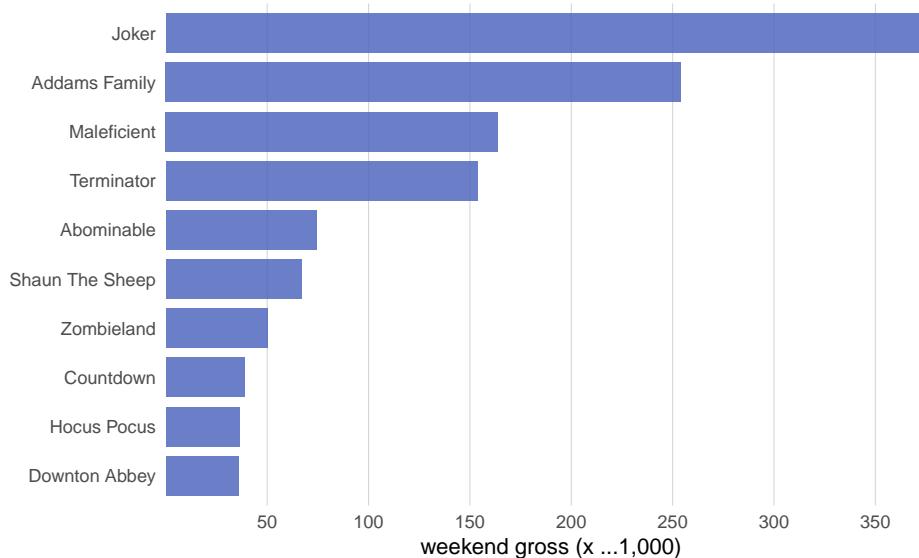


Figure 4.10: (#fig:bar_standard_fill2)bar graph coloured

```
## substituted for <ac>
```

4.1.2 Different orderings

In the previous example, we reordered the bars. We should only do this when there is no natural ordering to the categories the bars represent. Whenever there is a natural ordering (i.e. when our categorical variable is an ordered factor) we should retain that ordering in the visualization.

For example, this figure shows the age profiles of grooms in registered marriages in Ireland in 2018, ordered by age groups. In this case, the bars should be arranged in order of increasing age. Sorting by bar height while shuffling the age groups makes no sense. The data for this plot was obtained from the [Central Statistics Office] (<https://statbank.cso.ie/px/pxeirestat/statire>SelectTable/Omrade0.asp?Planguage=0>)

```
bride_groom_age <- read.csv("../data/ire_2018_bride_groom_age.csv")

library(ggplot2)
library (dplyr)

bride_groom_age$age <- factor(bride_groom_age$age, levels = c("< 20", "20 - 24", "25 - 29", "30 - 34", "35 - 39", "40 - 44", "45 - 49", "50 - 54", "55 - 59", "60 - 64", "65 - 69", "70 - 74", "75 - 79", "80 - 84", "85 - 89", "90 - 94"))

bride_groom_age_g<-bride_groom_age%>%filter(bride_groom=="Groom")
```

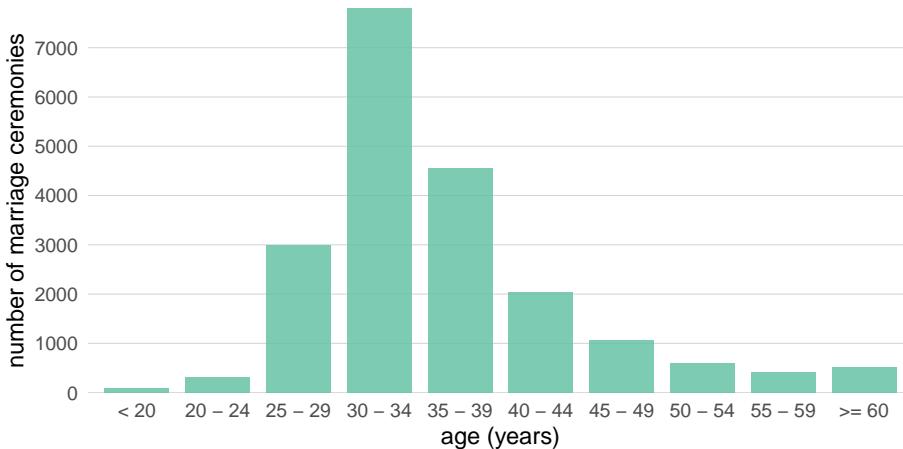


Figure 4.11: Marriages Registered in Ireland 2018: Groom Age

```
theme_set(theme_classic())

ggplot(bride_groom_age_g, (aes(x= age, y=ceremonies))) +
  geom_col(fill="#66c2a5", alpha=0.85, width =0.8) +
  scale_y_continuous(
    breaks = c(0, 1000, 2000, 3000, 4000, 5000, 6000, 7000),
    expand = c(0, 0),
    name = "number of marriage ceremonies") +
  xlab(label = "age (years)") +
  theme(
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    panel.grid.major.y = element_line(size = 0.2, linetype = 'solid', colour = "lightgrey"))
```

Here is the same data plotted using the ‘Tufte’ gridlines approach.

```
library(ggplot2)

theme_set(theme_classic())

ggplot(bride_groom_age_g, (aes(x= age, y=ceremonies))) +
  geom_col(fill="#66c2a5", width =0.8) +
  scale_y_continuous(
    breaks = c(0, 1000, 2000, 3000, 4000, 5000, 6000, 7000),
```

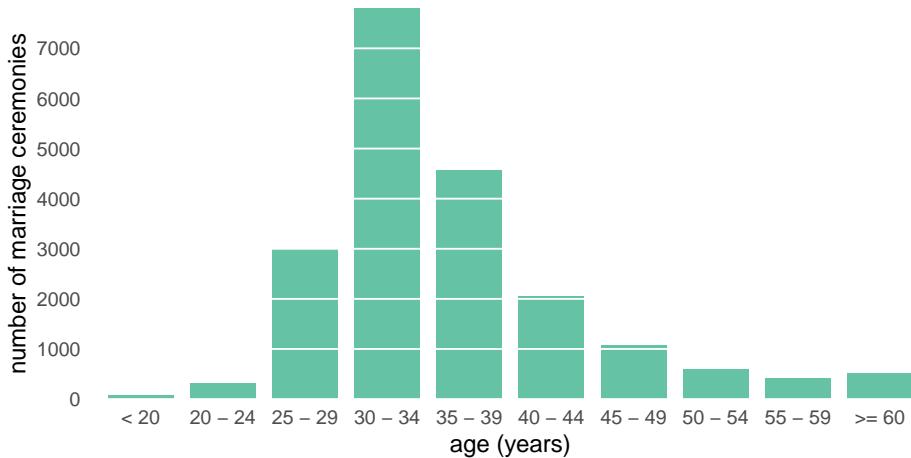


Figure 4.12: Marriages Registered in Ireland 2018: Groom Age

```

expand = c(0, 0),
name = "number of marriage ceremonies") +
xlab(label = "age (years)") +  
  

theme(
  axis.line.y = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  panel.background = element_blank(),
  panel.grid.major.y = element_line(size = 0.4, linetype = 'solid', colour = "white"),
  panel.on top = TRUE)

```

4.1.3 Grouped and Stacked bars

Frequently, however, we are interested in two categorical variables at the same time. In a grouped bar plot, we draw a group of bars at each position along the x axis, determined by one categorical variable, and then we draw bars within each group according to the other categorical variable.

In the next plot, we will include the age profile of the brides in registered marriage ceremonies in Ireland.

We use colour fill as an aesthetic so that bars representing groom data will be one colour and bars representing bride data will be another. In the `aes` function we set `fill = bride_groom`, the variable designating whether a data point is for bride or groom.

You must use the `position = "dodge"` in the `geom_col` function, which tells the bars to avoid each other horizontally.

Check to what happens when you don't use `position = "dodge"`

The default ggplot colours aren't great. We can use the `scale_fill_manual` function to set our own pair of colour values, which are a pair of colour-blind friendly qualitative colours selected from the <http://colorbrewer2.org> website. I have set `name = NULL` in this function to tell ggplot not to render the name of the legend, which is often not required.

In order to reduce the width of the overall plot, I have moved the legend inside the plot from its default position on the right hand-side outside the plot. It is OK to do this, as long as the legend cannot be confused with or interfere with the perception of the data elements of the plot.

```
library(ggplot2)
library (dplyr)

theme_set(theme_classic())

ggplot(bride_groom_age, (aes(x= age, y=ceremonies, fill=bride_groom))) +
  geom_col(position="dodge", alpha=0.85) +
  scale_y_continuous(
    breaks = c(0, 1000, 2000, 3000, 4000, 5000, 6000, 7000),
    expand = c(0, 0),
    name = "number of marriage ceremonies") +
  xlab(label = "age (years)") +
  scale_fill_manual(values = c("#fc8d62","#66c2a5" ), name = NULL) +
  theme(
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    panel.background = element_blank(),
    panel.grid.major.y = element_line(size = 0.4, linetype = 'solid', colour = "white"),
    panel.on top = TRUE,
    legend.position= c(0.8, 0.9), legend.direction="horizontal")
```

Here is another example, but this time with multiple bars. The data is from the Central Statistics Office and gives the distribution of principle economic status values per quarter in 2016.

By running the `head` function you can see that the data is in *wide form*, with Q1, Q2 etc represented as columns. ggplot tends to work best with data in *long form*, so we have to transform the dataframe.

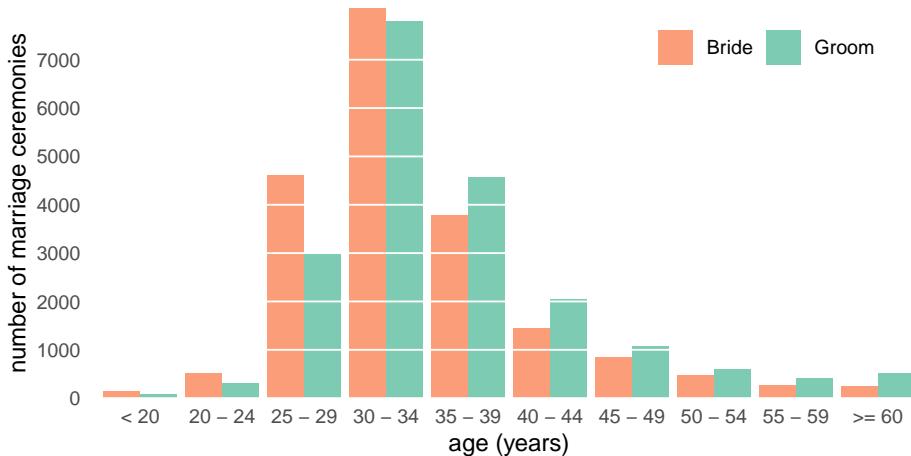


Figure 4.13: Marriages Registered in Ireland 2018: Ages of Bride and Groom

Luckily, there is a `tidyverse` library called `tidyr` that allows us to do this easily using the `gather` function. Using this function, I create two new columns *quarter* and *value*, which will represent the transformed data in the new dataframe. You can run `head` on the new dataframe to see the transformation.

There is a nice explanation on `tidyr` here

I have used colours from the colourblind friendly palette of *Okabe and Ito* for this visualisation, but I have *desaturated* them a little as they were a little bright for my taste.

Note that this is an example where you have to choose your break intervals sensitively. Five out six bars have values below 500 and one bar has a value over 1750. If your break frequency is too great, the gridlines make the overall plot look too busy; At the same time, if your breaks are too far apart, say at levels 1000 and 2000, then it will be difficult to resolve values for the shorter bars against the axis.

As before, in order to reduce the width of the overall plot, I have moved the legend inside the plot from its default position on the right hand-side outside the plot.

```
library(ggplot2)
library(dplyr)
library(tidyr)
library(colorspace)

econ_stat_per_q <- read.csv("../data/2016PrincipalEconomicStatusPerQuarter.csv")
```

```

head(econ_stat_per_q)

##                                     status Q1.2016 Q2.2016 Q3.2016 Q4.2016
## 1           At work    1926.1   1941.1   1961.7   1987.1
## 2      Unemployed     222.7    221.8    220.6    194.1
## 3        Student      419.6    408.8    392.9    402.8
## 4 Engaged on home duties    455.0    454.6    445.2    436.0
## 5 Retired from employment    441.7    444.3    457.7    467.1
## 6          Other       161.1    167.1    170.6    167.8

# convert the data from wide from to long form, which is required by ggplot
econ_stat_per_q %>% gather(quarter, value, Q1.2016:Q4.2016) -> econ_stat_per_q_long

head(econ_stat_per_q_long)

##                                     status quarter  value
## 1           At work Q1.2016 1926.1
## 2      Unemployed Q1.2016  222.7
## 3        Student Q1.2016  419.6
## 4 Engaged on home duties Q1.2016  455.0
## 5 Retired from employment Q1.2016  441.7
## 6          Other Q1.2016  161.1

theme_set(theme_classic())

# colour blind friendly palette - slightly desaturated
cbPalette <- desaturate(c("#999999", "#E69F00", "#56B4E9", "#009E73", "#FOE442", "#0072B2", "#D55588"))

ggplot(econ_stat_per_q_long, (aes(x= quarter, y=value, fill=status))) +
  geom_col(position="dodge", alpha=0.85) +
  scale_y_continuous(
    breaks = c(0, 250, 500, 750, 1000, 1250, 1500, 1750, 2000),
    expand = c(0, 0),
    name = "person aged 15 years and over (Thousand)") +
  xlab(label = "Year : 2016") +
  scale_fill_manual(values = cbPalette[2:8], name = NULL, labels = c("At work","Home duties","Other"))
  scale_x_discrete(labels = c("Q1","Q2","Q3","Q4"))+
  theme(
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.title = element_text(size=9),
    panel.grid.major.y = element_line(size = 0.2, linetype = 'solid', colour = "lightgrey"),
    panel.grid.minor.y = element_line(size = 0.2, linetype = 'solid', colour = "lightgrey"),
    panel.grid.major.x = element_line(size = 0.2, linetype = 'solid', colour = "lightgrey"),
    panel.grid.minor.x = element_line(size = 0.2, linetype = 'solid', colour = "lightgrey"))

```

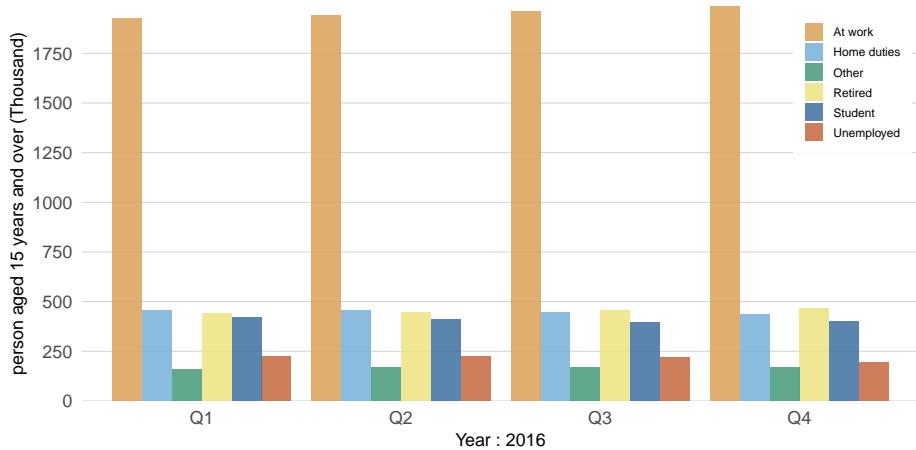


Figure 4.14: Person aged 15 years and over (Thousands) Principal Economic Status and Quarter

```
legend.position= c(0.92, 0.82),
legend.text = element_text(size = 6), # legend text was a little large
legend.key.size = unit(0.7, "lines"), # legend key size was a little large
legend.title = element_text(size =10)
```

I have produced a Tufte version of the same plot, which I think works.

```
library(ggplot2)
library (dplyr)
library(tidyr)
library(colorspace)

theme_set(theme_classic())

# colour blind friendly palette - slightly desaturated
cbPalette <- desaturate(c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0077B6"))

ggplot(econ_stat_per_q_long, (aes(x= quarter, y=value, fill=status))) +
  geom_col(position="dodge") +
  scale_y_continuous(
    breaks = c(0, 250, 500, 750, 1000, 1250, 1500, 1750, 2000),
    expand = c(0, 0),
    name = "person aged 15 years and over (Thousands)") +
  xlab(label = "Year : 2016") +
  scale_fill_manual(values = cbPalette[2:8], name = NULL, labels = c("At work","Home duties","Other","Retired","Student","Unemployed"))+
  scale_x_discrete(labels = c("Q1","Q2","Q3","Q4"))+
```

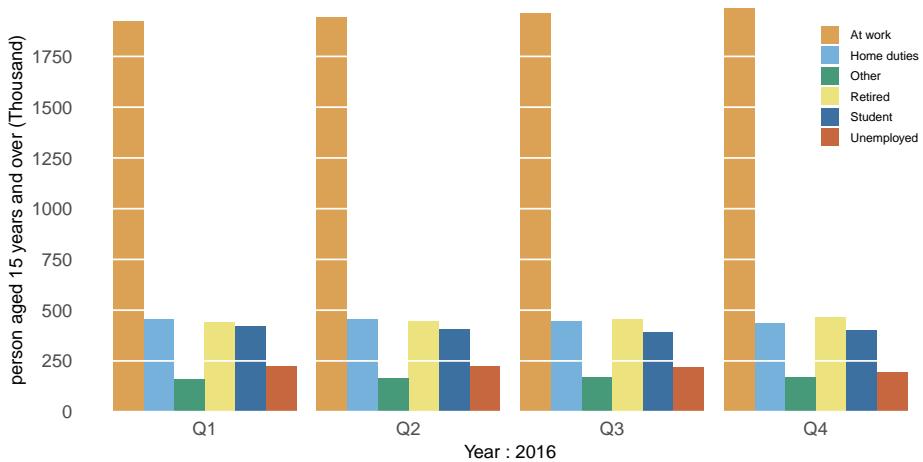


Figure 4.15: Person aged 15 years and over (Thousands) Principal Economic Status and Quarter

```
theme(
  axis.line.y = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  axis.title = element_text(size=9),
  panel.background=element_blank(),
  panel.grid.major.y = element_line(size = 0.4, linetype = 'solid', colour = "white"),
  panel.on top = TRUE,
  legend.position= c(0.92, 0.82),
  legend.text = element_text(size = 6), # legend text was a little large
  legend.key.size = unit(0.7, "lines"), # legend key size was a little large
  legend.title = element_text(size =10))# legend title was a little large)
```

Grouped bar plots show a lot of information at once and they can be confusing. It is a little difficult is difficult to compare numbers of persons across quarters, although admittedly the numbers are relatively static.

If we care more about the overall pattern per economic status group over a given time span, we can group the bars by quarter, clearly showing the change in each group per quarter.

As such, the next plot provides an alternative bar grouping of the same data, where the reader is assumed to be more interested in comparing the quarterly differences within each socio-economic class

```

library(ggplot2)
library(dplyr)
library(tidyr)
library(colorspace)

theme_set(theme_classic())

# colour blind friendly palette - slightly desaturated
cbPalette <- desaturate(c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0077B6"))

ggplot(econ_stat_per_q_long, (aes(x= status, y=value, fill=quarter))) +
  geom_col(position="dodge") +
  scale_y_continuous(
    breaks = c(0, 250, 500, 750, 1000, 1250, 1500, 1750, 2000),
    expand = c(0, 0),
    name = "person aged 15 years and over (Thousand)") +
  scale_fill_manual(values = cbPalette[2:8], name = "Year : 2016", labels = c("Q1","Q2","Q3","Q4")),
  scale_x_discrete(labels = c("At work","Home duties","Other","Retired","Student","Unemployed"))

theme(
  axis.line.y = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  axis.title.x = element_blank(),
  axis.title.y = element_text(size = 9),
  panel.background=element_blank(),
  panel.grid.major.y = element_line(size = 0.4, linetype = 'solid',colour = "white"),
  panel.on top = TRUE,
  legend.position= c(0.92, 0.82),
  legend.text = element_text(size = 6), # legend text was a little large
  legend.key.size = unit(0.7, "lines"), # legend key size was a little large
  legend.title = element_text(size =10))# legend title was a little large)

```

The plot below provides an extended view of the data in the figure above. It provides an average Principal Economic Status figure for each group per year. The average is calculated from the quarterly figures per year.

As the bars within each group are ordered (by year), I want colour to indicate some progression in time.

As such I considered using a *sequential palette* (e.g. from light blue to dark blue). However, as we have 6 years to colour, I found the lightest colour in the sequence was too light. I decided to use the *viridis* palette as even the palette for the discrete scale suggests a sequence. This palette also has the added bonus of being colour blind friendly.

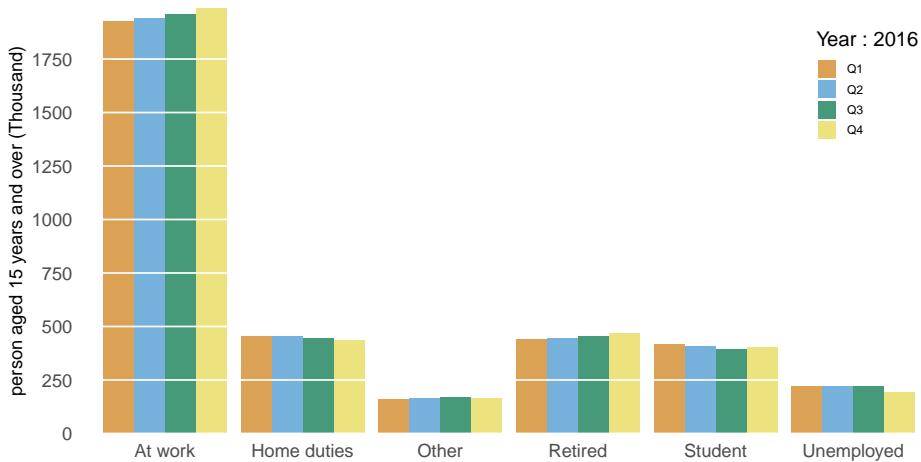


Figure 4.16: Person aged 15 years and over (Thousands) Principal Economic Status and Quarter

In the code below, I once again transform the data from *wide form/* to *long form*.

In the previous plot, you may have noticed that the ordering of the values of the status factor is alphabetical - *At work*, *Home Duties*, *Other*, etc. Having the value *Other* in the middle of the more meaningful values is a bit counter-intuitive. It would typically be the last category anyone might be interested in. Although the *status* variable is an unordered factor, we can impose a specific ordering on it that reflects the interest of our readers. I do this by explicitly resetting the ordering of the levels of the *status* factor in the dataframe.

```
library(RColorBrewer)

econ_stat <- read.csv("../data/2012-2017-principle_econ_year.csv")

head(econ_stat)

##          status    y2012    y2013    y2014    y2015    y2016    y2017
## 1      At work 1780.100 1816.650 1853.575 1908.225 1954.000 1999.15
## 2   Unemployed  361.175  325.600  290.825  246.150  214.800  185.10
## 3     Student  409.675  414.175  412.400  406.675  406.025  415.20
## 4 Engaged on home duties  504.000  481.800  475.200  471.925  447.700  429.65
## 5 Retired from employment  392.175  404.275  413.050  423.475  452.700  472.50
## 6        Other  147.250  150.275  151.850  155.100  166.650  172.45
# convert the data from wide from to long form, which is required by ggplot
econ_stat %>% gather(year, value, y2012:y2017) -> econ_stat_long
```

```

head(econ_stat_long)

##          status   year   value
## 1      At work y2012 1780.100
## 2     Unemployed y2012  361.175
## 3      Student y2012  409.675
## 4 Engaged on home duties y2012  504.000
## 5 Retired from employment y2012  392.175
## 6      Other y2012  147.250

# set the ordering of status factor

levels(econ_stat_long$status) # This gives you the current level ordering (alphabetic)

## [1] "At work"                  "Engaged on home duties"
## [3] "Other"                    "Retired from employment"
## [5] "Student"                  "Unemployed"

econ_stat_long$status <- factor(econ_stat_long$status, levels = c("At work", "Unemployed"))

levels(econ_stat_long$status) # This shows you the revised ordering (custom)

## [1] "At work"                  "Unemployed"
## [3] "Engaged on home duties"  "Student"
## [5] "Retired from employment" "Other"

theme_set(theme_classic())

fig<- ggplot(econ_stat_long, (aes(x= status, y=value, fill=year))) +
  geom_col(position="dodge", alpha=0.85) +
  scale_y_continuous(
    limits=c(0,2001),
    breaks = seq(0, 2000, by =200),
    expand = c(0, 0),
    name = "person aged 15 years and over (Thousand)") +
  scale_fill_viridis_d(name = NULL, end = 0.95, direction =-1, labels = c("2012","2013"))

scale_x_discrete(labels = c("At work", "Unemployed", "Home duties", "Student", "Retired"))

guides(fill = guide_legend(nrow = 1)) # ggplot will tend to wrap long horizontal labels

theme(
  axis.line.y = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  plot.title = element_text(hjust = 0.5))

```

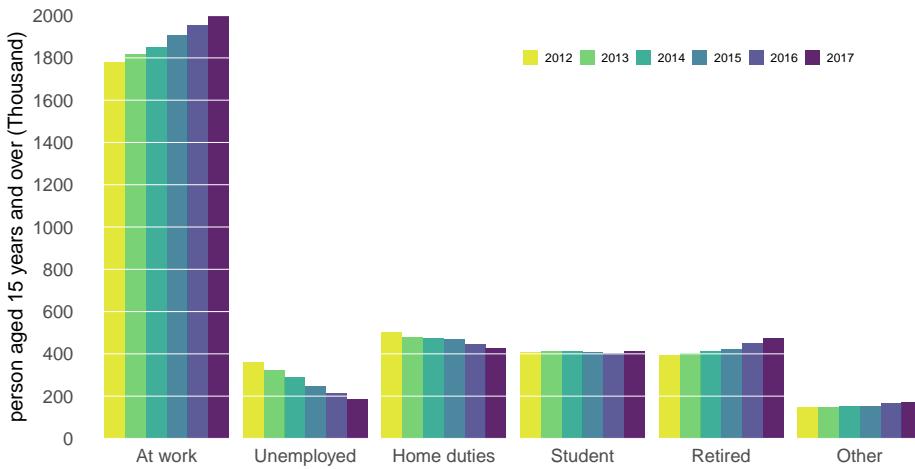


Figure 4.17: Person aged 15 years and over (Thousands) Principal Economic Status 2012-2017

```

axis.title = element_text(size=10),
panel.background=element_blank(),
panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
panel.on top = TRUE,
plot.margin = margin(10, 6, 3, 3),
legend.position= c(0.7, 0.9), legend.direction="horizontal",
legend.text = element_text(size = 6), # legend text was a little large
legend.key.size = unit(0.6, "lines"),# legend keys were a little large
legend.spacing.x = unit(0.1, 'cm'))
```

fig

I'll do a quick test to see how this plot renders for CVD vision using the `cvd_grid` function from the `colorblindr` package. It seems to be acceptable. The axis text is too big - but I'm not worried. I am just using this plot to check how the colours appear for someone with colourblindness.

```

library(colorblindr)
cvd_grid(fig)
```

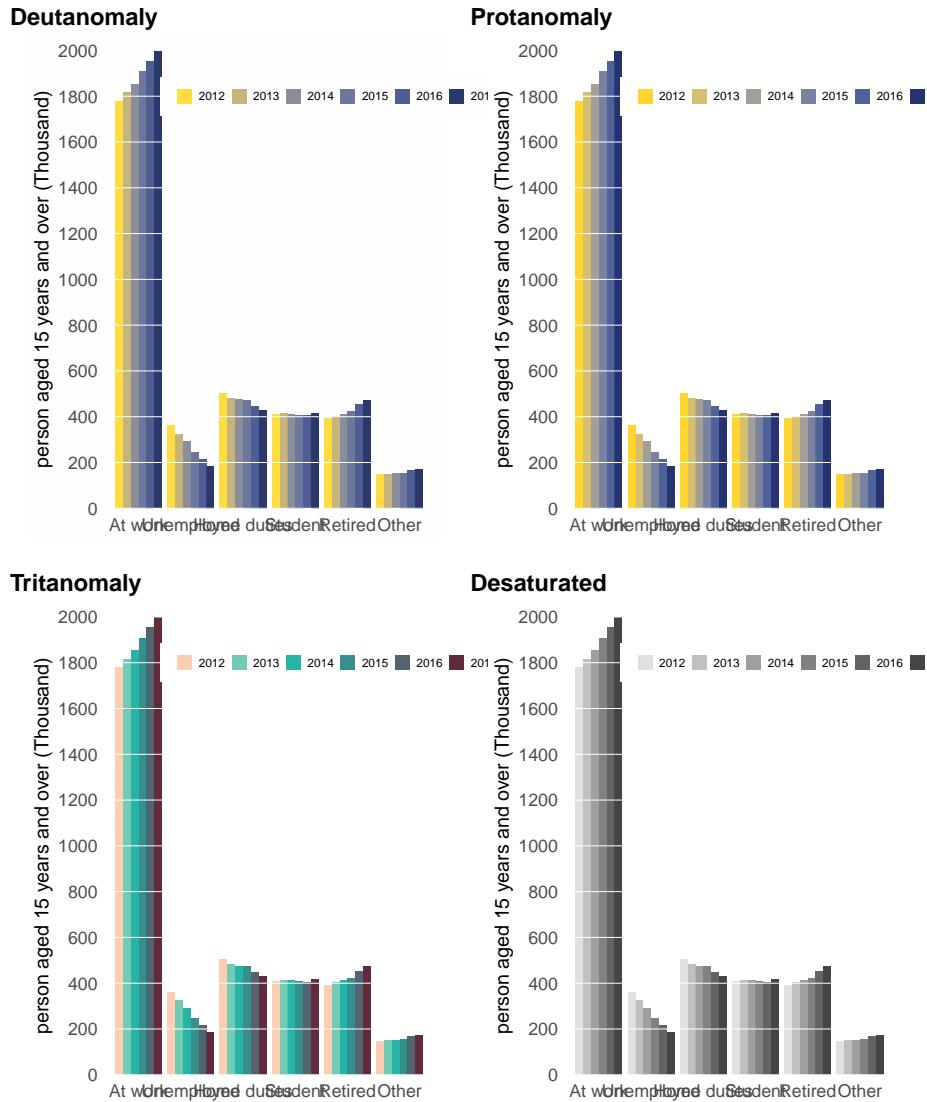


Figure 4.18: The plot rendered with a colour blindness simulation

4.1.4 Faceted View

Both Figures encode a single categorical variable by position along the x axis and another by bar color. The encoding by position is easy to read while the encoding by bar color requires more cognitive effort, as we have to visually match the colors of the bars against the colours in the legend.

An approach that reduces this is to separate out the grouped plots into a panel of regular bar plots. In ggplot, this is known as *faceting* - and it is quite easy to facet an existing plot. The key idea is that you choose a variable whose values will determine the number of plots to be created. In this case, we will facet by the *status* variable, which will mean that ggplot will create 6 individual plots - one for each *status* value.

The principle changes here are as follows:

- in the `aes` function `x` is set to *year* and `y` is set to *value*. This is because we want each panel to show values for the years 2012 - 2017.
- we set the fill colour to static colour as we would do for a single (non-grouped) bar plot
- add the function `facet_wrap(~status, scales = "free_x")` to the plot. This tells ggplot to create a set of plots (facets) — one for each value of the *status* variable. The facets can be placed next to each other, wrapping with a certain number of columns or rows.

The `scales` parameter is set to *free_x*. Normally, the axis scales on each facet are fixed, which means that they have the same size and range. They can be made independent, by setting scales to *free*, *free_x*, or *free_y*. In this case, we know that the year variable has the same values for each status value - so the use of `scales = "free_x"` would seem to be redundant. However, by including it, it forces ggplot to include the *x-axis* scale under each plot, which I think is easier to read. By default, it would otherwise put the *x-axis* scale at the bottom of the panel. Remove it and see for yourself. Finally, I've made some modifications to the title of each facet - known as *strip* text. I've done this in the `theme` function.

The r graphics cookbook has a useful primer on faceting in ggplot [http://www.cookbook-r.com/Graphs/Facets_\(ggplot2\)/](http://www.cookbook-r.com/Graphs/Facets_(ggplot2)/)

```
#library(RColorBrewer)

theme_set(theme_classic())

fig<- ggplot(econ_stat_long, (aes(x= year, y=value))) +
  geom_col(fill="#5069be", alpha=0.85) +
  scale_y_continuous(
    limits=c(0,2001),
```

```

breaks = c(0, 500, 1000, 1500, 2000),
expand = c(0, 0),
name = "person aged 15 years and over (Thousands)" + 

scale_x_discrete(labels = c("2012", "2013", "2014", "2015", "2016", "2017"), name = NULL)

facet_wrap(~status, ncol=3, scales = "free_x") + 

theme(
  axis.line.y = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  axis.title.x = element_text(size=8),
  axis.title.y = element_text(size=8),
  axis.text.x = element_text(size=6),
  axis.text.y = element_text(size=6),

  panel.spacing.y = grid::unit(14, "pt"),
  strip.text = element_text(size = 8),
  strip.background = element_blank(),
  plot.margin = margin(10, 6, 3, 3),
  panel.background = element_blank(),
  panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
  panel.on top = TRUE)

fig

```

4.1.5 Colouring bars by a second variable

You may want to colour bars according to a second variable, typically a categorical variable. In this example, we'll use the *mtcars* data set, a data set built into R.

```

# Load data
data("mtcars")
dfm <- mtcars
# Convert the cyl variable to a factor
dfm$cyl <- as.factor(dfm$cyl)
# Add the name column
dfm$name <- rownames(dfm)
# Inspect the data
head(dfm[, c("name", "wt", "mpg", "cyl")])

```

	name	wt	mpg	cyl
## Mazda RX4	Mazda RX4	2.620	21.0	6

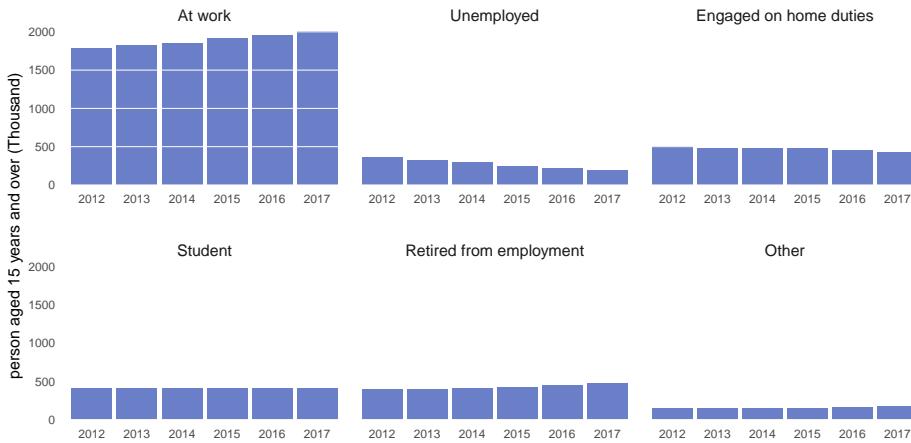


Figure 4.19: Person aged 15 years and over (Thousands) Principal Economic Status 2012-2017

```
## Mazda RX4 Wag      Mazda RX4 Wag 2.875 21.0   6
## Datsun 710         Datsun 710 2.320 22.8   4
## Hornet 4 Drive     Hornet 4 Drive 3.215 21.4   6
## Hornet Sportabout  Hornet Sportabout 3.440 18.7   8
## Valiant             Valiant 3.460 18.1   6
```

When I am working on a plot, I like to occasionally check the colours in the palette I am using. The scales package provides a function called `show_col` for a quick and dirty way to visualise any vector of colours.

```
library(scales)

theme_set(theme_classic())

# colour blind friendly palette
cbPal <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

# scales function
show_col(cbPal)
```

This allows me to select the three colours from the vector of colour blind friendly colour values that I use regularly.

```
# I need 3 colours only for the cyl variable
cb3 <- cbPal[c(6,2,1)]

ggplot(df, (aes(x= reorder(name, -mpg) , y=mpg, fill=cyl))) +
  geom_col(width=0.7) +
```



Figure 4.20: Bar graph with formatted y-axis

```

scale_y_continuous(limits = c(0, 35),
                   breaks = c( 10, 20, 30),
                   name = "miles per gallon") +
scale_fill_manual(values=cb3, name="Number of cylinders") +  
  

theme(  

  axis.line.y = element_blank(),  

  axis.ticks.y = element_blank(),  

  axis.line.x = element_blank(),  

  axis.ticks.x = element_blank(),  

  axis.text.x = element_text(size = 6, angle = 90, vjust = 1, hjust = 1),  

  axis.title.x = element_blank(),  

  plot.margin = margin(6, 6, 3, 3),  

  panel.background=element_blank(),  

  panel.grid.major.y = element_line(size = 0.2, linetype = 'solid', colour = "white"),
  panel.on top = TRUE,  

  legend.position= c(0.6, 0.92), legend.direction="horizontal",
  legend.text = element_text(size = 6), # legend text was a little large
  legend.key.size = unit(0.6, "lines"),
  legend.title = element_text(size =10))# legend keys were a little large

```

If we have space we can reorient this plot into a vertical alignment, which will make the car names makes easier to read.

While this is much easier to read – it also takes up at least twice as much vertical space. I've increased the number of breaks here also – because your eye has further to travel from the top of the bar vertically down to the x axis.

Which version you use really depends on the space you have available and the priority you place on the reader being able to read quickly through the rank order of the cars.

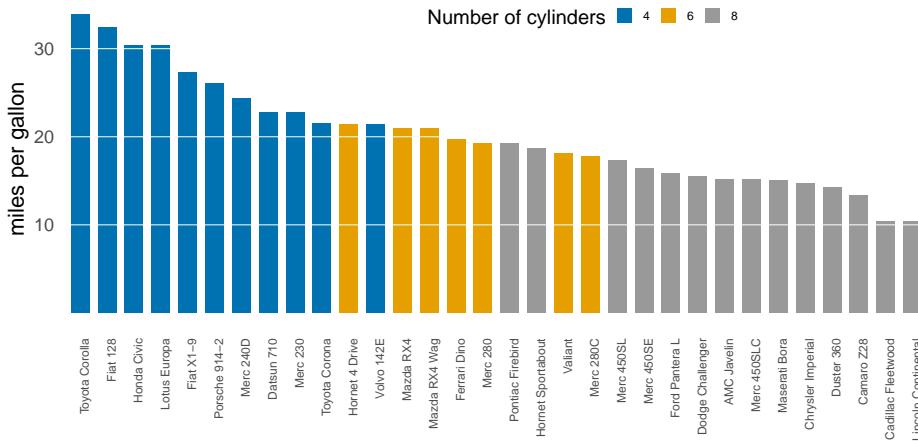


Figure 4.21: Bar graph coloured with second category, with formatted y-axis

```
ggplot(dfm, (aes(x= reorder(name, mpg) , y=mpg, fill=cyl))) +
  geom_col(width=0.8) +
  coord_flip(clip = "off") +
  scale_y_continuous(limits = c(0, 35),
                     breaks = seq(0,35, by=5),
                     name = "Miles per gallon",
                     expand = c(0, 0)) +
  scale_fill_manual(values=cb3, name="Number of cylinders") +
  theme(
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.text.x = element_text(size = 7),
    axis.text.y = element_text(size = 7),
    #axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    plot.margin = margin(6, 6, 3, 3),
    panel.background=element_blank(),
    panel.grid.major.x = element_line(size = 0.2, linetype = 'solid', colour = "white"),
    panel.on top = TRUE,
    legend.position= "top", legend.direction="horizontal",
    legend.text = element_text(size = 7), # legend text was a little large
    legend.key.size = unit(0.7, "lines"), # legend key size was a little large
```

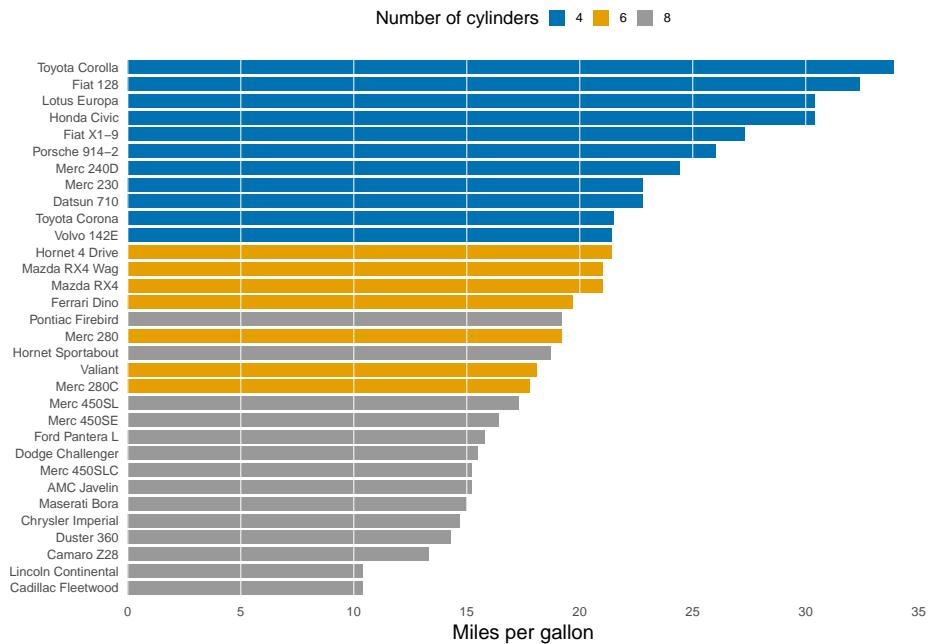


Figure 4.22: Bar graph coloured with second category, with formatted y-axis

```
legend.title = element_text(size = 10)) # legend title was a little large
```

4.1.6 Ordering within groups

In the two plots of the car data produced so far, the bars are coloured according to the values of the *cyl* variable.

We can add secondary ordering which might be useful in some contexts, where we order the *mpg* variable *within* the bars grouped by the cylinder variable

This requires telling ggplot the order we want the *x-axis* values plotted in. I use the `order` function to order the `name` values by *cyl* and then by *mpg*

I then explicitly set the factor levels of the *name* variable to this order. As I mentioned earlier, ggplot will plot the values of a categorical variable according to the order of its levels.

```
# names sorted by first by name and then by mpg
nameorder <- dfm$name[order(dfm$cyl, dfm$mpg)]

# Now we turn name into a factor, with levels in the order of name order
dfm$name <- factor(dfm$name, levels = nameorder)
```

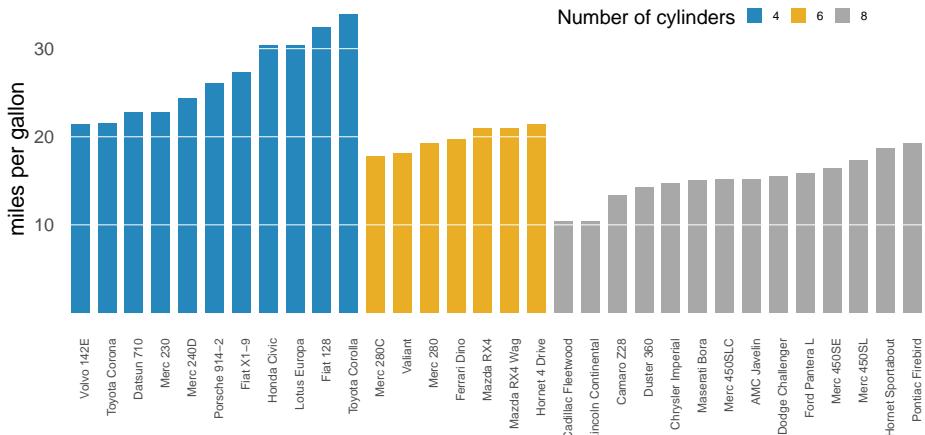


Figure 4.23: Bar graph coloured with second category, with formatted y-axis

```
ggplot(dfm, (aes(x= name, y=mpg, fill=cyl))) +
  geom_col(alpha = 0.85, width=0.7) +
  scale_y_continuous(limits = c(0, 35),
                     breaks = c( 10, 20, 30),
                     name = "miles per gallon") +
  scale_fill_manual(values=cb3, name="Number of cylinders") +
  theme(
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.text.x = element_text(size = 6, angle = 90, vjust = 1, hjust = 1),
    axis.title.x = element_blank(),
    plot.margin = margin(6, 6, 3, 3),
    panel.background=element_blank(),
    panel.grid.major.y = element_line(size = 0.2, linetype = 'solid', colour = "white"),
    panel.on top = TRUE,
    legend.position= c(0.75, 0.92), legend.direction="horizontal",
    legend.text = element_text(size = 6), # legend text was a little large
    legend.key.size = unit(0.6, "lines"),
    legend.title = element_text(size =10))# legend keys were a little large
```

4.1.7 Diverging Bar Charts

A Diverging Bar Chart or a Deviation Chart a is a bar chart that can handle both negative and positive values. The graph shows the divergence of quantitative values from a reference value — this can be zero or a mean value

We'll use a subset of the climate dataset in the *gcookbook* package. This data set includes estimated global temperature anomaly data for the years 1800 through 2011. The anomaly is the difference from the baseline temperature, which is the mean of the yearly temperatures from 1951-1980.

We will visualise the values of the *Anomaly10y* field which has positive and negative values. The *Anomaly10y* field gives the temperature anomaly in Celsius, smoothed over ten years.

```
library(gcookbook)

data(climate)

head(climate)

##      Source Year Anomaly1y Anomaly5y Anomaly10y Unc10y
## 1 Berkeley 1800      NA       NA -0.435  0.505
## 2 Berkeley 1801      NA       NA -0.453  0.493
## 3 Berkeley 1802      NA       NA -0.460  0.486
## 4 Berkeley 1803      NA       NA -0.493  0.489
## 5 Berkeley 1804      NA       NA -0.536  0.483
## 6 Berkeley 1805      NA       NA -0.541  0.475
```

A diverging bar chart typically has a colour coding for positive and negative values. To represent this easily we will create a new column *pos* containing boolean values — *TRUE* when an *Anomaly10y* value is positive and *FALSE* when it is negative. This will allow us to map colour as an aesthetic to the values of the *pos* field. We will then manually specify the two colours that will represent the boolean values of the *pos* variable.

```
library(gcookbook)
library(dplyr)

attach(climate)

# using dplyr to create a subset of data to visualise
climate_sub <- climate %>%
  filter(Source=="Berkeley" & Year >= 1900) %>% # filter rows by Source=="Berkeley" &
  mutate(pos = Anomaly10y >= 0) # create a new column called pos with boolean values equal to TRUE if Anomaly10y is greater than or equal to 0, and FALSE otherwise

# This is a non-dplyr way to do the very same thing. Dplyr is very handy!
climate_sub <- climate[which(Source=="Berkeley" & Year >= 1900),]
climate_sub$pos <- factor(ifelse(climate_sub$Anomaly10y >= 0, "TRUE", "FALSE"), levels = c("TRUE", "FALSE"))
```

```
# just removing the cols I won't be using.
climate_sub <- climate_sub[,c(1,2,5,7)]

detach(climate)
```

Now we create the diverging bar chart. The *x-axis* represents the *Year* values. The *y-axis* represents the *Anomaly10y* values; and we will colour the bars according to the values in the *pos* column.

Note that we have used `position = "identity"` within the `geom_col` function. This prevents an warning message being generated about stacking not being well defined for negative values

I've used the `scales` package to format the *y-axis* in the `scale_y_continuous` function (instead of using scientific notation). I could also simply have specified a vector of labels from -0.3 to 0.9.

```
library(scales)

# I need 2 colours for the pos variable which I select from the previously defined palette
cb2 <- cbPal[c(7,6)]

ggplot(climate_sub, aes(x = Year, y = Anomaly10y, fill = pos)) +
  geom_col(position = "identity", alpha = 0.85, width=0.7) +
  
  scale_y_continuous(limits = c(-0.35, 1),
                     breaks = seq(-0.3, 0.9, by= 0.1) ,
                     name = "Anomaly10y",
                     expand=c(0,0),
                     labels = scales::number_format(accuracy = 0.1)) +
  #scale_y_continuous(sec.axis = dup_axis(), breaks = c( -0.25, 0, 0.25, 0.5, 0.75), expand=c(0,
  
  scale_x_continuous(breaks = c( 1900, 1920, 1940, 1960, 1980, 2000)) +
  scale_fill_manual(values=cb2) +
  
  theme(
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    #axis.ticks.x = element_blank(),
    #axis.text.x = element_text(angle = 90, vjust = 1, hjust = 1),
    axis.title.x = element_blank(),
    plot.margin = margin(6, 6, 3, 3),
    panel.background=element_blank(),
    panel.grid.major.y = element_line(size = 0.2, linetype = 'solid', colour = "white"),
    panel.on top = TRUE,
```

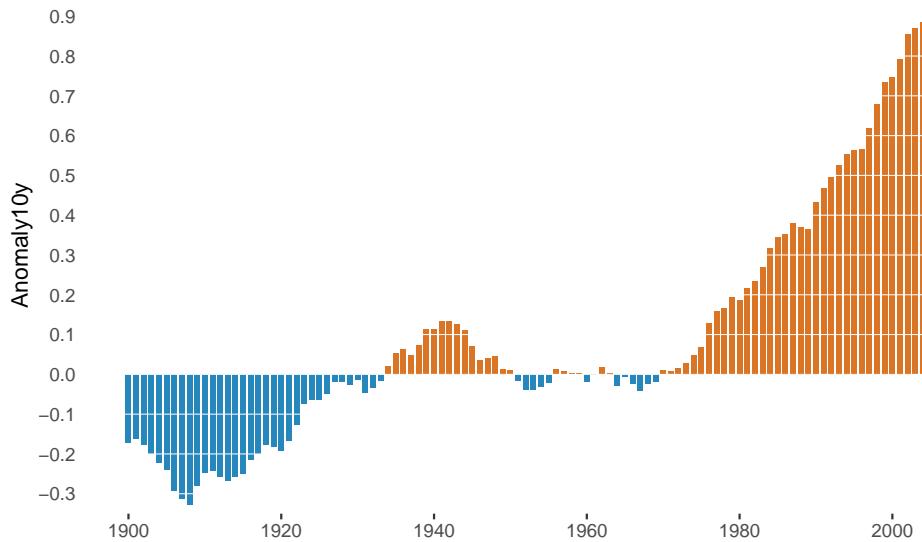


Figure 4.24: A diverging bar chart showing temperature anomaly data values from 1900 to 2014

```

    legend.position = "none"
)

ggplot(climate_sub, aes(x = Year, y = Anomaly10y, fill = pos)) +
  geom_col(position = "identity", alpha = 0.85, width=0.7) +
  scale_y_continuous(sec.axis = sec_axis(~ ., breaks = seq(-0.3, 0.9, by= 0.1)),labels =
  scale_x_continuous(breaks = c( 1900, 1920, 1940, 1960, 1980, 2000)) +
  scale_fill_manual(values=cb2) +
  theme(
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    #axis.ticks.x = element_blank(),
    #axis.text.x = element_text(angle = 90, vjust = 1, hjust = 1),
    axis.title.x = element_blank(),
    plot.margin = margin(6, 6, 3, 3),
    panel.background=element_blank(),
    panel.grid.major.y = element_line(size = 0.2, linetype = 'solid',colour = "white")
  )
}

```

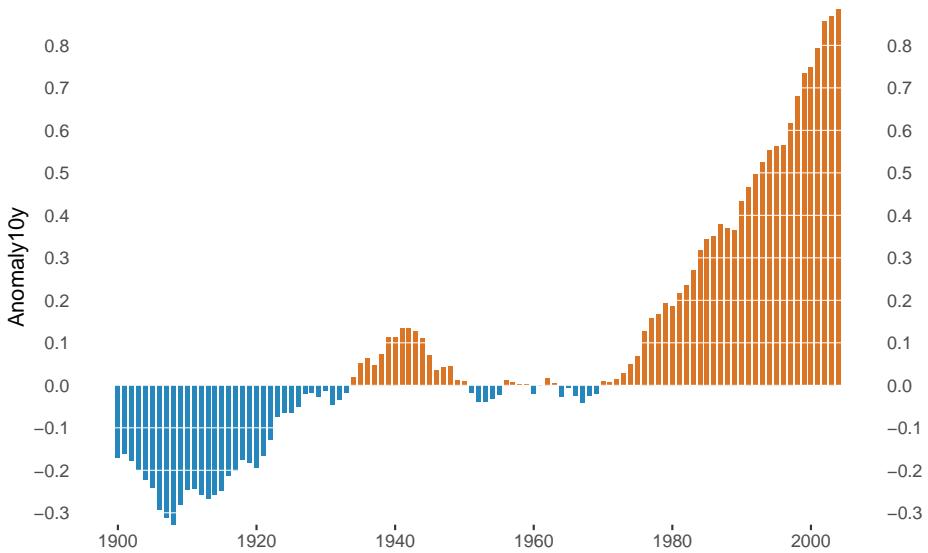


Figure 4.25: A diverging bar chart with extra y-axis, showing temperature anomaly data values from 1900 to 2014

```
panel.on top = TRUE,
legend.position = "none"
)
```

4.1.8 The Pareto Chart

A Pareto chart is a bar chart (with the bars ordered) combined with a line chart that shows the cumulative total or frequency of the bar values.

It is very useful for seeing at a glance what proportion of the data is represented by the first n bars.

It allows us to visualise how ranked quantities contribute to the whole

It is useful for revealing something like the 80-20 rule—e.g. 80% of the accidents are due to 20% of the possible reasons.

The data for the example below came from the Road Safety Authority (RSA) of Ireland from the report on its web site on Vehicle factors in fatal accidents on Irish roads 2012 to 2018.

The visualisation shows the disproportionate involvement of private cars in fatal road accidents in the period.

There is a little bit of data preparation required for the Pareto chart. We need to represent the cumulative frequency of occurrence of the category values we wish to plot.

```
# load packages
library(dplyr)
library(ggplot2)

fatal_acc <- read.csv("../data/RSA_vehicle_fatal_accidents-2012-2018xlsx.csv")
# str gives the type of the data container (a data.frame), the variables and types
str(fatal_acc)

## 'data.frame':   9 obs. of  2 variables:
## $ vehicle: Factor w/ 9 levels "HGV","Motorcycle",...: 4 1 2 8 3 5 7 6 9
## $ n      : int  831 103 96 94 37 27 17 16 9
# If we want to plot this as a pareto chart we need to represent the cumulative frequency

fatal_acc_df_cumulative <- fatal_acc %>%
  mutate(relative_freq = n/sum(n), cumulative_freq = cumsum(relative_freq))

head(fatal_acc_df_cumulative)

## #> #>       vehicle    n relative_freq cumulative_freq
## #> 1     Private Car 831    0.67560976    0.6756098
## #> 2             HGV 103    0.08373984    0.7593496
## #> 3     Motorcycle 96    0.07804878    0.8373984
## #> 4            Van 94    0.07642276    0.9138211
## #> 5    Pedal Cycle 37    0.03008130    0.9439024
## #> 6 PSV Bus/MiniBus 27    0.02195122    0.9658537
```

The first plot is a standard bar chart of the data. It's fine. However, we are after a chart that shows us the breakdown of values per vehicle type - but also shows us the cumulative contribution of the top ranked categories to the whole.

```
library(scales)

# plot
p <-
  ggplot(fatal_acc_df_cumulative, aes(x = reorder(vehicle, -n), y = n)) +
  geom_col(width = 1, fill = "#0571b0", color="white") +
  scale_y_continuous(breaks=seq(0,1000, by =100)) +
  ggttitle("vehicles involved fatal road accidents 2012-2018 ") +
  #labs(x = "", y = "Number of vehicles") +
  theme(plot.title = element_text(hjust = 0.5),
```

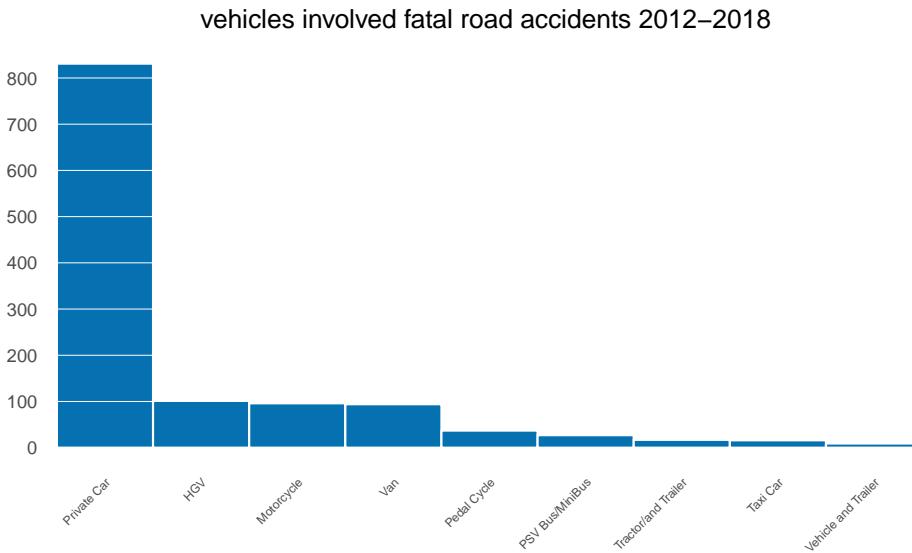


Figure 4.26: A standard bar chart illustrating the vehicle causes of road fatalities on Irish roads, 2012 to 2018

```

axis.line.y = element_blank(),
axis.ticks.y = element_blank(),
axis.line.x = element_blank(),
axis.ticks.x = element_blank(),
axis.title.y = element_blank(),
axis.title.x = element_blank(),
axis.text.x = element_text(size = 6, angle = 45, vjust = 1, hjust = 1),
panel.background=element_blank(),
panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
panel.on top = TRUE
)
p

```

This is a Pareto chart of the data.

The Pareto chart adds a line that runs from the top of the first bar with subsequent points indicating the cumulative percentage of each bar.

This allows us to immediately see and state that 91% of fatal accidents in the period were caused by the first four categories of motorised vehicles.

The line giving the cumulative percentages is a minor enhancement to the bar plot – but offers major gains in being able to interpret and make statements about contributions of the largest categories to the whole.

The main points to observe in the ggplot code are

Four geoms are used. `geom_col` is used to represent the bars. `geom_line` is used to represent the Pareto line. `geom_point` is used for the grey points on the line. `geom_text` is used to label the point positions. Note how these geoms have their own `aes` functions. This overrides the `aes` settings in the `ggplot` function. `geom_line`, `geom_point` and `geom_text` have their own `aes` function because their `y` values represent *cumulative* totals.

By default ggplot would plot the line directly on top of (invisible) points vertically aligned with `x-axis` breaks. E.g. the line would start at the `x` value indicated by the value *Private Car*. However, it is typical for Pareto charts to start at the right most edge of the bar. To achieve that I have had to nudge the (invisible) points underpinning the line using `position=position_nudge(x = 0.5, y = 0)`. As the `geom_point` and `geom_text` elements follow the line, I have done the same with their positioning.

```
library(scales)

total_accidents <- sum(fatal_acc_df_cumulative$n)

# plot
p <-
  ggplot(fatal_acc_df_cumulative, aes(x = reorder(vehicle, -n), y = n)) +
  geom_col(width = 1, fill = "#0571b0", color="white") +
  scale_y_continuous(breaks=seq(0,1200, by =100)) +
  #scale_y_continuous(sec.axis = sec_axis(~./total_accidents)*100, name = "Percentage")

  # the pareto line
  geom_line(aes(x = reorder(vehicle, -n), y = cumulative_freq*total_accidents), position = "identity")
  # points on the pareto line
  geom_point(aes(x = reorder(vehicle, -n), y = cumulative_freq*total_accidents), position = "identity")
  geom_text(aes(x = reorder(vehicle, -n), y = cumulative_freq*total_accidents, label = vehicle),
            position = "identity",
            hjust = 0.5, vjust = 0.5, angle = 45)
```

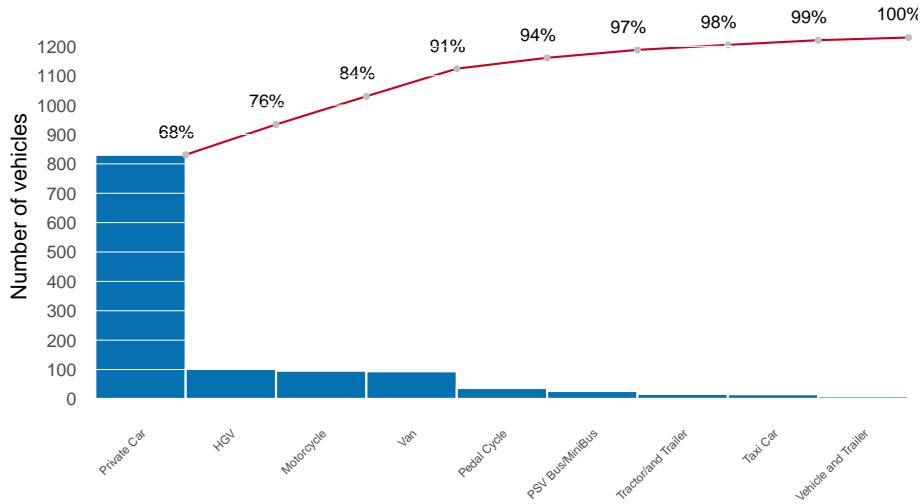


Figure 4.27: A pareto chart illustrating the cunmulative contribution to the whole of vehicle causes of road fatalities on Irish roads, 2012 to 2018

```

panel.background=element_blank(),
panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
panel.on top = TRUE

)

p

```

4.1.9 Weaknesses of bars

There are situations where bars don't work. You may recall at the start of video one this week that you learned that the primary aesthetic that a bar chart relies on is the *length* of the bar measured from the from the zero point.

There are situations where the values that we want to visualise have very small differences between them. If we visualise them using bar length we will have little sense of the differences between the values.

This is a dataset that I downloaded from athletics Ireland.

The view of the data I want to visualise gives the top 10 Irish female athletes in the hundred meters sprint in 2019.

You can see in the snippet of data on the slides that the running times vary from 11.33 seconds to 11.61 seconds.

Visualising this on a bar chart would look like this:

```
library(ggplot2)

ire_100m_2019 <- read.csv("../data/ire-2019-100m-best.csv")

# select the top female 100 m runners in 2019 by time
ire_100m_2019_f <- ire_100m_2019 %>% filter(competition == "Women") %>% top_n(10, wt = -time)

head(ire_100m_2019_f)

##   rank   time    PB      name      date competition
## 1    1 11.33  TRUE Ciara Neville 27/07/2019     Women
## 2    2 11.45  TRUE Gina Akpe-Moses 01/06/2019     Women
## 3    3 11.51 FALSE Phil Healy 06/04/2019     Women
## 4    4 11.58  TRUE Molly Scott 27/07/2019     Women
## 5    5 11.59 FALSE Joan Healy 27/07/2019     Women
## 6    6 11.61 FALSE Patience Jumbo-Gula 27/07/2019     Women

theme_set(theme_classic())

ggplot(ire_100m_2019_f, (aes(x = reorder(name, -time) , y = time))) +
  geom_col(width = 0.6, fill = "#0571b0", color = "white") +
  scale_y_continuous(limits = c(0, 12),
                     expand = c(0, 0),
                     breaks = seq(2, 12, by = 2),
                     #labels = c("50", "100", "150", "200", "250", "300", "350"),
                     name = "seconds") +
  ggtitle("2019 100m Women Irish National Performance Best ") +
  coord_flip() +
  theme(
    axis.title.x = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.title.y = element_blank(),
    plot.title = element_text(size = 11.5),
    plot.margin = margin(3, 6, 3, 3),
    panel.background = element_blank(),
    panel.grid.major.x = element_line(size = 0.4, linetype = 'solid', colour = "white"),
    panel.on top = TRUE
```

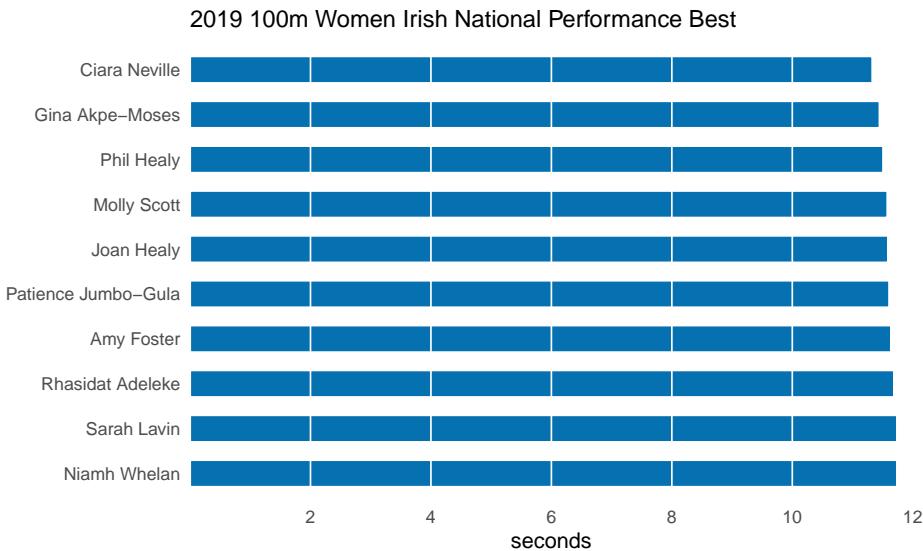


Figure 4.28: 2019 Top 10 Irish female athletes: hundred metres sprint

)

However, while the next plot tries to address this, it is completely wrong. Remember, the aesthetic to represent quantity is *length*.

The visual message conveyed by this chart is that Niamh Whelan's running times are 4 or 5 times faster than Ciara Neville's – when, in fact, there is a bout 0.5 of a second between them.

You have to refer to the axis to figure this out.

Never truncate a bar.

```
library(ggplot2)

ire_100m_2019 <- read.csv("../data/ire-2019-100m-best.csv")

# select the top female 100 m runners in 2019 by time
ire_100m_2019_f <- ire_100m_2019 %>% filter(competition=="Women") %>% top_n(10, wt=-time)

head(ire_100m_2019_f)

##   rank   time    PB      name      date competition
## 1    1 11.33  TRUE  Ciara Neville 27/07/2019     Women
## 2    2 11.45  TRUE  Gina Akpe-Moses 01/06/2019     Women
## 3    3 11.51 FALSE  Phil Healy 06/04/2019     Women
```

```

## 4    4 11.58 TRUE      Molly Scott 27/07/2019      Women
## 5    5 11.59 FALSE     Joan Healy 27/07/2019      Women
## 6    6 11.61 FALSE Patience Jumbo-Gula 27/07/2019 Women
theme_set(theme_classic())

ggplot(ire_100m_2019_f, (aes(x= reorder(name, -time) , y=time))) +
  geom_col( width=0.6, fill = "#0571b0", color="white") +
  scale_y_continuous(limits = c(11.25, 11.75), oob = rescale_none,
                      expand = c(0, 0),
                      #breaks = c( 5e4, 1.0e5, 1.5e5, 2.0e5, 2.5e5, 3e5, 3.5e5 ),
                      #labels = c("50", "100", "150", "200", "250", "300", "350" ),
                      name = "seconds") +
  ggttitle("2019 100m Women Irish National Performance Best ") +
  coord_flip() +
  theme(
    #axis.title.x = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.title.y = element_blank(),
    plot.title = element_text(size = 11.5),
    plot.margin = margin(3, 6, 3, 3),
    panel.background = element_blank(),
    panel.grid.major.x = element_line(size = 0.4, linetype = 'solid', colour = "white"),
    panel.on top = TRUE
  )

```

4.2 Dot plots

Cleveland Dot Plots are dot plots where the dots represent a value associated with a category value. They are an alternative to bar plots. They can reduce visual clutter and can be easier to read. The name comes from the statistician from William Cleveland (1993) Visualizing Data; Hobart Press

Dot plots are useful when all the values fall within a narrow range far from zero

In bar graphs, the *length* of the bar accurately communicates quantity only if it starts from zero

In dot plots the *position* of the dot communicates quantity. Therefore, the

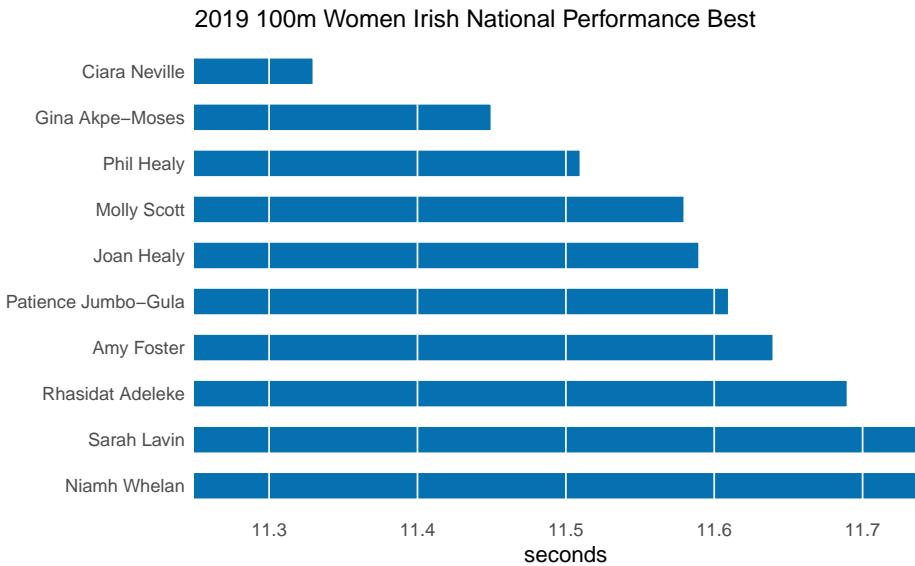


Figure 4.29: an example of a truncated bar plot. This is incorrect because the bar length is misrepresented, which misrepresents the quantity being encoded

quantitative scales in dot plots do not need to begin at zero

You can zoom into the range containing the data

The simplest way to create a dot plot is with `geom_point()`

```
ggplot(ire_100m_2019_f, aes(x=time, y = name)) +
  geom_point()
```

However this is not a particular useful plot. Dot plots are often generally ordered by the variable on the x-axis.

```
ggplot(ire_100m_2019_f, aes(x=time, y = reorder(name,-time))) +
  geom_point() +
  scale_x_continuous(limits = c(11.25, 11.78),
                     expand = c(0, 0),
                     breaks = seq(11.3, 11.7, by = 0.1),
                     name = "seconds") +
  theme(axis.title.y = element_blank())
```

The problem with this plot is that it is hard to relate each dot to the Athlete's name on the *y-axis*.

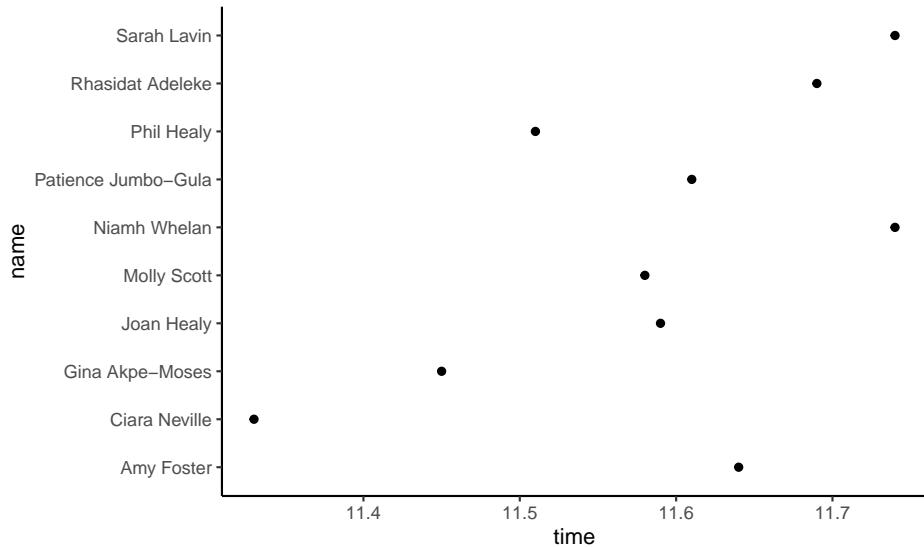


Figure 4.30: An overly simple dot plot

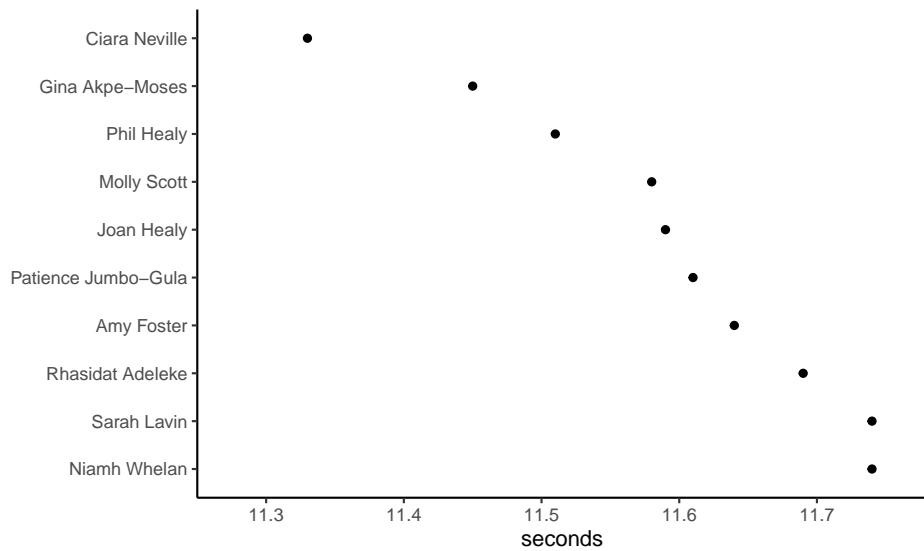


Figure 4.31: A slightly better dot plot

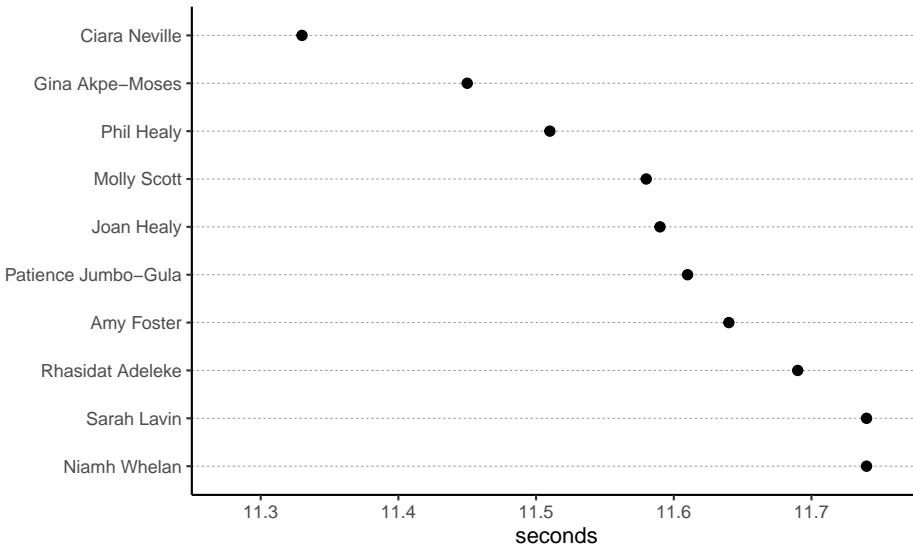


Figure 4.32: (#fig:athletics_dot3)A Cleveland plot of 2019 Top 10 Female 100m times

We will then modify this by adding a custom theme where we make the vertical lines disappear and the horizontal lines become dashed lines

We will also make the dot larger by increasing the size value in geom_point.

This is what is often known as a Cleveland dot plot.

```
ggplot(ire_100m_2019_f, aes(x=time, y = reorder(name,-time))) +
  geom_point(size=2) +
  scale_x_continuous(limits = c(11.25, 11.78),
                     expand = c(0, 0),
                     breaks = seq(11.3, 11.7, by = 0.1),
                     name = "seconds") +
  theme(axis.title.y = element_blank(),
        panel.grid.major.x =element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.y = element_line(colour = "grey60", linetype = "dashed", size=0.15))
```

4.2.1 Lollipop plots : Separating and ordering by another category

You may want to order the data within a category

As an example we'll bring data from the 2019 men's 100m competition into the

plot.

In this case we want to sort first by the *competition* variable and then by *time*.

However, the **reorder** function only will allow ordering by one variable. So we have to use the **order** function

By default when we create a bar chart ggplot will order the category variable (in this case name) alphabetically; This is because the default factor ordering is alphabetical.

As we did in the last video when ordering the car data by *cylinder* and then by *mpg* we create a new factor ordering, for the name *variable). This will then we used by ggplot to order the names on the y-axis

Instead of using the guidelines as before, we will add a new geom type **geom_segment** which will run from the dot to the y axis

These plots are sometimes called lollipop plots - for obvious reasons.

```
library(dplyr)

# select the top male and female 100 m runners in 2019 by time

ire_100m_2019_m <- ire_100m_2019 %>% filter(competition == "Men") %>% top_n(10, wt = -time)

ire_100m_2019_m_f <- ire_100m_2019_m %>% bind_rows(ire_100m_2019_f)

# names sorted by first by competition and then by time
nameorder <- ire_100m_2019_m_f$name[order(ire_100m_2019_m_f$competition, ire_100m_2019_m_f$name)]

# Now we turn name into a factor, with levels in the order of name order
ire_100m_2019_m_f$name <- factor(ire_100m_2019_m_f$name, levels = nameorder)

# Now ggplot will display the names in the order specified by nameorder
ggplot(ire_100m_2019_m_f, aes(x = time, y = name)) +
  geom_segment(aes(yend = name), xend = 0, colour = "grey60", size = 0.15, linetype = "dotted")
  geom_point(size = 2.5, aes(colour = competition)) +
  scale_colour_manual(values = c("#ca0020", "#0571b0"), name = "") +
  scale_x_continuous(limits = c(10.2, 11.82),
                     expand = c(0, 0),
                     breaks = seq(10.4, 11.8, by = 0.2),
                     name = "seconds") +
  theme(axis.title.y = element_blank(),
        panel.grid.major.x = element_line(size = 0.02, colour = "grey50"),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.y = element_blank(),
```

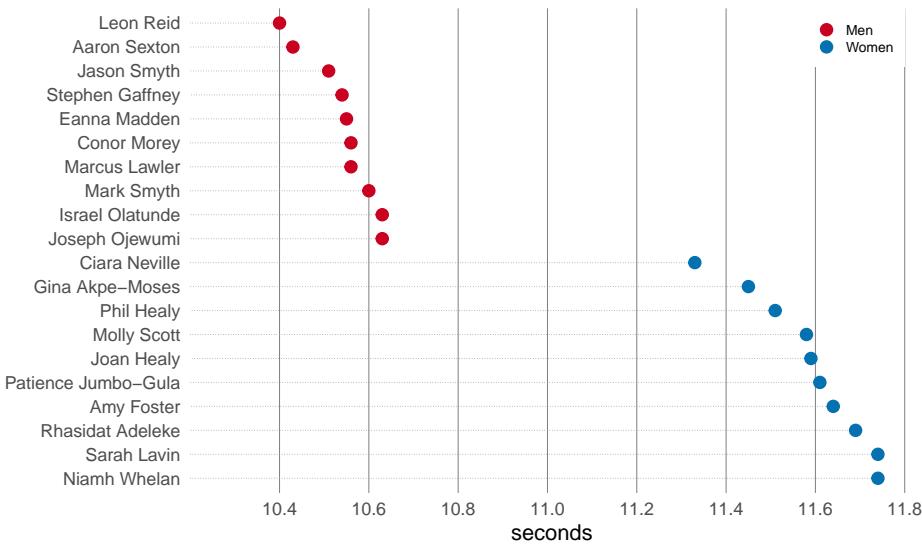


Figure 4.33: 2019 Top Irsih Female and Male 100m times

```

axis.line.y = element_blank(),
axis.line.x = element_blank(),
axis.ticks.y = element_blank(),
axis.ticks.x = element_blank(),
legend.position= c(0.92, 0.95),
legend.text = element_text(size = 7), # legend text was a little large
legend.key.size = unit(0.6, "lines"),
legend.title = element_blank()# legend keys were a little large

```

4.2.2 Combining and Comparing multiple points

The power of the dot plot becomes evident when we want to combine and compare multiple points of information.

Consider the case where we want to show running times for the top 10 Irish female athlete in 2019 compared to the 2018 season.

The first work we have to do is to transform the dataframe into long format where we have four columns : *name*, *pb*, *year* and *value*

Then I order the names by *2019 values* in descending order. That is because the intent of the visualisation is to highlight the *2019* values and use the *2019* values for points of comparison.

```

library(dplyr)

ire_100m_2018_2019 <- read.csv("../data/Athletics_Women_100m_2018_2019.csv")

head(ire_100m_2018_2019)

##   rank_2019      name X2018 X2019     pb
## 1       1 Ciara Neville 11.54 11.33 pb_2019
## 2       2 Gina Akpe-Moses 11.46 11.45 pb_2019
## 3       3 Phil Healy 11.28 11.51 pb_2018
## 4       4 Molly Scott 11.76 11.58 pb_2019
## 5       5 Joan Healy 11.57 11.59 pb_2018
## 6       6 Patience Jumbo-Gula 11.51 11.61 pb_2018

# This is in wide form. We will transform to a dataframe in long format
ire_100m_18_19_long <- ire_100m_2018_2019 %>% gather(year, value, X2018:X2019)
# This makes the year values more readable
ire_100m_18_19_long$year[ire_100m_18_19_long$year == "X2019"] <- "2019"
ire_100m_18_19_long$year[ire_100m_18_19_long$year == "X2018"] <- "2018"

# This gets the personal best information into a single column pb
ire_100m_18_19_long$pb <- ifelse((ire_100m_18_19_long$pb=="pb_2019" & ire_100m_18_19_long$year=="2019") | (ire_100m_18_19_long$pb=="pb_2018" & ire_100m_18_19_long$year=="2018"), "pb", "no pb")

# This is yet another way to order the level of a variable so that it displays on the axis
nameorder <- ire_100m_18_19_long %>% filter(year=="2019") %>%
  arrange(-value) %>%
  mutate(name = factor(name, levels = .$name))

ire_100m_18_19_long$name =factor(ire_100m_18_19_long$name, levels = nameorder$name)

head(ire_100m_18_19_long)

##   rank_2019      name pb year value
## 1       1 Ciara Neville 2018 11.54
## 2       2 Gina Akpe-Moses 2018 11.46
## 3       3 Phil Healy pb 2018 11.28
## 4       4 Molly Scott 2018 11.76
## 5       5 Joan Healy pb 2018 11.57
## 6       6 Patience Jumbo-Gula pb 2018 11.51

ggplot(ire_100m_18_19_long, aes(x = value, y= name)) +
  geom_line(aes(group = name), colour = "grey", size=0.5) +
  geom_point(aes(colour = year), size = 3, alpha = 0.7) +

```

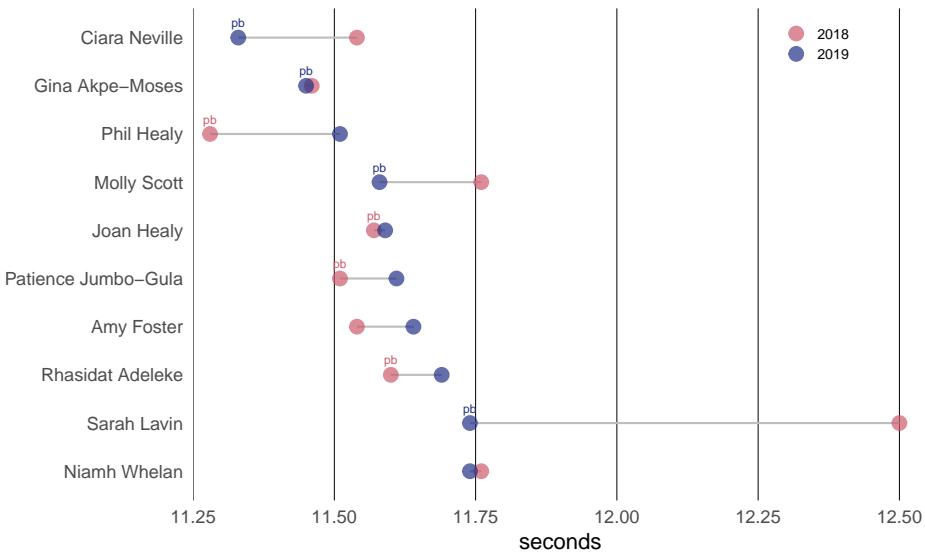


Figure 4.34: 2019 top Irish female 100 metres sprint : 2019 vs 2018 season bests

```

geom_text(aes(label=pb, colour = year), vjust=-1.2, size=2, na.rm = TRUE, show.legend = FALSE)

scale_colour_manual(values= c("#ce5a6c","#212f85"), name = "") +
scale_x_continuous(limits = c(11.25, 12.55),
                   expand = c(0, 0),
                   breaks = seq(11.25, 12.5, by = 0.25),
                   name = "seconds") +


theme(axis.title.y = element_blank(),
      panel.grid.major.x =element_line(size=0.03),
      panel.grid.minor.x = element_blank(),
      panel.grid.major.y = element_blank(),
      axis.line.y = element_blank(),
      axis.line.x = element_blank(),
      axis.ticks.y = element_blank(),
      axis.ticks.x = element_blank(),
      legend.position= c(0.85, 0.94),
      legend.text = element_text(size = 7), # legend text was a little large
      legend.key.size = unit(0.7, "lines"),
      legend.title = element_blank())# legend keys were a little large)

```

4.2.3 Diverging Dot Plots

This diverging dot plot shows a different view of the race time data - with a focus on improvement since 2018. The values on the x-axis now represents the difference in race times from 2018 to 2019 for each athlete.

I am using the original wide form of the data here. I use dplyr's mutate function to create a new column containing the difference between the 2018 and 2019 times

```
library(dplyr)

# I am using the orginal wide form of the data here. I use dplyr's mutate function to
# a new column containing the difference between the 2018 and 2019 times

ire_100m_2018_2019_diff <- ire_100m_2018_2019 %>%
  select(name, X2018, X2019) %>%
  mutate(diff = X2018-X2019,
        above = ifelse(diff > 0, TRUE, FALSE))%>%
  arrange(diff)%>%
  mutate(name = factor(name, levels = .$name))

ire_100m_19W_pb<- ire_100m_18_19_long%>%
  filter(year=="2019")%>%
  select(name,year, pb)

ire_100m_2018_2019_diff_pb<- merge(ire_100m_2018_2019_diff,ire_100m_19W_pb, by="name") 

ggplot(ire_100m_2018_2019_diff_pb, aes(x=diff, y=name, colour = above)) +
  geom_segment(aes(x = 0, y = name, xend = diff, yend = name), size = 0.6, color
  geom_point(size =2.5) +
  geom_text(aes(label=pb), vjust=-1.2, size=2, na.rm = TRUE, show.legend = FALSE)+

  scale_colour_manual(values= c("#ce5a6c","#212f85"), name = "") +
  scale_x_continuous(limits = c(-0.3, 0.82),
                     expand = c(0, 0),
                     name = "2018 vs 2019 difference in seconds",
                     breaks = seq(-0.2,0.8, by = 0.2)) +
  theme(axis.line.y = element_blank(),
        panel.grid.major.x = element_line(size=0.04, colour = "grey50"),
        axis.line.x = element_blank(),
        axis.ticks.y = element_blank(),
```

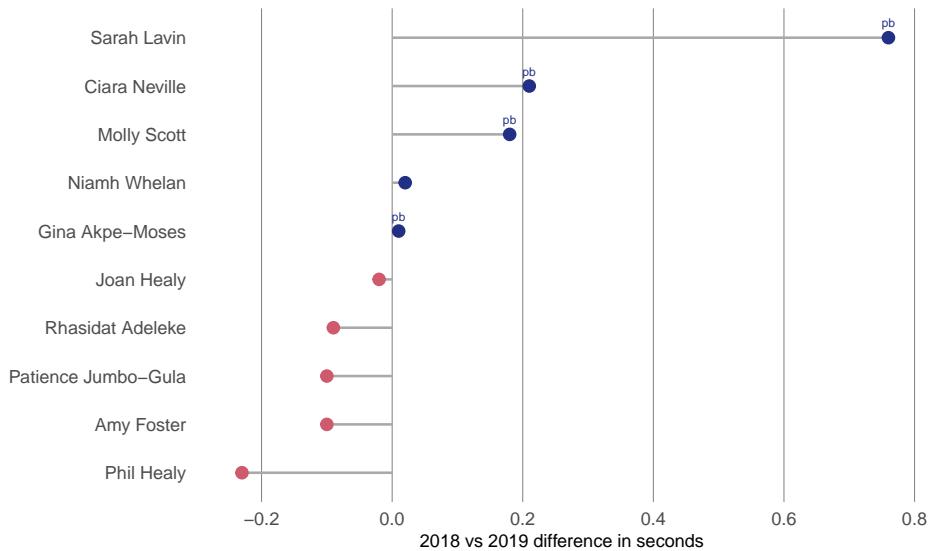


Figure 4.35: 2019 Top Female 100m sprint : Difference in Season best 2018-2019

```
axis.ticks.x = element_blank(),
axis.title.y = element_blank(),
legend.position = "None",
axis.title.x = element_text(size = 9))
```


Chapter 5

Pies and Proportions

We will look at how to represent basic proportions, emphasising alternatives to Pie charts

- Pie Charts
- Vertical Stacked Bars Charts
- Horizontal Stacked Bars Charts
- Pareto Charts
- Waffle Charts

5.1 The Pie Chart

I'll create a pie chart of the composition of the Irish parliament, the Dail, before the election in February 2020.

This data was scraped from the oireachtas web site. I do some preprocessing on it to amalgamate some of the smaller parties/ vacant seats and the ceann comhairle (speaker of the parliament) into an ‘other’ category

```
library(dplyr)
library(igraph)
library(forcats)

dail_32 <- read.csv("../data/ire_dail_parties-2016-20.csv")
dail_32<-dail_32%>%select(c(1,3))

# replace parties with 1 seat with a collective 'other parties'
oth<- dail_32%>%filter(X2020<2)%>%select(X2020)%>%summarise(sum(X2020))
oth$party <-"Other/Vacant"
oth<- oth %>% rename("X2020" ="sum(X2020)" )
```

```
# filter our these seats
dail_32<-dail_32%>%filter(X2020>=2)
# add the 'Other' row
dail_32<-rbind(dail_32, oth)

# calculate proportions
dail_32$X2020_prop <- round(dail_32$X2020*100/sum(dail_32$X2020),0)

# add a shortened name for each party
dail_32$short <- c("FG", "FF", "SF", "IND", "LAB", "S/PBP", "SDP", "GP", "OTH" )

# add a display label for each party
dail_32$label <- paste0(dail_32$short, ":", dail_32$X2020_prop, "%")

head(dail_32)
```

##		party	X2020	X2020_prop	short	label
## 1		Fine Gael	47	30	FG	FG:30%
## 2		Fianna Fáil	45	28	FF	FF:28%
## 3		Sinn Féin	22	14	SF	SF:14%
## 4		Independent	22	14	IND	IND:14%
## 5		Labour Party	7	4	LAB	LAB:4%
## 6		Solidarity-PBP	6	4	S/PBP	S/PBP:4%

I then looked at possible colours I might use for each party.

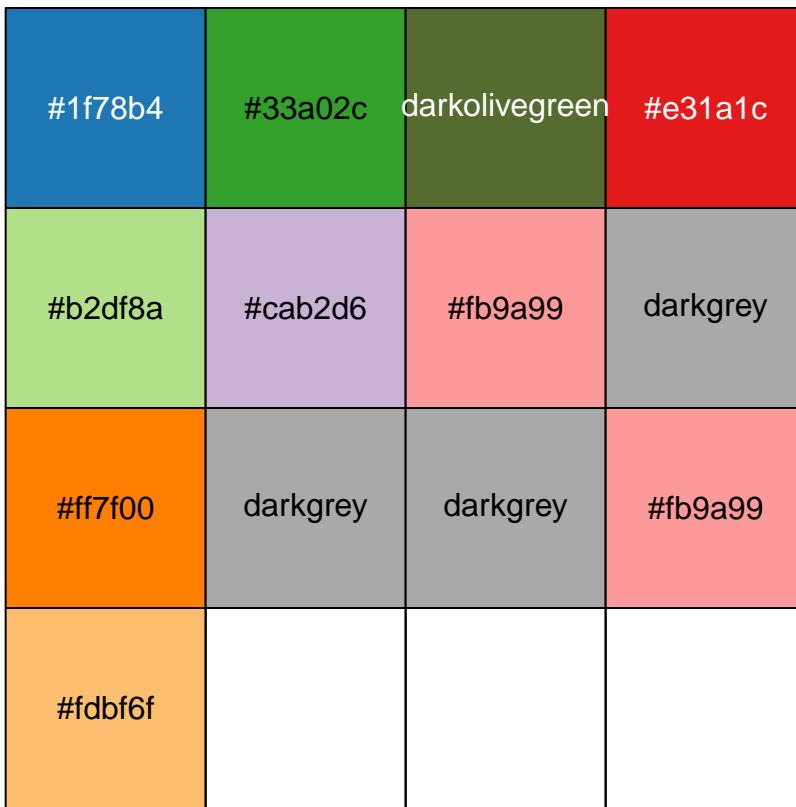
The `party.colour` list is where I make the assignment between the party short name and a colour

```
library(scales)

# possible colours
mycols <- c('#a6cee3', '#1f78b4', '#fb9a99', '#b2df8a', '#e31a1c', '#ff7f00', '#fdbf6f', '#33a02c')

party.colours <- c('FG' = '#1f78b4', 'FF' = '#33a02c', 'SF' = 'darkolivegreen', 'LAB' = 'darkblue')

# a handy way to see colours in your palette. show_col is in the scales package
show_col(party.colours)
```



5.1.1 Making the pie

In terms of creating a pie chart, ggplot doesn't have an *oven ready* version. Thanks to [this site]((<https://www.datanovia.com/en/blog/how-to-create-a-pie-chart-in-r-using-ggplot2/>)) for insight on how to create it

- Compute the position of the text labels as the cumulative sum of the proportion:
- Arrange the grouping variable (*short*) in descending order. This is important to compute the y coordinates of labels. Recall that *short* refers to the shortened party name I created earlier
- To put the labels in the center of pies, we'll use `cumsum(prop) - 0.5*prop` as label position.
- Key ggplot functions: `geom_bar() + coord_polar()`.
- Add text labels: `geom_text()`
- Change fill color manually: `scale_color_manual()`

- Apply `theme_void()` to remove axes, background, etc

```
library(ggrepel)

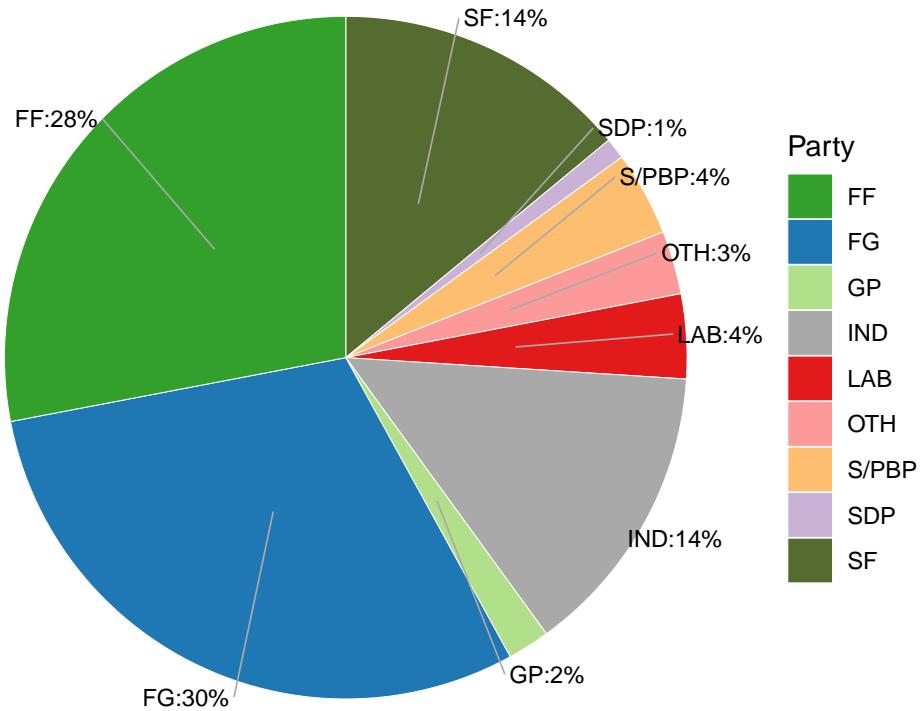
set.seed(42)

# sort by 2020 seats a
#dail_32<-dail_32%>%mutate(party = fct_reorder(party, -X2020))%>%arrange(-X2020)

# compute the position of the text labels as the cumulative sum of the proportion:
# Arrange the grouping variable (short) in descending order. This is important to comp

dail_32<- dail_32 %>%
  arrange(desc(short)) %>%
  mutate(lab.ypos = cumsum(X2020_prop) - 0.5*X2020_prop)

ggplot(dail_32, aes(x = "", y = X2020_prop, fill = short)) +
  geom_bar(width = 1, stat = "identity", color = "white", size=0.1, ) +
  coord_polar("y", start = 0) +
  #geom_text(aes(y = lab.ypos, label = label), color = "white") +
  geom_text_repel(aes(y = lab.ypos, label = label),size=3, segment.color = "darkgrey",
  scale_fill_manual(values = party.colours, name= "Party") +
  theme_void()
```



You can see that the big issue is the difficulty in judging the proportions of the smaller segments.

Also from a practical perspective, the small segments make labelling difficult.

I had to use the `ggrepel` library for this, which places labels on the plot so that they do not overlap.

5.2 Bar charts

We can address some of these issues by creating a bar chart instead.

This chart goes against some of the rules of use of colour data visualisation that I've tended to stress. Colour is being used here to emphasise the distinction between parties.

A minimalist approach, would have a non coloured chart with the labels on the x-axis. See Fig below

I think that colour is permissible when the colour strongly signifies an understood difference between the categories, such that the lack of colour or the use of wrong colour will be perceived as a distraction for your reader.

```
library(ggplot2)
library(forcats)
```

```

theme_set(theme_classic())

# handy way to reorder factors
dail_32<-dail_32%>%mutate(party = fct_reorder(party, -X2020))

ggplot(dail_32, (aes(x= party, y=X2020, fill=short))) +
  geom_col(width=1, colour="white") +
  geom_text(aes(label=label, group = party), colour="white", size =3, position = position_dodge(1))
  scale_y_continuous(limits = c(0, 50),
                     breaks = seq(5,50, by = 5),
                     name = "seats") +
  scale_fill_manual(values = party.colours) +
  ggtitle("32nd Dail: 2020 by party ") +
  theme(
    legend.position = "none",
    #axis.title.y = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    #axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1),
    axis.text.x = element_blank(),
    axis.title.x = element_blank(),
    plot.title = element_text(vjust = -8, hjust = 0.25, size = 11),
    plot.margin = margin(6, 6, 3, 3),
    panel.background = element_blank(),
    #panel.grid.major.y = element_line(size = 0.05, linetype = 'solid', colour = "white"),
    panel.grid.major.y = element_blank(),
    panel.on top = TRUE
  )
)

```

This visualization makes it easier to perform a direct comparison of the three groups, though it obscures other aspects of the data (Figure 10.3). Most importantly, in a side-by-side bar plot the relationship of each bar to the total is not visually obvious. Note that Gridlines are not needed if we have the number in the bar.

Here is the version with a single colour for all bars.

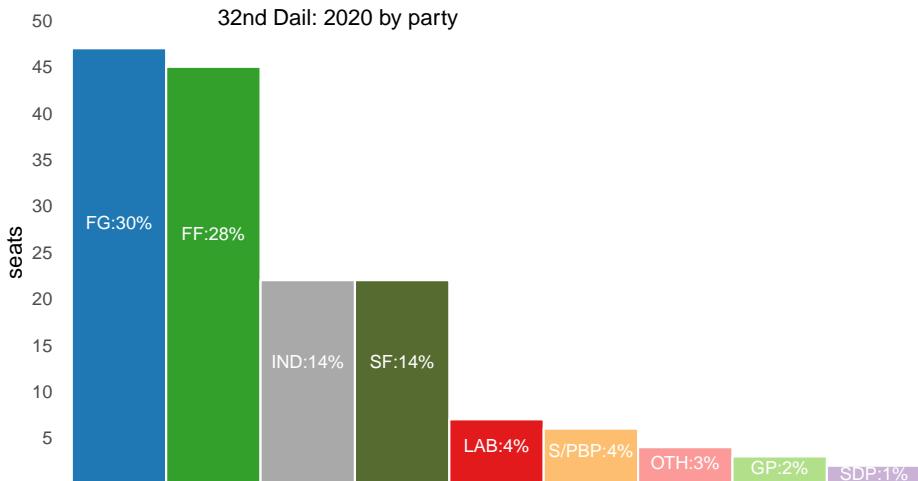


Figure 5.1: 32nd Dail, 2020. Party composition

```

library(ggplot2)
library(forcats)

theme_set(theme_classic())

# handy way to reorder factors
dail_32<-dail_32%>%mutate(party = fct_reorder(party, -X2020))

ggplot(dail_32, (aes(x= party, y=X2020))) +
  geom_col(width=1, colour="white", fill="#1f78b4") +
  geom_text(aes(label=label, group = party), colour="white", size =3, position = position_stack(vjust = 0)) +
  scale_y_continuous(limits = c(0, 50),
                     breaks = seq(5,50, by = 5),
                     name = "seats") +
  ggtitle("32nd Dail: 2020 by party ") +
  theme(
    legend.position = "none",
    #axis.title.y = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    )
  
```

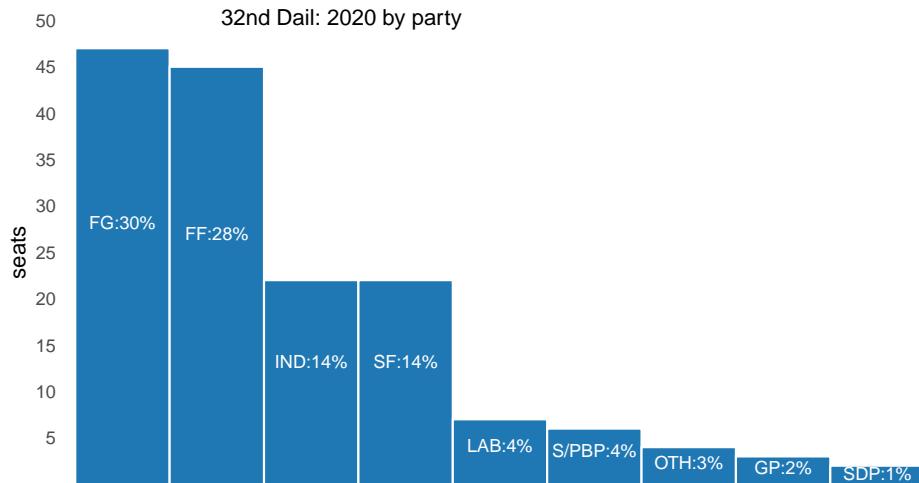


Figure 5.2: 32nd Dail, 2020. Party composition. No colouration

```

axis.ticks.x = element_blank(),
#axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1),
axis.text.x = element_blank(),
axis.title.x = element_blank(),
plot.title = element_text(vjust = -8, hjust = 0.25, size = 11),
plot.margin = margin(6, 6, 3, 3),
panel.background = element_blank(),
#panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
panel.grid.major.y = element_blank(),
panel.on top = TRUE
)

```

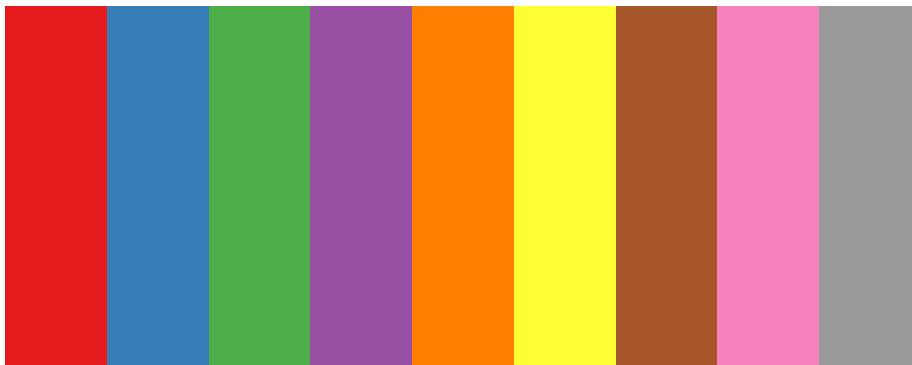
Here is the version with bars coloured according to the Set1 palette from RColorBrewer. I suggest that this no colouration of bars is better than inappropriate or wrong colouration.

```

library(ggplot2)
library(forcats)
library(RColorBrewer)

# View a single RColorBrewer palette by specifying its name
display.brewer.pal(n = 9, name = 'Set1')

```



Set1 (qualitative)

Figure 5.3: 32nd Dail, 2020. Party composition coloured by Set1 palette

```
# Hexadecimal color specification
brewer_cols <- brewer.pal(n = 9, name = "Set1")

theme_set(theme_classic())

# handy way to reorder factors
dail_32<-dail_32%>%mutate(party = fct_reorder(party, -X2020))

ggplot(dail_32, (aes(x= party, y=X2020, fill=short))) +
  geom_col(width=1, colour="white") +
  geom_text(aes(label=label, group = party), colour="white", size =3, position = position_stack(vjust = 0))
  scale_y_continuous(limits = c(0, 50),
                     breaks = seq(5,50, by = 5),
                     name = "seats") +
  scale_fill_brewer(type = "qual", palette="Set1")+
  ggtile("32nd Dail: 2020 by party ") +
  theme(
    legend.position = "none",
    #axis.title.y = element_blank(),
    axis.line.y = element_blank(),
```

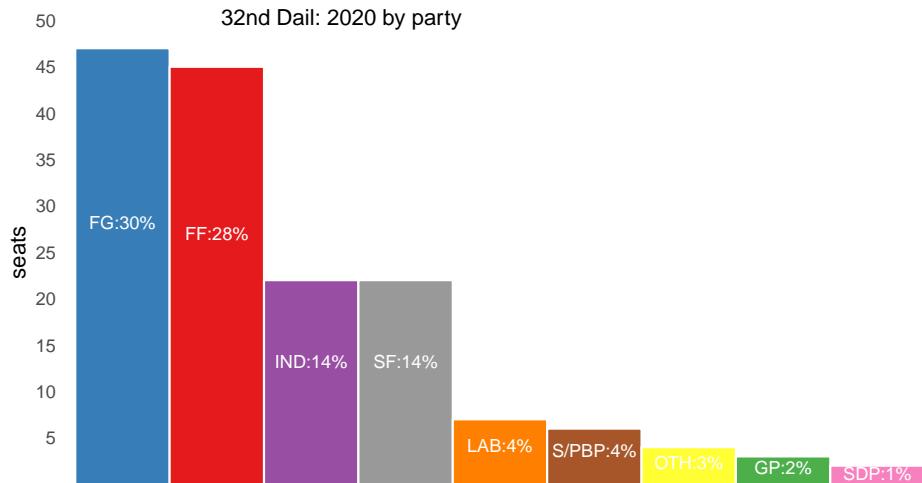


Figure 5.4: 32nd Dail, 2020. Party composition coloured by Set1 palette

```

axis.ticks.y = element_blank(),
axis.line.x = element_blank(),
axis.ticks.x = element_blank(),
#axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1),
axis.text.x = element_blank(),
axis.title.x = element_blank(),
plot.title = element_text(vjust = -8, hjust = 0.25, size = 11),
plot.margin = margin(6, 6, 3, 3),
panel.background = element_blank(),
#panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
panel.grid.major.y = element_blank(),
panel.on top = TRUE
)

```

5.2.1 Pareto Chart

By adding a line that tracks the cumulative percentages of the bars in descending order, we address the problem we first observed with a side by bar chart approach - that the relationship of each bar to the total is not visually obvious.

Here, the contribution of each bar to the cumulative total is clear. We can see immediately that the first four parties occupy 86% of all seats.

We can also judge that the Labour party occupies 4% of seats (90-86%).

```

library(ggplot2)
library(forcats)
library(dplyr)

dail_32 <- dail_32 %>% arrange(-X2020_prop, party)%>%
  mutate(cumulative = cumsum(X2020_prop))

total_seats <- sum(dail_32$X2020)

# redo the label to show num seats
dail_32$label <- paste0(dail_32$short, ":", dail_32$X2020)

theme_set(theme_classic())

# handy way to reorder factors
#dail_32<-dail_32%>%mutate(party = fct_reorder(party, -X2020))

ggplot(dail_32, (aes(x= party, y=X2020, fill=short))) +
  geom_col(width=1, colour="white") +
  #geom_text(aes(label=label, group = party), colour="white", size =2.7, position = position_stack(vjust=0))

# the pareto line
  geom_line(aes(x = party, y =(cumulative*total_seats)/100), position=position_nudge(x = 0.5, y = 0)) +
  # points on the pareto line
  geom_point(aes(x = party, y = (cumulative*total_seats)/100), position=position_nudge(x = 0.5, y = 0))

  geom_text(aes(x = party, y = (cumulative*total_seats)/100, label = sprintf("%1.2i%%", cumulative*total_seats)), position=position_nudge(x = 0.5, y = 0))

scale_y_continuous(limits = c(0, 170),
                   breaks = seq(10,160, by = 10),
                   name = "seats") +
  scale_x_discrete(labels = dail_32$label, expand = c(0,0.1))+

  scale_fill_manual(values = party.colours)+

  ggtitle("32nd Dail: 2020 by party ") +
  theme(
    legend.position = "none",
    #axis.title.y = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank())

```

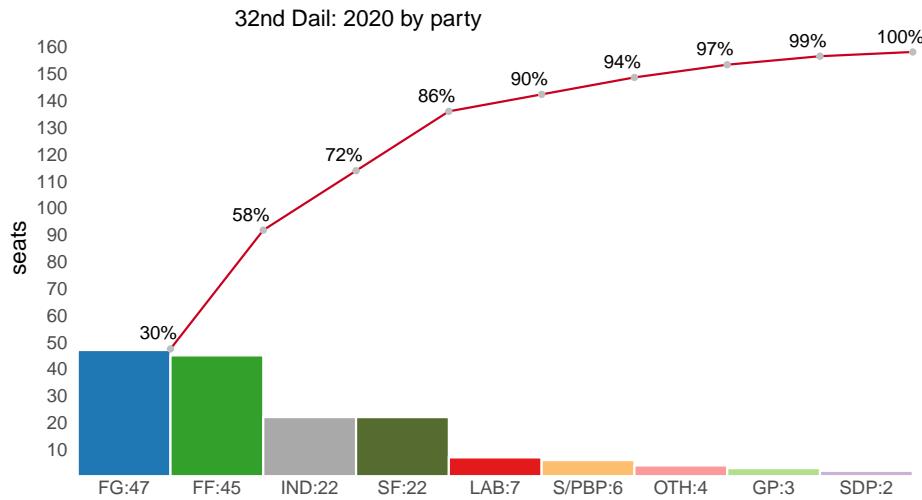


Figure 5.5: 32nd Dail, 2020. Party composition. Pareto Chart

```

axis.ticks.y = element_blank(),
axis.line.x = element_blank(),
axis.ticks.x = element_blank(),
axis.text.x = element_text(vjust = 8),

axis.title.x = element_blank(),
plot.title = element_text(vjust = -8, hjust = 0.25, size = 11),
plot.margin = margin(6, 6, 3, 3),
panel.background = element_blank(),
#panel.grid.major.y = element_line(size = 0.05, linetype = 'solid', colour = "white")
panel.grid.major.y= element_blank(),
panel.on top = TRUE

) +
expand_limits(x = c(3, 10))

```

5.2.2 Vertical Stacked Bar Chart

We can replace the side by side bar chart with a single stacked bar chart. It is better at giving visual sense of the proportions of each bar within the whole.

This is the bar equivalent of a pie chart, and it is better because it is easier to judge relative proportions of lengths than areas of a circle

I have ordered the bar bottom up in descending order – which means that the smaller proportions are together at the top of the chart and it is easier to judge

their relative proportions than if they were interspersed through out the bar

The plot here has labels within each bar segment showing the percentage values – which does give a sense, although not visual, of the proportions represented by each bar.

The plot has a legend, which adds extra cognitive load to the reader in reconciling the bar colours to the legend keys.

To create a stacked bar, you need to create a dummy variable on x axis. Here it is `x=2020`

To get the bar to be narrow like this, you have to set the width in your rendering environment. In the `{r}` section enclosing this code in the rmd file, I set the `fig.width=3`.

```
library(ggthemes)

mycols<-rev(mycols)

#handy way to reverse order of rows
dail_32<-dail_32%>%mutate(short = fct_reorder(short, X2020))%>%arrange(X2020)

ggplot(dail_32, aes(x = "2020", y = X2020, fill = short))+
  geom_col(width = 0.4) +
  geom_text(aes(label = paste0(X2020_prop, "%")),
            position = position_stack(vjust = 0.5), size=2.75, colour = "white") +
  scale_y_continuous(breaks = seq(0,160, by = 20), expand=c(0,0))+ 
  scale_x_discrete(expand = c(0, 0))+
  
  scale_fill_manual(values = party.colours)+ 
  theme(legend.position = "right",
        legend.title = element_blank()) +
  theme(axis.title.y = element_text(margin = margin(r = 10)),
        axis.title.x = element_blank(),
        axis.line.x = element_blank(),
        axis.line.y = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank()) +
  #theme(aspect.ratio = .2) +
  ylab("Seats") +
  xlab(NULL)
```

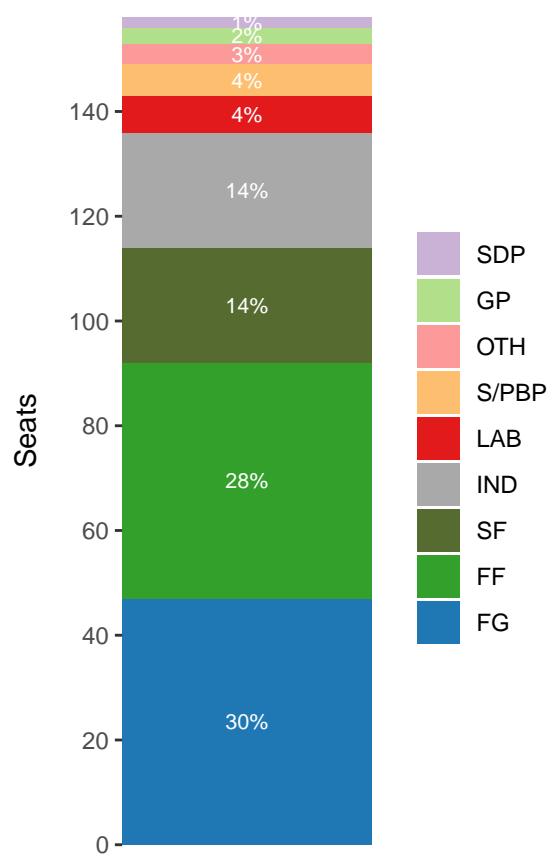


Figure 5.6: 32nd Dail, 2020. Party composition Vertical Stacked Bar Chart

In the next version, I've removed the legend and placed the party labels within the bars. In order to give a sense of the overall percentage of the whole I've put a second y axis on right of the figure.

This is better solution than the pie chart by a long way – but it is still not a perfect solution by any means.

Visually we can get a sense of the relative proportions – but it is hard to judge exact values. For instance, how many seats have Labour in the parliament?

And we have labelling issues at the top where the percentages become very small.

```
library(ggthemes)

mycols<-rev(mycols)

#handy way to reverse order of rows
dail_32<-dail_32%>%mutate(short = fct_reorder(short, X2020))%>%arrange(X2020)

ggplot(dail_32, aes(x = "2020", y = X2020, fill = short))+
  geom_col(width = 0.4) +
  geom_text(aes(label = short),
            position = position_stack(vjust = 0.5), size=2.6, colour = "white") +
  scale_y_continuous(breaks = seq(0,160, by = 20), expand=c(0,0), sec.axis = sec_axis(~ ./ total,
    name = "Percentage"))+
  scale_x_discrete(expand = c(0, 0))

#theme_economist(base_size = 8) +
scale_fill_manual(values = party.colours)+
theme(legend.position = "None",
      legend.title = element_blank()) +
theme(axis.title.y = element_text(margin = margin(r = 10)),
      axis.title.x = element_blank(),
      axis.line.x = element_blank(),
      axis.line.y = element_blank(),
      axis.text.x = element_blank(),
      axis.ticks.x = element_blank()) +
#theme(aspect.ratio = .2) +
ylab("Seats") +
xlab(NULL)
```

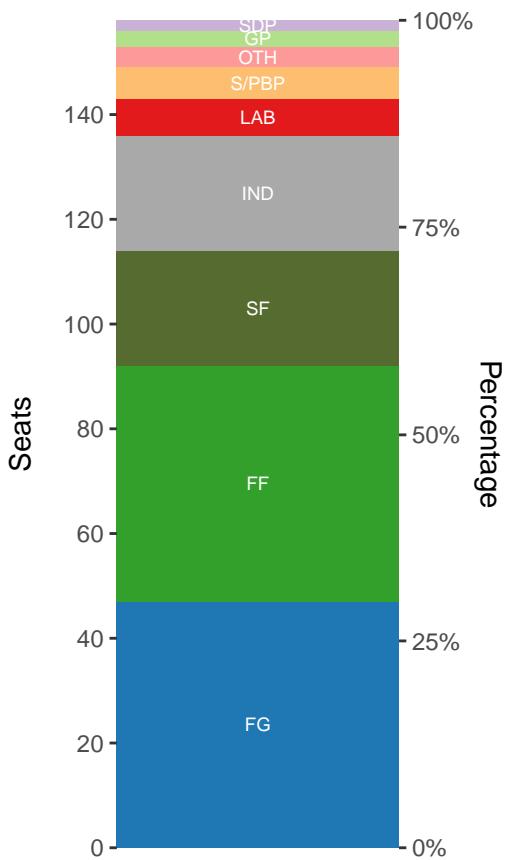


Figure 5.7: 32nd Dail, 2020. Party composition Vdsrtical Stacked Bar Chart with percentages

5.2.3 Horizontal Stacked Bar Chart

A similar solution is the horizontal stacked bar chart. The principle is the same as the vertical stacked bar but I suggest that this much easier to read.

As before the bars are ordered in descending order – but now, rather than from bottom to top we are reading from left to right, which is much more intuitive to western readers at least.

There is still a labelling issue on the right hand side. I have added labels connected by line segments. It's not an ideal solution - but where the bars become small and crowded there isn't a lot of choice.

```
library(ggplot2)
library(forcats)
library(dplyr)

dail_32<-dail_32 %>%
  mutate(year = as.factor("2020"))

#handy way to reverse order of rows
dail_32<-dail_32%>%mutate(short = fct_reorder(short, X2020))%>%arrange(X2020)

theme_set(theme_classic())

annotate_size <- 2.5
v_just= -5

ggplot(dail_32, aes(x=year, y=X2020, fill=short)) +
  geom_col(width = 0.2) +
  geom_text(aes(label=X2020, group = short), colour="white", size =3, position = position_stack(vjust = v_just))

scale_y_continuous(limits = c(0, 166),
                   expand = c(0.0, 0),
                   breaks = seq(5,160, by = 20),
                   name = "seats") +
  #scale_x_discrete(expand = c(0,0 ))+
  scale_fill_manual(values = party.colours) +
  ggtitle("32nd Dail: 2020 by party ") +
```

```

# Using manual annotation like this is far from ideal.

annotate("text", x="2020", vjust= v_just, y = 10, label = dail_32[9,4], size = annot

annotate("text", x="2020", vjust= v_just,y = 60, label = dail_32[8,4], size = annot

annotate("text", x="2020", vjust= v_just, y = 100, label = dail_32[7,4], size = annot

annotate("text", x="2020", vjust= v_just, y = 117, label = dail_32[6,4], size = annot

annotate("text", x="2020", vjust= v_just, y = 138, label = dail_32[5,4], size = annot

annotate("text", x="2020", vjust= v_just, y = 146, label = dail_32[4,4], size = annot

annotate("text", x="2020", vjust= v_just-2.5, y = 150, label = dail_32[3,4], size = annot

annotate("text", x="2020", vjust= v_just-2.5, y = 156, label = dail_32[2,4], size = annot

annotate("text", x="2020", vjust= v_just-2.5, y = 161, label = dail_32[1,4], size = annot

coord_flip() +

theme(
  legend.position = "none",
  axis.title.y = element_blank(),
  axis.line.y = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  #axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1),
  axis.text.y = element_blank(),
  axis.text.x = element_blank(),
  #axis.title.x = element_blank(),
  plot.title = element_text(vjust = -8, hjust = 0.25, size = 11),
  plot.margin = margin(6, 6, 3, 3),
  #panel.background = element_blank(),
  #panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
  # panel.on top = TRUE

)

```

The next horizontal Stacked bar is modification of the earlier version. I've included a percentage scale on the lower axis – which gives similar information

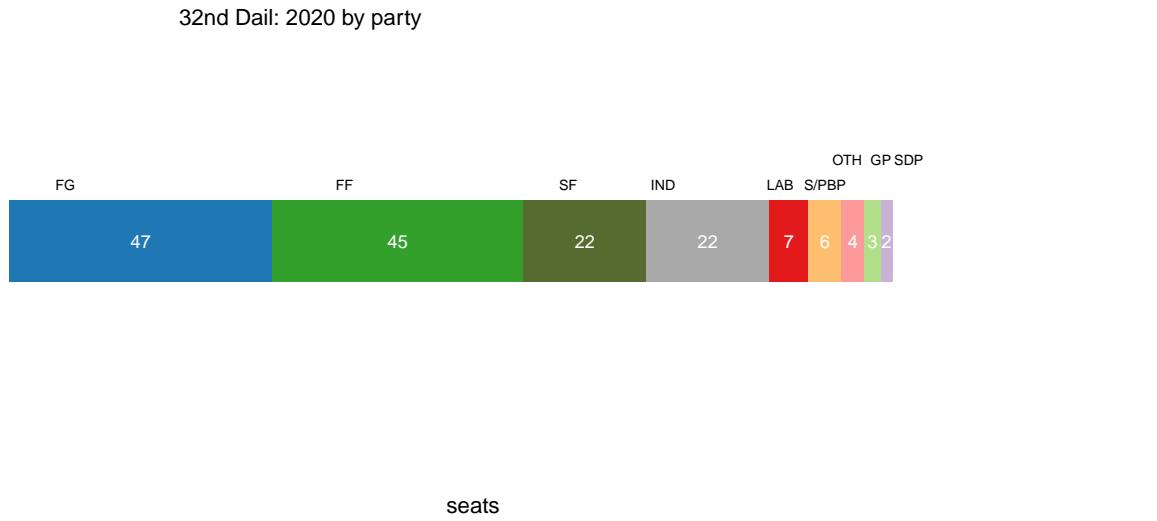


Figure 5.8: 32nd Dail, 2020. Party composition Horizontal Stacked Bar Chart

to the line in the Pareto plot.

However, while we can judge that the first 4 parties have 84 to 85% of the vote, We can't accurately say what percentage is occupied by Labour, which we could do in the Pareto plot.

The legend needs to be edited to remove the keys for the parties already labelled in the bar.

```
library(ggplot2)
library(forcats)
library(dplyr)
library(ggrepel)

dail_32<-dail_32 %>%
  mutate(year = as.factor("2020"))

#handy way to reverse order of rows
dail_32<-dail_32%>%mutate(short = fct_reorder(short, X2020))%>%arrange(X2020)

#dail_32<-dail_32%>%arrange(desc(short)) %>% mutate(pos = cumsum(X2020) - X2020/2)

theme_set(theme_classic())
```

```

annotate_size <- 2.5
v_just= -5

ggplot(dail_32, aes(x=year, y=X2020, fill=short, group=short)) +
  geom_col(width = 0.2) +
  geom_text(data=subset(dail_32, X2020>= 7), aes(x=year, y=X2020, label=short), colour="white")
#geom_text(data=subset(dail_32, short %in% c("LAB", "OTH", "GP", "SDP", "S/PBP")), aes(x=year, y=X2020, label=short), colour="white")

scale_y_continuous(position = "right", limits = c(0, total_seats),
                    expand = c(0.0, 0),
                    breaks = c(seq(0,140, by = 20), total_seats) ,
                    name = "Seats",sec.axis = sec_axis(~ ./ total_seats, labels = scales::percent))
#scale_y_continuous(position = "right", limits = c(0, total_seats),
#                    expand = c(0.0, 0),
#                    breaks = c(seq(0,140, by = 20), total_seats) ,
#                    name = "Percentage") + 

scale_x_discrete(expand = c(0,0 ))+
  scale_fill_manual(values = party.colours, name = "") +
  ggtitle("32nd Dail: 2020 by party ") +
  coord_flip() +
  theme(
    #legend.position="top",
    legend.key=element_blank(),
    #legend.background=element_rect(colour='black', fill='transparent'),
    legend.text=element_text(size=8),

    axis.title.y = element_blank(),
    axis.line.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.x = element_blank(),
    #axis.ticks.x = element_blank(),
    #axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1),
    axis.text.y = element_blank(),
    #axis.text.x = element_blank(),
    #axis.title.x = element_blank(),
    #plot.title = element_text(vjust = -8, hjust = 0.25, size = 11),
    )
  )

```

32nd Dail: 2020 by party

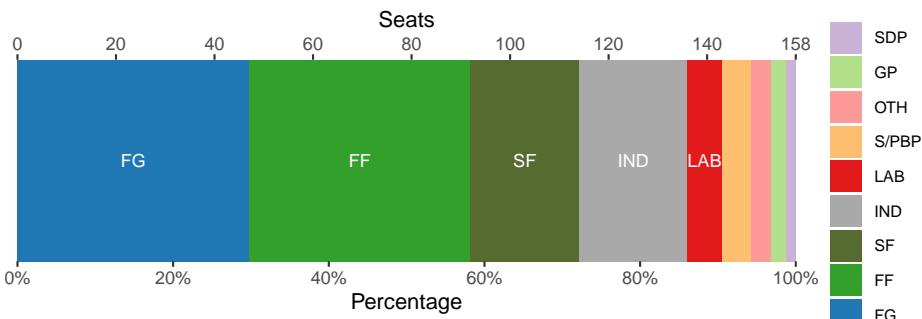


Figure 5.9: 32nd Dail, 2020. Party composition Horizontal Stacked Bar Chart

```

plot.margin = margin(6, 6, 3, 3),
#panel.background = element_blank(),
#panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
# panel.on top = TRUE

)

```

5.3 Waffle chart

A related chart to this approach is the so-called waffle chart – named because the squares appear like the patterns on waffles apparently.

In any case, visualisations like this are common in political analysis – as they convey the idea of actual seats occupied by each party. In the visualisation in the slide Fine Gael have 47 seats and therefore have 47 squares in the plot.

From a perceptual perspective, the reader is required to assess quantity by area occupied by the squares. Area is generally hard to assess quantitatively – but the presence of the little squares makes it a little easier.

Also, if you know the height of the plot – in this example, 6 squares – you can make a fairly good guess as to the quantities in each sub set of squares.

```

library(waffle)
library(magrittr)
library(hrbrthemes)
library(ggplot2)
library(dplyr)
library(waffle)

#reverse order of rows

```

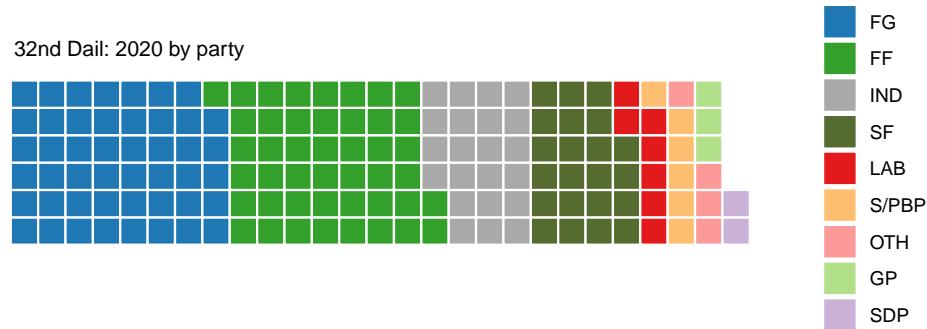


Figure 5.10: 32nd Dail, 2020. Party composition Waffle Chart

```
dail_32<-dail_32%>%mutate(short = fct_reorder(short, -X2020))%>%arrange(-X2020)

dail_32 %>%
  ggplot(aes(fill = short, values = X2020)) +
  geom_waffle(n_rows = 6, size = 0.5, colour = "white", flip = FALSE) +
  scale_fill_manual(
    name = NULL,
    values = party.colours
  ) +
  ggtitle("32nd Dail: 2020 by party ") +
  coord_equal() +
  theme_minimal()+
  theme_enhance_waffle() +
  theme(plot.title = element_text(size=10, hjust=0.07))
```

5.4 Don't forget Tables

Now while a table does not rely on the immediacy of our visual perceptual systems, when there is a small amount of data to present, I would suggest that it can sometimes do a better job than a purely visual solution.

```
library(knitr)
library(kableExtra)
library(dplyr)

values <- c(63,201,185)
categories<- c("A", "B", "C")
```

```
d<- data.frame(categories, values)
d$percent <- round((100*values)/sum(values),0)

d%>%arrange(-values) -> dd

kable(dd, digits = 2, format = "html", row.names = FALSE) %>%
  kable_styling(
    full_width = F,
    font_size = 12,
    position = "center")
```

categories

values

percent

B

201

45

C

185

41

A

63

14

Thanks to Claus Wilke for this summary:

	Pie chart	Stacked bars	Side-by-side bars
Clearly visualizes the data as proportions of a whole			
Allows easy visual comparison of the relative proportions			
Visually emphasizes simple fractions, such as 1/2, 1/3, 1/4			
Looks visually appealing even for very small datasets			

Pie chart	Stacked bars	Side-by-side bars
Works well when the whole is broken into many pieces		
	Works well for the visualization of many sets of proportions or time series of proportions	

Chapter 6

Comparing Proportions

Very often we want to compare the relative proportions of several quantities that may have been measured at different times, places or contexts.

We might want to visualise how proportions of two quantities vary according to the values of another variable.

As a running example, I will visualise a dataset downloaded from the Central statistics office website. It shows the number of people in Ireland in 2016 that have self-reported in the 2016 census having a Masters of science or PhD degree or higher. The data is segregated by gender and age group.

I want to produce a visualisation that shows the proportional difference by gender of people holding higher degrees within the various age groups.

6.1 Stacked Bars in Series

To start with we can use what we have learned in the section on stacked bars.

We can easily create a series of stacked bars representing the proportion of female to male degree holders per age group. You can achieve the simple Stacked Bar Graph simply by omitting the `dodge` attribute from `geom_bar`

The objective of a plot like this is not only to show the proportions per age-group but to be able to compare these proportions across the age groups.

```
library(ggplot2)

ire_education <- read.csv("../data/ire-2016-Masters, Doctorate (Phd) or higher.csv")

ggplot(ire_education, aes(x = age, y=number, fill=gender)) +
  geom_bar( stat="identity") +
```

```

scale_y_continuous(name = "", breaks = seq(0,100000, by=20000), labels = seq(0,100, 10))

#scale_fill_brewer(palette = "Set1", name = "Uploads") +
  scale_fill_manual(values = c( "#1b9e77", "#d95f02"),
                     name = NULL) +
  ggtitle("Masters, Doctorate (Phd) or higher (in thousands)") +
  theme_classic() +
  theme(
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.y= element_blank(),
    axis.title.x=element_blank(),
    axis.text.x = element_text( vjust = 5, size=9, face="bold"),
    axis.text.y = element_text( size=7),
    axis.title.y = element_text(size=9, face="bold"),
    legend.text = element_text(size=8),
    legend.title = element_blank(),
    legend.position = c(0.8,0.9),
    legend.key.size = unit(0.8,"line"),
    panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "grey"),
    panel.grid.minor.y = element_line(size = 0.1, linetype = 'solid', colour = "grey"),
    plot.title=element_text( hjust=0.00, face='bold', size=11)
  )

```

Looking at this graph, we can easily assess the trend of the values at the base of each stack, these are the orange bars representing the number of male degree holders.

We can see that the number of male degree holders reaches a peak in the 35 to 44 age-group and declines after that. We can also see that there are few degree holders in the 85 and all age group, but the bar is so small that we cannot judge any quantity from it.

When looking at the Green bars representing the numbers of female degree holders, it is much harder to make an accurate assessment. We can easily see that for the first four bars, there are a higher proportion of female degree holders than male degree holders. However, while we do see the downward trend after the 35 to 44 age group, it is much harder to be quantitative of about the trend.

For example, it is hard to determine by what amount the proportion of female

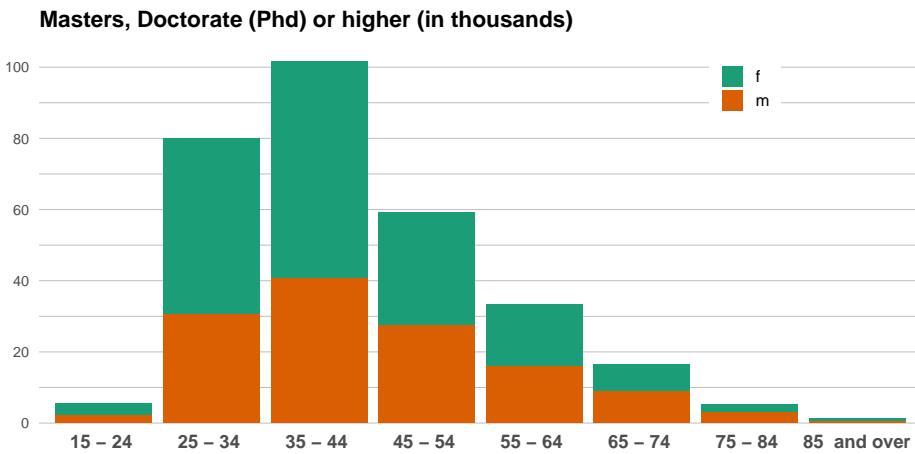


Figure 6.1: Ireland 2016, Holders of Masters, Doctorate (Phd) degree or higher (in thousands) by age group, gender

higher degree holders changes from the 25 to 34 age group to the 35 to 44 age group.

This is because the green bars do not start at common level unlike the orange bars, which start at the zero point on the y-axis. Also, as the age profile increases and the number of degree holders falls, it becomes harder to judge the proportion of male to female degree holders.

6.1.1 100% or Proportional Stacked Bar Graph

It may be useful to show the proportion out of 100% per age group. This is known as a proportional stacked bar graph or a 100% stacked bar graph.

To do this, you can use `geom_col` instead of `geom_bar` and set the `position` attribute to `fill`. With `position = fill`, the y values are scaled from 0 to 1.

To print the labels as percentages set `labels = scales::percent` in `scale_y_continuous`. You should reference the `scales` library first though

```
library(scales)

ggplot(ire_education, aes(x = age, y=number, fill=gender)) +
  geom_col(position="fill") +
  scale_y_continuous(name = "", labels = scales::percent) +
  #scale_fill_brewer(palette = "Set1", name = "Uploads") +
```

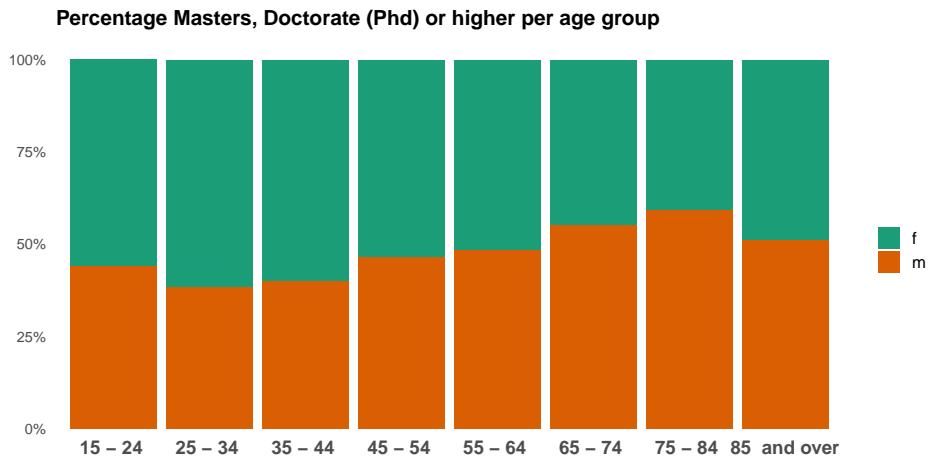


Figure 6.2: Ireland 2016, Holders of Masters, Doctorate (Phd) degree or higher gender proportion by age group

```

scale_fill_manual(values = c( "#1b9e77", "#d95f02"),
                  name = NULL) +
  ggtitle("Percentage Masters, Doctorate (Phd) or higher per age group") +
  theme_classic() +
  theme(
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.y= element_blank(),
    axis.title.x=element_blank(),
    axis.text.x = element_text( vjust = 5, size=9, face="bold"),
    axis.text.y = element_text( size=7),
    axis.title.y = element_text(size=9, face="bold"),
    legend.text = element_text(size=8),
    legend.title = element_blank(),
    #legend.position = c(0.8,0.9),
    legend.key.size = unit(0.8,"line"),
    plot.title=element_text( hjust=0.00, face='bold', size=10)
  )

```

6.2 The Stacked Area graph

The proportional stacked Area graph is conceptually like the proportional stacked bar graph'. With the stacked area graph the object is to show both trend and proportional difference over time time for a number of data series.

The entire graph represents the total of all the data plotted.

Stacked Area Graphs also use the areas to convey whole numbers, so they do not work for negative values. Overall, they are useful for comparing multiple variables changing over an interval.

The Stacked Area graph has its standard and proportional form - analagous to the forms of the stacked bar chart

This relies upon the `geom_area` geometric. It requires that the xaxis is coontinuous so we need to create a set of continuous points. I choose the mid points of the age intervals as the values for the continuous scale. We can then use the `geom_area` geometric in ggplot to draw and colour an area under the line to connect the points

The stacked area graph can be misleading. If some was to read the peak number for female graduates as indicated by the y axis – you might mistakenly conclude that there are over 100,000 female graduates. However, this not true, the ridge of green is stacked on top of orange ridge so you need to subtract the orange value from any value you read on the y-axis for green.

```
library(scales)

#lower part of age range
ire_education$age_continuous <- c(19, 29, 39, 49, 59, 69, 79, 89)

ggplot(ire_education, aes(x = age_continuous, y=number, fill=gender, colour = gender)) +
  geom_area(stat="identity", colour="black", size = .2) +
  scale_x_continuous(name = "age (years)", expand = c(-0.05, 0), breaks=c(19,29,39,49,59,69,79,89),
                     labels = c("19", "29", "39", "49", "59", "69", "79", "89")) +
  scale_y_continuous(name = "", breaks = seq(0,100000, by=20000), labels = seq(0,100, by=20), expand = c(0, 0)) +
  #scale_fill_brewer(palette = "Set1", name = "Uploads") +
  scale_fill_manual(values = c( "#1b9e77", "#d95f02")) +
  scale_colour_manual(values = c( "#1b9e77", "#d95f02")) +
  ggttitle("Masters, Doctorate (Phd) or higher (in thousands)") +
```

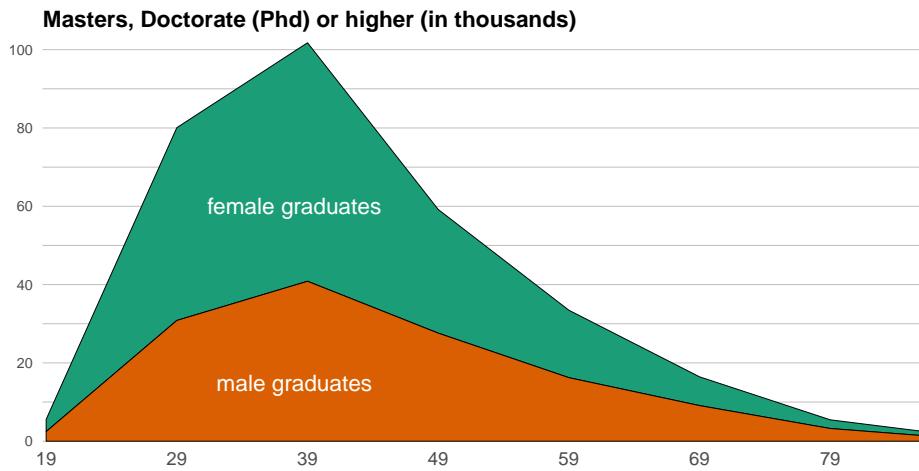


Figure 6.3: (#fig:stacked_area2) Ireland 2016, Holders of Masters, Doctorate (Phd) degree or higher by age and gender

```

annotate("text", x=38, y = 60000, label = "female graduates", size = 4, family = "Helvetica")
annotate("text", x=38, y = 15000, label = "male graduates", size = 4, family = "Helvetica")

theme_classic() +
  theme(
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.y = element_blank(),
    axis.title.x = element_blank(),
    axis.text.x = element_text(size=9),
    axis.text.y = element_text(size=7),
    axis.title.y = element_text(size=9),
    legend.text = element_text(size=8),
    legend.title = element_blank(),
    legend.position = "none",
    legend.key.size = unit(0.8,"line"),
    panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "grey"),
    panel.grid.minor.y = element_line(size = 0.1, linetype = 'solid', colour = "grey"),
    plot.title = element_text(hjust=0.00, face='bold', size=11)
  )

```

6.2.1 The Proportional Stacked Area graph

We can produce a proportional version of this also – which visually emphasises the difference in overall proportion and the trend.

We have to be very careful using the proportional however.

While the message seems very clear it is also misleading to some extent. Because the graphic gives no indication that there are relatively few Data points in the older age groups. Without context, readers may interpret that there are as many graduates in the *25 to 34* age group as there are in the *85 and over* age group. And we know that is not the case

```
library(scales)

ire_education$age_continuous <- c(19,29,39,49,59,69,79,89)

ggplot(ire_education, aes(x = age_continuous, y=number, fill=gender, colour = gender)) +
  geom_area(position="fill", colour="black", size = .2) +
  scale_x_continuous(name = "age (years)", expand = c(0, 0), breaks=c(19,29,39,49,59,69,79,89), lab=
  scale_y_continuous(expand = c(0, 0), labels = scales::percent, name = "") +
  annotate("text", x=55, y = 0.7, label = "female graduates", size = 4, family = "Helvetica", colo
  annotate("text", x=55, y = 0.3, label = "male graduates", size = 4, family = "Helvetica", colo

  scale_fill_manual(values = c( "#1b9e77", "#d95f02")) +
  scale_colour_manual(values = c( "#1b9e77", "#d95f02")) +
  ggttitle("Percentage Masters, Doctorate (Phd) or higher per age group") +
  theme_classic() +
  theme(
    axis.line.x = element_blank(),
    axis.ticks.x = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.y= element_blank(),
    axis.title.x=element_blank(),
    axis.text.x = element_text(size=9),
    axis.text.y = element_text(size=7),
    axis.title.y = element_text(size=9),
```

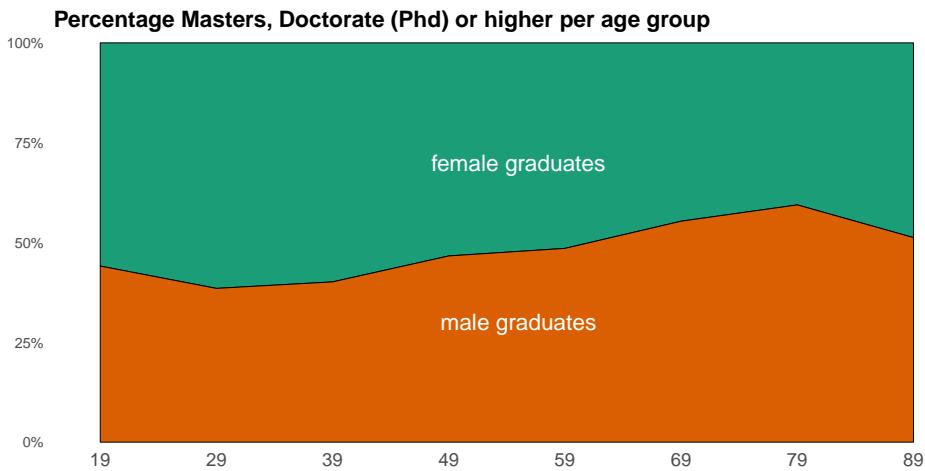


Figure 6.4: Ireland 2016, Holders of Masters, Doctorate (Phd) degree or higher gender proportion by age

```
legend.text = element_text(size=8),
legend.title = element_blank(),
legend.position = "none",
legend.key.size = unit(0.8,"line"),
plot.title=element_text( hjust=0.00, face='bold', size=11)

)
```

6.3 Grouped Bar Chart

We should not overlook the grouped bar chart approach that we saw earlier. I think that this approach works fairly well when we are comparing two categories. However the grouped bar approach still does not give a good sense of the overall proportion of male vs female graduates per time interval..

```
library(ggplot2)

ggplot(ire_education, aes(x = age, y=number, fill=gender)) +
  geom_bar(position="dodge", stat="identity") +
  scale_y_continuous(name = "", breaks = seq(0,60000, by=10000)) +
  #scale_fill_brewer(palette = "Set1", name = "Uploads") +
  scale_fill_manual(values = c( "#1b9e77", "#d95f02"),
```

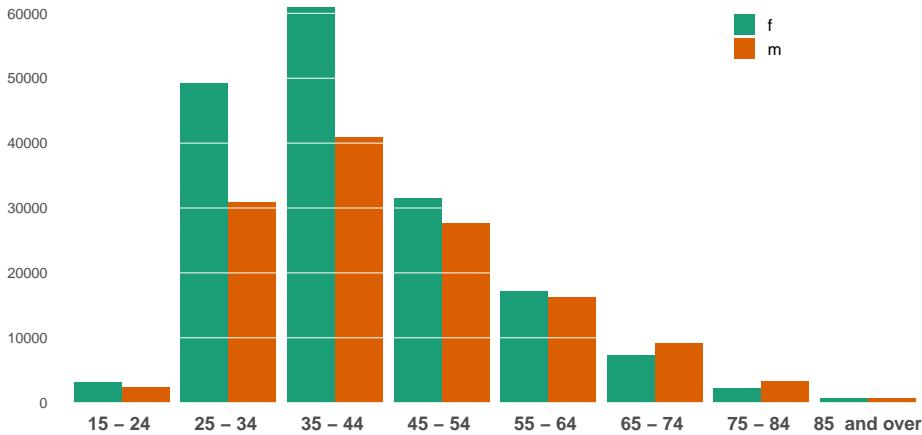


Figure 6.5: Ireland 2016, Holders of Masters, Doctorate (Phd)degree or higher, by gender and age group

```

name = NULL) +
# ggtitle("2016 Archive Upload Figures") +
theme_classic() +
theme(
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.y= element_blank(),
  axis.title.x=element_blank(),
  axis.text.x = element_text( vjust = 5, size=9, face="bold"),
  axis.text.y = element_text( size=7),
  axis.title.y = element_text(size=9, face="bold"),
  legend.text = element_text(size=8),
  legend.title = element_blank(),
  legend.position = c(0.8,0.9),
  legend.key.size = unit(0.8,"line"),
  plot.title=element_text( hjust=0.00, face='bold', size=11),
  panel.background = element_blank(),
  panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
  panel.on top = TRUE
)

```

While this shows the change in proportion of higher level graduates per age

group - it does not show the relative proportions per age group in the whole population.

A better way to do this is to show the proportion of the whole per age group represented by either gender.

6.4 Overlapping Area Plot

A side-by-side overlapping area plot where we show the proportion of each gender to the total may offer us a solution. Here we show the proportion of male graduates to the total number of graduates on the left and the proportion of female graduates to the total number of higher level graduates on the right.

This graphic allows us to see the relative proportion of each gender to the whole and allows us to compare the trends per gender.

The only difficulty here is that these may be interpreted as stacked area plots, which they are not. Each series visualised — male, female and total start at a common baseline, $y = 0$.

To create this plot, I had to add an extra value to the `status` field in the dataset representing the total male and female values. I could have done this using `dplyr` — but it was late! Will update it another time.

```
library(scales)
library(dplyr)

ire_education2 <- read.csv("../data/ire-2016-Masters, Doctorate (Phd) or higher2.csv")

ire_education2$age_continuous <- c(19, 29, 39, 49, 59, 69, 79, 89)

data2 = ire_education2 %>% filter(gender == "m&f") %>% select(number, age_continuous)

data1 = filter(ire_education2, gender != "m&f")

data1$gender <- factor(data1$gender, levels=c("f", "m"))
levels(data1$gender) <- list("Male Graduate" = "m", "Female Graduate" = "f")

ggplot(data1, aes(x = age_continuous, y=number)) +
  geom_area(stat="identity", data = data2, aes(fill = "all people surveyed"), colour = "#000000")
  geom_area(aes(fill="Selected Group", colour = gender), stat="identity", colour="transparent")
```

```

scale_x_continuous(name = "age (years)", expand = c(0, 0), breaks=c(19,29,39,49,59,69,79,89), lab
scale_y_continuous(expand = c(0, 0), name = "") +
#annotate("text", x=55, y = 0.7, label = "female graduates", size = 4, family = "Helvetica", co
# annotate("text", x=55, y = 0.3, label = "male graduates", size = 4, family = "Helvetica", co
facet_wrap(~gender, nrow = 1) +
scale_fill_manual(values = c("#b3b3b3a0", "#2b8cbed0")) +
#scale_colour_manual(values = c( "#1b9e77", "#d95f02")) +
# ggttitle("2016 Archive Upload Figures") +
theme_classic() +
theme(
  axis.line.x = element_blank(),
  axis.ticks.x = element_blank(),
  axis.ticks.y = element_blank(),
  axis.line.y= element_blank(),
  #axis.title.x=element_blank(),
  axis.text.x = element_text(size=9),
  axis.text.y = element_text(size=7),
  axis.title.y = element_text(size=9),
  legend.text = element_text(size=8),
  legend.title = element_blank(),
  legend.position = "none",
  legend.key.size = unit(0.8,"line"),
  plot.title=element_text( hjust=0.00, face='bold', size=11),
  strip.background = element_blank(),
#strip.text.x = element_blank(),
panel.background=element_blank(),
  panel.grid.major.y = element_line(size = 0.1, linetype = 'solid',colour = "grey"),
  panel.on top = TRUE
)

```

An alternative approach to the overlapping area graph shows the relative proportion of each gender to the total – but plots everything in the same 2D space.

This is created using the excellent `ggridges` package from Clause Wilke. We will see more of this package in later lectures.

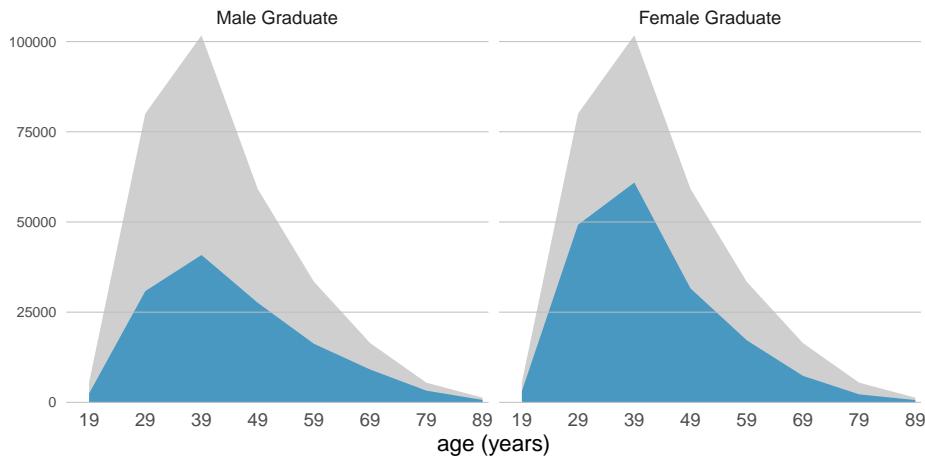


Figure 6.6: Ireland 2016, Holders of Masters, Doctorate (Phd) degree or higher gender proportion by age

This appears to solve a lot of our problems. The plots for female and male data both start from the zero point on the y axis. The proportions of female to male graduate numbers can be easily seen and quantified at different ages – and the proportion of each gender to the total number can also be seen.

However, this only works as long as the readers realises that this is *not* a stacked area plot – that in fact each of the filled areas start from the common baseline of $y = 0$, and that the areas in the front are transparent.

Otherwise the reader may assume that the ridge of blue on top of the green ridge in front – indicates a small proportion of female graduates - when the opposite is in fact true.

To try and emphasise that each ridge is unique and not just built upon the values of the one below it, I have used different line types

```
library(ggridges)

##
## Attaching package: 'gggridges'

## The following object is masked from 'package:ggplot2':
##
##     scale_discrete_manual

library(dplyr)

ire_education2<- ire_education2%>%mutate(gender=factor(ire_education2$gender, levels=c
```

```

ggplot(ire_education2, aes(x=age_continuous, y = 0, group = gender, alpha = gender, height = number))
  geom_ridgeline( size = 0.2) +
  scale_linetype_manual(values=c("solid", "twodash", "dotted"),name = "")+
  scale_alpha_manual(values = c(0.2,0.1, 0.2))+

  scale_x_continuous(name = "age (years)", expand = c(0, 0), breaks=c(19,29,39,49,59,69,79,89), labels = seq(19,89))

  scale_y_continuous(name = "", breaks = seq(0,100000, by=20000), labels = seq(0,100, by=20), expand = c(0,0))

  scale_fill_brewer(palette = "Dark2", type="qual")+

  annotate("text", x=30, y = 15000, hjust = 0, label = "male", size = 3, family = "Helvetica", color="grey")
  annotate("text", x=30, y = 45000, hjust = 0, label = "female", size = 3, family = "Helvetica", color="darkgreen")
  annotate("text", x=30, y = 70000, hjust = 0, label = "female and male", size = 3, family = "Helvetica", color="darkred")

  ggtitle("Number of Masters, Doctorate (Phd) or higher (x 1000): by gender, by age") +
  theme_minimal() + 

  theme(
    legend.position = "none",
    panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "grey"),
    panel.grid.minor.y = element_blank(),
    panel.grid.major.x = element_line(size = 0.1, linetype = 'solid', colour = "grey"),
    panel.grid.minor.x = element_blank(),
    plot.title = element_text(size =10)
  )

```

6.5 Proportional Stacked Bar again

The proportional stacked bar chart can be misleading because it does not indicate the amount of data within each bar.

However, in this example we can use it to show the proportion of male to female TDs (members of the Irish parliament, the Dail) from 1970 to 2020. This is because the number of TDs from 1970 2020 has been more or less consistent.

```

library(dplyr)
library(tidyr)
library(cowplot)

ire_women_tds <- read.csv("../data/ire_women_tds_dail._percent.csv")

```

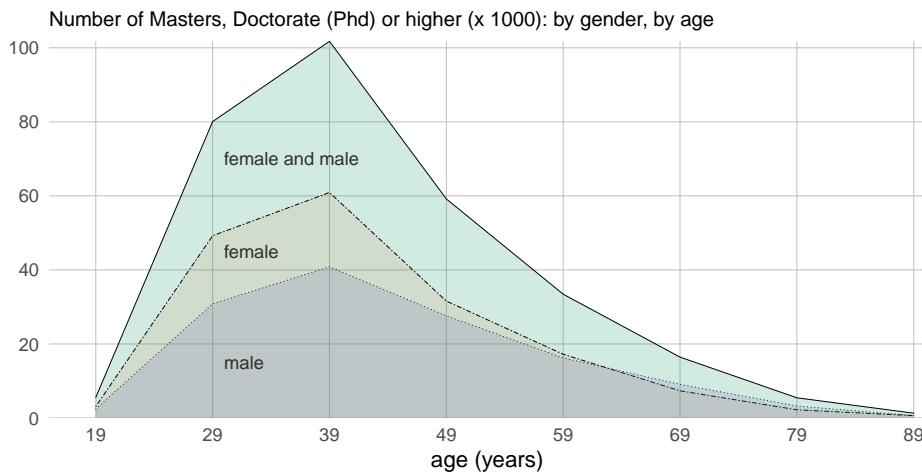


Figure 6.7: Ireland 2016, Holders of Masters, Doctorate (Phd) degree or higher gender proportion by age

```

ire_women_tds %>%
  filter( year >= 1970 ) %>%
  mutate(women = perc_women, men = 100 - perc_women) %>%
  select(-perc_women) %>%
  gather(gender, percent, women, men) %>%
  mutate(gender = factor(gender, levels = c("women", "men")))%>%filter(gender == "women")

ggplot(ire_women_tds_only, aes(x = year, y = percent, fill = gender)) +
  geom_col(width = 1, color = "#FFFFFF", size = .3, alpha = 0.66) +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(
    name = "percentage",
    breaks=(seq(0,25,by=5)),
    labels = scales::percent_format(accuracy = 1, scale = 1),
    expand = c(0, 0)
  ) +
  scale_fill_manual(values = c("#D55E00E0", "#0072B2E0"), guide = "none") +
  coord_cartesian(clip = "off") +
  ggtitle("Percentage of female TDs in the Dail: 1970 to present")+
  theme(
    panel.background=element_blank(),

```

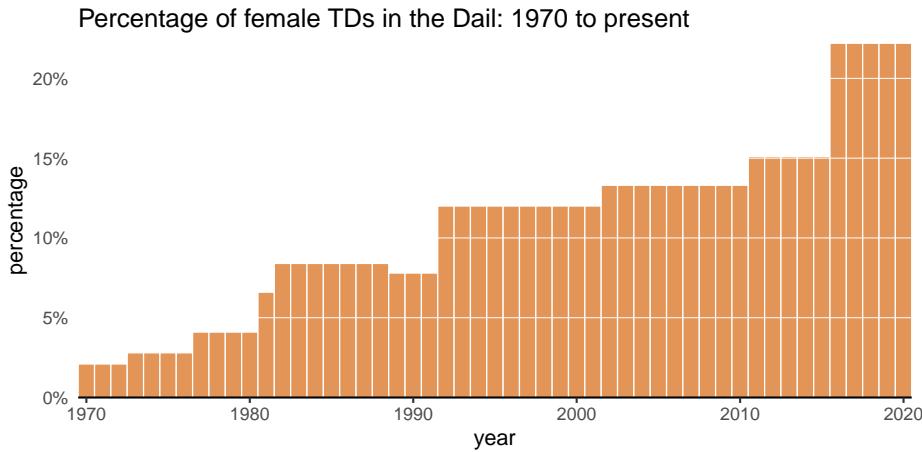


Figure 6.8: Ireland: percentage of female members of parliament 1970–2020

```

  panel.grid.major.y = element_line(size = 0.1, linetype = 'solid', colour = "white"),
  panel.on top = TRUE,
  panel.grid.minor.y = element_blank(),
  panel.grid.major.x = element_blank(),
  panel.grid.minor.x = element_blank(),

  axis.ticks.y = element_blank(),
  #axis.ticks.x = element_blank(),
  #axis.line.x = element_blank(),
  axis.line.y = element_blank(),
  #plot.margin = margin(14, 1.5, 3, 1.5)
)

```

Furthermore, when we show the data like this we present a more immediate sense of the relative lack of female representation in Irish politics .

```

library(dplyr)
library(tidyr)
library(cowplot)

ire_women_tds %>%
  filter( year >= 1970) %>%
  mutate(women = perc_women, men = 100 - perc_women) %>%
  select(-perc_women) %>%
  gather(gender, percent, women, men) %>%
  mutate(gender = factor(gender, levels = c("women", "men")))) -> ire_women_men_tds

```

```

# calculate label position
labels <- filter(ire_women_men_tds, year == max(year)) %>%
  mutate(pos = 100 - cumsum(percent) + 0.5*percent)

plot <- ggplot(ire_women_men_tds, aes(x = year, y = percent, fill = gender)) +
  geom_col(width = 1, color = "#FFFFFF", size = .3, alpha = 0.66) +
  # use position = position_stack(reverse = TRUE) to reverse intental stack ordering
  geom_hline(
    yintercept = c(50),
    color = "#000000FF", size = 0.3, linetype = 2
    #color = "#FFFFFFFA0"
  ) +
  geom_hline(yintercept = 100, color = "black") +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(
    name = "relative proportion",
    labels = scales::percent_format(accuracy = 1, scale = 1),
    expand = c(0, 0)
  ) +
  scale_fill_manual(values = c("#D55E00E0", "#0072B2E0"), guide = "none") +
  coord_cartesian(clip = "off") +
  ggtitle("Proportion of Female TDs : 1970 to present") +
  theme(
    axis.ticks.y = element_blank(),
    axis.ticks.x = element_blank(),
    axis.line.x = element_blank(),
    axis.line.y = element_blank(),
    plot.margin = margin(14, 1.5, 3, 1.5)
    plot.title = element_text(size = 11)
  )

# Generates a canvas onto which one can draw axis-like objects like annotations
yax <- axis_canvas(plot, axis = "y") +
  geom_text(data = labels, aes(y = pos, label = paste0(" ", gender)),
            x = 0, hjust = 0, size = 11/.pt)

ggdraw(insert_yaxis_grob(plot, yax, grid::unit(.15, "null")))

```

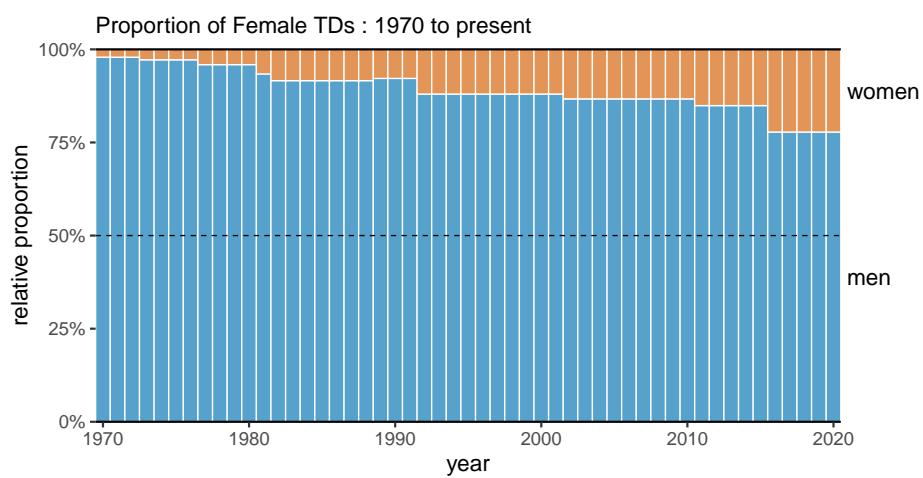


Figure 6.9: Ireland: proportion of female to male members of parliament 1970-2020

Chapter 7

Hierarchical Proportions

Some data is hierarchical and visualizing the overall proportions within each category cannot be achieved using the techniques we've seen up to now

The traditional approach would be to represent tree structures like a file system as a rooted directed graph with the root node at the top of the page and the child nodes below the parent with lines connecting them.

7.1 Tree maps

A Tree Map is a 2D representation of hierarchical information. It represents the quantities for each category using area size.

Each category is assigned a rectangle area with subcategory rectangles nested inside it. The area assigned to a category is in proportion to the quantity value of the category . The area assigned to a category is in proportion to the quantity value of the category, and is proportional to other quantities within the parent category. The area size of the parent category is the total of its subcategories.

Tree maps have a lot of information to show and can provide an intuitive overview of the categorical proportions in hierarchical data. There are some drawbacks.

- All Labels may not display
- While our brains are good at comparing the lengths of objects (e.g bars) or the positions of objects (e.g. dots) we compare sizes of rectangles with less precision

This is really a case where the visualisation gives us the overall gist of the story

Let's now examine TreeMap to see if you can derive quantitative information from it

I've provided a dataset of the counties of Ireland, north and south. Each county in the data set is represented by a county name, a Province, a county area, county population and population density.

```
library(ggplot2)
library(treemapify)
library(dplyr)

ire_counties <- read.csv("../data/ire_counties.csv")

ire_counties <- ire_counties %>% rename("Density" = "Density....km..")%>%rename("Area"= "Area....km..")

ire_counties$Province <- factor(ire_counties$Province, levels = c("Connacht", "Ulster"))

head(ire_counties)

##   Rank    County Area Density Province Population
## 1    1      Cork  7500    72.3 Munster     542868
## 2    2    Galway  6149    42.0 Connacht    258058
## 3    3      Mayo  5586    23.3 Connacht    130507
## 4    4  Donegal  4861    32.6 Ulster     159192
## 5    5     Kerry  4807    30.7 Munster     147707
## 6    6 Tipperary  4305    37.2 Munster     159553
```

Thanks to Claus Wilke who provided the solution on Stack Overflow on how to colour the tree map using several sequential scales.

```
https://stackoverflow.com/questions/50163072/different-colors-with-gradient-for-subgroups-on-a-treemap

#ire_counties$Province <- factor(ire_counties$Province, levels = c("Athena", "Hermes", "Zeus"))

# code to add colors to data frame follows
# first the additional packages needed

library(colorspace)
library(scales)

# number of palettes needed

n <- length(unique(ire_counties$Province))

# now calculate the colors for each data point
ire_counties_df2 <- ire_counties %>%
  mutate(index = as.numeric(factor(Province))- 1) %>%
  group_by(index) %>%
```

```

mutate(
  max_area = max(Area),
  colour = gradient_n_pal(
    sequential_hcl(
      6,
      h = 360 * index[1]/n,
      c = c(45, 20),
      l = c(30, 80),
      power = 0.6)
    )(1 - (Area/max_area))
)

ggplot(data = ire_counties_df2 , aes(area = Population , fill = colour, subgroup = Province)) +
  geom_treemap(colour = "white", size = 0.5*.pt, alpha = NA) +
  geom_treemap_text(aes(label = County), colour = "black" , size =10, place = "topleft",fontface =
  geom_treemap_subgroup_border(colour = "white", size =0.5) +
  geom_treemap_subgroup_text(grow = FALSE, colour = "#FAFAFA", size = 42, place ="bottomleft",
  scale_fill_identity()+
  coord_cartesian(clip = "off") +
  guides(colour = "none", fill = "none")

```

There are three aesthetics at play here.

1. Tile area, representing quantity *population*. The *population* for each county and province, is represented by portion of area it occupies in the overall 2D space.
2. A qualitative colour scale representing the parent level categories – e. olive green for Ulster, purple for Leinster, Crimson for Connact and Turquoise for Munster
3. sequential colour scales – one associated with the colour assigned to each province. Each colour scale represents quantity *county area*. In the colour gradient used, a dark shade indicates a high value and a light shade indicates a low value of *county area*.

7.1.1 Adding indications of quantity

The treemap gives you no indication of scale. In terms of mapping tile area to quantity, I feel that adding a legend (for example where a square of given area indicates a particular value) would NOT help.



Figure 7.1: Ireland counties and Provinces by population and geographical area

For that matter, adding 4 legends – one for each sequential scale used, would make the graphic cluttered looking, apart from the fact that matching colours is an imprecise task at the best of times.

There isn't necessarily a solution to this as visualising nested proportions can be tricky.

However, it is important to have some sense of scale, and most tree maps include an indication of quantity in at least some of the tiles

The values included in the following tree map are the population values ‘and the idea is that they are confirmatory indications of quantity rather than the primary indicators.

```
ggplot(data = ire_counties_df2 , aes(area = Population , fill = colour, subgroup = Province)) +  
  geom_treemap(colour = "white", size = 0.5*pt, alpha = NA) +  
  geom_treemap_text(aes(label = County), colour = "black" , size =10, place = "topleft") +  
  geom_treemap_text(aes(label = format(Population, nsmall=0, big.mark=",", trim=TRUE)), color = "black", size = 8, place = "topleft") +  
  #geom_treemap_text(aes(label = Area), color = "black", size = 8, place = "topleft") +  
  geom_treemap_subgroup_border(colour = "white", size =0.5) +
```

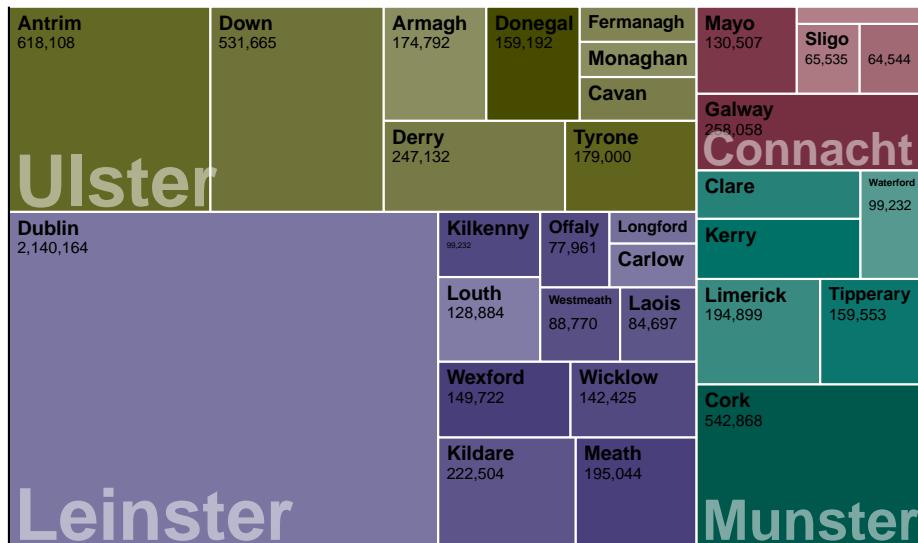


Figure 7.2: Ireland counties and Provinces by population and geographical area

```
geom_treemap_subgroup_text(grow = FALSE, colour = "#FAFAFA", size = 36, place = "bottomleft",
  scale_fill_identity() +
  coord_cartesian(clip = "off") +
  guides(colour = "none", fill = "none")
```

Here is an example of a tree map using widely used treemap conventions.

The principle difference from the last example is that the area of each province is outlined by thick grey borders and expanded text labels

It is also typical that the font size expands or contracts to fill the space available in sub-category tiles, in this case the tiles representing counties

While this graphic easier to achieve (it is essentially the default in the `treemapify` library), it has a number of issues

The province labels intrude upon the county areas and labels.

The varying size of the county labels conflict with the tile area size. For example, the label for *Dublin* has the same size as the label for *Westmeath*, one of the smallest tiles

However, with a single hue, it is arguably easier to compare colour gradient values across all tiles.



Figure 7.3: Ireland counties and Provinces by population and geographical area

```

library(RColorBrewer)

ggplot(data = ire_counties_df2 , aes(area = Population , fill = Area, subgroup = Province)
+   geom_treemap(colour = "white", size = 0.15*pt, alpha = 1) +
+   geom_treemap_text(aes(label = County), colour = "black" , grow = FALSE, place = "centre")
+
#geom_treemap_text(aes(label = Area), color = "black", size = 8, place = "topleft",
+ geom_treemap_subgroup_border(colour = "darkgrey", size =6) +
+ geom_treemap_subgroup_text(grow = TRUE, place = "centre", colour = "#FAFAFA", min.size = 10)
+
#scale_fill_distiller(palette ="Blues", direction = 1) +
+ scale_fill_gradient( low = "#56B1F7", high = "#132B43")+
+ coord_cartesian(clip = "off") +
+ guides(colour = "none", fill = "none")

```

Chapter 8

Visualising Time Series Data

In this chapter we will look at visualizing data where a value or a set of values are ordered by a time or interval variable. Data in this form is quite common, so we are going to look at the standard ways of representing it, plus a few extensions. We'll also mention how to handle missing data, which is a fairly common aspect of time series analysis.

8.1 Data

As a first example of a time series visualisation, we will use data from the Elsevier's Scopus repository.

The Scopus keeps track of how many citations an academic research article receives from other articles. An article that receives a lot of citations is considered influential. And journals, which contain collections of articles, are considered influential if their articles are highly cited.

Our data contains the citescapes from 2011 to 2018 for the top ranked journals in computer science in 2018.

The data in the CSV file has column names with year values. R doesn't accept column names starting with a number, so on importing this data, it has prefixed the column name with an X.

Data in this wide format is quite common for time series data

Looking at the titles of the top journals in computer science in 2018, We can see that the topics of machine learning, data analytics, pattern recognition and computer vision are predominant.

In the visualisation exercise that follows, I want to provide a visual analysis of how consistent cite scores have been for these topics from 2011 to 2018.

```
library(dplyr)
library(tidyr)
library(knitr)
library(kableExtra)
scopus_citescore_wide <- read.csv("../data/scopus_comp_sci_top_citescore_2018.csv")

kable(scopus_citescore_wide, digits = 2, format = "html", row.names = TRUE) %>%
  kable_styling( bootstrap_options = c("striped"), full_width = T,
                 font_size = 12) %>%
  column_spec(column = 2, width = "18em")
```

Title	
ShortTitle	
X2018	
X2017	
X2016	
X2015	
X2014	
X2013	
X2012	
X2011	
1	
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition	
CVPR	
37.26	
18.18	
12.72	
6.19	
5.05	
5.25	
3.23	

2.09

2

Journal of Statistical Software

JSS

25.02

16.32

7.71

5.02

6.37

7.12

4.93

4.59

3

IEEE Transactions on Pattern Analysis and Machine Intelligence

PAMI

19.67

12.75

13.29

12.66

11.05

11.80

10.09

8.89

4

Foundations and Trends in Machine Learning

FTML

19.00

10.90

10.00

17.30

16.00

31.80

15.82

8.50

5

Proceedings of the IEEE International Conference on Computer Vision

ICCV

16.64

11.67

9.92

3.90

2.49

3.16

1.64

2.54

- Then convert the data set into the long format preferred by ggplot
- Convert the year values (e.g. *X2014*) into date objects in R
- Although the data just gives the date as a year, an R date object needs day and month values. I make each year value equal to the 1st of January in that year. While that seems like an arbitrary decision - it will effect where ggplot places the ticks in the subsequent plots I make, if I choose 1st of June in each year, the tick on the axis representing each year will lie midway the year and the following year.

```
scopus_citescore_long<-scopus_citescore_wide %>%
  gather(key = year, value=citescore, 3:10) %>%
  mutate(year = case_when(
    year == "X2011" ~ "2011",
    year == "X2012" ~ "2012",
    year == "X2013" ~ "2013",
    year == "X2014" ~ "2014",
    year == "X2015" ~ "2015",
    year == "X2016" ~ "2016",
    year == "X2017" ~ "2017",
    year == "X2018" ~ "2018",
  )) %>%
  mutate(year=paste0(year, "-01-01")) %>%
  mutate(year= as.Date(year, format="%Y-%m-%d"))
```

```
# show as a table

kable(scopus_citescore_long, digits = 2, format = "html", row.names = TRUE) %>%
  kable_styling( bootstrap_options = c("striped"), full_width = T,
    font_size = 12) %>%
  column_spec(column = 2, width = "30em")%>%
  scroll_box(height = "300px")
```

Title

ShortTitle

year

citescore

1

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2018-01-01

37.26

2

Journal of Statistical Software

JSS

2018-01-01

25.02

3

IEEE Transactions on Pattern Analysis and Machine Intelligence

PAMI

2018-01-01

19.67

4

Foundations and Trends in Machine Learning

FTML

2018-01-01

19.00

5

Proceedings of the IEEE International Conference on Computer Vision
ICCV

2018-01-01

16.64

6

Proceedings of the IEEE Computer Society Conference on Computer Vision and
Pattern Recognition

CVPR

2017-01-01

18.18

7

Journal of Statistical Software

JSS

2017-01-01

16.32

8

IEEE Transactions on Pattern Analysis and Machine Intelligence

PAMI

2017-01-01

12.75

9

Foundations and Trends in Machine Learning

FTML

2017-01-01

10.90

10

Proceedings of the IEEE International Conference on Computer Vision
ICCV

2017-01-01

11.67

11

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2016-01-01

12.72

12

Journal of Statistical Software

JSS

2016-01-01

7.71

13

IEEE Transactions on Pattern Analysis and Machine Intelligence

PAMI

2016-01-01

13.29

14

Foundations and Trends in Machine Learning

FTML

2016-01-01

10.00

15

Proceedings of the IEEE International Conference on Computer Vision

ICCV

2016-01-01

9.92

16

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2015-01-01

6.19

17

Journal of Statistical Software

JSS

2015-01-01

5.02

18

IEEE Transactions on Pattern Analysis and Machine Intelligence

PAMI

2015-01-01

12.66

19

Foundations and Trends in Machine Learning

FTML

2015-01-01

17.30

20

Proceedings of the IEEE International Conference on Computer Vision

ICCV

2015-01-01

3.90

21

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2014-01-01

5.05

22

Journal of Statistical Software

JSS

2014-01-01

- 6.37
23
IEEE Transactions on Pattern Analysis and Machine Intelligence
PAMI
2014-01-01
11.05
24
Foundations and Trends in Machine Learning
FTML
2014-01-01
16.00
25
Proceedings of the IEEE International Conference on Computer Vision
ICCV
2014-01-01
2.49
26
Proceedings of the IEEE Computer Society Conference on Computer Vision and
Pattern Recognition
CVPR
2013-01-01
5.25
27
Journal of Statistical Software
JSS
2013-01-01
7.12
28
IEEE Transactions on Pattern Analysis and Machine Intelligence
PAMI
2013-01-01

11.80

29

Foundations and Trends in Machine Learning

FTML

2013-01-01

31.80

30

Proceedings of the IEEE International Conference on Computer Vision

ICCV

2013-01-01

3.16

31

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2012-01-01

3.23

32

Journal of Statistical Software

JSS

2012-01-01

4.93

33

IEEE Transactions on Pattern Analysis and Machine Intelligence

PAMI

2012-01-01

10.09

34

Foundations and Trends in Machine Learning

FTML

2012-01-01

15.82

35

Proceedings of the IEEE International Conference on Computer Vision

ICCV

2012-01-01

1.64

36

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2011-01-01

2.09

37

Journal of Statistical Software

JSS

2011-01-01

4.59

38

IEEE Transactions on Pattern Analysis and Machine Intelligence

PAMI

2011-01-01

8.89

39

Foundations and Trends in Machine Learning

FTML

2011-01-01

8.50

40

Proceedings of the IEEE International Conference on Computer Vision

ICCV

2011-01-01

2.54

8.2 Line Graphs

To start with let's look at the top ranked journal : Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition

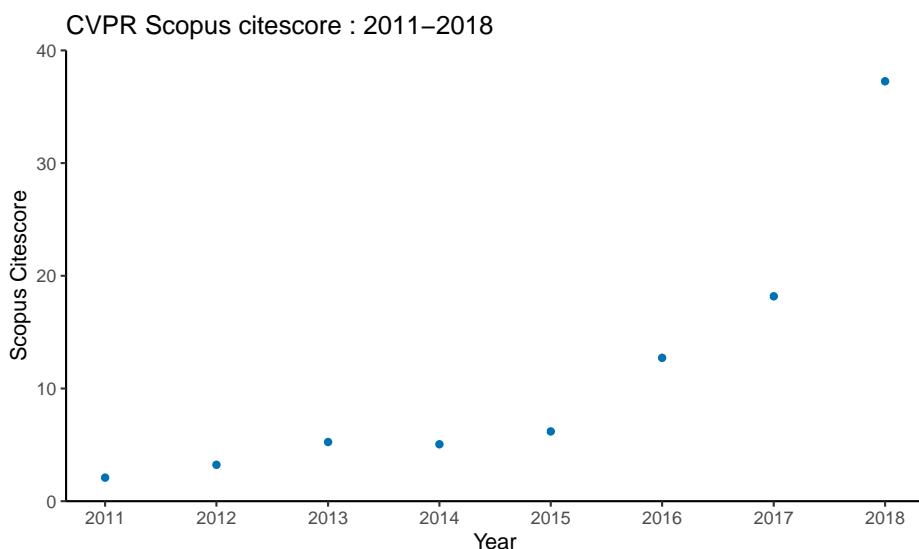
Which we'll refer to be its abbreviation: CVPR

We can visualize this growth by making a form of scatter plot where we draw dots representing the citescore for each year.

Unlike the scatter-plots we saw in lecture 2 the dots here are spaced evenly along the x axis.

```
library(scales)
scopus_citescore_long %>% filter(ShortTitle == "CVPR") ->CVPR_growth

ggplot(CVPR_growth, aes(year, citescore)) +
  geom_point(color = "white", fill = "#0072B2", shape = 21, size = 2) +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus Citescore") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  ggtitle("CVPR Scopus citescore : 2011-2018") +
  theme_classic() +
  theme(plot.margin = margin(7, 7, 3, 1.5))
```



We can visually emphasize the order and the trend by connecting points

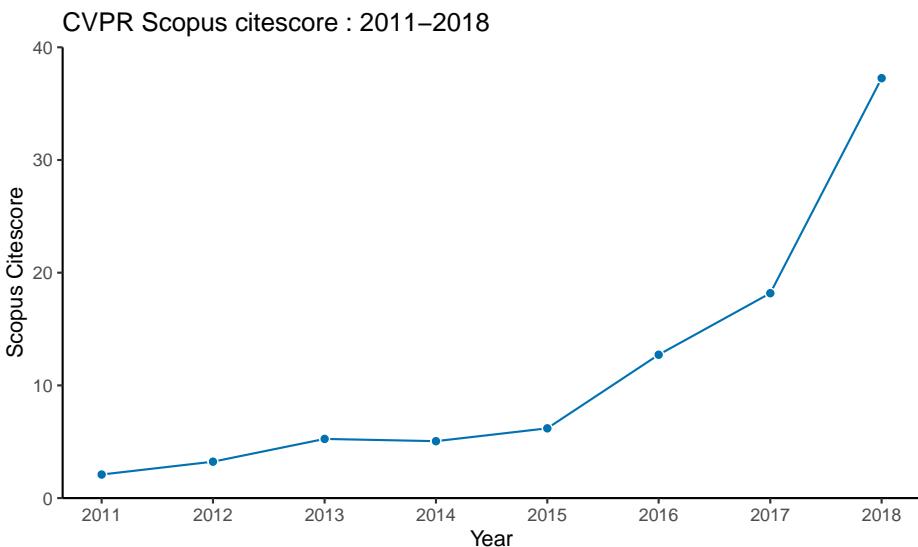
This is called a Line Plot

The lines do not represent data – visual aid only

We can see a very clear upward trend in citation score for this journal with a strong increase from 2017 to 2018

```
library(scales)

ggplot(CVPR_growth, aes(year, citescore)) +
  geom_line(color = "#0072B2") +
  geom_point(color = "white", fill = "#0072B2", shape = 21, size = 2) +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus Citescore") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  ggtitle("CVPR Scopus citescore : 2011–2018") +
  theme_classic() +
  theme(plot.margin = margin(7, 7, 3, 1.5))
```



8.3 Missing Data

Missing data is often an issue in time series data. If you have missing dates in your data, perhaps there were no observations on that day, you should enter the missing observation values as NA

NA can mean “

- “Not Available” e.g. the sensor was down at the time of the measure,

- “Not Applicable”: e.g. when asking someone who is renting when they bought their house
- “No Answer”

A common mistake is to plot data like this using a line graph.

- A line graph suggests a smooth transition from value to value. In reality, data is missing.
- In these situations, *do not* use a line to connect the points.

Our data set doesn’t have missing values, so I’ve artificially removed a number of years for example purposes. Were I faced with this data set, I would enter the missing values for the missing years as a NA value

```
#create artificial gaps in the data
scopus_citescore_long %>% filter(ShortTitle == "CVPR") %>%
  filter(year < as.Date("2013-01-02") | year >= as.Date("2016-01-01")) -> CVPR_growth_gap1

kable(CVPR_growth_gap1, digits = 2, format = "html", row.names = TRUE) %>%
  kable_styling(bootstrap_options = c("striped"), full_width = T,
                font_size = 12) %>%
  column_spec(column = 2, width = "30em")
```

Title	
ShortTitle	
year	
citescore	
1	
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition	
CVPR	
2018-01-01	
37.26	
2	
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition	
CVPR	
2017-01-01	
18.18	
3	

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2016-01-01

12.72

4

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2013-01-01

5.25

5

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2012-01-01

3.23

6

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2011-01-01

2.09

```
#create artifical gaps in the data
scopus_citescore_long %>%filter(ShortTitle == "CVPR") %>%
  mutate(citescore = case_when(
    #year == as.Date("2013-01-01") ~ as.numeric(NA),
    year == as.Date("2014-01-01") ~ as.numeric(NA),
    year == as.Date("2015-01-01") ~ as.numeric(NA),
    TRUE ~ citescore ))-> CVPR_growth_gap2

kable( CVPR_growth_gap2, digits = 2, format = "html", row.names = TRUE) %>%
  kable_styling( bootstrap_options = c("striped"), full_width = T,
                font_size = 12) %>%
  column_spec(column = 2, width = "30em")
```

Title	
ShortTitle	
year	
citescore	
1	
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition	
CVPR	
2018-01-01	
37.26	
2	
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition	
CVPR	
2017-01-01	
18.18	
3	
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition	
CVPR	
2016-01-01	
12.72	
4	
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition	
CVPR	
2015-01-01	
NA	
5	
Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition	
CVPR	

2014-01-01

NA

6

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2013-01-01

5.25

7

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2012-01-01

3.23

8

Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition

CVPR

2011-01-01

2.09

Do not use lines to join the dots when there are gaps in the time interval.

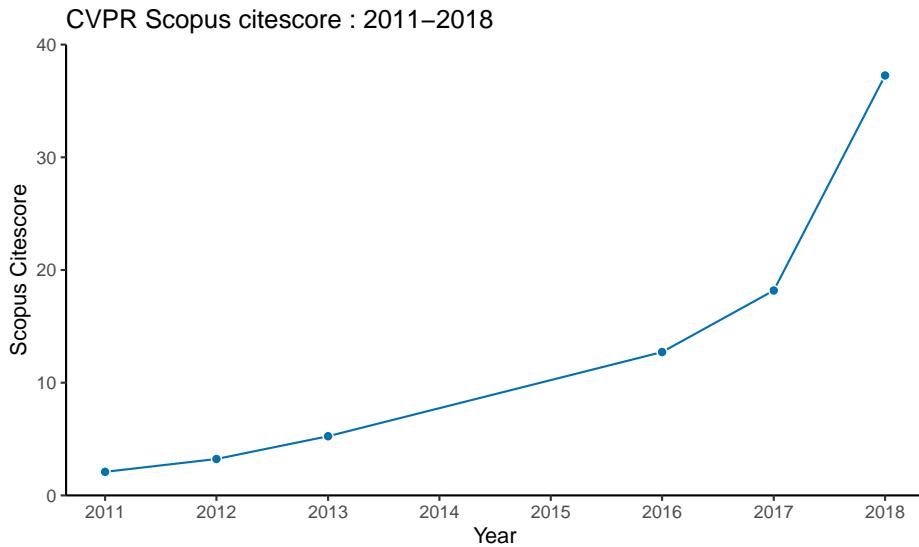
The plot below illustrated how *NOT* to do it.

```
library(scales)

scopus_citescore_long %>% filter(ShortTitle == "CVPR") ->CVPR_growth

ggplot(CVPR_growth_gap1, aes(year, citescore)) +
  geom_line(color = "#0072B2") +
  geom_point(color = "white", fill = "#0072B2", shape = 21, size = 2) +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus Citescore") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  ggtitle("CVPR Scopus citescore : 2011-2018") +
```

```
theme_classic() +
  theme(plot.margin = margin(7, 7, 3, 1.5))
```



You should make sure your missing values are represented as NAs in the data. When you plot you still get lines where there are contiguous values - but broken lines where the values are missing. This is preferable.

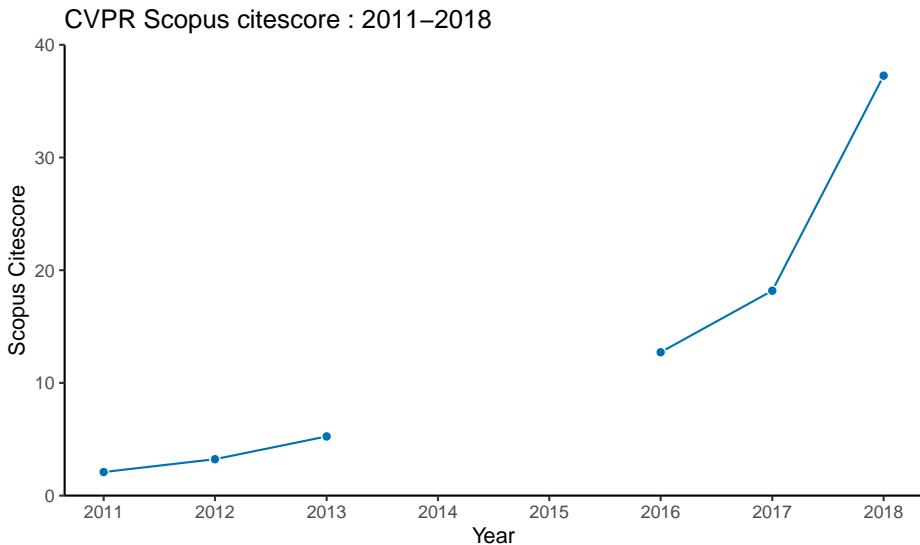
This is a true representation of the data – and the reader can see where is no data

```
library(scales)

scopus_citescore_long %>% filter(ShortTitle == "CVPR") -> CVPR_growth

ggplot(CVPR_growth_gap2, aes(year, citescore)) +
  geom_line(color = "#0072B2") +
  geom_point(color = "white", fill = "#0072B2", shape = 21, size = 2) +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus Citescore") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  ggtitle("CVPR Scopus citescore : 2011-2018") +
  theme_classic() +
  theme(plot.margin = margin(7, 7, 3, 1.5))

## Warning: Removed 2 rows containing missing values (geom_point).
```



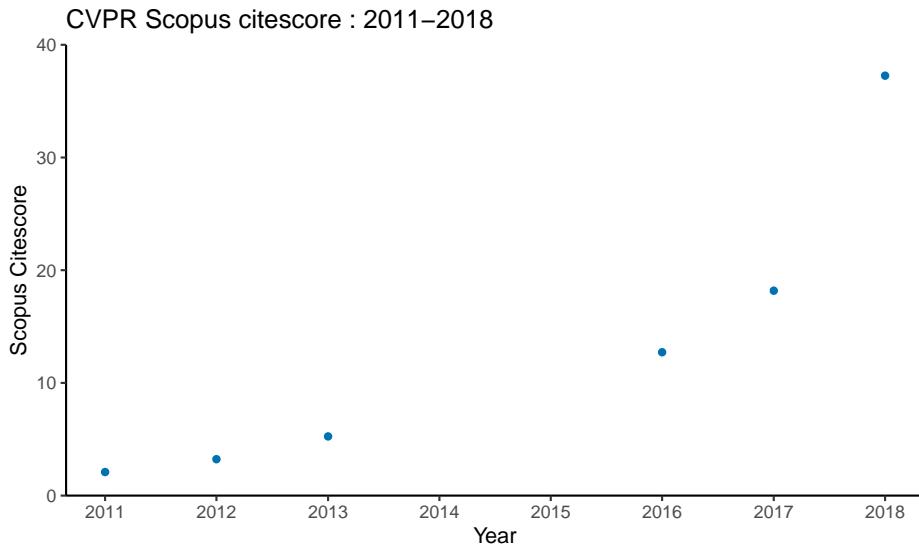
Alternatively, omit the line entirely if there are missing values.

There is a whole topic in statistics on the imputation of missing values

This is not within the scope of this module

```
ggplot(CVPR_growth_gap2, aes(year, citescore)) +
  geom_point(color = "white", fill = "#0072B2", shape = 21, size = 2) +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus Citescore") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  ggtitle("CVPR Scopus citescore : 2011–2018") +
  theme_classic() +
  theme(plot.margin = margin(7, 7, 3, 1.5))

## Warning: Removed 2 rows containing missing values (geom_point).
```



8.4 Line graphs emphasise trend

I'm visualising the data in its original state again – so no missing values.

Often the data points are omitted.

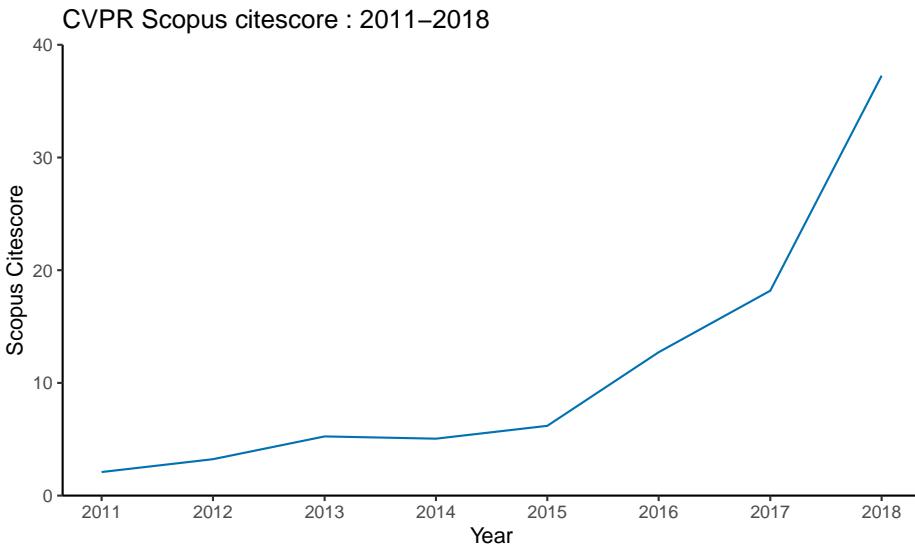
While this loses the detail of the actual position of the data points, *it emphasises the trend in the data* .

Such a line graph is a typical way of representing time series data.

Where there are many observations in the data, it is more important to show the trend rather than individual data points

```
scopus_citescore_long %>% filter(ShortTitle == "CVPR") ->CVPR_growth

ggplot(CVPR_growth, aes(year, citescore)) +
  geom_line(color = "#0072B2") +
  #geom_point(color = "white", fill = "#0072B2", shape = 21, size = 2) +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus Citescore") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  ggttitle("CVPR Scopus citescore : 2011-2018") +
  theme_classic() +
  theme(plot.margin = margin(7, 7, 3, 1.5))
```



8.5 Line graph with a fill

If you have only a single data series to plot, a fill can help to emphasise the shape and the separation of the space above and below the line. It also gives a visual quantity of area below the line - which indicates the summation of all values represented by the line.

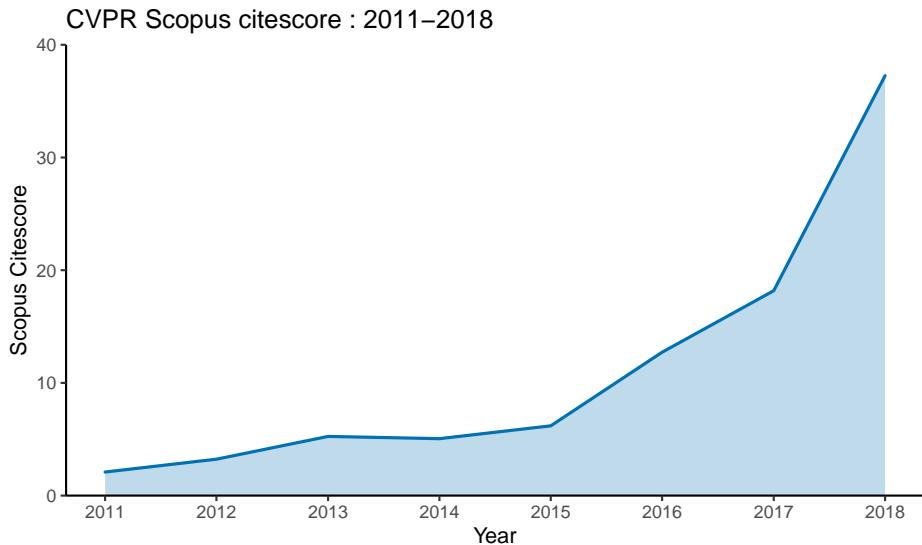
However, as with bar plots, this is valid if the y axis starts at 0, so that the height of the line is a correct visual representation of the data

This plot uses the `geom_ridgeline` function we saw last week.

This is handy of way drawing a line with a fill below it.

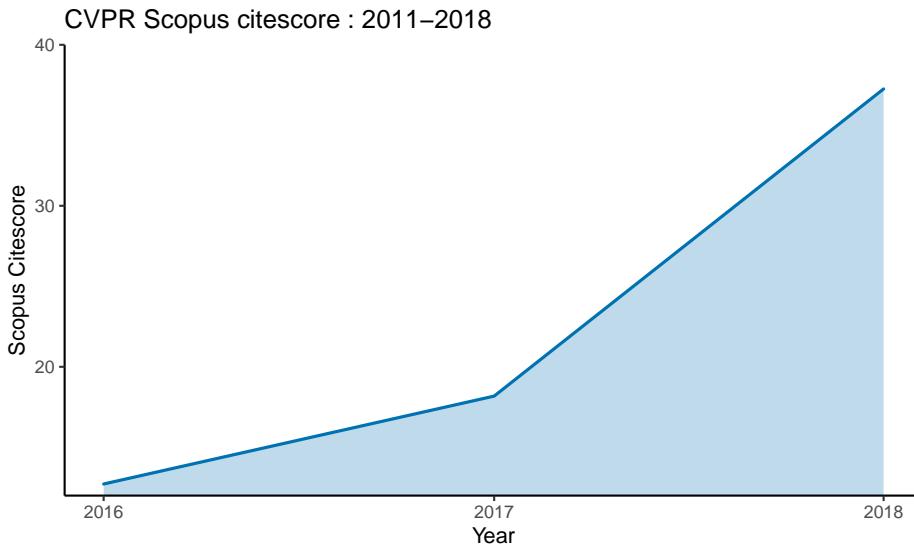
```
scopus_citescore_long %>% filter(ShortTitle == "CVPR") ->CVPR_growth

ggplot(CVPR_growth, aes(year, citescore)) +
  geom_ridgeline(aes(height = citescore, y=0), color = "#0072B2", fill = "#0072B240", size = .75)
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus Citescore") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  ggtitle("CVPR Scopus citescore : 2011-2018")+
  theme_classic() +
  theme(plot.margin = margin(7, 7, 3, 1.5))
```



However, the plot below is completely misleading as it does not measure height from 0 on the y-axis

```
ggplot(CVPR_growth, aes(year, citescore)) +
  geom_ridgeline(aes(height = citescore, y=0), color = "#0072B2", fill = "#0072B240",
                 alpha = 0.5) +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus Citescore") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y"), limits = as.Date(c("2011-01-01", "2018-12-31")))
  ggtitle("CVPR Scopus citescore : 2011-2018") +
  coord_cartesian(ylim=c(12, 40)) +
  theme_classic() +
  theme(plot.margin = margin(7, 7, 3, 1.5))
```



8.6 Visualising Multiple Time Series

Often, we want to compare several time series.

As you saw in the first assignment, using a dot plot to present multiple categories of data is challenging. The points impinge on one another and your eye finds it hard to separate one series from another.

Here, I've plotted the citescores for each of the journal we are interested in – using colour and shape to initially separate them

With a time series plot, the individual time series will impinge on one another and your eye will find it hard to separate one series from another.

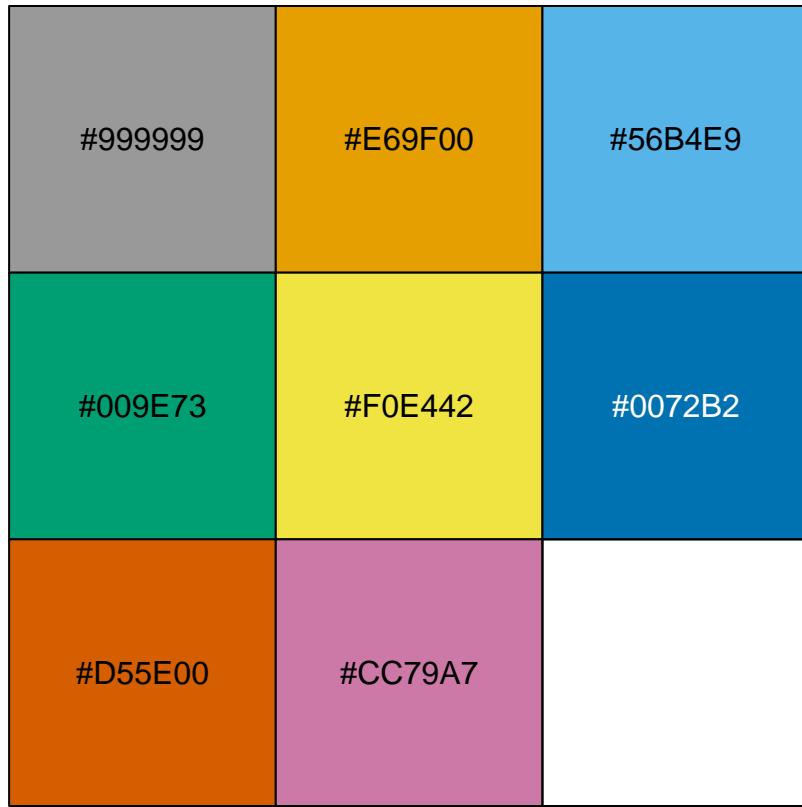
We could spend time teasing the points apart with different visual properties, as you did in your first assignment, but it will be easier to add a line to each series, which will help your eye trace each.

I sometimes throw up a palette to view the colours before deciding which ones to use.

```
library(colorspace)
library(scales)

cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#FOE442", "#0072B2", "#D55E00", "#CC7700")

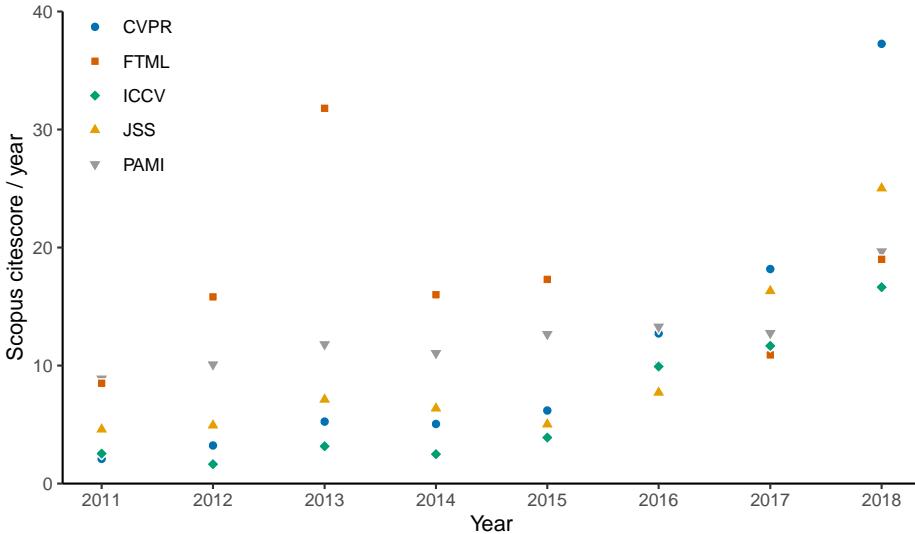
show_col(cbPalette)
```



```
p <- ggplot(scopus_citescore_long, aes(year, citescore, color = ShortTitle, fill = ShortTitle))
  geom_point(color = "white", size = 2) +
  scale_shape_manual(values = c(21, 22, 23, 24, 25),
                     name = NULL) +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus citescore / year") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),
                     name = NULL) +
  scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),
                    name = NULL) +
  theme_classic() +
  theme(legend.title.align = 0.5,
        legend.position = c(0.005, 1.05),
```

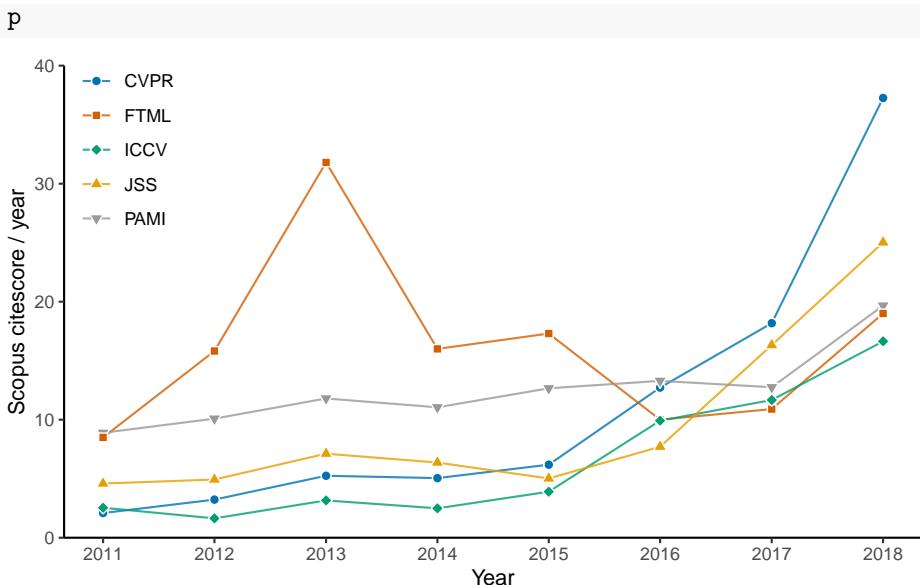
```
legend.just = c(0, 1),
plot.margin = margin(14, 7, 3, 1.5))
```

p



We could spend time teasing these apart with different visual properties, but it will be easier to add a line to each series, which will help your eye trace each.

```
p <- ggplot(scopus_citescore_long, aes(year, citescore, color = ShortTitle, fill = ShortTitle, shape = ShortTitle))
geom_point(color = "white", size = 2) +
  scale_shape_manual(values = c(21, 22, 23, 24, 25),
                     name = NULL) +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus citescore / year") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),
                     name = NULL) +
  scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),
                     name = NULL) +
  theme_classic() +
  theme(legend.title.align = 0.5,
        legend.position = c(0.005, 1.05),
        legend.just = c(0, 1),
        plot.margin = margin(14, 7, 3, 1.5))
```



This is better but is the presence of the points and legends make the plot cluttered.

In the plot below, I've removed the points and the trends in the time series are much easier to follow

I've also removed the legend and instead labelled each series by inserting an invisible second y axis on the right.

The labels are specified as the labels on selected break points on this axis.

However, it doesn't quite work as the final values for PAMI and FTML are too close together. While the associated labels are just readable, the overlap looks visually unattractive.

```
# gives the final count value that you want to show
scopus_citescore_final <- filter(scopus_citescore_long, year == ymd("2018-01-01"))

p <- ggplot(scopus_citescore_long, aes(year, citescore, color = ShortTitle)) +
  geom_line(size = 1, alpha = 0.7) +
  #geom_point(color = "white", size = 2) +
  scale_shape_manual(values = c(21, 22, 23, 24, 25),
                     name = NULL) +
  # adds a second axis which is a duplicate (dup_axis) of the primary axis
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus citescore / year",
```

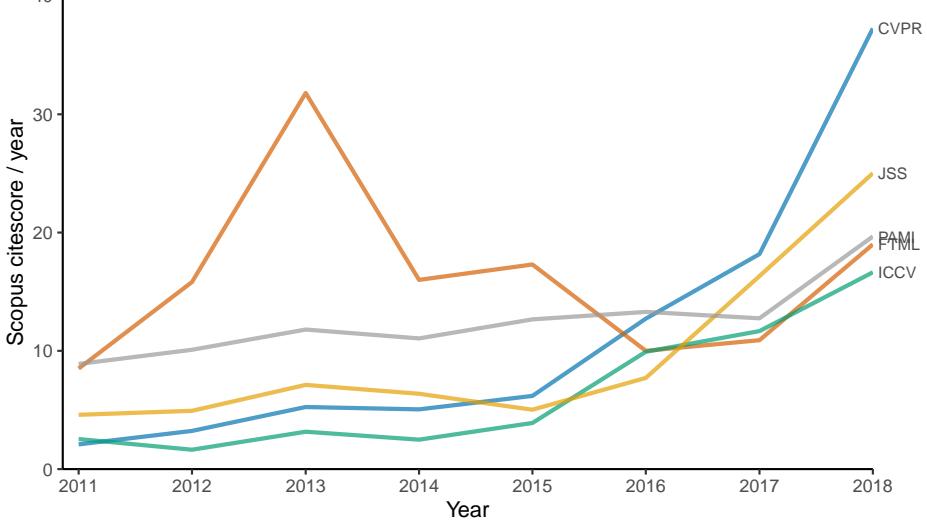
```

sec.axis = dup_axis(
  breaks = scopus_citescore_final$citescore,
  labels = scopus_citescore_final$ShortTitle,
  name = NULL)) +
scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y"), expand = expand_scale)

scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),
  name = NULL) +
theme_classic() +

```

p



8.7 Using a hidden second y-axis to label series

The labels for PAMI and FTML overlap. While they are readable the look unattractive. The positions of the axis labels are decided by the value of the associated variable, which is *citescore*. In order to separate these two labels we can create another field called *pos* which indicates the positions on the y -axis for the labels. The only changes I will make will effect the labels PAMI and FTML labels.

```
# creating the pos field
scopus_citescore_final<- scopus_citescore_final%>%
  mutate(pos=citescore)%>%mutate(pos = case_when(
    ShortTitle=="PAMI" ~ pos+0.4,
    ShortTitle=="FTML" ~ pos-0.4,
    TRUE ~ pos)
  )
kable(scopus_citescore_final, digits = 2, format = "html", row.names = TRUE) %>%
  kable_styling( bootstrap_options = c("striped"), full_width = T,
    font_size = 12) %>%
  column_spec(column = 2, width = "30em")%>%
  column_spec(column = 6, color = "red", background = "#d3d3d3")
```

Title
 ShortTitle
 year
 citescore
 pos
 1
 Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition
 CVPR
 2018-01-01
 37.26
 37.26
 2
 Journal of Statistical Software
 JSS
 2018-01-01
 25.02
 25.02
 3
 IEEE Transactions on Pattern Analysis and Machine Intelligence

PAMI

2018-01-01

19.67

20.07

4

Foundations and Trends in Machine Learning

FTML

2018-01-01

19.00

18.60

5

Proceedings of the IEEE International Conference on Computer Vision

ICCV

2018-01-01

16.64

16.64

The current positions of the axis labels on the right are decided by the values of *citescore* in 2018; These are the last values in the series,

In order to separate these two labels, I have created an artificial field, *pos*, to indicate the positions on the y-axis for the labels. The values for PAMI and FTML must be sufficiently separated that the labels do not overlap

The *pos* field will have the same values as *citescore*, except for the 2018 PAMI and FTML data points.

I increase slightly the value for PAMI in *pos* ,and decrease the value for FTML in order to separate them.

These changes will effect the labels for PAMI and FTML only,

This can be viewed as a non-data adjustment, as the label positions are determined by *pos*, which is an artificial field, created to solve an overlapping label problem.

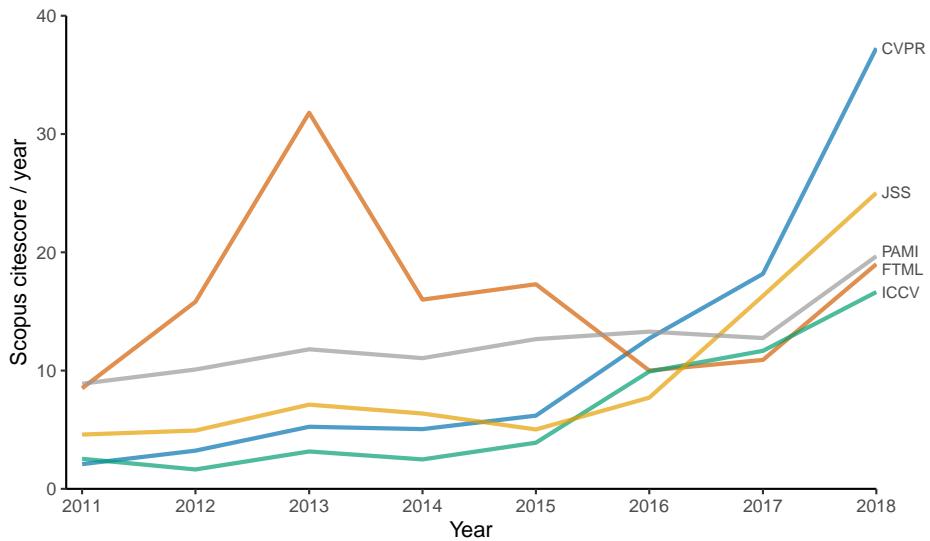
The positions of the lines representing each data series are of course unaltered .

These are determined by the *citescore* values. You should never alter data in order to make a non-data adjustment – for example, the placing of labels.

```

p <- ggplot(scopus_citescore_long, aes(year, citescore, color = ShortTitle)) +
  geom_line(size = 1, alpha = 0.7) +
  #geom_point(color = "white", size = 2) +
  # adds a second axis which is a duplicate (dup_axis) of the primary axis
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus citescore / year",
                     sec.axis = dup_axis(
                       breaks = scopus_citescore_final$pos,
                       labels = scopus_citescore_final$ShortTitle,
                       name = NULL)) +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y"), expand = expand)
  scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),
                     name = NULL) +
  theme_classic() +
  theme(legend.position = "none",
        plot.margin = margin(14, 7, 3, 1.5),
        axis.line.y.right = element_blank(),
        axis.ticks.y.right = element_blank(),
        axis.text.y.right = element_text(margin = margin(0, 0, 0, 0), size = 8))
)
p

```



8.8 A faceted approach to multiple time series

Facetting is where we partition a plot into a number of separate subplots. Each subplot contains the data associated with a value of a faceting or partitioning variable. In this case, each panel contains data associated with the values of the *ShortTitle* variable.

- Advantages – each series is clearly represented and can be compared to each other on the same scale
- We can see immediately see information that may not have stood out on inspection of the multi-series plot.

E.g. ICCV started with a relatively low citescore in 2011, as did CPVR – big contrast in their growth rates. PAMI’s growth rate has been comparatively modest up to 2017. in 2018, it doubled its previous growth/

- Disadvantage - false perception of similar trends due to narrowing of x-axis. A quick inspection would suggest that that the growth rates for CVPR and FTML are correlated but that’s not the case, FTML’s spike of growth occurred from 2011 to 2013 and CVPR’s occurred from 2015 to 2018. Visually they look like they are tracking each other – and this is a visual artifact of having a narrow x axis scale so that the panels fit in a horizontal row. Clearly, this technique is best when there aren’t too many data series to plot simultaneously.

```
p <- ggplot(scopus_citescore_long, aes(year, citescore, color = ShortTitle))+
  geom_line(size = 0.6) +
  #geom_point(color = "white", size = 2) +
  scale_shape_manual(values = c(21, 22, 23, 24, 25),
                     name = NULL) +
  # adds a second axis which is a duplicate (dup_axis) of the primary axis
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus citescore / year",
                     breaks=seq(0,40,by=5)
  ) +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y"), expand = expand_scale())
  scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"), name = NULL)
  theme_classic() +
  theme(legend.position = "none",
        plot.margin = margin(14, 7, 7, 1.5),
        axis.title.x = element_blank(),
        axis.title.y=element_text(margin = margin(t = 0, r = 10, b = 0, l = 0)),
```



8.9 A faceted approach with fill

We can improve the visual separation of the plots by using the fill technique we saw earlier.

The fill and the presence of the vertical fill outline at the start and end of each series makes it easier to visually determine the relative x axis positions of the visual features of each line.

We can see more easily that the spikes in the CVPR and FTML data occur at different times.

The fill also introduces the perception of area under the line – and it more clearly allows to make observations that

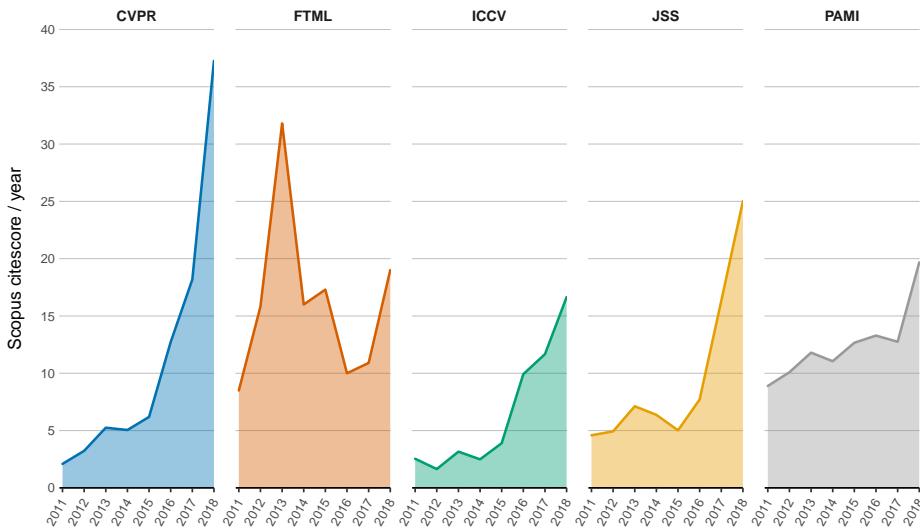
PAMI's citescore has been fairly consistent throughout the period – and that CPVR's huge growth is relatively late

Overall, we can see that FTML has the largest area under the curve indicating the largest cumulative citescore in the period, followed by PAMI.

This is information that is available in the first multi-series plot we generated – but not immediately perceptible

```
p <- ggplot(scopus_citescore_long, aes(year, citescore, color = ShortTitle, fill=ShortTitle))+
  geom_ridgeline(aes(height = citescore, y=0), size = 0.75, alpha=0.4) +
  #geom_point(color = "white", size = 2) +
  # adds a second axis which is a duplicate (dup_axis) of the primary axis
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus citescore / year",
                     breaks=seq(0,40,by=5)
  ) +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y"), expand = expand_scale(
    0, 1
  )),
  scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"), name = NULL),
  scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"), name = NULL),
  theme_classic() +
  theme(legend.position = "none",
        plot.margin = margin(14, 7, 7, 1.5),
        axis.title.x = element_blank(),
        axis.title.y=element_text(margin = margin(t = 0, r = 10, b = 0, l = 0)),
        axis.line.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.y= element_text(margin = margin(0, 0, 0, 0) ,size = 7.5),
        axis.text.x= element_text(angle=60, hjust = 1.2, size=7.5, margin = margin(0, 0, 0, 0) ),
        panel.grid.major.y = element_line(colour="grey", size = 0.1),
        strip.text.x = element_text(size=9, face="bold"),
        strip.background = element_blank(),
        panel.spacing.x = unit(4.8, "mm")
  ) +
  facet_grid(cols = vars(ShortTitle))
```

p



8.10 Bar Graphs for emphasizing and comparing individual values

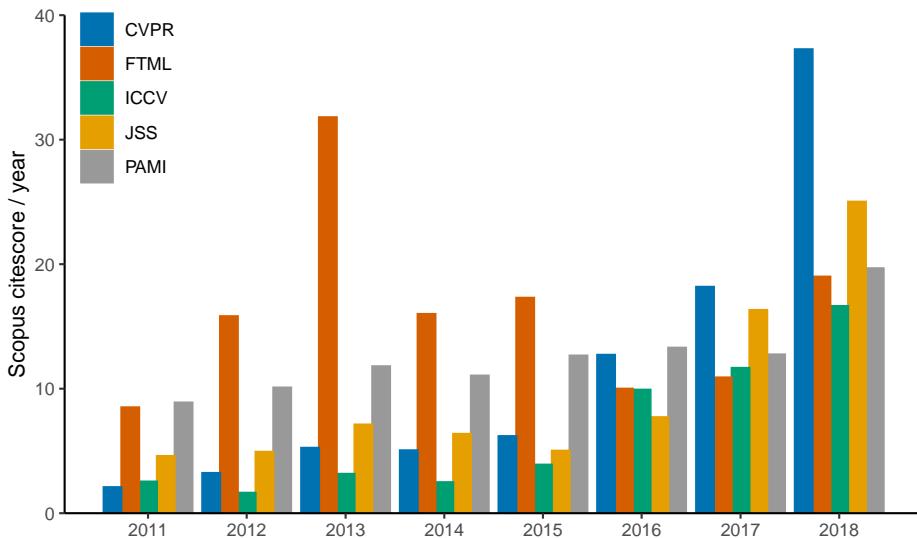
This is an example of the grouped bar chart that we saw last week. I've applied it to our times series citescore data. A citescore value for each journal is represented by a bar at each in each year.

This is a direct translation of the multi-series line graph we saw earlier to bar graph format

- The classic Bar Chart uses either horizontal or vertical bars (column chart) to show discrete, numerical comparisons across categories.
- One axis of the chart shows the specific categories being compared and the other axis represents a value scale.
- The length of bars encode values that allows accurate comparison of individual values to one and other.
- If your primary aim is to compare values on, say, a monthly/yearly basis rather than show trend - then a bar graph will work

ggplot items to notes * replaced geom_line with geom_bar * Within geom_bar I use the following attributes `stat = identity` and `position = dodge` * `identity` means that we use the raw variable values. The alternative is `bin`. * `dodge` means that the bars do not overlap as in a stacked bar chart * you could also add `colour = black` to put a line around each bar * I use `scale_fill_manual` to specify the bar fill colours

```
p <- ggplot(scopus_citescore_long, aes(year, citescore, color = ShortTitle, fill = ShortTitle)) +
  geom_bar(position="dodge", stat="identity") +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus citescore / year") +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),
                     name = NULL) +
  scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),
                    name = NULL) +
  theme_classic() +
  theme(
    axis.title.x = element_blank(),
    legend.title.align = 0.5,
    legend.position = c(0.005, 1.05),
    legend.just = c(0, 1),
    plot.margin = margin(14, 7, 3, 1.5))
p
```



8.11 A faceted approach

The next graph was produced by facetting the grouped bar chart.

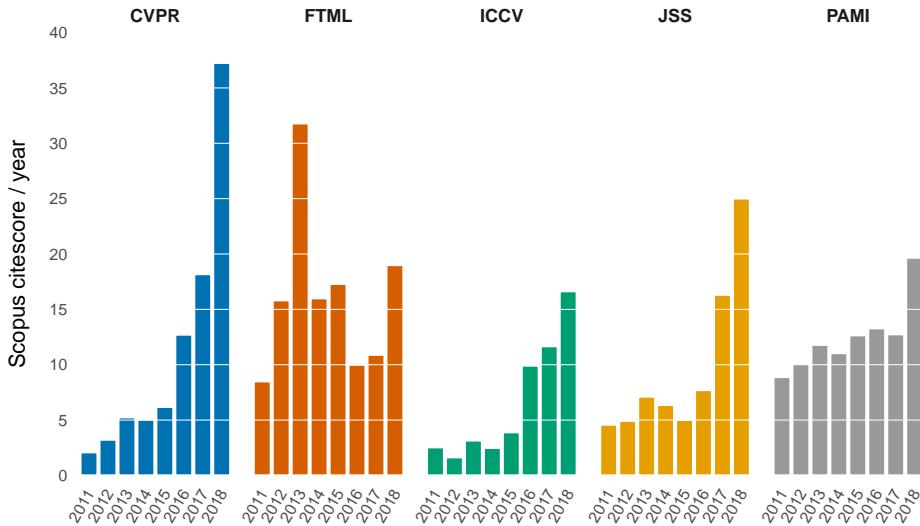
This is a translation to bar form of the faceted line graph approach we saw near the end of video It is an easier way to assess the differences between series. Each series has its own panel and its own x axis , but they share a y axis and this allow comparison of bar length across series.

This allows you to be much more quantitative - you can see trends but also quantities and proportions are easier to perceive because we are dealing with bar lengths

There is really no need for colour and I've coloured the bars to keep continuity with the previous analysis. However, we could just as easily represent this graph in dark grey

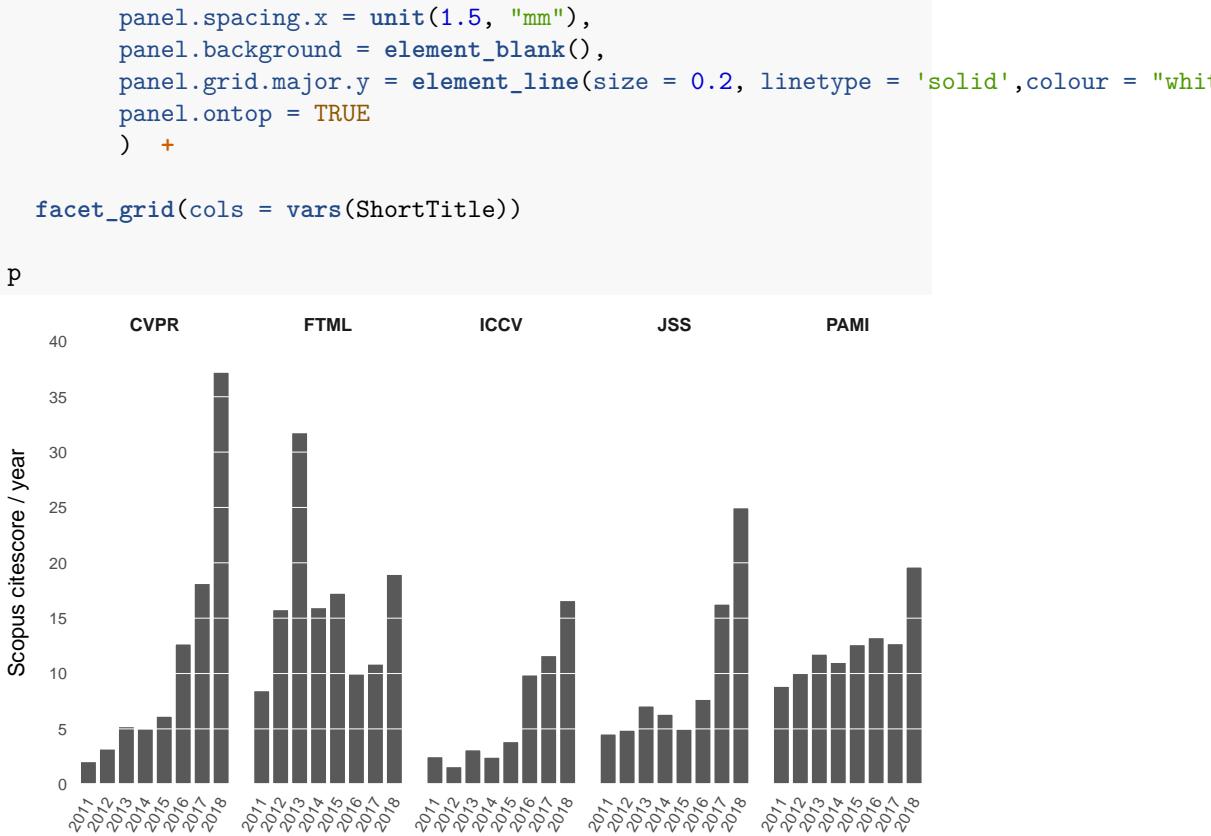
```
p <- ggplot(scopus_citescore_long, aes(year, citescore, fill = ShortTitle))+
  geom_bar(position="dodge", stat="identity", colour="white") +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus citescore / year", breaks=seq(0,40,by=5)) +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),
                    name = NULL) +
  theme_classic() +
  theme(legend.position = "none",
        plot.margin = margin(14, 7, 7, 1.5),
        axis.title.x = element_blank(),
        axis.title.y=element_text(margin = margin(t = 0, r = 10, b = 0, l = 0)),
        axis.line.y = element_blank(),
        axis.line.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y= element_text(margin = margin(0, 0, 0, 0) ,size = 7.5),
        axis.text.x= element_text(angle=60, hjust = 1.2, size=7.5, margin = margin(0, 0, 0, 0)),
        #panel.grid.major.y = element_line(colour="grey", size = 0.1),
        strip.text.x = element_text(size=9, face="bold"),
        strip.background = element_blank(),
        panel.spacing.x = unit(1.5, "mm"),
        panel.background = element_blank(),
        panel.grid.major.y = element_line(size = 0.2, linetype = 'solid', colour = "white"),
        panel.on top = TRUE
      ) +
  facet_grid(cols = vars(ShortTitle))
```

p



Here is the same graph without colour.

```
p <- ggplot(scopus_citescore_long, aes(year, citescore)) +
  geom_bar(position="dodge", stat="identity", colour="white") +
  scale_y_continuous(limits = c(0, 40), expand = c(0, 0),
                     name = "Scopus citesscore / year", breaks=seq(0,40,by=5)) +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y")) +
  #scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"), name = NULL) +
  theme_classic() +
  theme(legend.position = "none",
        plot.margin = margin(14, 7, 7, 1.5),
        axis.title.x = element_blank(),
        axis.title.y=element_text(margin = margin(t = 0, r = 10, b = 0, l = 0)),
        axis.line.y = element_blank(),
        axis.line.x = element_blank(),
        axis.ticks.y = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y= element_text(margin = margin(0, 0, 0, 0) ,size = 7.5),
        axis.text.x= element_text(angle=60, hjust = 1.2, size=7.5, margin = margin(0, 0, 0, 0) ),
        #panel.grid.major.y = element_line(colour="grey", size = 0.1),
        strip.text.x = element_text(size=9, face="bold"),
        strip.background = element_blank(),
```

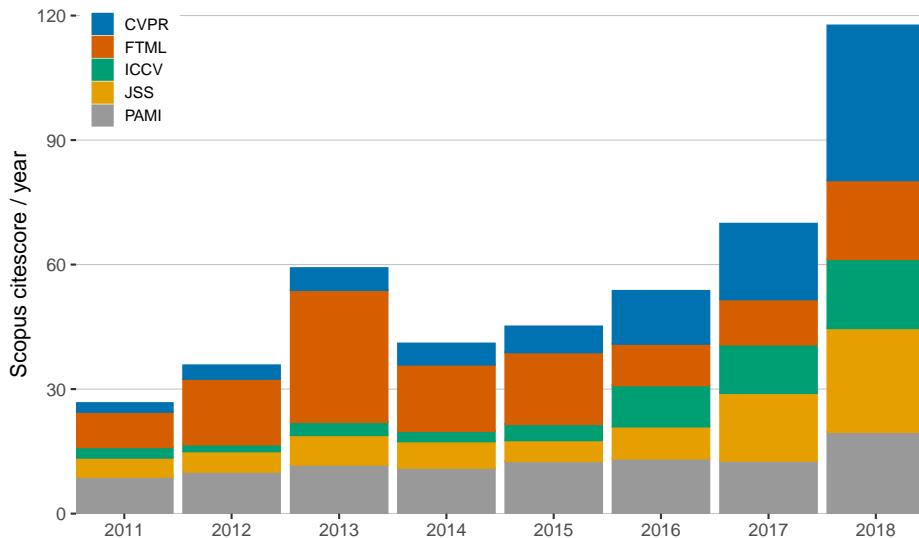


8.12 Stacked Bar Graphs

- Stacked Bar Graphs segment each bar according to the values in the categories in the data under comparison
- They are often used to show how a larger category is divided into smaller categories and what the relationship of each part has on the total amount.
- There are two types of Stacked Bar Graphs:
- **Simple Stacked Bar Graphs** place each value for the segment after the previous one. The total value of the bar is all the segment values added together. Ideal for comparing the total amounts across each group/segmented bar.
- **100% Stack Bar Graphs or Proportional Stack Bar Graphs** show the percentage-of-the-whole of each group and are plotted by the percentage of each value to the total amount in each group. This makes it easier to see the relative differences between quantities in each group.

- You can achieve the simple Stacked Bar Graph simply by omitting the `dodge` attribute from `geom_bar`

```
p <- ggplot(scopus_citescore_long, aes(year, citescore, color = ShortTitle, fill = ShortTitle)) +  
  geom_bar(stat="identity") +  
  
  scale_y_continuous(limits = c(0, 120), expand = c(0, 0),  
                     name = "Scopus citescore / year") +  
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y"), expand = c(0, 0) ) +  
  
  scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),  
                     name = NULL) +  
  scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"),  
                     name = NULL) +  
  
  theme_classic() +  
  
  theme(legend.title.align = 0.5,  
        legend.position = c(0.005, 1.05),  
        legend.just = c(0, 1),  
        legend.text = element_text(size=8),  
        legend.title = element_blank(),  
        legend.key.size = unit(0.8,"line"),  
        axis.title.x = element_blank(),  
        axis.line.x = element_blank(),  
        axis.line.y = element_blank(),  
        panel.grid.major.y = element_line(size = 0.1, colour = "grey"),  
        plot.margin = margin(14, 7, 3, 1.5))  
p
```



We can see the proportional trends here – some of which we spotted in our line graph analysis. The CVPR journal has relatively low values to start with but large growth after 2015, the PALMI journal relatively consistent values.

But this is less than satisfactory form a quantitative perspective, as it is next to impossible to fix a value on the internal segments of each bar – for example the green segments representing the ICCV journal.

From a quantitative accuracy perspective, I feel that this really only works where we have *two* categories to compare over time - particularly where one category is much larger than the other.

For visualising the relative proportions of multiple categories over time, the proportional stacked bar and its cousin the proportional stacked area bar are better - but really only still give us the gist of the relative proportions. We've come across the graphs in the chapter on proportions.

8.13 100% or Proportional Stacked Bar Graph For Time Series

- The proportional stacked bar graph or a 100% stacked bar graph show the proportional quantities (out of 100%) per category per year.
- To do this, you can use `geom_col` instead of `geom_bar` and set the `position` attribute to `fill`. With `position = fill`, the y values are scaled from 0 to 1.

To print the labels as percentages set `labels = scales:percent` in `scale_y_continuous`. You should reference the `scales` library first though

```

library(scales)

p <- ggplot(scopus_citescore_long, aes(x=year, y=citescore, color = ShortTitle, fill = ShortTitle))

  geom_col(position = "fill") + 

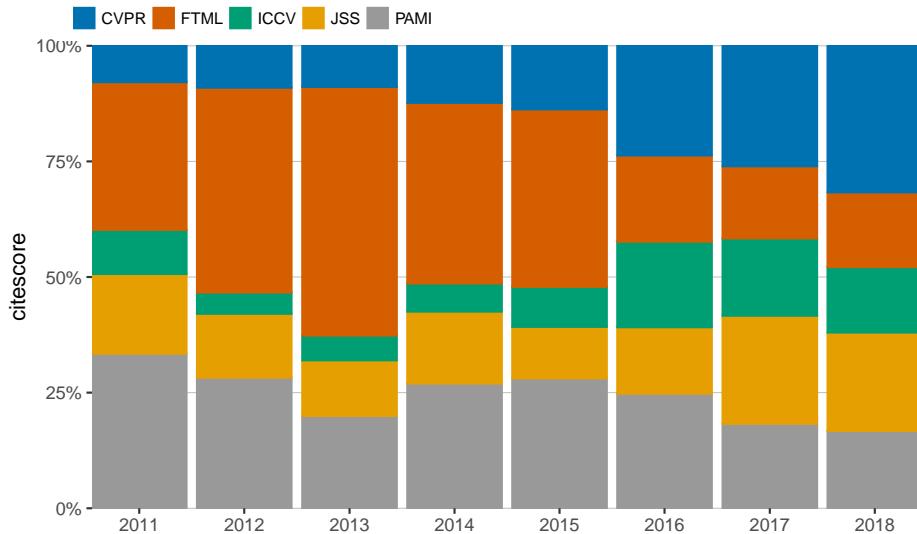
  scale_y_continuous( expand = c(0, 0),
                     labels = scales::percent) +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y"),expand = c(0, 0) ) + 

  scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00","#999999"),
                     name = NULL) +
  scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00","#999999"),
                     name = NULL) + 

  theme_classic() + 

  theme(legend.text = element_text(size=8),
        legend.title = element_blank(),
        legend.position = c(0.19,1.06),
        legend.key.size = unit(0.8,"line"),
        legend.direction = "horizontal",
        legend.spacing.x = unit(0.1, 'cm'),
        axis.line.x = element_blank(),
        axis.line.y = element_blank(),
        axis.title.x = element_blank(),
        panel.grid.major.y = element_line(size = 0.1, colour = "grey"),
        plot.margin = margin(30, 8, 8, 7))
p

```



8.14 Proportional Stacked Area Graph For Time Series

The idea here is the same as for the Proportional Stacked Bar Graph - to illustrate the percentage proportional difference between series over time. The objective is to show both *trend* and *proportional difference* over time

There are several trends we can see in the graph below such as

- The relatively low cite scores for ICCV (green) up to 2015
- The relatively constant scores for PAMI (grey)
- The dominance of FTML (orange) in 2013 and the fact it is the category that occupies the most area reflecting the fact that it has the overall most citations in the time period 2011 to 2018, followed by PAMI
- And the relatively late increase in citation values for CVPR

A plot like this doesn't really give us a strong sense of quantity - but it does give us a story based on relative proportions and trends.

Some ggplot points to note"

- `position="fill in geom_area`
- The code is the same as for the stacked bar plot used except that I have replaced `geom_bar` with `geom_area`
- I have added a thin black line between each area - the attributes for these are set within `geom_area`
- The presence of the legend causes some cognitive load - we can drop it entirely and label the areas.

8.14. PROPORTIONAL STACKED AREA GRAPH FOR TIME SERIES 185

- The easiest way to do this with full control over where the labels go is to use `annotate`

```

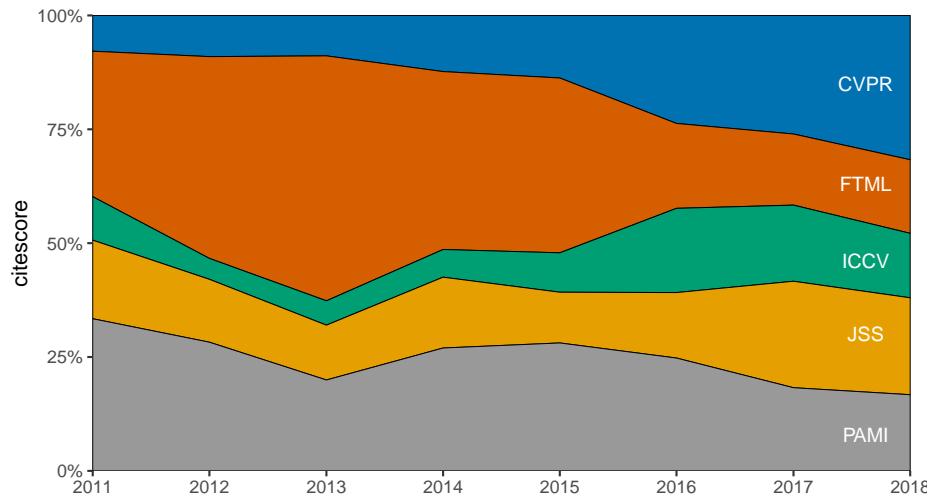
library(scales)

xpos<-as.Date("2017-08-15")

p <- ggplot(scopus_citescore_long, aes(x=year, y=citescore, color = ShortTitle, fill = ShortTitle))

  geom_area(position="fill", colour="black", size = .2) +
  
  scale_y_continuous( expand = c(0, 0),
                     labels = scales::percent) +
  scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y"),expand = c(0, 0)) +
  
  scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00","#999999"),
                     name = NULL) +
  scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00","#999999"),
                     name = NULL) +
  
  annotate("text",  x=xpos, y = 0.85, label = "CVPR", size = 3.5, family = "Helvetica", colour =
  
  annotate("text",  x=xpos, y = 0.63, label = "FTML", size = 3.5, family = "Helvetica", colour =
  
  annotate("text",  x=xpos, y = 0.46, label = "ICCV", size = 3.5, family = "Helvetica", colour =
  annotate("text",  x=xpos, y = 0.3, label = "JSS", size = 3.5, family = "Helvetica", colour =
  
  annotate("text",  x=xpos, y = 0.08, label = "PAMI", size = 3.5, family = "Helvetica", colour =
  
  theme_classic() +
  
  theme(legend.position = "none",
        #legend.text = element_text(size=8),
        #legend.title = element_blank(),
        #legend.position = c(0.19,1.06),
        #legend.key.size = unit(0.8,"line"),
        #legend.direction = "horizontal",
        #legend.spacing.x = unit(0.1, 'cm'),
        axis.line.x = element_blank(),
        axis.line.y = element_blank(),
        axis.title.x = element_blank(),
        panel.grid.major.y = element_line(size = 0.1, colour = "grey"),
        plot.margin = margin(30, 8, 8, 7))
p

```



8.15 Steam Graphs

An alternative would be to try a so called steam graph.

The Stream Graph is a variation of a Stacked Area Graph,

Stream Graphs display the changes in data over time of different categories through the use of flowing, shapes that somewhat resemble a river-like stream.

In a Stream Graph, the size of each individual stream shape is proportional to the values in each category. Typically the horizontal axis is used to represent the timescale.

Stream Graphs are ideal for displaying high-volume datasets, in order to discover trends and patterns over time across several categories.

The downside to Stream Graphs is that they can be very cluttered when there are very many categories. The categories with smaller values are often drowned out to make way for categories with much larger values, making it impossible to see all the data. Also, it's impossible to read the exact values visualised in a Stream Graph, as there is no axis to use as a reference.

Therefore, Stream Graphs are best used to audiences who don't intend to spend much time deciphering the graph and exploring its data. Stream Graphs are better for giving a more general view of the data.

```
library(ggTimeSeries)

p <- ggplot(scopus_citescore_long, aes(year, citescore, color = ShortTitle, fill = ShortTitle))

  stat_steamgraph(colour="black", size = .2) +
```

```

scale_y_continuous( expand = c(0, 0),
  name = "Scopus citescore / year") +
scale_x_date(name = "Year", breaks = "1 year", labels=date_format("%Y"),expand = c(0, 0) ) +  

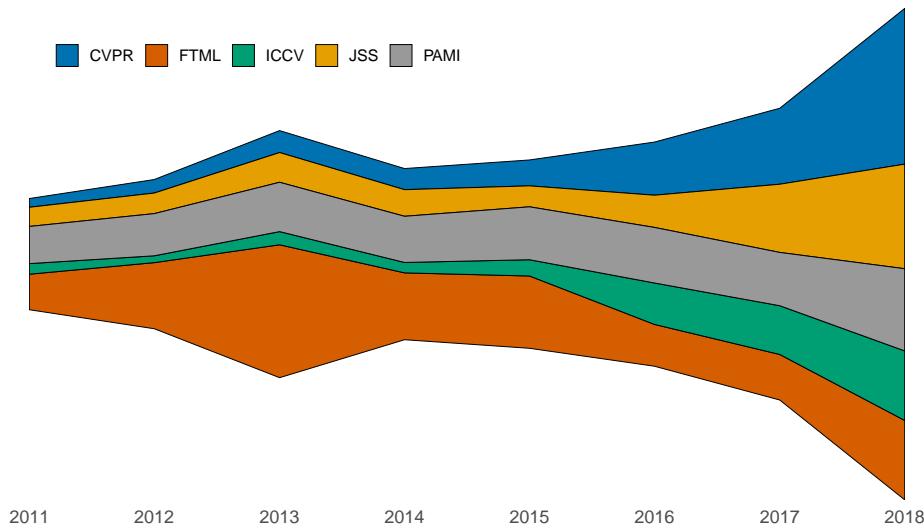
#scale_color_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00","#999999"), name = N  

scale_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00","#999999"),
  name = NULL) +  

theme_classic() +  

theme(legend.title.align = 0.5,
  legend.position = c(0.005, 0.95),
  legend.just = c(0, 1),
  legend.text = element_text(size=8),
  legend.direction = "horizontal",
  legend.title = element_blank(),
  legend.key.size = unit(0.8,"line"),
  axis.line.x = element_blank(),
  axis.line.y = element_blank(),
  axis.text.y = element_blank(),
  axis.ticks.x = element_blank(),
  axis.ticks.y = element_blank(),
  axis.title.y = element_blank(),
  axis.title.x = element_blank(),
  panel.grid.major.y = element_blank(),
  plot.margin = margin(14, 14, 8, 14))
p

```



8.16 Interactive Steam graph

The next second steam graph notes is interactive. If you hover over any part off the graph, it will tell you the value you are looking at.

This is a nice feature and certainly makes you want to play with it but I feel it has limited value in terms of communicating key messages in the data.

Even as a means of data exploration, unless values can be retained visually somehow, you will find yourself forgetting very quickly the values that you hover over a few seconds earlier.

For that reason, I prefer the static non-dynamic stream graph

```
# Libraries
#devtools::install_github("hrbrmstr/streamgraph")

library(streamgraph)
library(tidyr)

scopus_citescore_long %>%
  streamgraph( key="ShortTitle", value="citescore", date="year", interactive=TRUE) %>%
  sg_axis_x(1, "year", "%Y") %>%
  sg_fill_manual(values = c("#0072b2", "#D55E00", "#009e73", "#E69F00", "#999999"))%>%
  sg_legend(TRUE, "Journal: ")

## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is
```

8.17 Handling overplotting with time series data

In the previous example, we compared 5 time series on a single plot. However with much greater numbers of time series, representing the data on single plot is challenging. Lets consider a data set from the World Bank that gives Gross Domestic Product (GDP) per Capita for each country from 1990 to 2014

```
library(readr)
library(gridExtra)
library(dbplyr)

nations_data <- read_csv("../data/nations.csv", col_types = cols(population=col_double))

nations_data <- nations_data[complete.cases(nations_data[,c(3,4,5,10)]),]

nations_data_view<- nations_data[(nations_data$country=="Ireland"), ][1:5, c(4,3,10,5)]

t <- tableGrob(nations_data_view)
grid.arrange(t)
```

year	country	region	gdp_percap
1 2000-01-01	Ireland	Europe & Central Asia	29612.17
2 1996-01-01	Ireland	Europe & Central Asia	19974.53
3 1991-01-01	Ireland	Europe & Central Asia	14015.82
4 2012-01-01	Ireland	Europe & Central Asia	46063.34
5 1993-01-01	Ireland	Europe & Central Asia	15392.42

```
#nations_data<- nations_data[1:1000, c(4,3,10,5)]
```

This is fairly raw plot generated to just view the data.

Despite the mess, there is some discernible information that will help us when making visualisation decisions. Clearly there is wide variance. Some countries have an extremely high GDP and judging by the line density very many have quite low GDP.

including a mean line gives a clearer indication of the variance in the data.

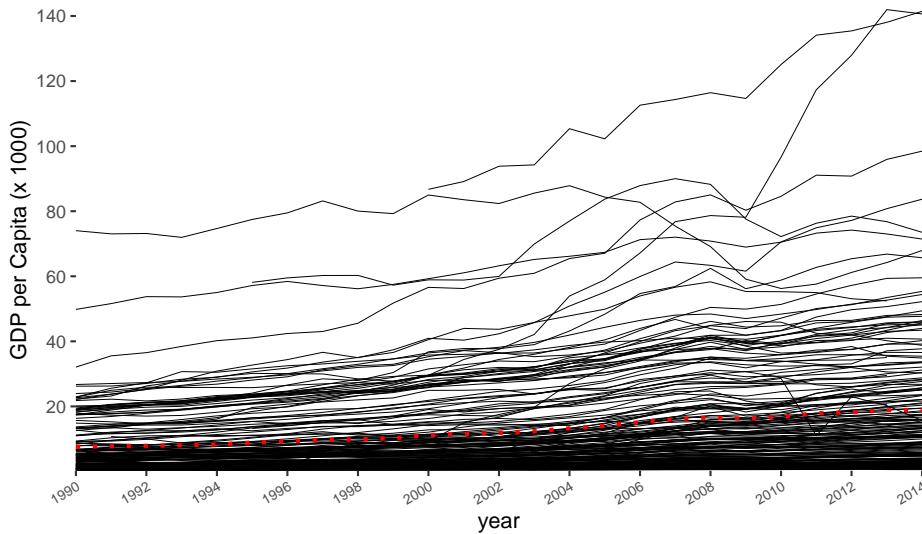
We can see that below the mean, the lines are densely backed close to the x axis. While above the mean, the line densities are much less and there some extremely high GDP values

```
nations_data %>% group_by(year) %>% summarise(mean = mean(gdp_percap)) ->mean_annual_gdp

mean_annual_gdp <-as.data.frame(mean_annual_gdp)

p_nations<- ggplot(nations_data, aes(x =year, y=gdp_percap, group = country)) +
  geom_line( size= 0.1, na.rm = TRUE ) +
  scale_y_continuous(breaks=seq(0,140000, by = 20000), name = "GDP per Capita (x 1000)", labels =
    scale_x_date(name = "year", breaks = "2 year", labels=date_format("%Y"), expand=c(0,0) ) +
  geom_line(data =mean_annual_gdp, aes(x=year, y=mean), inherit.aes = FALSE, col="red", size = 1,
  theme_classic() +
  theme(legend.position = "none",
        plot.margin = margin(14, 7, 3, 1.5),
        axis.text.x = element_text(angle=30, hjust = 1, size=6.5),
        axis.line = element_blank())

p_nations
```



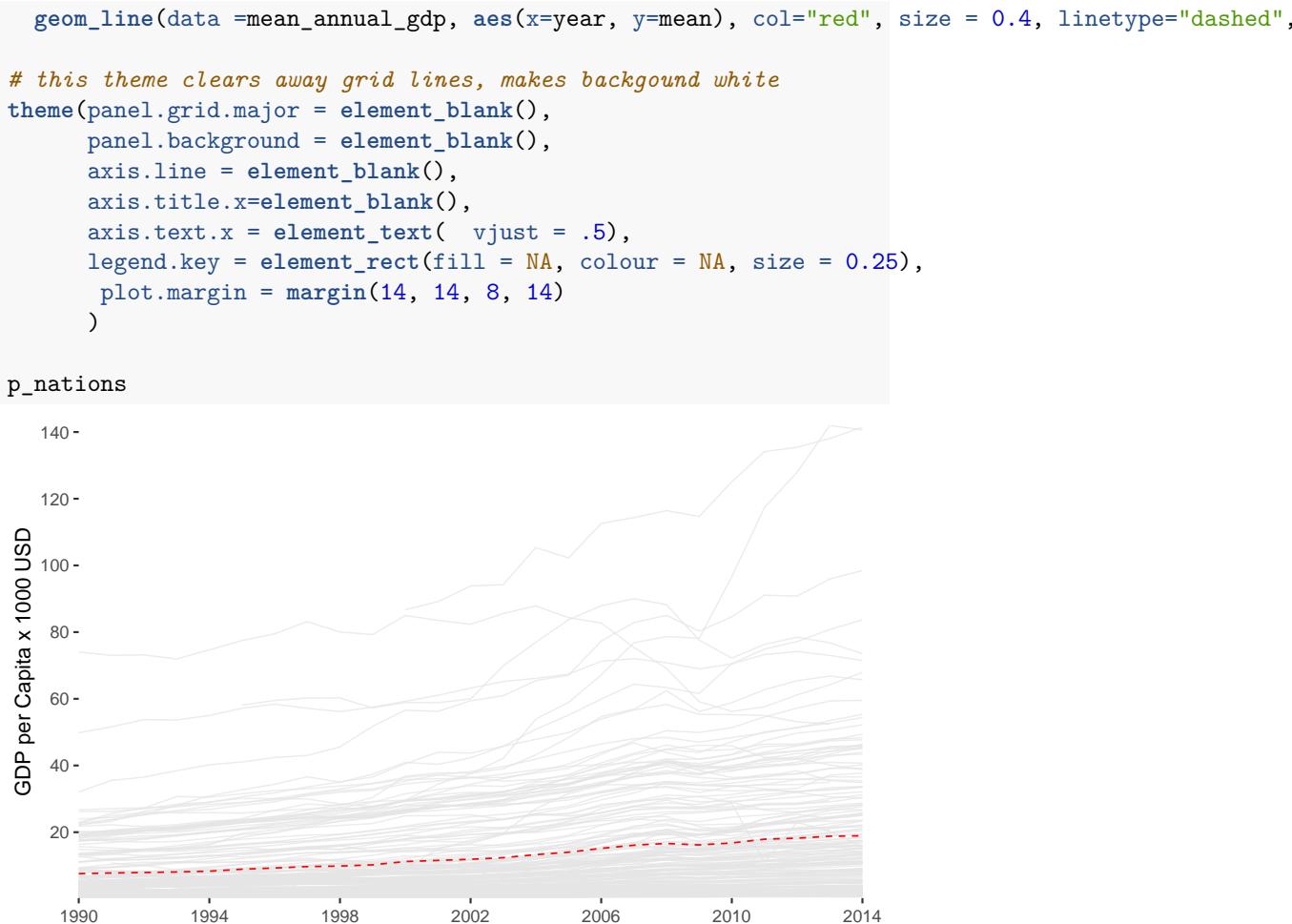
8.17.1 Approach

The data from every country cannot be shown at once. The intended audience of the visualisation is important here. Which countries must you represent? Ireland, UK and the US - but you also don't want to lose the context of the world

- Create a separate data set with the countries you want to show: your foreground data set
 - Create another data set minus the data points of the countries you wish to show: your background data set
1. Plot your background dataset first.
 2. Set all lines on it to colour grey with a low alpha value and low line size value. This is your plot background.
 3. Then add your foreground layer

To start with we'll plot the background layer:

```
# Plot Your background layer
p_nations<- ggplot(nations_data, aes(x =year, y=gdp_percap)) +
  scale_x_date(name = "year", breaks = "4 year", labels=date_format("%Y"), expand=c(0,0))
  scale_y_continuous(labels = seq(from = 0, to =140, by=20), breaks =seq(from = 0, to =140, by=20))
  p_nations<- p_nations +
    geom_line( aes(group = country),size= 0.35, na.rm = TRUE, color="grey90", alpha =0.7)
  #mean line
```



This background is pale enough that the foreground colours will still have high contrast, but also dark enough that we can clearly see where there are regions of overlap leading to almost a total fill of grey.

Now add your foreground layer:

The first lines of code help you to select the foreground countries and then to select a subset of data for those countries alone. This is the data that will be used for the foreground lines.

I've used colour, line width to foreground the target countries of interest

```

#foreground countries
target_countries <- c("Ireland", "United States", "United Kingdom")

#foreground data

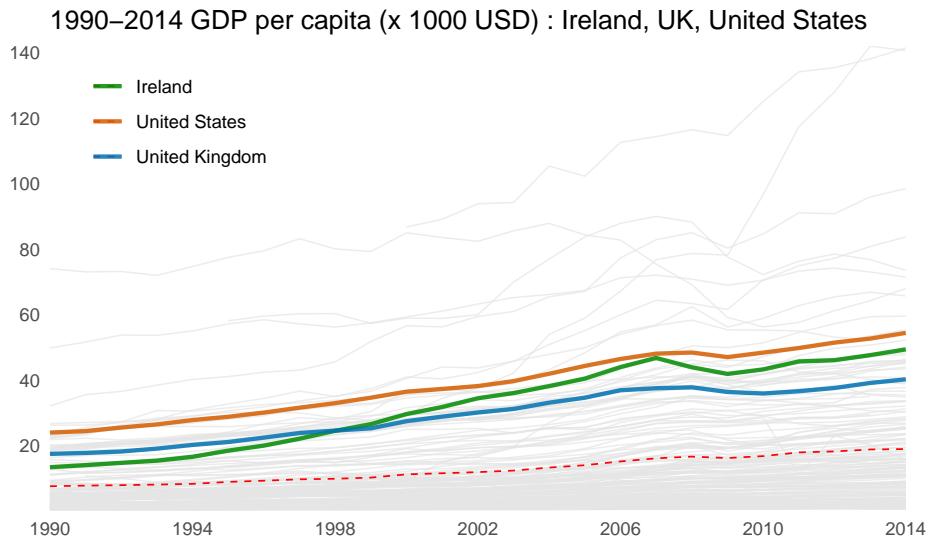
```

```

nations_data_targets<- subset(nations_data, country %in% target_countries)

#Your foreground layer
p_nations2 <- p_nations +
  geom_line(data=nations_data_targets, size =1, alpha=0.85, show.legend = TRUE, (aes(x
    scale_colour_manual(values = c("green4","#D55E00", "#0072b2"),name = NULL, limits =
    ggttitle("1990–2014 GDP per capita (x 1000 USD) : Ireland, UK, United States") +
    theme(
      axis.ticks.y.right = element_blank(),
      axis.ticks.y = element_blank(),
      axis.ticks.x = element_blank(),
      axis.title.y= element_blank(),
      axis.text.y.right = element_text(colour="black", size =8),
      legend.key = element_rect(fill = NA, colour = NA, size = 0.25),
      legend.position = c(0.15, .85)
    )
  )
p_nations2

```



The final plot achieves what set I out to do - to show the GDP for Ireland in comparison to the UK and the US – but also preserving the data from the every other country as a background – which unavoidably, but unobtrusively shows the extreme variance in wealth creation in the modern world economy.

The plot below is a second version of the same plot - this time without a legend. I've used the same trick I used earlier of creating a hidden second y-axis and labelling only 3 breaks at the final GDP values of Ireland, UK and the US.

The mean line is labelled using an annotation.

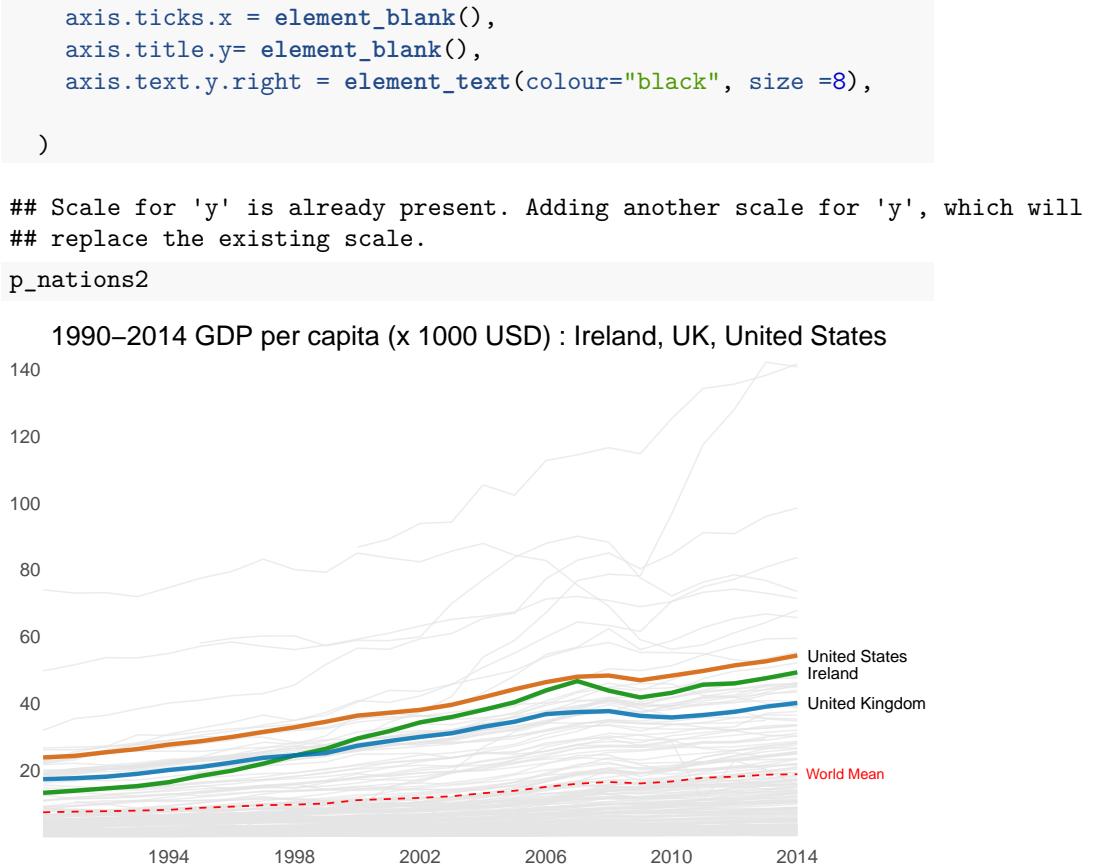
```
#foreground countries
target_countries <- c("Ireland", "United States", "United Kingdom")

#foreground data
nations_data_targets <- subset(nations_data, country %in% target_countries)

# gives the final count value that you want to show on the right axis
nations_data_targets_final <- filter(nations_data_targets, year == ymd("2014-01-01"))

mean_val_final <- mean_annual_gdp[which(mean_annual_gdp$year==as.Date("2014-01-01")),]$mean

#Your foreground layer
p_nations2 <- p_nations +
  geom_line(data=nations_data_targets, size = 1, alpha=0.85, show.legend = TRUE, (aes(x =year, y=gdp_per_cap))
# note how the limits attribute allows us to specify the order of items in the legend
  scale_colour_manual(values = c("green4", "#D55E00", "#0072b2"), name = NULL, limits = c("Ireland", "United Kingdom", "United States"))
  scale_y_continuous(labels = seq(from = 0, to = 140, by=20), breaks = seq(from = 0, to = 140000, by=20000),
    breaks = nations_data_targets_final$gdp_percap,
    labels = nations_data_targets_final$country,
    name = NULL,) )+
  ggtree("1990-2014 GDP per capita (x 1000 USD) : Ireland, UK, United States") +
  annotate("text", x=as.Date("2015-07-15"), y = mean_val_final, label = "World Mean", size = 2.5)
  coord_cartesian(xlim=c(as.Date("1990-04-01"), as.Date("2014-01-01"))), clip = 'off') +
  theme(
    legend.position = "none",
    axis.ticks.y.right = element_blank(),
    axis.ticks.y = element_blank(),
```



8.18 Using faceting for multiple time series

If you have to show many series simultaneously, faceting offers a solution

However, be aware of the cognitive overload in comparing multiple plots arranged over a large 2D space.

As the number of facet panels grows, the benefits of visualisation may lessen. With a busy plot, it will be important to strip the visualisation down as much as possible so there are no distracting visual artifacts

The visualisation here shows the GDP per European country as a black line from 1990 to 2014. The red dotted line indicates the mean GDP per year for Europe.

The GDP traces for each country are shown as a background trace – and I've ordered the facet panels in ascending order of final GDP, left to right, top to bottom.

The default ordering was alphabetical. However, ordering in ascending order allows you to perceive a gradual increase in the GGDG slope as you read left to right. I found that this was easier to read. Of course the ordering should be determined by the objective of the analysis – and another type of analysis may require alphabetical ordering so the reader can find the plot of particular country quickly.

A plot like this is about presenting trends - and not really about quantities or precise dates.

As I've limited the background data to the European countries and ordered the panels by final GDP – you get to see which country has the maximum value that is so apparent in the in background trace all the plots – Luxembourg

The trickiest part about producing this plot is to prevent ggplot trying to facet the background trace by country. You will notice that the background remains constant while each panel has a different foreground series.

```
library(scales) #date format function is in scales

#subset of data
nations_data_europe<- subset(nations_data, region=="Europe & Central Asia")

# mean gdp per year for europe data
nations_data_europe %>% group_by(year) %>% summarise(mean = mean(gdp_per_cap)) ->mean_annual_gdp_europe
mean_annual_gdp_europe <-as.data.frame(mean_annual_gdp_europe)

# select a list of countries ordered by their GDP in 2014
nations_data_europe%>%filter(year == ymd("2014-01-01"))%>%arrange(gdp_per_cap)%>%select(country)->nations_data_europe_levels

# This bit was tricky as the nations_data_europe_levels had to be converted to a df, a list and then a factor
nations_data_europe_levels<- unlist(as.list(as.data.frame(nations_data_europe_levels)))

# convert the country column to factor
nations_data_europe$country<- factor(nations_data_europe$country)

# reset the levels of the factor to the country order in nations_data_europe_levels
nations_data_europe$country<-factor(nations_data_europe$country, levels=nations_data_europe_levels)

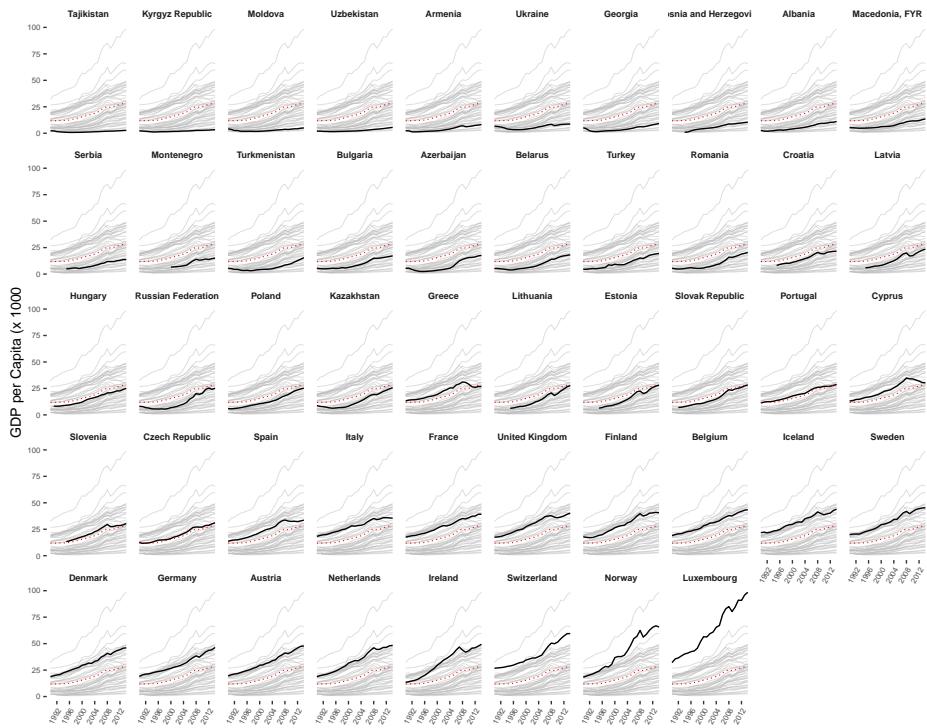
# plotting background reference lines across each facet:
backgrounddata <- nations_data_europe
colnames(backgrounddata)[3] <- "groupVar" # rename the 'country' column on the referenceLines to groupVar

# the reference plot is not faceted. The faceting should apply to data from nations_data
backgroundlines<- geom_line(data = backgrounddata, na.rm = TRUE, aes(x =year, y=gdp_per_cap, group=groupVar))
```

```
p_nations<- ggplot(nations_data_europe, aes(x =year, y=gdp_percap)) +
  backgroundlines +
  geom_line(size= 0.5, colour = "black", na.rm = TRUE ) +
  geom_line(data =mean_annual_gdp_europe, aes(x =year, y=mean), col="red", size = 0.4,
  scale_x_date(name = "year", breaks = "4 year", labels=date_format("%Y")) +
  scale_y_continuous(name = "GDP per Capita (x 1000", breaks=seq(0,100000, by= 25000),
  theme_bw() +
  theme(legend.position = "none",
        plot.margin = margin(14, 7, 3, 1.5),
        axis.text.x = element_text(angle=60, hjust = 1, size=6),
        axis.text.y = element_text( size=6),
        axis.title.x = element_blank(),
        strip.text.x = element_text(size=7, face="bold"),
        strip.background = element_blank(),
        panel.spacing.x = unit(1.5, "mm"),
        panel.spacing.y = unit(2, "mm"),
        panel.grid = element_blank(),
        panel.border = element_blank()

    ) +
  facet_wrap(vars(country),ncol=10)

p_nations
```



Chapter 9

Tutorials: Creating a Bubble Plot

A bubble chart is basically a scatterplot with a third numeric variable encoded as point size.

The first plot we will build shows Life Expectancy vs GDP per Capita for the different countries in the data for a single year. The point size will represent the country's population.

Our objective is to create a plot like the following:

9.1 Objective: Make bubble plot to examine Life expectancy vs GDP per capita

Using the nations data set, our goal is to plot a bubble chart that illustrates the relationship vs GDP for the different regions of the world. We want also want the plot to illustrate the population of each country. In the second tutorial. We examine the trends over time.

We will produce: - a bubble plot for a given year where each point represents a country and is coloured according to region. In this plot, each point is sized according to population. We want to annotate some of the more extreme values with country names to give the reader some basis for comparison. We also want to pinpoint Ireland.

We also want to produce a faceted view of the plot where each facet represents the countries within a given region

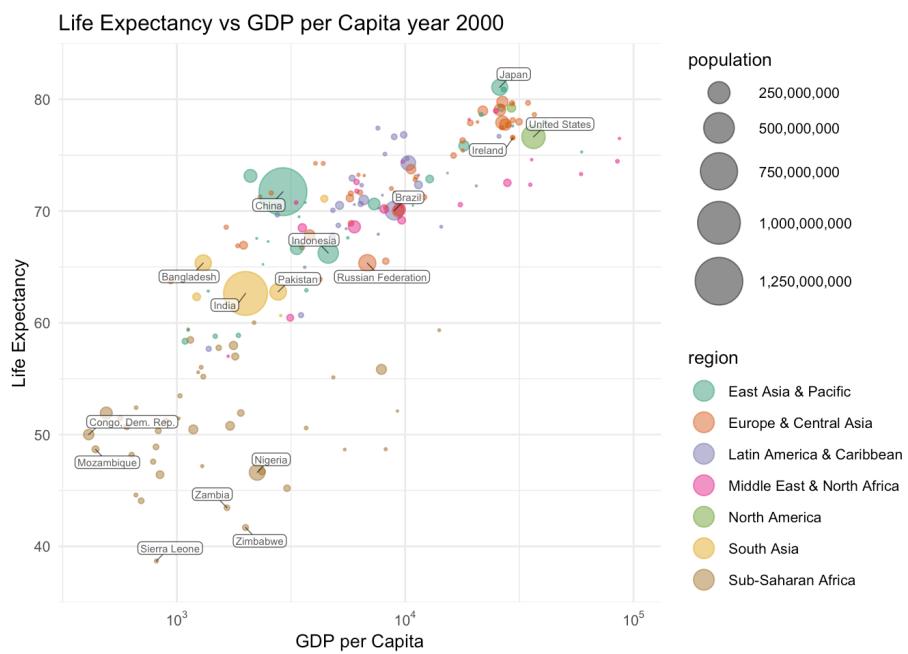


Figure 9.1: Life Expectancy vs GDP, year 2000

9.1. OBJECTIVE: MAKE BUBBLE PLOT TO EXAMINE LIFE EXPECTANCY VS GDP PER CAPITA201

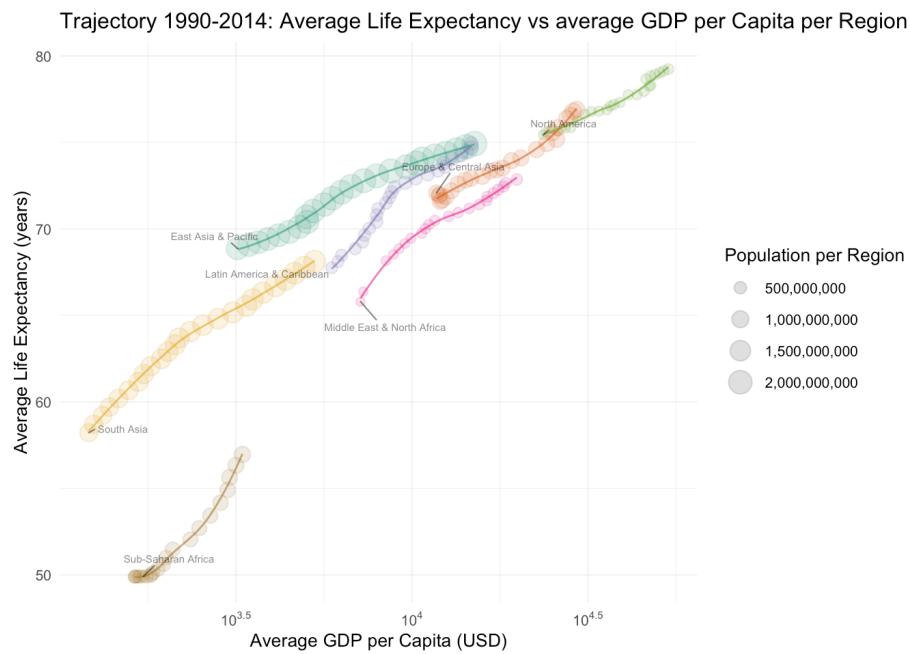


Figure 9.2: 1990-2014 Life Expectancy vs GDP

9.2 Loading Data

nations.csv : Data from the World Bank Indicator's data portal. This data set contains the following fields:

- **iso2c iso3c** Two- and Three-letter codes for each country, assigned by the International Organization for Standardization.
- **country** Country name.
- **year**
- **population** Estimated total population at mid-year, including all residents apart from refugees.
- **gdp_per_cap** Gross Domestic Product per capita in current international dollars, corrected for purchasing power in different territories.
- **life_expect** Life expectancy at birth, in years.
- **population** Estimated total population at mid-year, including all residents apart from refugees.
- **birth_rate** Live births during the year per 1,000 people, based on mid-year population estimate.
- **neonatal_mortal_rate** Neonatal mortality rate: babies dying before reaching 28 days of age, per 1,000 live births in a given year.
- **region** income World Bank regions and income groups, explained here.

9.3 Load required packages and the data

Download the data from Blackboard. Save it to a folder that you can access again. If you are using a lab machine, save it to a network drive.

Now, open a new R script in RStudio, save the blank script to the folder with the data for this week

Set set your working directory to this location by selecting from the top menu Session>Set Working Directory>To Source File Location.

Now copy the following code into your script and run to load `readr`, `ggplot2`, and then load the `nations.csv` data

```
# if you haven't already installed the readr package you can do so now
# readr allows you to easily read and write data from/to a file

#install.packages("readr")

# load required packages
library(tidyverse)
library(ggplot2 )
library(readr)

# load disease and democracy data
nations_data <- read_csv("../data/nations.csv")
```

```
library(kableExtra)
kable(head(nations_data, 100), digits = 2, format = "html", row.names = TRUE) %>%
  kable_styling( bootstrap_options = c("striped"), full_width = T,
                 font_size = 10) %>%
  scroll_box(height = "300px")
```

iso2c
iso3c
country
year
gdp_per_cap
life_expect
population
birth_rate
neonat_mortal_rate
region
income
1
AD
AND
Andorra
1996
NA
NA
64291
10.90
2.8
Europe & Central Asia
High income
2
AD

AND

Andorra

1994

NA

NA

62707

10.90

3.2

Europe & Central Asia

High income

3

AD

AND

Andorra

2003

NA

NA

74783

10.30

2.0

Europe & Central Asia

High income

4

AD

AND

Andorra

1990

NA

NA

54511

11.90

4.3

Europe & Central Asia

High income

5

AD

AND

Andorra

2009

NA

NA

85474

9.90

1.7

Europe & Central Asia

High income

6

AD

AND

Andorra

2011

NA

NA

82326

NA

1.6

Europe & Central Asia

High income

7

AD

AND

Andorra

2004

NA

NA

78337

10.90

2.0

Europe & Central Asia

High income

8

AD

AND

Andorra

2010

NA

NA

84419

9.80

1.7

Europe & Central Asia

High income

9

AD

AND

Andorra

2001

NA

NA

67770

11.80

2.1

Europe & Central Asia

High income

10

AD

AND

Andorra

2002

NA

NA

71046

11.20

2.1

Europe & Central Asia

High income

11

AD

AND

Andorra

1997

NA

NA

64147

11.20

2.6

Europe & Central Asia

High income

12

AD

AND

Andorra

1993

NA

NA

61003

11.40

3.4

Europe & Central Asia

High income

13

AD

AND

Andorra

2008

NA

NA

85616

10.40

1.8

Europe & Central Asia

High income

14

AD

AND

Andorra

1999

NA

NA

64161

12.60

2.3

Europe & Central Asia

High income

15

AD
AND
Andorra
2014
NA
NA
72786
NA
1.5
Europe & Central Asia
High income
16
AD
AND
Andorra
2005
NA
NA
81223
10.70
1.9
Europe & Central Asia
High income
17
AD
AND
Andorra
2012
NA
NA
79316

9.50

1.6

Europe & Central Asia

High income

18

AD

AND

Andorra

2013

NA

NA

75902

NA

1.5

Europe & Central Asia

High income

19

AD

AND

Andorra

1992

NA

NA

58904

12.10

3.7

Europe & Central Asia

High income

20

AD

AND

Andorra

1995

NA

NA

63854

11.00

3.0

Europe & Central Asia

High income

21

AD

AND

Andorra

1998

NA

NA

63888

11.90

2.4

Europe & Central Asia

High income

22

AD

AND

Andorra

1991

NA

NA

56674

11.90

4.0

Europe & Central Asia

High income

23

AD

AND

Andorra

2006

NA

NA

83373

10.60

1.9

Europe & Central Asia

High income

24

AD

AND

Andorra

2007

NA

NA

84878

10.10

1.8

Europe & Central Asia

High income

25

AD

AND

Andorra

2000

NA
NA
65399
11.30
2.2
Europe & Central Asia
High income
26
AE
ARE
United Arab Emirates
1991
73037.00
72.01
1913190
24.65
7.9
Middle East & North Africa
High income
27
AE
ARE
United Arab Emirates
1993
71959.88
72.57
2127863
22.36
7.3
Middle East & North Africa
High income

28

AE

ARE

United Arab Emirates

2001

83533.56

74.71

3217865

15.83

5.5

Middle East & North Africa

High income

29

AE

ARE

United Arab Emirates

1992

73154.28

72.30

2019014

23.50

7.6

Middle East & North Africa

High income

30

AE

ARE

United Arab Emirates

1994

74683.89

72.85

2238281
21.26
6.9
Middle East & North Africa
High income
31
AE
ARE
United Arab Emirates
2007
75426.79
76.11
6010100
12.81
4.7
Middle East & North Africa
High income
32
AE
ARE
United Arab Emirates
2004
87843.71
75.46
3975945
14.24
5.1
Middle East & North Africa
High income
33
AE

ARE

United Arab Emirates

1996

79480.47

73.38

2467726

19.31

6.4

Middle East & North Africa

High income

34

AE

ARE

United Arab Emirates

2006

82754.28

75.90

5171255

13.26

4.9

Middle East & North Africa

High income

35

AE

ARE

United Arab Emirates

2000

84974.89

74.45

3050128

16.40

5.6

Middle East & North Africa

High income

36

AE

ARE

United Arab Emirates

1999

79262.39

74.19

2884188

17.02

5.8

Middle East & North Africa

High income

37

AE

ARE

United Arab Emirates

1990

74017.30

71.72

1811458

25.77

8.2

Middle East & North Africa

High income

38

AE

ARE

United Arab Emirates

1997

83165.26

73.65

2595220

18.47

6.2

Middle East & North Africa

High income

39

AE

ARE

United Arab Emirates

2005

84338.36

75.69

4481976

13.73

5.0

Middle East & North Africa

High income

40

AE

ARE

United Arab Emirates

1998

80040.22

73.92

2733770

17.70

6.0

Middle East & North Africa

High income
41
AE
ARE
United Arab Emirates
2008
69124.40
76.31
6900142
12.42
4.6
Middle East & North Africa
High income
42
AE
ARE
United Arab Emirates
2012
61172.99
77.02
8952542
11.27
4.0
Middle East & North Africa
High income
43
AE
ARE
United Arab Emirates
2011
57594.13

76.85

8734722

11.51

4.1

Middle East & North Africa

High income

44

AE

ARE

United Arab Emirates

2009

59100.38

76.49

7705423

12.08

4.5

Middle East & North Africa

High income

45

AE

ARE

United Arab Emirates

1995

77467.24

73.12

2350192

20.24

6.6

Middle East & North Africa

High income

46

AE

ARE

United Arab Emirates

2002

82369.69

74.97

3394060

15.29

5.4

Middle East & North Africa

High income

47

AE

ARE

United Arab Emirates

2014

67921.40

77.37

9086139

10.82

3.6

Middle East & North Africa

High income

48

AE

ARE

United Arab Emirates

2013

64229.46

77.20

9039978

11.04

3.8

Middle East & North Africa

High income

49

AE

ARE

United Arab Emirates

2003

85563.64

75.22

3625798

14.76

5.2

Middle East & North Africa

High income

50

AE

ARE

United Arab Emirates

2010

56245.48

76.68

8329453

11.78

4.3

Middle East & North Africa

High income

51

AF

AFG

Afghanistan

2009

1525.52

58.60

27207291

40.27

39.1

South Asia

Low income

52

AF

AFG

Afghanistan

2011

1712.59

59.33

28809167

37.64

38.1

South Asia

Low income

53

AF

AFG

Afghanistan

1996

NA

53.60

17481800

49.04

47.5

South Asia

Low income

54

AF

AFG

Afghanistan

1993

NA

52.02

14824371

48.84

49.9

South Asia

Low income

55

AF

AFG

Afghanistan

1994

NA

52.61

15869967

48.90

49.1

South Asia

Low income

56

AF

AFG

Afghanistan

1991

NA

50.64

12789374

48.90

51.9

South Asia

Low income

57

AF

AFG

Afghanistan

2002

895.60

55.86

21487079

47.23

43.9

South Asia

Low income

58

AF

AFG

Afghanistan

2012

1934.29

59.68

29726803

36.40

37.4

South Asia

Low income

59
AF
AFG
Afghanistan
2005
1039.41
57.03
24399948
44.89
41.7
South Asia
Low income
60
AF
AFG
Afghanistan
2010
1629.17
58.97
27962207
38.94
38.7
South Asia
Low income
61
AF
AFG
Afghanistan
2006
1095.66
57.43

25183615

43.89

41.0

South Asia

Low income

62

AF

AFG

Afghanistan

2004

940.48

56.63

23499850

45.76

42.5

South Asia

Low income

63

AF

AFG

Afghanistan

1992

NA

51.36

13745630

48.83

50.9

South Asia

Low income

64

AF

AFG
Afghanistan
1997
NA
54.02
18034130
49.04
47.0
South Asia
Low income
65
AF
AFG
Afghanistan
1995
NA
53.14
16772522
48.98
48.2
South Asia
Low income
66
AF
AFG
Afghanistan
2014
1939.95
60.37
31627506
34.23

36.1

South Asia

Low income

67

AF

AFG

Afghanistan

2003

945.69

56.24

22507368

46.54

43.2

South Asia

Low income

68

AF

AFG

Afghanistan

2001

NA

55.49

20531160

47.84

44.5

South Asia

Low income

69

AF

AFG

Afghanistan

2000
NA
55.13
19701940
48.33
45.2
South Asia
Low income
70
AF
AFG
Afghanistan
1990
NA
49.86
12067570
49.03
52.8
South Asia
Low income
71
AF
AFG
Afghanistan
2007
1245.06
57.83
25877544
42.78
40.4
South Asia

Low income

72

AF

AFG

Afghanistan

2008

1283.04

58.23

26528741

41.56

39.8

South Asia

Low income

73

AF

AFG

Afghanistan

1999

NA

54.77

19038420

48.70

45.6

South Asia

Low income

74

AF

AFG

Afghanistan

2013

1941.90

60.03
30682500
35.25
36.8
South Asia
Low income
75
AF
AFG
Afghanistan
1998
NA
54.40
18511480
48.93
46.1
South Asia
Low income
76
AG
ATG
Antigua and Barbuda
1997
13732.77
72.76
72232
20.25
10.7
Latin America & Caribbean
Upper middle income
77

AG
ATG
Antigua and Barbuda
1991
11594.58
71.40
62412
18.88
14.1
Latin America & Caribbean
Upper middle income
78
AG
ATG
Antigua and Barbuda
1992
11766.78
71.63
63434
19.05
13.5
Latin America & Caribbean
Upper middle income
79
AG
ATG
Antigua and Barbuda
2011
19987.92
75.47
88152

16.76
5.8
Latin America & Caribbean
Upper middle income
80
AG
ATG
Antigua and Barbuda
1996
13236.87
72.54
70245
20.06
11.2
Latin America & Caribbean
Upper middle income
81
AG
ATG
Antigua and Barbuda
2000
15312.57
73.40
77648
20.24
9.4
Latin America & Caribbean
Upper middle income
82
AG
ATG

Antigua and Barbuda
1998
14108.11
72.97
74206
20.35
10.3
Latin America & Caribbean
Upper middle income
83
AG
ATG
Antigua and Barbuda
1999
14554.48
73.19
76041
20.35
9.9
Latin America & Caribbean
Upper middle income
84
AG
ATG
Antigua and Barbuda
2014
22157.58
75.94
90900
16.32
5.1

Latin America & Caribbean

Upper middle income

85

AG

ATG

Antigua and Barbuda

1990

11087.02

71.16

61906

18.78

14.7

Latin America & Caribbean

Upper middle income

86

AG

ATG

Antigua and Barbuda

2004

17593.88

74.23

81718

18.99

7.9

Latin America & Caribbean

Upper middle income

87

AG

ATG

Antigua and Barbuda

2013

21008.15
75.78
89985
16.45
5.3
Latin America & Caribbean
Upper middle income
88
AG
ATG
Antigua and Barbuda
2009
21671.38
75.15
86300
17.22
6.3
Latin America & Caribbean
Upper middle income
89
AG
ATG
Antigua and Barbuda
1993
12415.41
71.87
64868
19.27
12.9
Latin America & Caribbean
Upper middle income

90
AG
ATG
Antigua and Barbuda
1994
13141.20
72.09
66550
19.54
12.3
Latin America & Caribbean
Upper middle income
91
AG
ATG
Antigua and Barbuda
2005
19067.07
74.43
82565
18.60
7.5
Latin America & Caribbean
Upper middle income
92
AG
ATG
Antigua and Barbuda
2002
15372.58
73.82

80030
19.72
8.6
Latin America & Caribbean
Upper middle income
93
AG
ATG
Antigua and Barbuda
2003
16426.92
74.02
80904
19.37
8.2
Latin America & Caribbean
Upper middle income
94
AG
ATG
Antigua and Barbuda
2007
24503.84
74.80
84397
17.85
6.9
Latin America & Caribbean
Upper middle income
95
AG

ATG

Antigua and Barbuda

2012

20956.34

75.62

89069

16.59

5.5

Latin America & Caribbean

Upper middle income

96

AG

ATG

Antigua and Barbuda

1995

12516.60

72.32

68349

19.82

11.7

Latin America & Caribbean

Upper middle income

97

AG

ATG

Antigua and Barbuda

2008

24723.11

74.98

85350

17.51

6.6

Latin America & Caribbean

Upper middle income

98

AG

ATG

Antigua and Barbuda

2010

20151.31

75.31

87233

16.97

6.0

Latin America & Caribbean

Upper middle income

99

AG

ATG

Antigua and Barbuda

2001

14907.07

73.61

78972

20.02

9.0

Latin America & Caribbean

Upper middle income

100

AG

ATG

Antigua and Barbuda

```

2006
22040.91
74.62
83467
18.21
7.2
Latin America & Caribbean
Upper middle income

```

9.4 Create a subset of the data to plot

To start, we will use the year 2000 as the first date we wish to analyse.

Create a subset of the data set which contains the data for the year 2000

```
nations_2000<- subset(nations_data, year==2000)
```

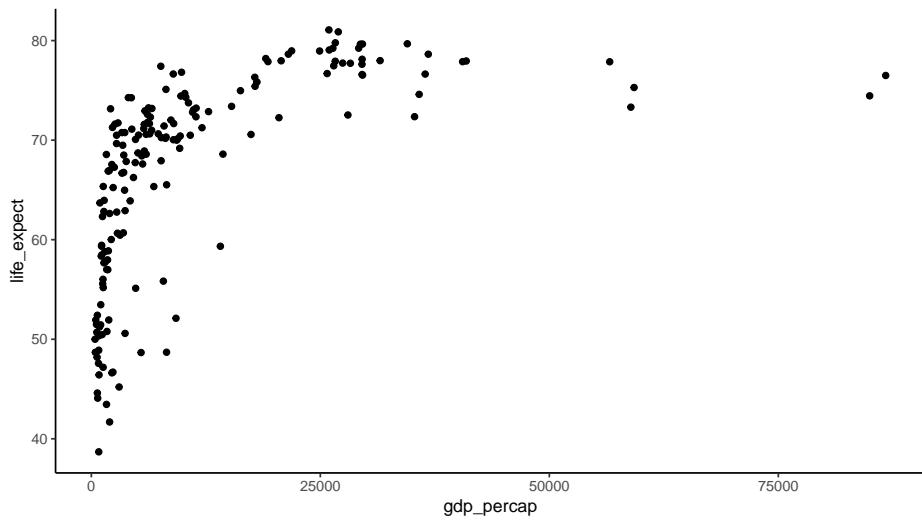
9.5 Create the basic plot

```

p <- ggplot(nations_2000, aes(x=gdp_per_cap, y=life_expect))
p <- p + geom_point()
p

```

Warning: Removed 32 rows containing missing values (geom_point).



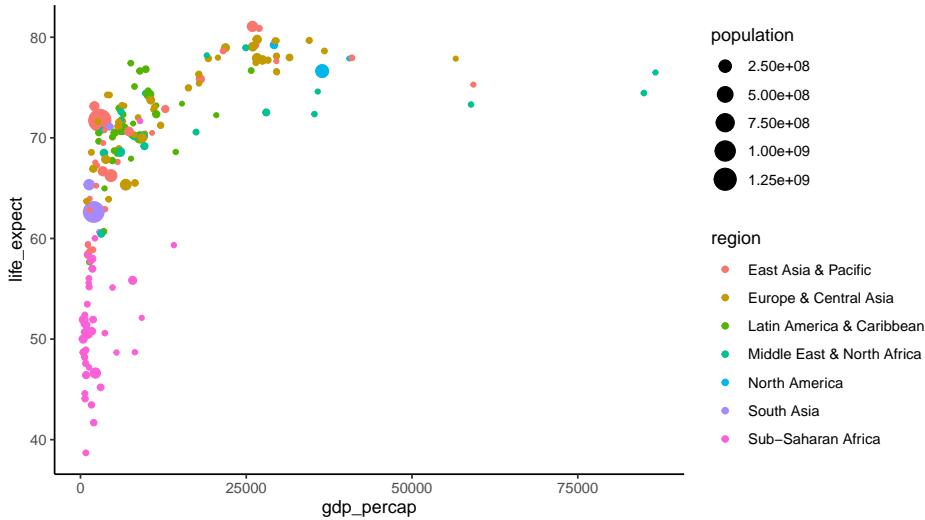
9.6. IDENTIFY THE FEATURES YOU NEED TO FIX/ADJUST WITH THE BASIC PLOT 243

Now we can group up the countries by region

We can also size each point according to population of the country it represents

```
p <- ggplot(nations_2000, aes(x=gdp_per_cap, y=life_expect, size=population, color=region))  
p + geom_point()
```

```
## Warning: Removed 32 rows containing missing values (geom_point).
```

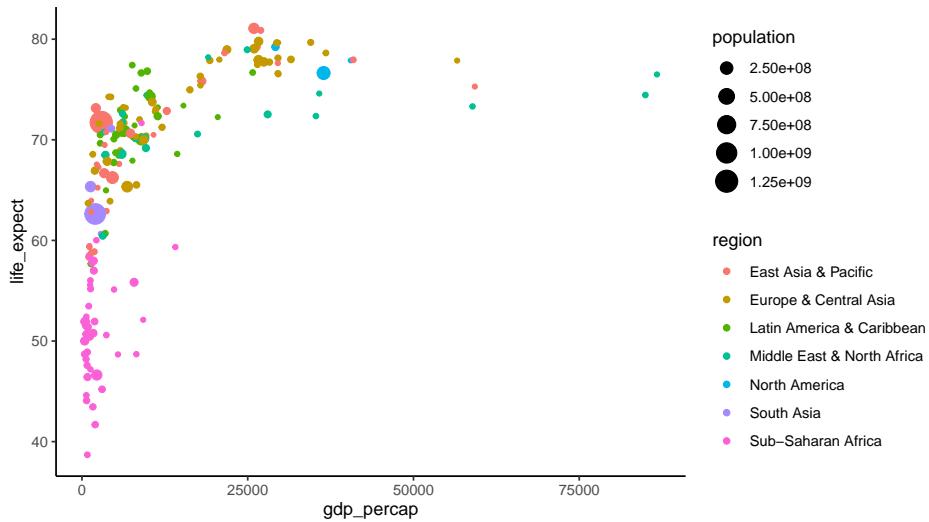


9.6 Identify the features you need to fix/adjust with the basic plot

- Fix the missing data messages
- chart title
- axis labels
- axis scale
- overlap
- colour
- point area scale
- legend - use non scientific notation
- annotated labels
- Grid layout

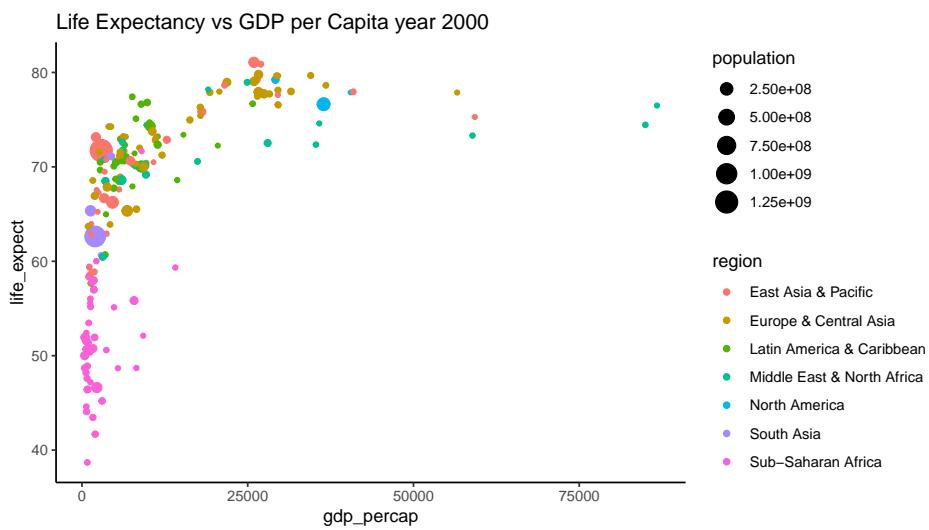
9.7 Fix the missing data messages

```
p + geom_point(na.rm = TRUE) # na.rm=TRUE removes rows with missing values from the analysis
```

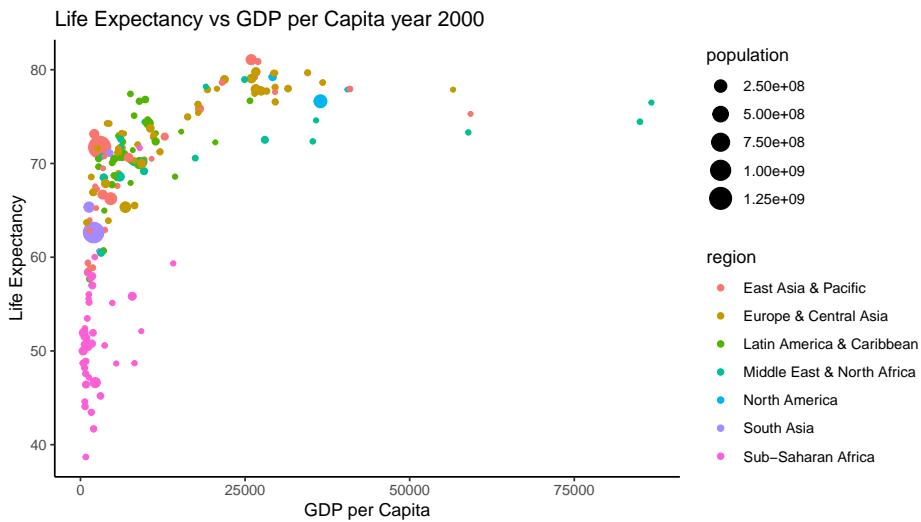


9.8 Add the chart title

```
p + geom_point(na.rm = TRUE) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") # adds a title
```



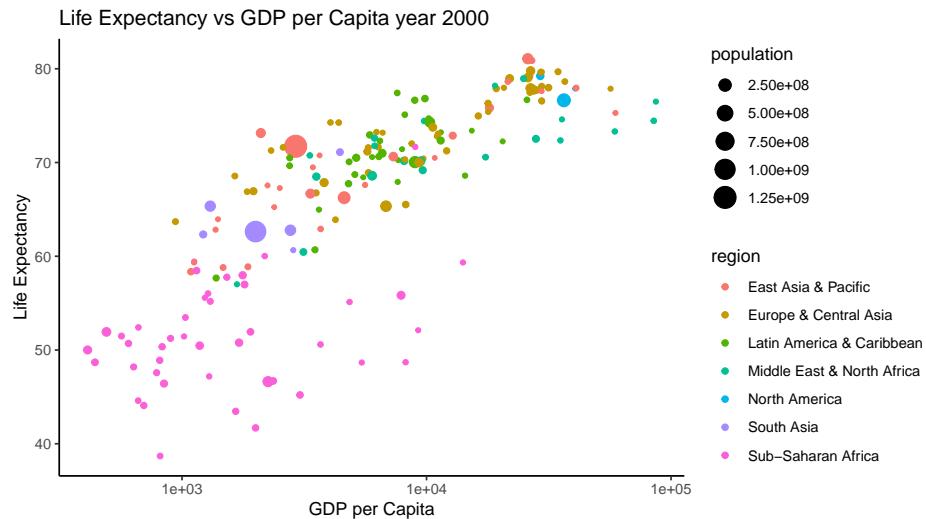
```
p + geom_point(na.rm = TRUE) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") + # adds a title
  xlab("GDP per Capita") +
  ylab("Life Expectancy")
```



9.10 Axis scale

Since many countries have a low GDP compared to the richest few countries, many of the data points are bunched up on the left of the chart. Using a log scale on the x-axis, will allow us to spread the data more evenly across the chart.

```
p + geom_point(na.rm = TRUE) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") + # adds a title
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10()
```



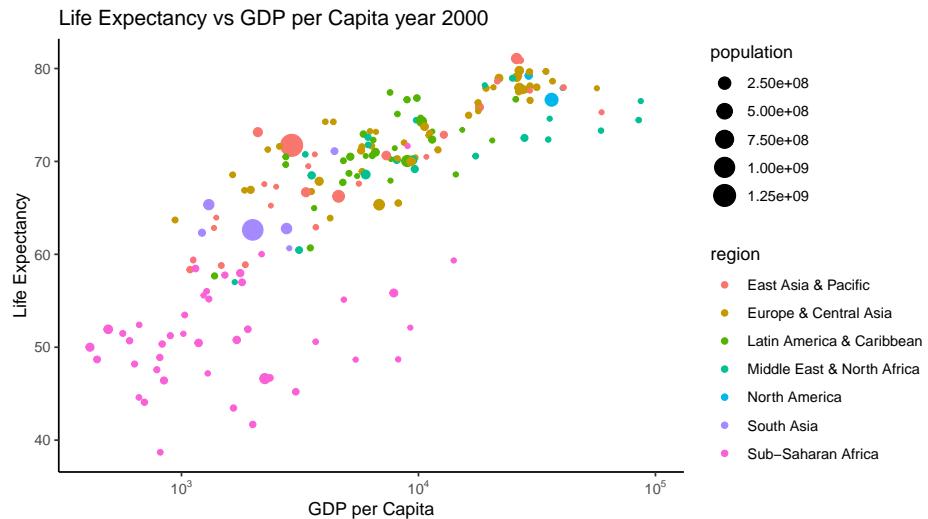
You may prefer to show the x-axis values in log 10 format. To do that you can specify the label format usng `trans_format` function. Look up the R documentation for more information on this function

At the point you can decide the upper and lower limits of the x-axis. If you omit this, R will decide these for you. However, there may be cases when you want to show the full extent of the x (and y axis).

Typically, this is done using the `xlim` and `ylim` functions, but when you are using `scale_x_log10` to scale the axis, you specify the axis limits as an argument to this function

```
library(scales)

p + geom_point(na.rm = TRUE) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") + # adds a title
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5))
```



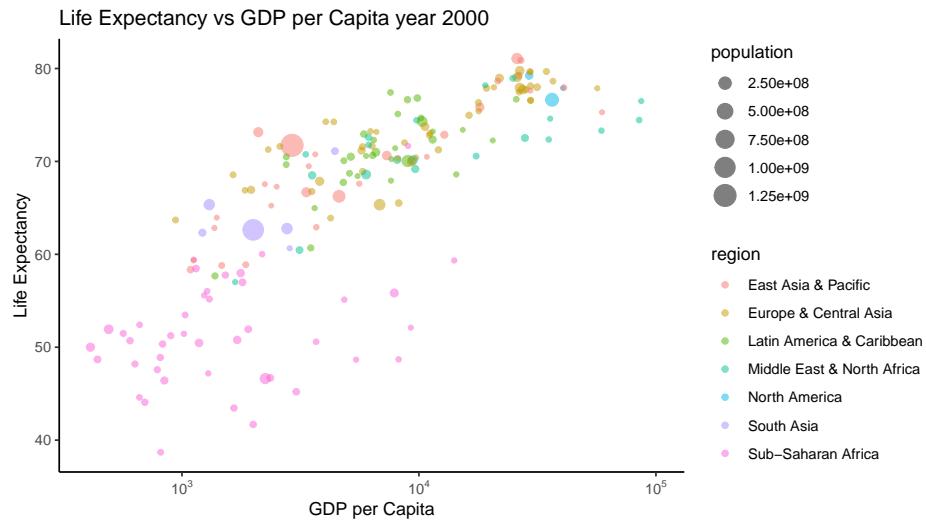
9.11 Colour and overlap

There are two issues we should deal with here

- There is a lot of overlap between points. In this situation, we should show some degree of transparency so it is clear where points overlap
- The default colour allocation is somewhat gaudy, so we should choose a more appropriate palette suited to the categorical data of the `region` field

Firstly, setting transparency using the alpha field of geopoint

```
p + geom_point(na.rm = TRUE, alpha=0.5) + # alpha (transparency) set to 0.5
  ggtitle("Life Expectancy vs GDP per Capita year 2000") + # adds a title
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5))
```



Selecting a palette

```
### Load the RcolorBrewer package or install if not present
##if (!require("RColorBrewer")) {
#install.packages("RColorBrewer")
#}

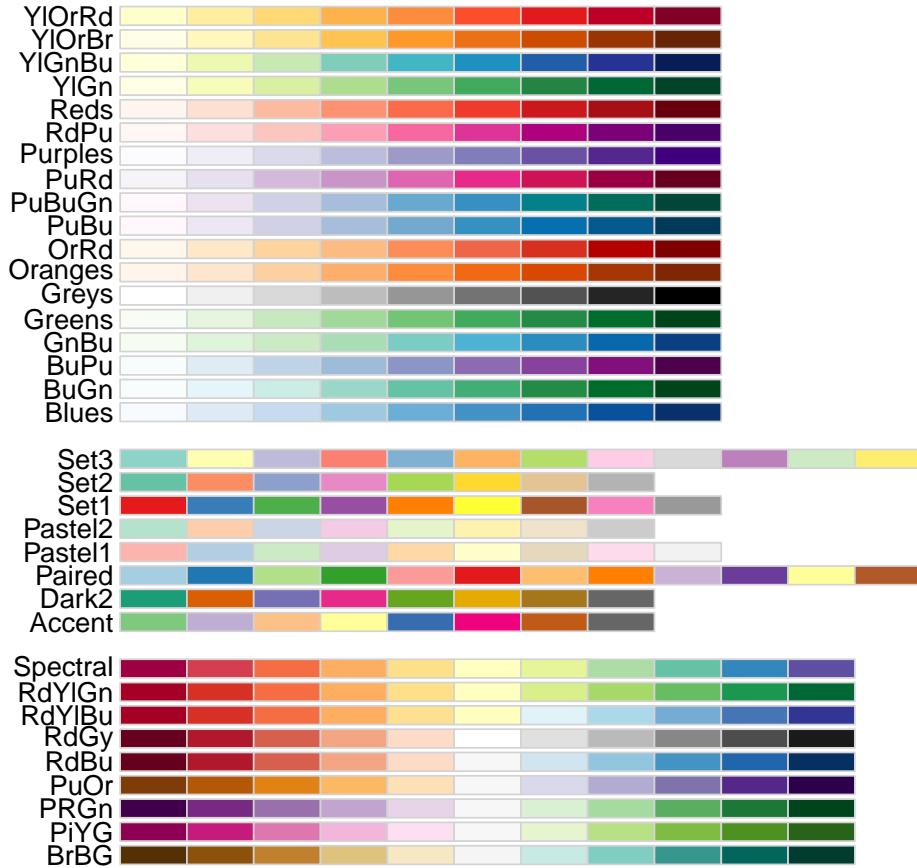
library(RColorBrewer)
```

RColorBrewer library has 3 groups of palettes

- **Sequential** : *Sequential palettes are suited to ordered data that progress from low to high.* Lightness steps dominate the look of these schemes, with light colors for low data values to dark colors for high data values.
- **Qualitative** : do not imply magnitude differences between legend classes, and hues are used to create the maximum visual differences between classes. *Qualitative schemes are best suited to representing nominal or categorical data.*
- **Diverging** : Diverging palettes put equal emphasis on mid-range critical values and extremes at both ends of the data range. The critical class or break in the middle of the legend is emphasized with light colors and low and high extremes are emphasized with dark colors that have contrasting hues. *In short, Light colours for mid-range data, low and high contrasting dark colours*

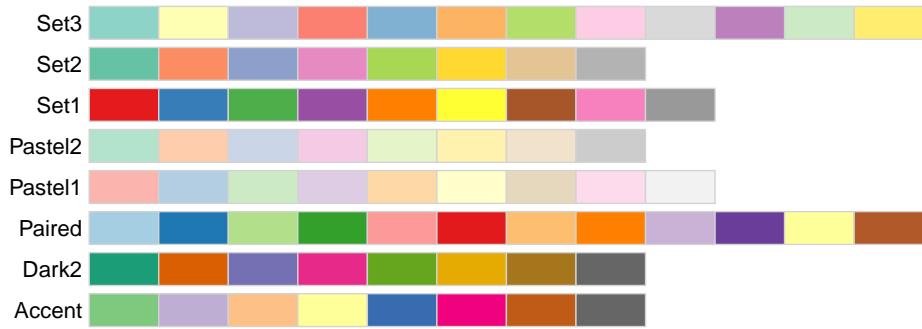
You can view the 3 groups by calling the following function

```
display.brewer.all()
```



As we are colouring the data points by `region`, which is a categorical variable, we should select one of the qualitative palettes. My preference is for “dark2”, which has 8 colours - which is a good fit for the set of values (7) of the `region` variable

```
display.brewer.all(type="qual") # just displays the qualitative palettes
```



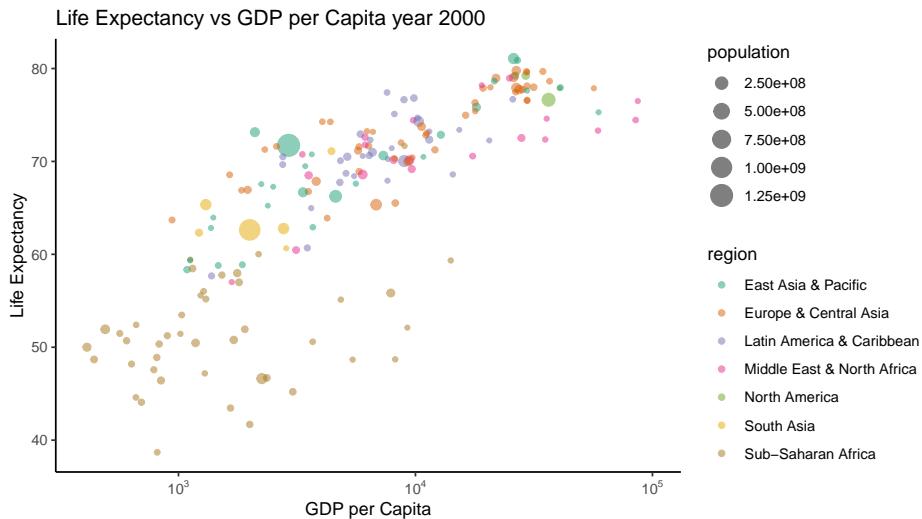
```
display.brewer.pal(7,"Dark2") # displays the Dark2 palette for 7 colours
```



Dark2 (qualitative)

We now add this palette to our plot

```
p + geom_point(na.rm = TRUE, alpha=0.5) + # alpha (transparency) set to 0.5
  ggtitle("Life Expectancy vs GDP per Capita year 2000") + # adds a title
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5))
  scale_color_brewer(palette='Dark2') # this sets the colour palette to 'Dark2'
```



9.12 Refining Point Area

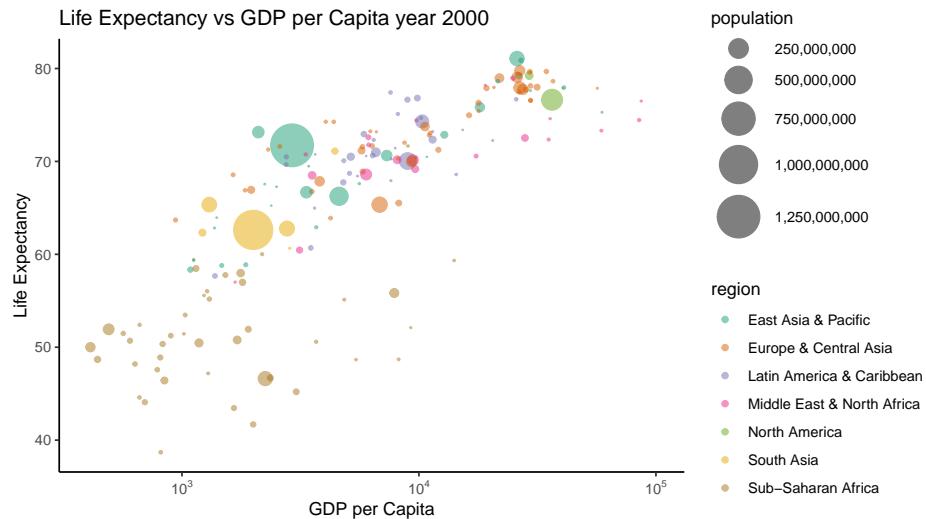
The default area of the data points does not visually reflect the comparative population sizes of various countries. As such we will use ‘`scale_size_area`’ to directly scale the area of each data point according to its population value.

This function can take `max_size` and `labels` attributes. `max_size` indicates the maximum area size of the largest data point. The smaller data points are scaled from this maximum value.

Selecting the `max_size` value is something you do by trial and error

The `labels` attribute refers to the numeric labels of the legend. Currently, it is displaying numbers in scientific notation. For more general consumption, we could set this to show numbers where units are separated by commas

```
p + geom_point(na.rm = TRUE, alpha=0.5) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") +
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5)) +
  scale_color_brewer(palette='Dark2') +
  scale_size_area(max_size=12, labels = comma)
```



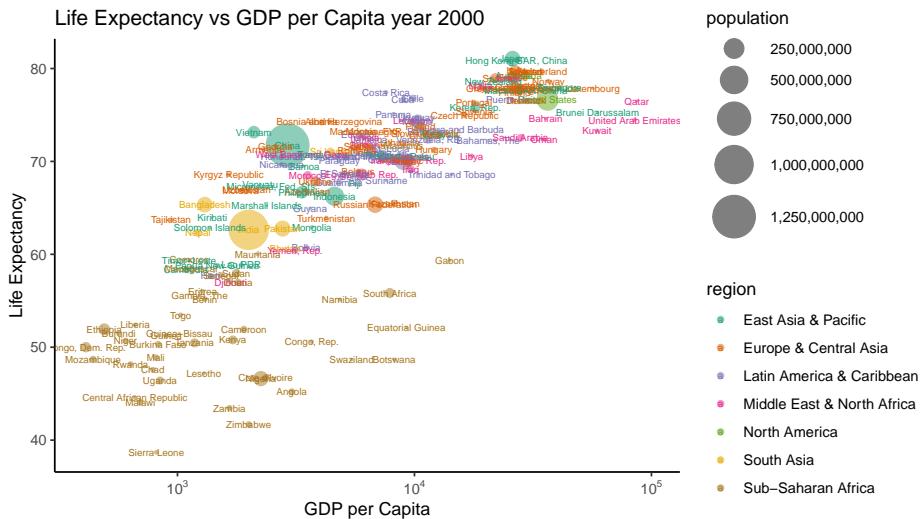
9.13 Labelling Data points

Examine the current state of the chart. What information does it communicate?

- The most dramatic piece of information is the link between low life expectancy and low GDP for countries in Sub-Saharan Africa.
- We can see relatively high life expectancy for countries in North America and East Asia - but which countries?

We can label countries using the `geom_text` function and specifying the value to display for each data point, as well as its size, and vertical and horizontal offset from the point. In this case, we want the label to display the value of the `country` feature for each point

```
p + geom_point(na.rm = TRUE, alpha=0.5) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") +
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5)) +
  scale_color_brewer(palette='Dark2') +
  scale_size_area(max_size=12, labels = comma) + # scale each point according to its population
  geom_text(aes(label=country), size = 2,na.rm = TRUE)
```



Even with a small font size, which is hard to read, the result is a label hair ball - and many of the benefits of the visualisation have been lost.

9.13.1 What labels to display?

This depends entirely on your audience - and their objectives in reading information from a data plot.

A group of senior economists at the world bank will have different requirements than the readers of a daily news paper.

In any case, the message is that **we cannot label everything point on the plot**. We need to label those points that help **orient the reader toward the key pieces of information being communicated**.

We need to use our knowledge of the domain to make those labelling decisions.

Method 1:

manual annotation by specifying the precise location of a label using the x, y coordinate values from the plot. This gives you precise control over the location of each label but it is not very scalable.

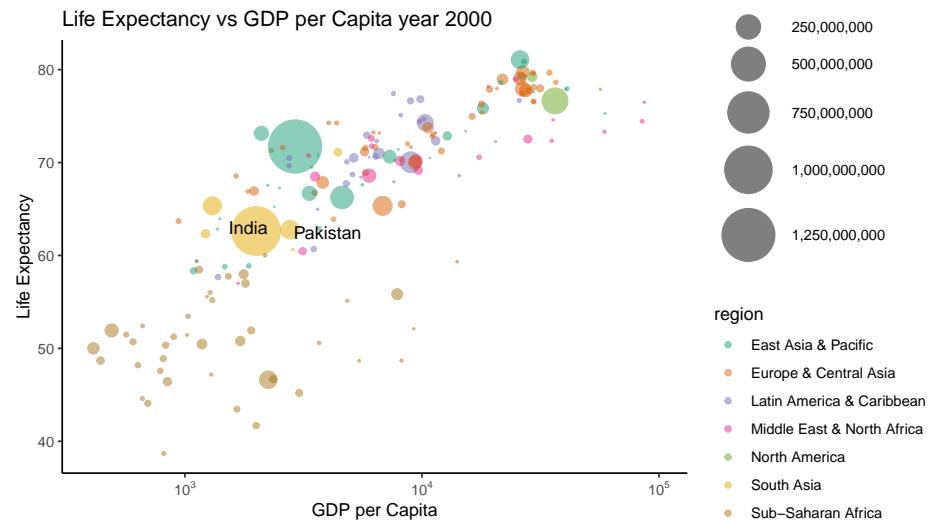
For this example we will label just India and Pakistan. To do that, we get the life_expect, gdp_per_cap values for each country.

```
nations_2000[nations_2000$country == 'India' | nations_2000$country == 'Pakistan', c('country', 'life
```

```
## # A tibble: 2 x 3
##   country  life_expect  gdp_per_cap
##   <chr>        <dbl>      <dbl>
## 1 India          62.6     1998.
## 2 Pakistan       62.8     2776.
```

To demonstrate this approach, I will annotate the plot using the `annotate` function

```
p + geom_point(na.rm = TRUE, alpha=0.5) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") +
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5)) +
  scale_color_brewer(palette='Dark2') +
  scale_size_area(max_size=15, labels = comma) +
  annotate("text", x=1850, y= 63, label="India") + # postioning by trial and error
  annotate("text", x=4000, y= 62.5 , label="Pakistan") # postioning by trial and error
```



9.13.2 How to selectively display labels

Method 2: use a function to define the most interesting data points and display only labels for these data points. Using this method, you can define the values that may be of interest to your readership - which may include you, of course.

For this plot, I am going to define the following criteria of ‘interestingness’ :

- population >120000000
- life_expect < 44
- gdp_percap < 450
- country ==Ireland

The more criteria you add or the wider your criteria, the more labels you will include - and you may lose readability. Defining criteria is based on an understanding of the domain and the constraints of the visualisation. It requires some

trial and error.

Using `ifelse` statement to encode these rules, we create a new field called `label_country` which we populate with the country names that fulfil the rule criteria. We then set these values as the label aesthetic for `geom_text`.

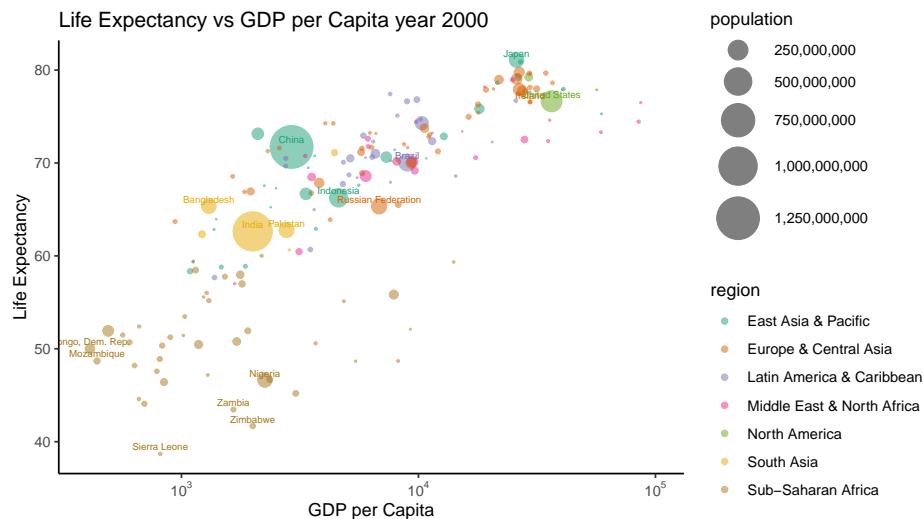
```
# Filter required rows.
```

```
nations_2000$label_country <- ifelse(nations_2000$population > 120000000 | nations_2000$life_exp
```

The full plot code is here:

```
p + geom_point(na.rm = TRUE, alpha=0.5) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") +
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5)) +
  scale_color_brewer(palette='Dark2') +
  scale_size_area(max_size=12, labels = comma) + # scale each point according to its population value

#Using geom_text to add text labels
geom_text(data= nations_2000,aes(label=label_country), vjust=-0.5, size=2, na.rm = TRUE, show.legend=FALSE)
```



Note the additional attributes used in `geom_text`

- `vjust` refers to the vertical offset of the label;
- `x` refers to its x positioning. If the x-axis were scaled uniformly, the x position of the label would be the x position of the data point +/- a

number (e.g `x = gdp_percap + 55`)). However, as the axis has a log scale, the x offset has to be based on a multiplicative function

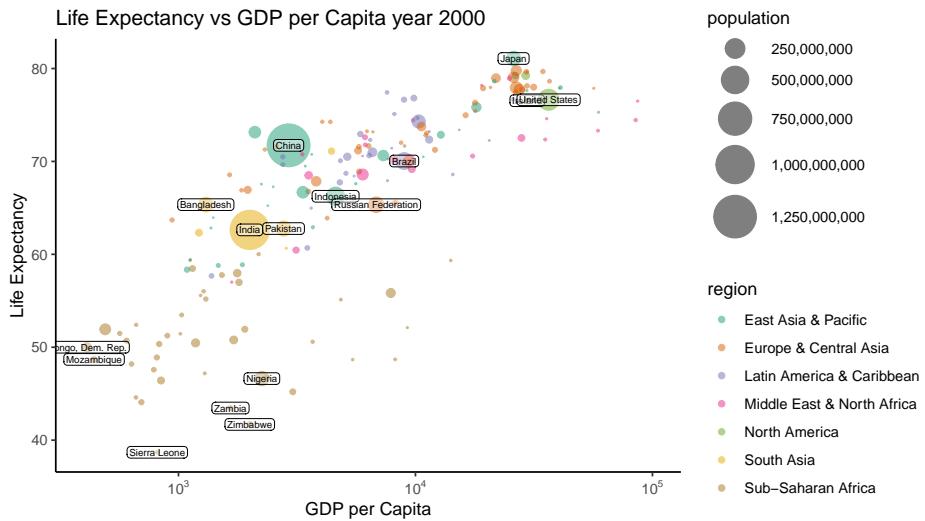
- `size` the font size of the label
- `na.rm = TRUE` the standard command to ignore rows with missing values
- `show.legend = FALSE` unless we include this another legend based on the text labels will be overlaid on the color legend

Note that this automated approach to label generation has problems. You do not have full control over where the labels will be placed - and in this case, you can see that the labels for India and Pakistan clash. However, the benefit is that you can easily run this plot script for different years of analysis and quickly produce a series of labelled plots for comparison purposes.

9.13.3 Labels that repel each other

The user of `geom_label` is an alternative labelling solution to `geom_text`. By default each label is enclosed in a label area.

```
p + geom_point(na.rm = TRUE, alpha=0.5) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") +
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5))
  scale_color_brewer(palette='Dark2') +
  scale_size_area(max_size=12, labels = comma) + # scale each point according to its population
  #Using geom_text to add text labels
  geom_label(data= nations_2000,aes(label=label_country), size= 2, label.padding = unit
```



However, we still have a problem with overlapping labels. We can allow ggplot solve this by using the `ggrepel` library and the `geom_label_repel` or `geom_text_repel` functions.

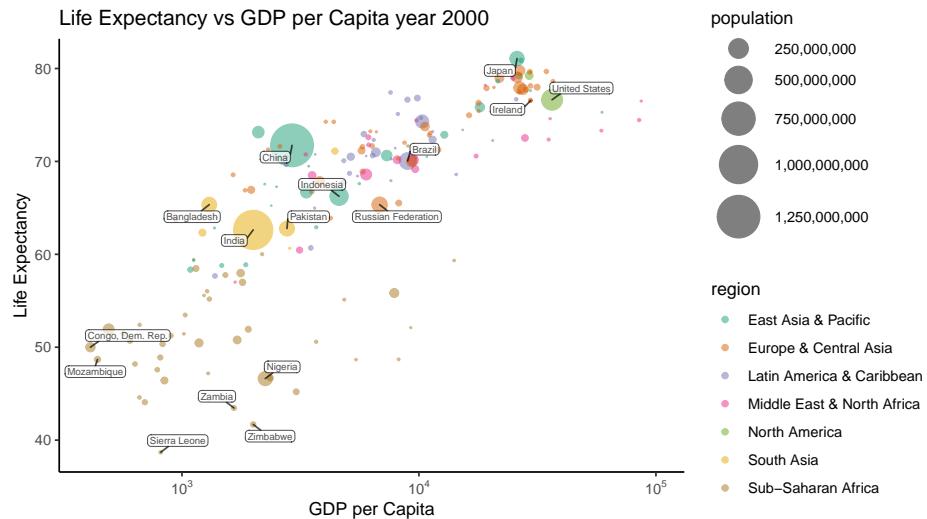
`geom_label_repel` will add a line between the label and the point according to the value of the `min.segment.length` attribute.

If you set this value to be zero, then it will add a line between each point and its label.

```
library(ggrepel)

p + geom_point(na.rm = TRUE, alpha=0.5) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") +
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5)) +
  scale_color_brewer(palette='Dark2') +
  scale_size_area(max_size=12, labels = comma) + # scale each point according to its population value

#Using geom_text to add text labels
geom_label_repel(data= nations_2000,aes(label=label_country), min.segment.length = unit(0, 'lines'))
```



The final step in this visualisation are minor tweaks

- Change the default ‘grey’ theme using a theme
- Increase the size of the points of the colour legend using the guides function

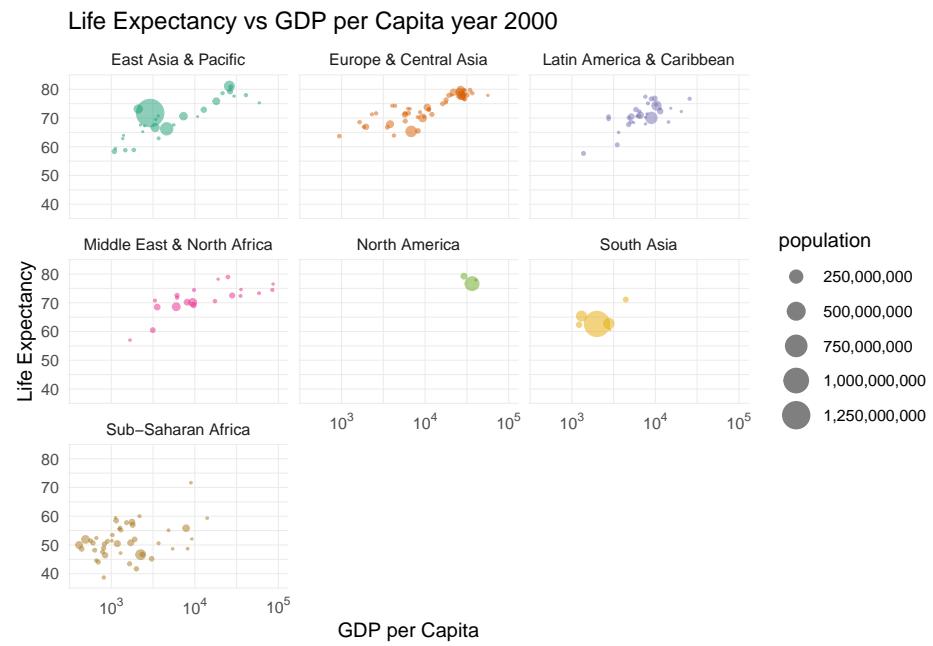
```
library(ggrepel)

pp<-p + geom_point(na.rm = TRUE, alpha=0.5) +
  ggtitle("Life Expectancy vs GDP per Capita year 2000") +
  xlab("GDP per Capita") +
  ylab("Life Expectancy") +
  scale_x_log10(labels = trans_format("log10", math_format(10^.x)), limits = c(400,1e5)) +
  scale_y_continuous(breaks=c(20,30,40,50,60,70,80,90), limits=c(35,85), expand=c(0,0)) +
  scale_color_brewer(palette='Dark2') +
  scale_size_area(max_size=12, labels = comma) + # scale each point according to its pop

  #Using geom_text to add text labels
  geom_label_repel(data= nations_2000,aes(label=label_country), size= 2, min.segment.length= 10) +
  theme_minimal(base_size = 10) +
  theme(panel.grid.minor = element_blank() ) +
  # increase the size of the legend colour points
  guides(colour = guide_legend(override.aes = list(size = 5)))
```

pp





Chapter 10

Visualising Trends

10.1 Representing Trends

Often we are more interested in the overarching trend of the data than in the specific detail of where each individual data point lies.

By drawing the trend on top of or instead of the actual data points, usually in the form of a straight or curved line, we can create a visualization that helps the reader immediately see key features of the data.

```
library(tidyverse)
library(lubridate)

iseq_all <- read_csv("../data/ISEQ All Share_quote_chart.csv") %>%
  mutate(
    date = as.Date(dmy_hm(time)),
    close= `ISEQ All Share`
  ) %>%
  select(date,close )

## Parsed with column specification:
## cols(
##   time = col_character(),
##   `ISEQ All Share` = col_double(),
##   volume = col_double()
## )
```

Daily closing values of the ISEQ for the year up to Wednesday, February 26th, 2020. Data source



Figure 10.1: Dow Jones Industrial Index

```

startdate<-"31-12-2018"
enddate<-"26-02-2020"

iseq_all_1920<- iseq_all %>% filter(date > dmy(startdate) & date <= dmy(enddate))

ggplot(iseq_all_1920, aes(date, close)) +
  geom_line(color = "grey20", size = .5) +
  scale_x_date(limits = c(dmy(startdate), dmy(enddate)), expand = c(0, 0), date_breaks =
  xlab(NULL) + ylab("ISEQ Closing Prices") +
  theme_minimal()+
  theme(
    plot.margin = margin(3, 12, 3, 1.5),
    panel.grid.minor.x = element_blank(),
    panel.grid.minor.y = element_blank()
)
  
```

10.2 Moving Average

Smoothing produces a function that captures key patterns in the data while removing irrelevant minor detail or noise.

To generate a moving average, we take a time window, say the first 20 days in the time series, calculate the average price over these 20 days, then move the time window by one day, so it now spans the 2nd to 21st day, move the time window again, and so on.

To plot this sequence of moving averages, we need to decide which specific time point to associate with the average for each time window.

We can plot the average at the center of the time window, which results in a curve that overlays perfectly on the original data

```
library(knitr)
library(kableExtra)
library(e1071)

source("moving_ave.R", echo = T)

## 
## > moving_ave <- function(date, value, range, center = TRUE) {
## +   if (isTRUE(center)) {
## +     offset <- ceiling(range/2)
## +   }
## +   else {
## +     .... [TRUNCATED]

iseq_all_1920_mov_ave <- iseq_all %>% filter(date > dmy(startdate) & date <= dmy(enddate)) %>%
  mutate(
    close_20d_ave = moving_ave(date, close, 20, center = TRUE),
    close_50d_ave = moving_ave(date, close, 50, center = TRUE),
    close_100d_ave = moving_ave(date, close, 100, center = TRUE)
  )

# kurtosis values for orginal data vs smoothed data

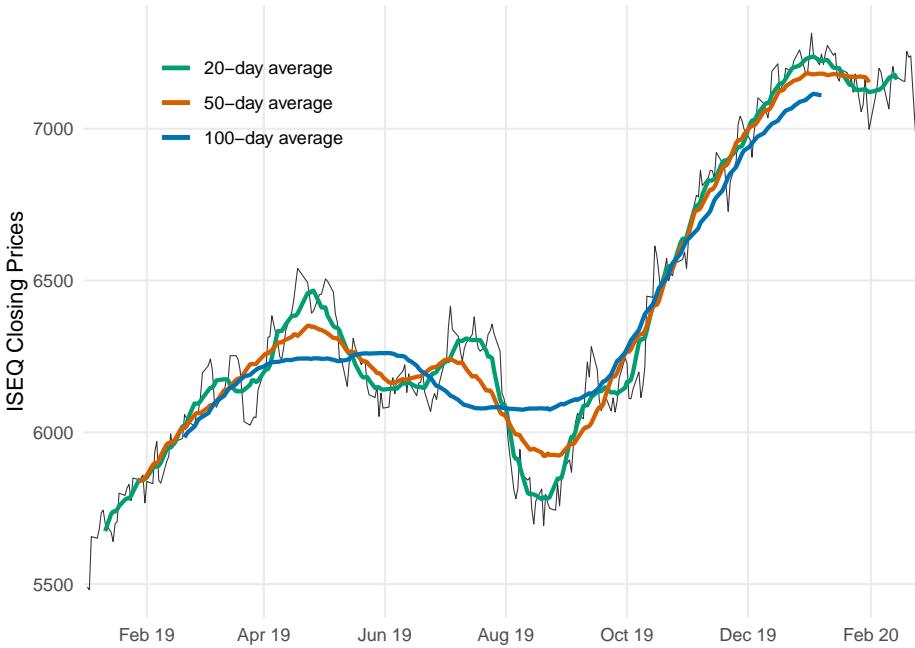
k0<-kurtosis(iseq_all_1920_mov_ave$close)
k20<-kurtosis(iseq_all_1920_mov_ave$close_20d_ave, na.rm=TRUE)
k50<-kurtosis(iseq_all_1920_mov_ave$close_50d_ave,na.rm=TRUE)
k100<-kurtosis(iseq_all_1920_mov_ave$close_100d_ave,na.rm=TRUE)

k<- data.frame("win" = c(0,20,50,100), "kurtosis" = c(k0,k20,k50,k100))
```

```
kable(k) %>%
  kable_styling( full_width = F)
```

win	kurtosis
0	-0.9308360
20	-0.8397314
50	-0.5764831
100	0.1269679

```
ggplot(iseq_all_1920_mov_ave, aes(date, close)) +
  geom_line(color = "grey20", size = .25) +
  geom_line(aes(date, close_20d_ave, color = "20d"), size = 1, na.rm = TRUE) +
  geom_line(aes(date, close_50d_ave, color = "50d"), size = 1, na.rm = TRUE) +
  geom_line(aes(date, close_100d_ave, color = "100d"), size = 1, na.rm = TRUE) +
  scale_color_manual(
    values = c(
      `20d` = "#009e73",
      `50d` = "#d55e00",
      `100d` = "#0072b2"
    ),
    breaks = c("20d", "50d", "100d"),
    labels = c("20-day average", "50-day average", "100-day average"),
    name = NULL
  ) +
  scale_x_date(limits = c(dmy(startdate), dmy(enddate)), expand = c(0, 0), date_breaks =
  xlab(NULL) + ylab("ISEQ Closing Prices") +
  theme_minimal() +
  theme(
    plot.margin = margin(3, 12, 3, 1.5),
    panel.grid.minor.x = element_blank(),
    panel.grid.minor.y = element_blank(),
    legend.position = c(0.2, 0.85)
  )
)
```



The moving average is the most simplistic approach to smoothing, and it has some obvious limitations. First, it results in a smoothed curve that is shorter than the original curve. Parts are missing at either the beginning or the end or both.

Second, even with a large averaging window, a moving average is not necessarily that smooth.

10.3 Loess Smoother

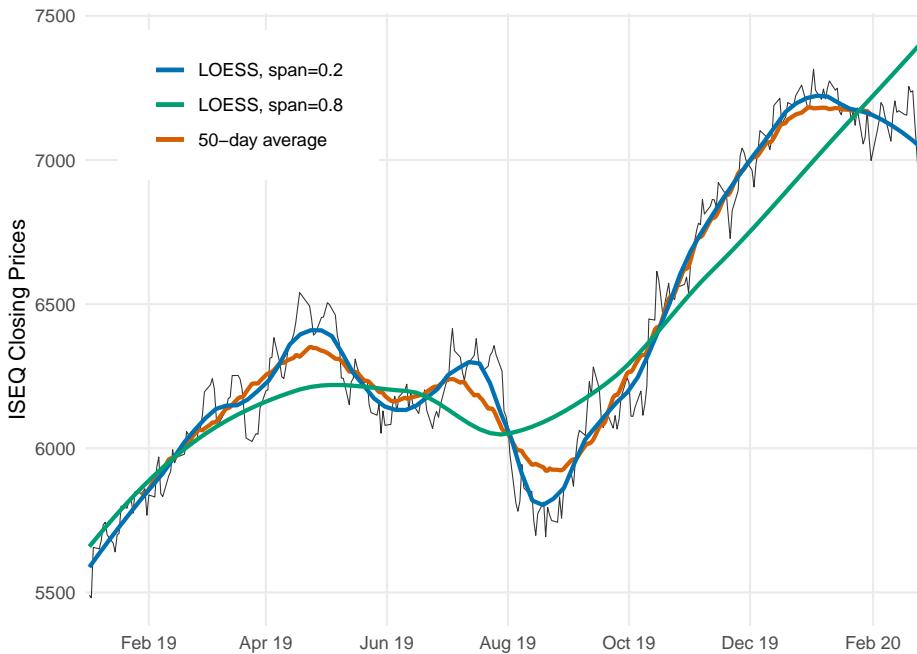
One widely used method is LOESS (locally estimated scatterplot smoothing, W. S. Cleveland (1979)), which fits low-degree polynomials to subsets of the data.

```
loess_span1 = 0.2
loess_span2 = 0.8

iseq_all_1920 %>%
  mutate(
    close_50d_ave = moving_ave(date, close, 50)
  ) %>%
  ggplot(aes(date, close)) +
  geom_line(color = "grey20", size = .25) +
  geom_line(aes(date, close_50d_ave, color = "100d"), size = 1, na.rm = TRUE) +
  geom_smooth(aes(color = "smooth1"), method="loess", span= loess_span1, size = 1, na.rm = TRUE)
```

```
geom_smooth(aes(color = "smooth2"), method="loess", span= loess_span2, size = 1, na.rm = TRUE)

scale_color_manual(
  values = c(
    `100d` = "#d55e00",
    smooth1 = "#0072b2",
    smooth2 = "#009E73"
  ),
  breaks = c("smooth1", "smooth2", "100d"),
  labels = c(paste0("LOESS, span=", loess_span1), paste0("LOESS, span=", loess_span2),
             name = NULL
  ) +
  scale_x_date(limits = c(dmy(startdate), dmy(enddate)), expand = c(0, 0), date_breaks = "1 year",
               date_labels = "%Y-%m"),
  xlab(NULL) + ylab("ISEQ Closing Prices") +
  theme_minimal() +
  theme(
    legend.justification = c(1, 0.5),
    legend.box.background = element_rect(fill = "white", color = NA),
    legend.box.margin = margin(0, 12, 6, 12),
    plot.margin = margin(3, 12, 3, 1.5),
    panel.grid.minor.x = element_blank(),
    panel.grid.minor.y = element_blank(),
    legend.position = c(0.35,0.85)
  )
)
```



10.4 4 Irish Tech Stocks

Lets look at stock prices for 4 successful Irish companies trading on the Irish stock market

```
# assuming company is ordered alphabetically
company.colours<- c("firebrick4", "#009E73", "#E69F00", "#56B4E9")

iseq_4companies <- read_csv("../data/IRE_ISEQ_Companies.csv") %>%
  mutate(
    date = as.Date(dmy_hm(time)),
    close= Close,
    company = Company
  ) %>%
  select(date,company, close)

## Parsed with column specification:
## cols(
##   time = col_character(),
##   Close = col_double(),
##   Volume = col_double(),
##   Company = col_character()
## )
```

```

lastdate<-max(iseq_4companies$date)

iseq_4companies_final <- filter(iseq_4companies, date == lastdate)

# if you are colouring the axis text, the colurts need to be in the same order that the
axis.color.order <- company.colours[order(iseq_4companies_final$close)]

ggplot(iseq_4companies, aes(date, close, colour = company)) +
  geom_line( size = .5, alpha= 0.6) +
  geom_smooth(aes(group = company), colour = "black", alpha = 0.2, method="loess", span = 3)

  scale_color_manual(values=company.colours,
                     name="",
                     breaks=c("AIB", "APPLEGREEN", "GLANBIA", "RYANAIR"),
                     labels=c("AIB", "Applegreen", "Glanbia", "Ryanair")) +

  scale_y_continuous(name="ISEQ closing share prices",
                     limits = c(0, 20),
                     expand=c(0,0),
                     sec.axis = dup_axis(
                     breaks = iseq_4companies_final$close,
                     labels = iseq_4companies_final$company,
                     name = NULL)) +

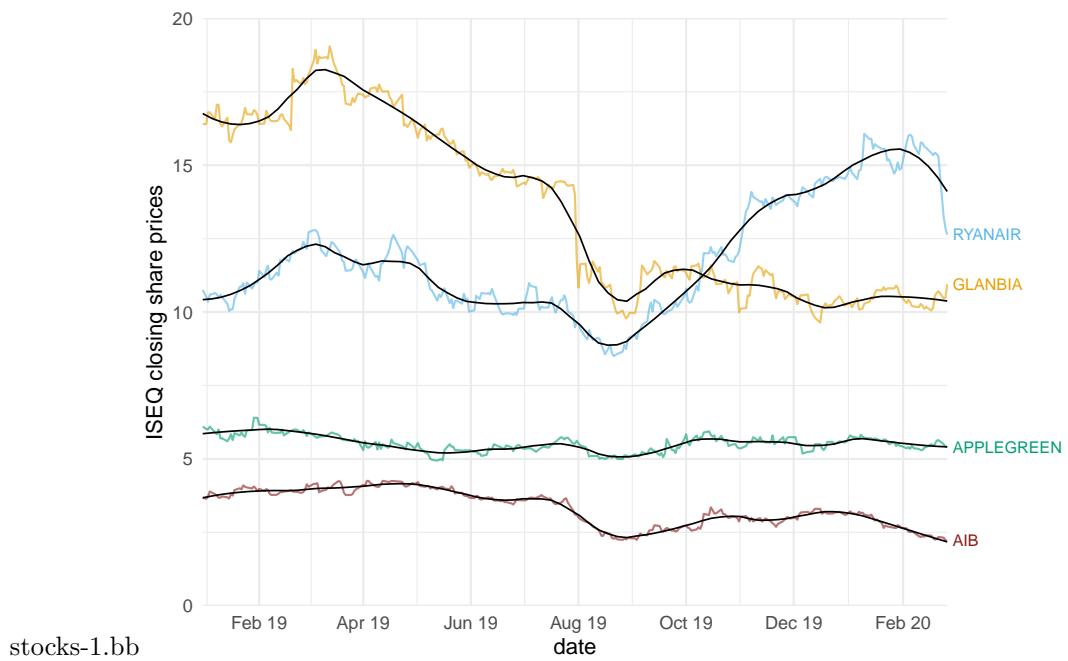
  scale_x_date(limits = c(dmy(startdate), dmy(enddate)), date_breaks = "2 month", date_labels = "%Y-%m")

  theme_minimal()+
  theme(
    legend.position = "None",
    plot.margin = margin(3, 12, 3, 1.5),
    axis.line.y.right = element_blank(),
    axis.ticks.y.right = element_blank(),
    axis.text.y.right = element_text(margin = margin(0, 0, 0, 0) ,size = 8, colour = "black"))

)

## Warning: Removed 1967 rows containing missing values (geom_path).

```



Chapter 11

UK Election Polls Case Study

11.1 Read in the data

Thank you to Daniel Levitt of The Guardian newspaper who provided me with this data set.

```
library(readr)
library(ggplot2)
library(dplyr)
library(tidyr)
library(lubridate)
library(scales)
library(kableExtra)

ukpoll <- read_csv("../data/UKPoll-2020-continuous-series-1.csv")

kable(ukpoll,
      digits = 2,
      format = "html",
      row.names = TRUE) %>%
  kable_styling(
    bootstrap_options = c("striped"),
    full_width = F,
    font_size = 12
  ) %>%
  column_spec(column = 3, width = "8em") %>%
  scroll_box(height = "400px")
```

date

pollster

con

lab

ldem

grn

brx

oth

1

08/06/2017

2017 general election

43

41

8

2

NA

6

2

21/06/2017

Panelbase

41

46

6

1

NA

7

3

29/06/2017

Opinium

39

45

5
2
NA
9
4
01/07/2017
YouGov
41
44
7
2
NA
6
5
06/07/2017
YouGov
38
46
6
1
NA
9
6
11/07/2017
YouGov
40
45
7
1
NA
6

7

14/07/2017

Opinium

41

43

5

2

NA

8

8

16/07/2017

ICM

42

43

7

2

NA

7

9

18/07/2017

Ipsos MORI

41

42

9

2

NA

7

10

19/07/2017

YouGov

41

43
6
2
NA
7
11
11/08/2017
BMG
42
39
7
3
NA
8
12
18/08/2017
Opinium
40
43
6
2
NA
10
13
22/08/2017
YouGov
41
42
8
1
NA

8

14

28/08/2017

ICM

42

42

7

3

NA

5

15

31/08/2017

YouGov

41

42

6

2

NA

8

16

10/09/2017

ICM

42

42

7

3

NA

7

17

13/09/2017

YouGov

41
42
7
2
NA
8
18
15/09/2017
BMG
39
38
8
4
NA
10
19
15/09/2017
Opinium
41
41
5
3
NA
10
20
18/09/2017
Ipsos MORI
40
44
9
1

NA

6

21

22/09/2017

Opinium

42

40

6

2

NA

10

22

24/09/2017

ICM

40

42

8

2

NA

9

23

24/09/2017

YouGov

39

43

7

2

NA

8

24

29/09/2017

BMG

37

42

10

3

NA

8

25

01/10/2017

Ipsos MORI

38

40

9

3

NA

10

26

05/10/2017

YouGov

40

42

7

2

NA

9

27

06/10/2017

Opinium

40

42

5

2
NA
10
28
08/10/2017
ICM
41
41
7
2
NA
9
29
11/10/2017
YouGov
39
42
8
2
NA
8
30
19/10/2017
YouGov
40
42
8
2
NA
9
31

23/10/2017

ICM

42

42

7

2

NA

6

32

24/10/2017

YouGov

41

43

7

2

NA

7

33

01/11/2017

ICM

40

41

8

3

NA

7

34

08/11/2017

YouGov

40

43

6
2
NA
9
35
12/11/2017
ICM
41
41
7
2
NA
8
36
16/11/2017
Opinium
40
42
6
2
NA
10
37
20/11/2017
YouGov
40
43
7
2
NA
7

38

20/11/2017

Kantar

42

38

9

3

NA

8

39

23/11/2017

YouGov

39

41

7

3

NA

9

40

26/11/2017

ICM

41

41

7

3

NA

8

41

28/11/2017

Ipsos MORI

37

39

9

4

NA

10

42

05/12/2017

YouGov

40

41

7

2

NA

9

43

08/12/2017

BMG

37

40

9

3

NA

10

44

10/12/2017

ICM

42

40

8

2

NA

9
45
11/12/2017

YouGov

42
41
7
2
NA
10
46
14/12/2017

ICM

41
42
7
3
NA
7
47
14/12/2017

Opinium

39
41
7
2
NA
12
48
20/12/2017

YouGov

40

42

7

1

NA

9

49

08/01/2018

YouGov

40

41

9

2

NA

7

50

12/01/2018

Opinium

40

40

6

3

NA

10

51

12/01/2018

BMG

40

41

8

2

NA

9

52

14/01/2018

ICM

40

41

7

3

NA

8

53

17/01/2018

YouGov

41

42

7

2

NA

7

54

19/01/2018

ICM

41

41

7

3

NA

7

55

23/01/2018

Ipsos MORI

39

42

9

2

NA

8

56

29/01/2018

YouGov

42

42

6

3

NA

7

57

04/02/2018

ICM

41

40

8

3

NA

8

58

06/02/2018

YouGov

43

39

8

3
NA
7
59
08/02/2018
Opinium
42
39
7
2
NA
11
60
09/02/2018
BMG
40
40
8
4
NA
8
61
12/02/2018
Kantar
39
39
8
2
NA
11
62

13/02/2018

YouGov

40

41

8

2

NA

8

63

19/02/2018

ICM

42

43

7

2

NA

6

64

20/02/2018

YouGov

40

42

8

2

NA

8

65

27/02/2018

YouGov

41

42

7
2
NA
8
66
04/03/2018
ICM
43
42
7
3
NA
5
67
06/03/2018
YouGov
41
43
7
2
NA
7
68
07/03/2018
Ipsos MORI
43
42
6
2
NA
7

69

15/03/2018

YouGov

42

39

7

3

NA

9

70

15/03/2018

Opinium

42

40

6

3

NA

9

71

16/03/2018

BMG

38

40

10

3

NA

9

72

18/03/2018

ICM

44

41
8
2
NA
6
73
27/03/2018
YouGov
43
39
8
2
NA
7
74
01/04/2018
YouGov
42
38
7
3
NA
8
75
05/04/2018
YouGov
42
41
7
2
NA

9

76

08/04/2018

ICM

42

41

7

3

NA

7

77

10/04/2018

YouGov

40

40

9

2

NA

9

78

12/04/2018

ComRes

40

41

7

2

NA

9

79

12/04/2018

Opinium

40
40
7
2
NA
11
80
13/04/2018
BMG
39
38
11
4
NA
8
81
17/04/2018
YouGov
43
38
8
3
NA
8
82
24/04/2018
Ipsos MORI
41
40
10
2

NA

7

83

25/04/2018

YouGov

43

38

8

3

NA

7

84

29/04/2018

ComRes

40

40

9

3

NA

9

85

29/04/2018

ICM

42

39

8

3

NA

8

86

01/05/2018

Deltapoll

41

41

6

2

NA

10

87

04/05/2018

BMG

39

39

10

3

NA

10

88

09/05/2018

YouGov

43

38

9

2

NA

7

89

13/05/2018

ICM

43

40

8

3

NA

7

90

14/05/2018

YouGov

43

38

9

3

NA

7

91

16/05/2018

Opinium

43

39

6

3

NA

10

92

21/05/2018

YouGov

42

38

9

3

NA

7

93

22/05/2018

Ipsos MORI

40

40

7

5

NA

7

94

29/05/2018

YouGov

42

39

9

2

NA

9

95

29/05/2018

ICM

43

40

8

2

NA

8

96

05/06/2018

YouGov

44

37

8

3

NA

7

97

07/06/2018

Opinium

42

40

7

2

NA

11

98

08/06/2018

BMG

38

41

11

2

NA

7

99

10/06/2018

ICM

42

40

8

3

NA

6

100

12/06/2018

YouGov

42

39

8

2

NA

8

101

19/06/2018

YouGov

42

40

9

2

NA

7

102

24/06/2018

ICM

41

40

9

3

NA

8

103

26/06/2018

YouGov

42

37

9

3

NA

9

104

27/06/2018

Ipsos MORI

41

38

7

4

NA

10

105

04/07/2018

YouGov

41

40

9

2

NA

8

106

05/07/2018

BMG

39

37

10

4

NA

8

107

09/07/2018

YouGov

39

39

9

3

NA

10

108

09/07/2018

ICM

41

39

9

3

NA

8

109

09/07/2018

Kantar

40

38

9

3

NA

10

110

11/07/2018

YouGov

37

39

10

3

NA

11

111

13/07/2018

Opinium

36

40

8

3

NA

14

112

14/07/2018

Deltapoll

37

42

7

3

NA

11

113

17/07/2018

YouGov

36

41

9

2

NA

11

114

20/07/2018

YouGov

38

39

9

2

NA

12

115

22/07/2018

ICM

40

41

8

3

NA

9

116

23/07/2018

YouGov

38

38

10

3

NA

10

117

24/07/2018

Ipsos MORI

38

38

10

3

NA

12

118

31/07/2018

YouGov

38

38

10

3

NA

11

119

05/08/2018

ICM

39

40

7

3

NA

10

120

09/08/2018

YouGov

39

35

10

3
NA
13
121
10/08/2018
BMG
37
39
10
5
NA
8
122
13/08/2018
Kantar
40
39
9
3
NA
10
123
14/08/2018
YouGov
41
38
8
3
NA
11
124

16/08/2018

Deltapoll

37

40

8

5

NA

11

125

17/08/2018

Opinium

39

38

7

3

NA

13

126

19/08/2018

ICM

40

40

8

2

NA

10

127

21/08/2018

YouGov

40

37

9
2
NA
13
128
29/08/2018
YouGov
39
37
10
3
NA
11
129
01/09/2018
YouGov
42
36
9
2
NA
11
130
04/09/2018
YouGov
39
35
11
4
NA
11

131

07/09/2018

BMG

37

38

11

4

NA

11

132

09/09/2018

ICM

42

39

8

3

NA

7

133

10/09/2018

Kantar

40

35

10

4

NA

10

134

13/09/2018

YouGov

40

36
11
3
NA
9
135
13/09/2018
Opinium
39
38
7
3
NA
12
136
18/09/2018
Ipsos MORI
39
37
13
5
NA
6
137
19/09/2018
YouGov
40
36
11
2
NA

11
138
20/09/2018
Opinium
37
39
9
2
NA
14
139
22/09/2018
BMG
38
38
10
4
NA
9
140
24/09/2018
ICM
41
40
9
3
NA
7
141
25/09/2018
YouGov

42
36
11
2
NA
8
142
27/09/2018
ComRes
39
40
9
2
NA
10
143
28/09/2018
Opinium
39
36
9
3
NA
11
144
29/09/2018
BMG
35
40
12
3

NA

11

145

05/10/2018

BMG

38

39

10

4

NA

9

146

05/10/2018

Opinium

39

39

7

3

NA

11

147

09/10/2018

YouGov

41

37

9

3

NA

11

148

12/10/2018

Opinium

41

37

8

3

NA

12

149

15/10/2018

YouGov

41

36

9

3

NA

9

150

15/10/2018

Kantar

41

36

10

4

NA

10

151

22/10/2018

Ipsos MORI

39

37

10

5

NA

10

152

23/10/2018

YouGov

41

36

8

4

NA

11

153

26/10/2018

Deltapoll

43

40

6

2

NA

9

154

28/10/2018

ICM

40

38

9

3

NA

9

155

30/10/2018

YouGov

41

39

7

2

NA

9

156

02/11/2018

ComRes

37

39

9

3

NA

10

157

05/11/2018

YouGov

41

37

8

4

NA

9

158

09/11/2018

BMG

36

37

12
4
NA
11
159
12/11/2018
Kantar
40
39
8
3
NA
10
160
15/11/2018
ComRes
36
40
9
3
NA
12
161
15/11/2018
Opinium
36
39
7
3
NA
15

162

19/11/2018

YouGov

39

36

8

4

NA

11

163

27/11/2018

YouGov

40

35

10

3

NA

11

164

04/12/2018

YouGov

40

38

9

4

NA

10

165

05/12/2018

Ipsos MORI

38

38

9

5

NA

10

166

06/12/2018

Kantar

38

38

9

5

NA

11

167

07/12/2018

YouGov

38

37

10

4

NA

9

168

07/12/2018

BMG

37

38

12

4

NA

9
169
10/12/2018
YouGov
39
38
9
4
NA
11
170
14/12/2018
Opinium
38
39
8
4
NA
12
171
17/12/2018
YouGov
41
39
7
3
NA
10
172
20/12/2018
Opinium

39

39

6

4

NA

11

173

01/01/2019

Opinium

41

34

8

4

NA

13

174

07/01/2019

YouGov

41

35

11

3

NA

9

175

11/01/2019

BMG

36

36

12

5

NA

11

176

14/01/2019

YouGov

39

34

11

4

NA

12

177

14/01/2019

Kantar

35

38

9

4

NA

14

178

15/01/2019

ComRes

37

39

8

3

NA

11

179

17/01/2019

ComRes

38

37

10

3

NA

11

180

18/01/2019

ICM

39

40

9

3

NA

9

181

18/01/2019

Opinium

37

40

7

4

NA

14

182

01/02/2019

Opinium

37

33

7

4
NA
19
183
04/02/2019
YouGov
41
34
10
4
NA
10
184
05/02/2019
Ipsos MORI
38
38
10
3
0
12
185
08/02/2019
BMG
38
35
13
5
0
10
186

11/02/2019

Kantar

40

35

10

4

NA

11

187

15/02/2019

Opinium

37

37

8

4

0

13

188

19/02/2019

YouGov

41

33

10

4

0

12

189

22/02/2019

Opinium

40

32

5
4
0
20
190
23/02/2019
YouGov
41
30
10
4
2
13
191
23/02/2019
Deltapoll
39
31
5
3
NA
21
192
04/03/2019
YouGov
40
31
11
4
3
11

193

08/03/2019

BMG

37

31

10

5

NA

16

194

11/03/2019

Kantar

41

31

8

6

NA

13

195

15/03/2019

YouGov

35

31

12

4

4

12

196

15/03/2019

Opinium

35

35
7
4
NA
19
197
17/03/2019
ComRes
34
35
8
3
NA
20
198
19/03/2019
Ipsos MORI
38
34
8
4
1
15
199
21/03/2019
ComRes
34
35
8
4
NA

20

200

22/03/2019

Opinium

36

35

7

4

NA

18

201

25/03/2019

YouGov

36

33

11

4

5

11

202

29/03/2019

Opinium

35

35

9

5

NA

16

203

30/03/2019

Deltapoll

32
35
7
2
6
19
204
03/04/2019
YouGov
32
31
12
4
5
16
205
05/04/2019
BMG
29
31
8
4
6
20
206
07/04/2019
ComRes
32
32
7
3

NA

24

207

08/04/2019

Kantar

32

35

11

4

NA

19

208

11/04/2019

YouGov

28

32

11

5

8

16

209

12/04/2019

Opinium

29

36

8

4

NA

23

210

17/04/2019

YouGov

29
30
10
5
12
14
211

23/04/2019

Opinium

26
33
6
4
17
15
212

24/04/2019

YouGov

27
30
11
5
14
13
213

24/04/2019

Panelbase

27
36
8

3

13

14

214

30/04/2019

YouGov

29

29

13

5

15

10

215

09/05/2019

YouGov

24

24

16

7

18

10

216

10/05/2019

Opinium

22

28

11

6

21

12

217

10/05/2019

BMG

27

30

18

6

10

9

218

12/05/2019

ComRes

20

27

13

4

20

14

219

13/05/2019

Hanbury

21

30

13

5

19

13

220

13/05/2019

Kantar

25

34

15
3
10
14
221
14/05/2019
YouGov
25
25
16
7
18
10
222
14/05/2019
Ipsos MORI
25
27
15
7
16
11
223
16/05/2019
Opinium
22
29
11
3
24
10

224

20/05/2019

Opinium

22

26

12

4

25

9

225

21/05/2019

Panelbase

21

31

13

5

19

11

226

29/05/2019

YouGov

19

19

24

8

22

8

227

30/05/2019

Deltapoll

20

26

16

5

24

11

228

30/05/2019

Opinium

17

22

16

11

26

8

229

05/06/2019

YouGov

18

19

22

9

25

7

230

06/06/2019

YouGov

18

20

20

9

26

6
231
07/06/2019
BMG
26
27
17
6
18
7
232
09/06/2019
ComRes
23
27
17
5
22
7
233
10/06/2019
YouGov
17
19
22
8
26
7
234
14/06/2019
YouGov

21

21

19

9

24

5

235

19/06/2019

YouGov

20

20

21

9

23

6

236

20/06/2019

Opinium

20

26

16

6

23

9

237

20/06/2019

Survation

24

26

18

6

20
7
238
25/06/2019

YouGov

22
20
19
10
22
7
239

25/06/2019

Ipsos MORI

26
24
22
8
12
6
240

03/07/2019

YouGov

24
18
20
9
23
6
241

05/07/2019

Opinium

23

25

15

8

22

8

242

05/07/2019

BMG

28

27

18

6

14

6

243

07/07/2019

ComRes

25

28

16

5

19

5

244

10/07/2019

YouGov

24

20

19

9
21
7
245
11/07/2019
ComRes
24
28
15
5
20
8
246
11/07/2019
Survation
23
29
19
3
20
6
247
17/07/2019
YouGov
25
21
20
8
19
6
248

24/07/2019

YouGov

25

19

23

9

17

7

249

25/07/2019

ComRes

28

27

19

4

16

5

250

26/07/2019

YouGov

31

21

20

8

13

7

251

26/07/2019

Opinium

30

28

16
5
15
8
252
27/07/2019
Deltapoll
30
25
18
4
14
9
253
30/07/2019
YouGov
32
22
19
8
13
6
254
30/07/2019
Ipsos MORI
34
24
20
6
9
7

255

06/08/2019

YouGov

31

22

21

7

14

5

256

09/08/2019

Opinium

31

28

13

5

16

6

257

11/08/2019

ComRes

31

27

16

4

16

6

258

12/08/2019

BMG

31

25
19
6
12
7
259
14/08/2019
YouGov
30
21
20
8
14
8
260
19/08/2019
Kantar
42
28
15
3
5
7
261
21/08/2019
YouGov
32
22
20
7
12

7

262

23/08/2019

YouGov

33

21

19

7

14

6

263

23/08/2019

Opinium

32

26

15

4

16

8

264

28/08/2019

YouGov

34

22

17

8

13

7

265

29/08/2019

YouGov

33
22
21
7
12
6
266
31/08/2019
Deltapoll
35
24
18
4
14
6
267
03/09/2019
YouGov
35
25
16
7
11
6
268
04/09/2019
Hanbury
33
26
17
3

14

7

269

06/09/2019

YouGov

35

21

19

7

12

6

270

06/09/2019

Panelbase

31

28

19

2

15

3

271

06/09/2019

Opinium

35

25

17

3

13

7

272

07/09/2019

Deltapoll

31

28

17

4

13

7

273

08/09/2019

ComRes

30

29

17

4

13

7

274

09/09/2019

Kantar

38

24

20

3

7

8

275

10/09/2019

YouGov

32

23

19

7

14

5

276

12/09/2019

ComRes

28

27

20

5

13

7

277

13/09/2019

Opinium

37

25

16

2

13

6

278

16/09/2019

Ipsos MORI

33

24

23

4

10

5

279

18/09/2019

YouGov

32

21

23

4

14

8

280

20/09/2019

Opinium

37

22

17

4

12

7

281

25/09/2019

YouGov

33

22

22

6

14

4

282

27/09/2019

Opinium

36

24

20

2

11

7

283

27/09/2019

YouGov

33

22

21

5

13

6

284

01/10/2019

YouGov

34

21

23

5

12

5

285

04/10/2019

BMG

31

26

20

7

11

4

286

04/10/2019

Opinium

38

23

15

4

12

6

287

06/10/2019

ComRes

33

27

19

3

13

7

288

09/10/2019

YouGov

35

22

20

6

12

6

289

10/10/2019

ComRes

33

27

18

4

12

6

290

11/10/2019

Panelbase

33

30

17

3

12

4

291

15/10/2019

Kantar

39

25

18

3

8

6

292

15/10/2019

YouGov

37

22

18

5

11

6
293
17/10/2019
Opinium
37
24
16
4
12
8
294
17/10/2019
ComRes
33
29
18
4
12
5
295
18/10/2019
Panelbase
36
27
17
3
11
4
296
21/10/2019
Deltapoll

37

24

19

3

11

6

297

21/10/2019

YouGov

37

22

19

7

11

5

298

25/10/2019

Opinium

40

24

15

3

10

7

299

25/10/2019

YouGov

36

23

18

6

12
5
300
28/10/2019
Ipsos MORI
41
24
20
3
7
5
301
30/10/2019
YouGov
36
21
18
6
13
5
302
31/10/2019
Panelbase
40
29
14
3
9
5
303
31/10/2019

ComRes

36

28

17

3

10

5

304

31/10/2019

ORB

36

28

14

4

12

6

305

01/11/2019

YouGov

39

27

16

4

7

6

306

01/11/2019

Opinium

42

26

16

2
9
6
307
02/11/2019
Deltapoll
40
28
14
2
11
5
308
04/11/2019
ICM
38
31
15
3
9
5
309
04/11/2019
YouGov
38
25
16
5
11
5
310

06/11/2019

YouGov

36

25

17

5

11

6

311

08/11/2019

Panelbase

40

30

15

3

8

4

312

08/11/2019

Opinium

41

29

15

2

6

6

313

08/11/2019

YouGov

39

26

17
3
10
5
314
08/11/2019
BMG
37
29
16
7
9
2
315
09/11/2019
Deltapoll
41
29
16
2
6
6
316
10/11/2019
ComRes
37
29
17
3
9
5

317

11/11/2019

ICM

39

31

15

3

8

4

318

11/11/2019

Kantar

37

27

17

3

9

6

319

12/11/2019

YouGov

42

28

15

4

4

7

320

12/11/2019

ComRes

40

30
16
3
7
4
321
14/11/2019
Panelbase
43
30
15
2
5
5
322
14/11/2019
ComRes
41
33
14
2
5
5
323
15/11/2019
BMG
37
29
16
5
9

4	
324	
15/11/2019	
Opinium	
44	
28	
14	
3	
6	
5	
325	
15/11/2019	
YouGov	
45	
28	
15	
3	
4	
5	
326	
16/11/2019	
Deltapoll	
45	
30	
11	
3	
6	
5	
327	
18/11/2019	
ICM	

42
32
13
3
5
5
328
18/11/2019
Kantar
45
27
16
3
2
7
329
19/11/2019
YouGov
42
30
15
4
4
5
330
19/11/2019
ComRes
42
31
15
2

5

5

331

19/11/2019

Ipsos MORI

44

28

16

3

3

6

332

20/11/2019

YouGov

43

29

15

3

4

6

333

21/11/2019

BMG

41

28

18

5

3

5

334

21/11/2019

ComRes

42
32
15
2
5
4
335

22/11/2019

Panelbase

42
32
14
3
3
6
336

22/11/2019

YouGov

42
30
16
4
3
5
337

22/11/2019

Opinium

47
28
12

3

3

7

338

23/11/2019

Deltapoll

43

30

16

3

3

5

339

25/11/2019

ICM

41

34

13

3

4

5

340

25/11/2019

Kantar

43

32

14

4

3

4

341

26/11/2019

YouGov

43

32

13

2

4

6

342

26/11/2019

ComRes

41

34

13

2

5

5

343

28/11/2019

Panelbase

42

34

13

3

4

4

344

28/11/2019

ComRes

43

33

13

3

4

4

345

29/11/2019

BMG

39

33

13

5

4

4

346

29/11/2019

YouGov

43

34

13

3

2

5

347

29/11/2019

Opinium

46

31

13

2

2

6

348

30/11/2019

Deltapoll

45

32

15

1

3

3

349

02/12/2019

ICM

42

35

13

2

3

4

350

02/12/2019

Kantar

44

32

15

3

2

4

351

03/12/2019

YouGov

42

33

12

4

4

6

352

03/12/2019

ComRes

42

32

12

2

3

9

353

04/12/2019

Ipsos MORI

44

32

13

3

2

6

354

11/12/2019

Ipsos Mori

44

33

12

3

2

5
355
11/12/2019
Panelbase
43
34
11
3
4
4
356
11/12/2019
Survation
45
34
9
3
3
5
357
11/12/2019
Opinium
45
33
12
2
2
4
358
11/12/2019
Deltapoll

45

35

10

3

3

4

359

11/12/2019

Kantar TNS

44

32

13

3

3

0

360

11/12/2019

BMG

41

32

14

4

3

0

361

10/12/2019

ComRes

41

36

12

2

3
4
362
09/12/2019
ICM
42
36
12
2
3
3
363
08/12/2019
ComRes
43
36
12
2
3
4
364
07/12/2019
Survation
45
31
11
2
4
4
365
07/12/2019

Deltapoll

44

33

11

NA

3

0

366

06/12/2019

BMG

41

32

14

4

4

3

367

06/12/2019

YouGov

43

33

13

3

3

0

368

05/12/2019

ComRes

41

33

12

```

2
3
4
369
05/12/2019
ComRes
42
36
11
2
4
NA

```

11.2 Transform the data to long format

We need to Transform the data from *wide format* to *long format*.

Making the *party* variable into an ordered factor allows me to control the order that the party will be ordered in the legend.

The data set contains polling data going back to 2017. We'll examine data for 2019.

```

#Transform the data from wide format to long format.
ukpoll %>% gather(party, rating, con:oth) -> ukpoll_long

#Make the *party* variable into an ordered factor
ukpoll_long$party <-
  factor(ukpoll_long$party,
         levels = c('con', 'lab', 'ldem', 'grn', 'brx', 'oth'))

# selecting data for 2019 only
ukpoll_long_2019 <-
  ukpoll_long %>% mutate(date = as.Date(date, format = "%d/%m/%Y")) %>% filter(date >= as.Date("2019-01-01"))

```

11.3 Calculate daily mean poll scores

As there are several polls presenting figures for the same day, I calculate an unweighted mean of their scores - so that each party will have a single score per day.

I will use the means as the data points on which to plot the *loess* smoothing curve.

```
# as there are some times several polls on the same day, I calculate an unweighted mean
ukpoll_long_2019_daily_mean<-ukpoll_long_2019 %>%
  group_by(date, party) %>%
  summarise(meanrating = mean(rating))
```

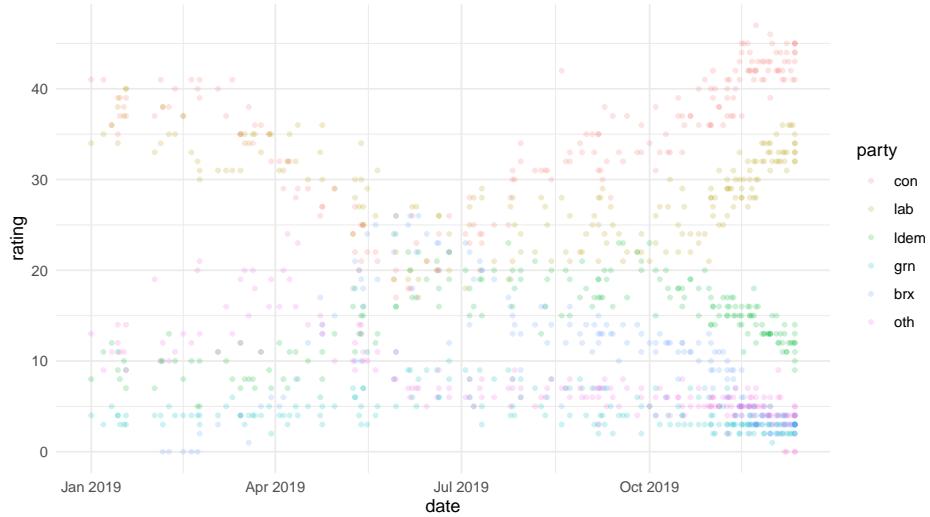
11.4 Make a basic plot

I'll make a basic plot laying out the data points and colouring by party.

I'll set the alpha value low as we will have quite a bit of overplotting

```
g <-
  ggplot(data = ukpoll_long_2019, aes(x = date, y = rating, colour = party)) +
  geom_point(alpha = 0.2,
             size = 1,
             na.rm = TRUE) +
  theme_minimal()

g
```



11.5 Add a smoothing line

The distribution of point colouration suggests several trends but nothing is very clear.

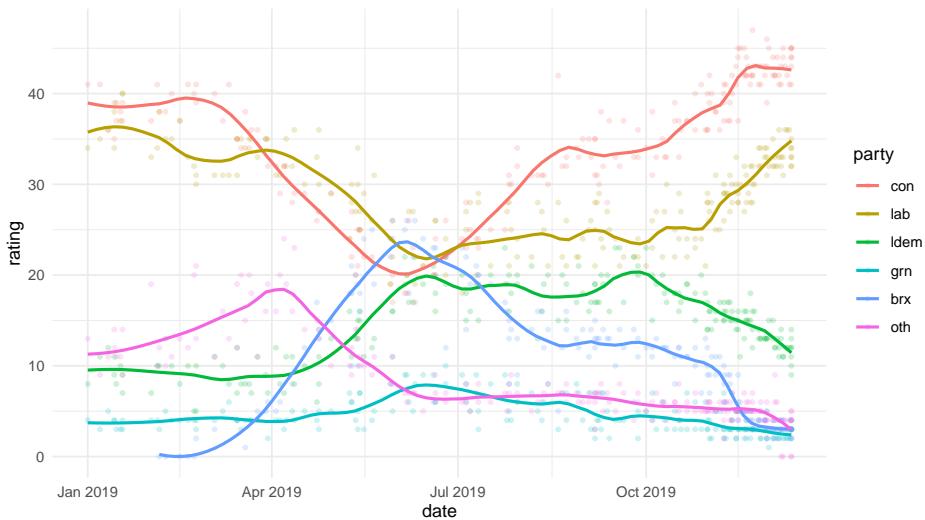
Now, I will superimpose a *loess* smoothing trend line on top of these points using `geom_smooth`

The data I use for the smoother is the daily mean data that I calculated calculated above. Notice how you can give a geometric its own data set and aesthetic mappings.

I've told ggplot that the aesthetic (`aes`) for this geometric should not inherit the global aesthetic mapping defined in the `ggplot` function - `inherit.aes = FALSE`

```
g <-
  g + geom_smooth(
    data = ukpoll_long_2019_daily_mean,
    aes(x = date, y = meanrating, color = party),
    method = "loess",
    span = 0.2,
    size = 0.9,
    na.rm = TRUE,
    se = FALSE,
    inherit.aes = FALSE
  )

g
```



You can see how in a few lines of code the trends in this data set can be outlined.

If this graph is part of exploratory analysis for our own consumption, we may not need to go much further. It gives us the main trends in the data.

However, there are still several things to do before graph is ready to present or publish to other people.

11.6 Further steps

- customise the party colour allocations
- customise the x axis breaks so that they are more frequent and show months only (year not needed)
- customise y axis title to “percentage”; remove x axis title.
- move the legend to the top and make it horizontal
- customise legend so it shows long form of part names
- customise gridlines, showing the very minimum needed
- add vertical lines indicating events of interest - Theresa May’s resignation, Boris Johnson’s appointment as PM, Election day
- add highlight of party polling scores on the eve of the election

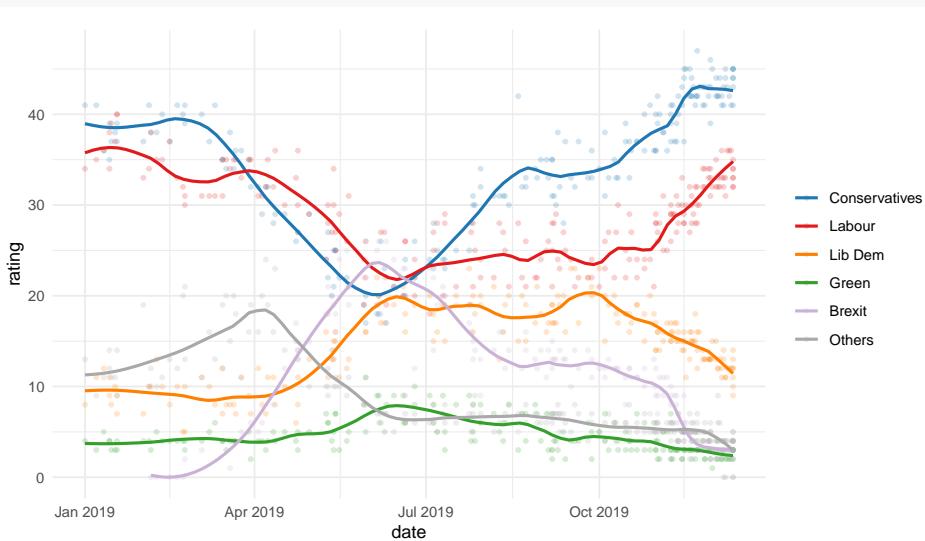
11.7 Customise colour allocation

Let’s first customise the party colour allocations to more appropriate colours. We use the `scale_colour_manual` to do that. At the same time, we can customise the labels associated with these colours by setting the `label` attribute to a vector of party long names. Setting the `name` attribute to *NULL* tells ggplot not to show the name of the variable represented by the colours.

```
party.colours <-
  c(
    'con' = '#1f78b4',
    'lab' = '#e31a1c',
    'ldem' = '#ff7f00',
    'brx' = '#cab2d6',
    'grn' = '#33a02c',
    'oth' = 'darkgrey'
  )

g <- g +
  scale_colour_manual(
    values = party.colours,
    labels = c(
      "Conservatives",
      "Labour",
      "Lib Dem",
      "Green",
      "Brexit",
      "Others"
    ),
    name = NULL
  )

g
```



11.8 Customise the x-axis

Next we'll customise the x-axis breaks so that they are more frequent and show months only (year not needed).

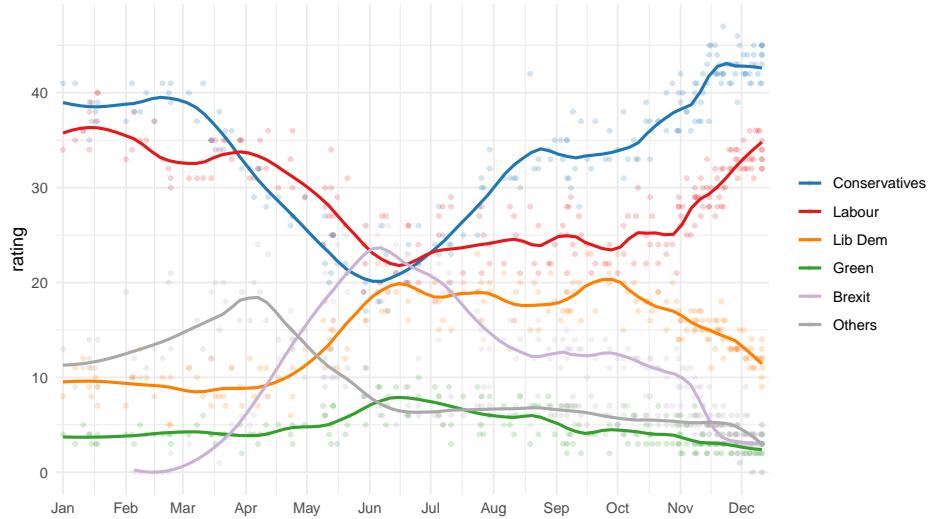
As the x-axis has date values, we have to use the `scale_x_date` function to override the ggplot defaults. This function has a special way of setting breaks. You simply specify the break interval you want in the form `1 month`, `2 month`, `1 year`, `2 week`, etc

We specify the `data_format` as `%b`, which represents the short format of a month. You can see examples of different date formats available here

Frankly, I find the `expand` attribute hard to explain. It's associated with the `scale_x_` and `scale_y_`. In general, setting its value to `c(0,0)` tightens the plot so that there is no distance between the plot and the axis in question. However, the function that controls how the values in the vector control the distance are somewhat arcane - and I tweak these values on a trial and error basis.

```
g <- g +
  scale_x_date(
    name = NULL,
    breaks = "1 month",
    labels = date_format("%b"),
    expand = c(0.01, 0)
  )

g
```

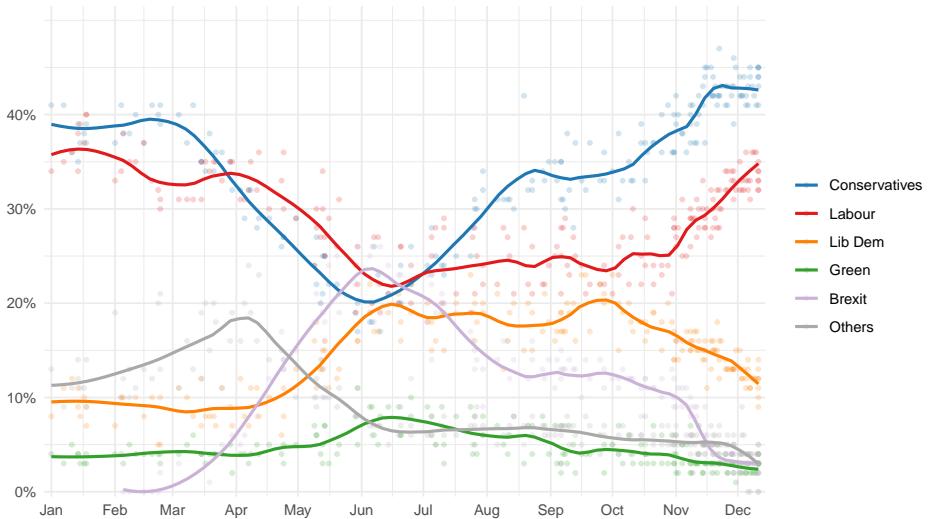


11.9 Customise the y-axis

We'll customise the y-axis. The main addition of interest here is the removal of the axis title, `name = NULL`. Instead of changing the title to *percentage*, I've directly labelled the values *as percentages*

```
g <- g +
  scale_y_continuous(
    limits = c(0, 51),
    breaks = seq(0, 40, by = 10),
    name = NULL,
    labels = paste0(seq(0, 40, by = 10), "%") ,
    expand = c(0.01, 0)
  )

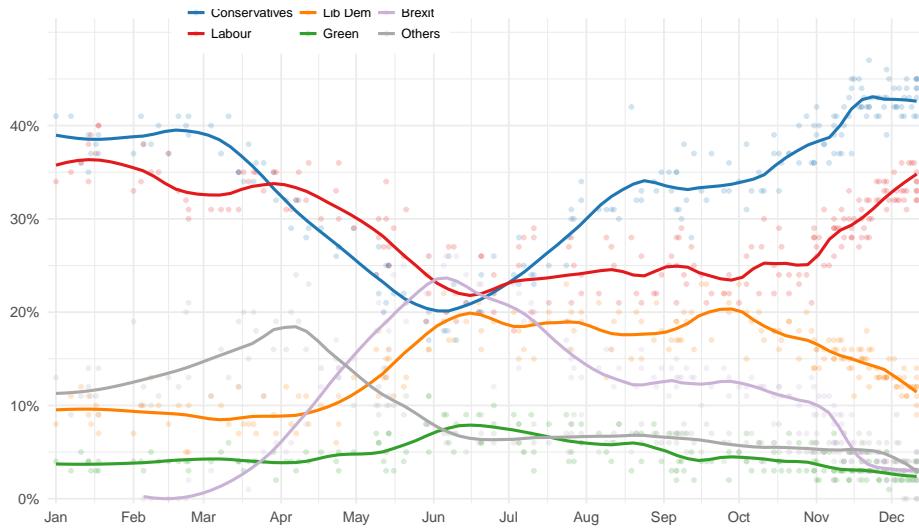
g
```



11.10 Customise the legend

Now, we'll make some customisations to the legend, which we do in the theme function.

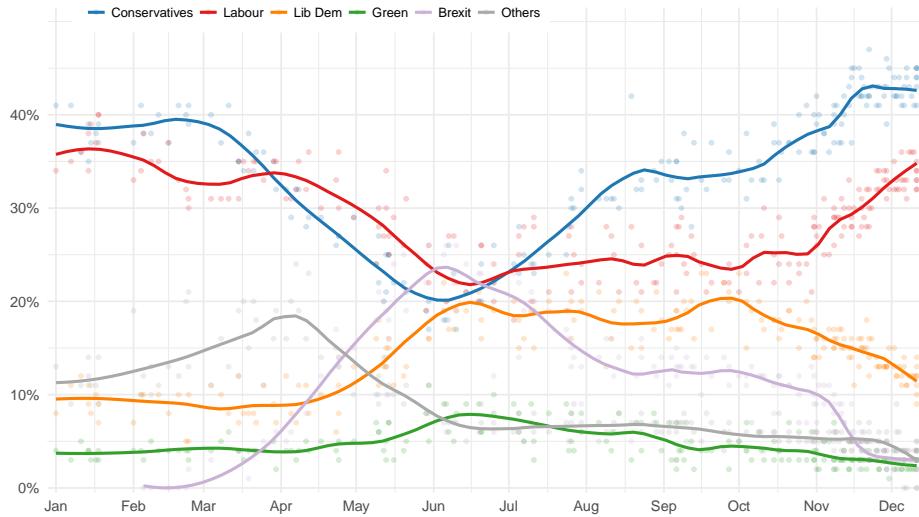
```
g <- g +
  theme(
    legend.text = element_text(size = 8),
    legend.title = element_blank(),
    legend.position = c(0.3, 0.99), # move to the top
    legend.direction = "horizontal", # make it horizontal
    legend.key.size = unit(0.9, "line"),
    legend.spacing.x = unit(0.1, 'cm'),
    legend.background = element_rect(
      fill = "white",
      size = 0.5,
      colour = "white"
    )
  )
g
```



While we've specified that the legend be horizontal, ggplot will wrap it into a second row by default. To override this we need to call upon the `guides` function. This is somewhat non-intuitive.

```
g <- g + guides(colour = guide_legend(nrow = 1))
```

```
g
```



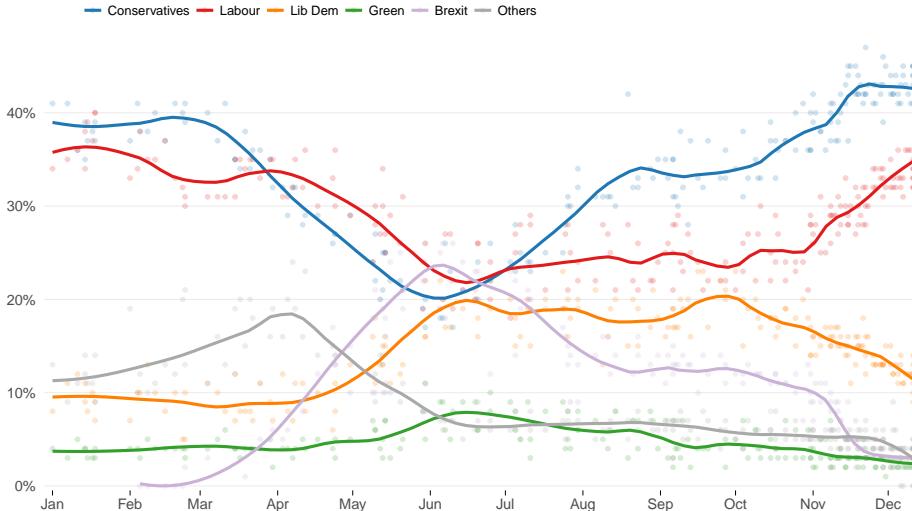
11.11 Customise the gridlines

We'll customise gridlines, showing the very minimum needed. I think we just need very light gridlines representing the break values on the y-axis. These

are the *major y* gridlines. As I've already removed the x-axis gridlines, I'll replace the x-axis ticks so the reader knows more precisely the point on the axis indicated by the month values

Note how you can add simply another theme function with an additional customisation - this time for the gridlines

```
g <- g +
  theme(
    panel.grid.minor.y = element_blank(),
    panel.grid.major.x = element_blank(),
    panel.grid.minor.x = element_blank(),
    panel.grid.major.y = element_line(size = 0.2),
    axis.ticks.x = element_line(size = 0.2)
  )
g
```



11.12 Add vertical lines with labels to represent events

Now we'll add vertical lines to indicate events of interest: Theresa May's resignation, Boris Johnson's appointment as PM, Election day.

For each event we'll add a vertical line and a label. We will use two geoms for this `geom_vline` and `geom_label`.

```
g <- g +
  # vertical line for Theresa' May's resignation
  geom_vline()
```

```

xintercept = as.Date("07/06/2019", format = "%d/%m/%Y") ,
linetype = "solid",
color = "grey",
size = 0.2,
alpha = 0.8
) +
# label for Theresa' May's resignation
geom_label(
  aes(
    x = as.Date("07/06/2019", format = "%d/%m/%Y"),
    y = 37,
    label = "Theresa May resigns"
  ),
  color = "darkgrey",
  fill = "white",
  size = 2.5
) +

# vertical line for Boris Johnson's resignation
geom_vline(
  xintercept = as.Date("24/07/2019", format = "%d/%m/%Y") ,
  linetype = "solid",
  color = "grey",
  size = 0.2,
  alpha = 0.8
) +

# label for Boris Johnson's resignation
geom_label(
  aes(
    x = as.Date("24/07/2019", format = "%d/%m/%Y"),
    y = 42,
    label = "Boris Johnson elected PM"
  ),
  color = "darkgrey",
  fill = "white",
  size = 2.5
) +

# to give better horizontal separation between vline and the finalpoints, I've 'move
geom_vline(
  xintercept = as.Date("14/12/2019", format = "%d/%m/%Y") ,
  linetype = "solid",
  color = "grey",
  size = 0.2,
)

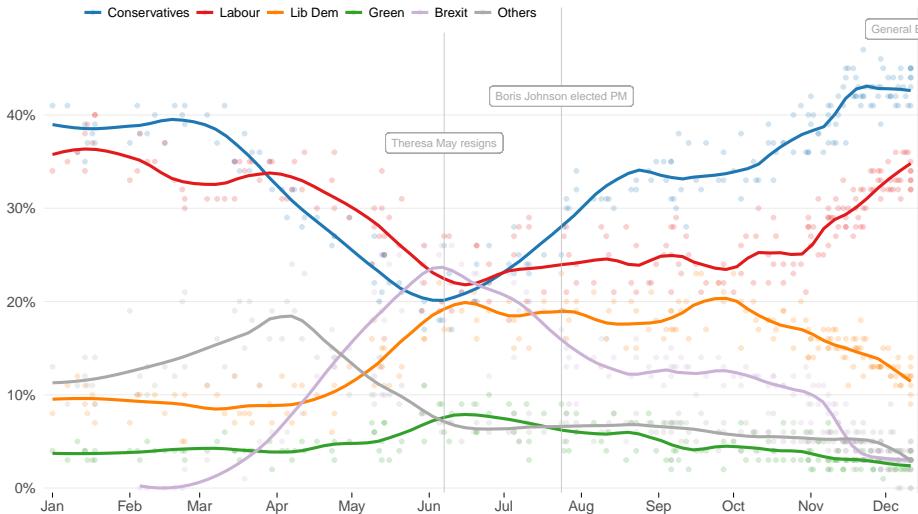
```

```

    alpha = 0.7
) +  
  

  geom_label(
    aes(
      x = as.Date("12/12/2019", format = "%d/%m/%Y"),
      y = 49.2,
      label = "General Election"
    ),
    color = "darkgrey",
    fill = "white",
    size = 2.5
  )
g

```



11.13 Turn ‘clipping’ off

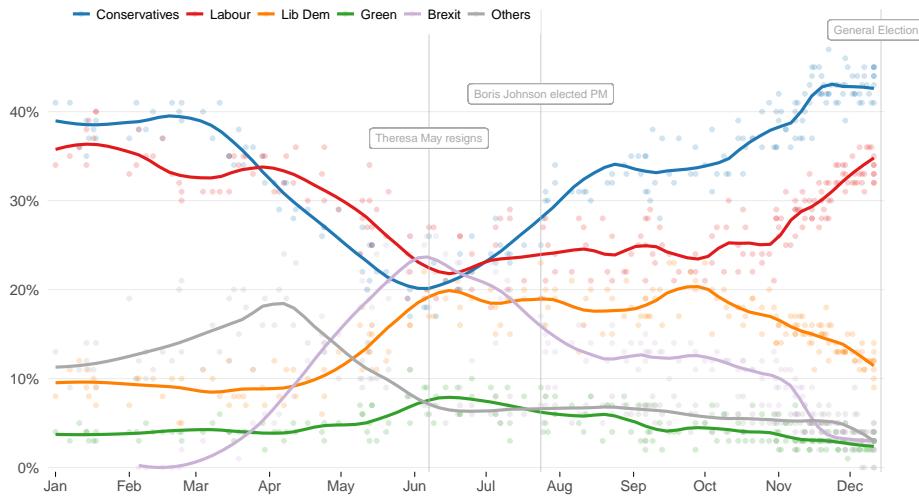
You’ll notice that the label for the general election runs outside the plot and is clipped. We’ll tell ggplot to turn clipping off and we’ll set some margin around the plot.

It seems that there is a bug in ggplot where adding subsequent separate theme elements removes the ticks in the x-axis, so I have to replace them again here.

```

g <- g +
  coord_cartesian(clip = 'off') +
  theme(plot.margin = unit(c(0.5, 1, 0.5, 0.5), "cm")) +
  theme(axis.ticks.x = element_line(size = 0.2))
g

```



11.14 Label the final points on the right hand-side

We are now going to label the final poll numbers on the eve of the general election December 11th. As several polls were conducted, we'll show the unweighted average score for each party. The idea is that we'll show these values on the right hand side, with each party value coloured in the party colours. As there were really 4 parties of significant interest in this election, we will show values for the Conservatives, Labour, the Lib Dems and the Brexit Party. This also avoids us having to sort out the potentially difficult task of labelling in a tight space all the parties that were predicted to receive about 2% of the vote.

11.15 Create a hidden second y-axis

I create a hidden secondary y axis on the right handside, specifying 4 breaks - one for the final (average) poll score for each of the 4 parties. Label each of these breaks in percentage format. Colour each value by the associated party value.

First I calculate a data frame containing the final values for the y-axis labels on the right but only selecting the parties of interest

```
# the last date in the series
lastdate <- max(ukpoll_long_2019_daily_mean$date)

# the final mean poll values for "con", "lab", "ldem" and "brx"
ukpoll_long_2019_daily_mean_final <-
  filter(
```

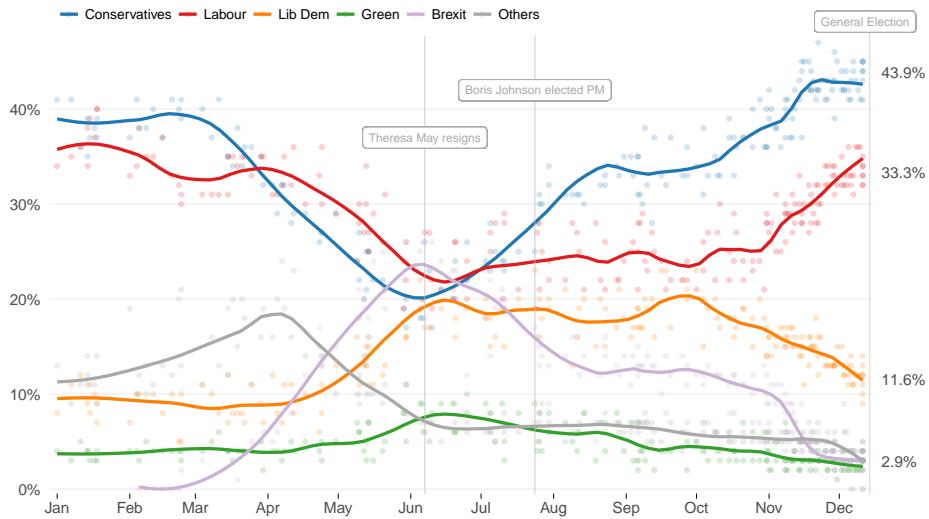
```
    ukpoll_long_2019_daily_mean,
    date == lastdate & party %in% c("con", "lab", "ldem", "brx")
)
```

I have been showing you the power of incrementally building a plot like this in ggplot. At this point, I have to specify a second y-axis, which I do with the `scale_y_continuous` function, that I've already defined. I am simply going to override that initial definition with another. However, this is for tutorial purposes and keeps the amount of code I have to show at each step to a minimum. In a reality, you would go back and modify your existing `scale_y_continuous` function.

```
g <- g +
  scale_y_continuous(
    limits = c(0, 50),
    breaks = seq(0, 40, by = 10),
    name = NULL,
    labels = paste0(seq(0, 40, by = 10), "%") ,
    expand = c(0.01, 0),
    sec.axis = dup_axis( # specifying the second y axis
      breaks = ukpoll_long_2019_daily_mean_final$meanrating,
      labels = paste0(round(
        ukpoll_long_2019_daily_mean_final$meanrating, 1
      ), "%"),
      name = NULL
    )
  )
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which will
## replace the existing scale.
```

```
g
```



The `theme_minimum` theme we are using as a base, suppresses the new axis line on the right. Otherwise we would have to explicitly set `axis.line.y.right` to `element_blank()` in a theme function.

11.16 Colour the right y-axis text

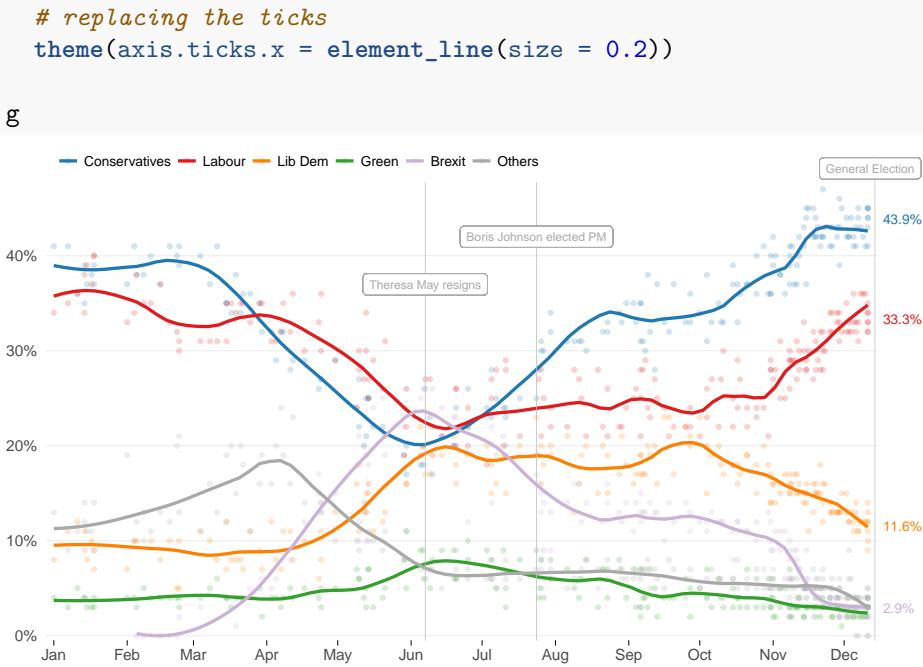
Now we will colour the final values in party colours. We select a subset of party colours defined earlier - in fact, the first four. Then we order them in the order they should appear on the right y-axis.

We then set the axis text colours to these subset of party colours in a theme function

```
# reordering a subset of the party colours

ukpoll.right.axis.colours <-
  party.colours[1:4] [order(-ukpoll_long_2019_daily_mean_final$meanrating)]

g <- g +
  theme(
    axis.text.y.right = element_text(
      margin = margin(0, 0, 0, 0) ,
      size = 8,
      colour = ukpoll.right.axis.colours
    ),
    axis.ticks.y.right =   element_line(size = 0.2)
  ) +
  # ggplot bug removes ticks every time new theme function is specified
```



11.17 The full code

Here is the entire code for this graphic in one place:

First the data processing steps:

```
library(readr)
library(ggplot2)
library(dplyr)
library(tidyr)
library(lubridate)
library(scales)
library(kableExtra)

ukpoll <- read_csv("../data/UKPoll-2020-continuous-series-1.csv")

#Transform the data from wide format to long format.
ukpoll %>% gather(party, rating, con:oth) -> ukpoll_long

#Make the *party* variable into an ordered factor
ukpoll_long$party <-
  factor(ukpoll_long$party,
        levels = c('con', 'lab', 'ldem', 'grn', 'brx', 'oth'))
```

```

# selecting data for 2019 only
ukpoll_long_2019 <-
  ukpoll_long %>% mutate(date = as.Date(date, format = "%d/%m/%Y")) %>% filter(date >=

# as there are some times several polls on the same day, I calculate an unweighted mean
ukpoll_long_2019_daily_mean<-ukpoll_long_2019 %>%
  group_by(date, party) %>%
  summarise(meanrating = mean(rating))

# we calculate a data frame containing the final values for the y-axis labels on the right axis
# but only select the parties of interest
lastdate<-max(ukpoll_long_2019_daily_mean$date)
ukpoll_long_2019_daily_mean_final <- filter(ukpoll_long_2019_daily_mean, date == lastdate)

party.colours <- c('con' = '#1f78b4', 'lab' = '#e31a1c', 'ldem' = '#ff7f00', 'brx' = '#c8512e')

ukpoll.right.axis.colours <- party.colours[1:4] [order(-ukpoll_long_2019_daily_mean_final$meanrating)]

```

The ggplot function:

```

ggplot(data = ukpoll_long_2019, aes(x = date, y = rating, colour = party)) +
  geom_point(alpha = 0.2,
             size = 1,
             na.rm = TRUE) +
  geom_smooth(
    data = ukpoll_long_2019_daily_mean,
    aes(x = date, y = meanrating, color = party),
    method = "loess",
    span = 0.2,
    size = 0.9,
    na.rm = TRUE,
    se = FALSE,
    inherit.aes = FALSE
  ) +
  scale_colour_manual(
    values = party.colours,
    labels = c(
      "Conservatives",
      "Labour",
      "Lib Dem",
      "Green",
      "Brexit",
      "Others"
    )
  )

```

```

),
name = NULL
) +
scale_x_date(
  name = NULL,
  breaks = "1 month",
  labels = date_format("%b"),
  expand = c(0.01, 0)
) +
scale_y_continuous(
  limits = c(0, 50),
  breaks = seq(0, 40, by = 10),
  name = NULL,
  labels = paste0(seq(0, 40, by = 10), "%") ,
  expand = c(0.01, 0),
  sec.axis = dup_axis(
    breaks = ukpoll_long_2019_daily_mean_final$meanrating,
    labels = paste0(round(
      ukpoll_long_2019_daily_mean_final$meanrating, 1
    ), "%"),
    name = NULL
  )
) +
# vertical line for Theresa' May's resignation
geom_vline(
  xintercept = as.Date("07/06/2019", format = "%d/%m/%Y") ,
  linetype = "solid",
  color = "grey",
  size = 0.2,
  alpha = 0.8
) +
# label for Theresa' May's resignation
geom_label(
  aes(
    x = as.Date("07/06/2019", format = "%d/%m/%Y"),
    y = 37,
    label = "Theresa May resigns"
  ),
  color = "darkgrey",
  fill = "white",
  size = 2.5
)

```

```

) +  

# vertical line for Boris Johnson's resignation  

geom_vline(  

  xintercept = as.Date("24/07/2019", format = "%d/%m/%Y") ,  

  linetype = "solid",  

  color = "grey",  

  size = 0.2,  

  alpha = 0.8  

) +  

# label for Boris Johnson's resignation  

geom_label(  

  aes(  

    x = as.Date("24/07/2019", format = "%d/%m/%Y"),  

    y = 42,  

    label = "Boris Johnson elected PM"  

  ),  

  color = "darkgrey",  

  fill = "white",  

  size = 2.5  

) +  

# to give better horizontal separation between vline and the finalpoints, I've 'moved'  

geom_vline(  

  xintercept = as.Date("14/12/2019", format = "%d/%m/%Y") ,  

  linetype = "solid",  

  color = "grey",  

  size = 0.2,  

  alpha = 0.7  

) +  

geom_label(  

  aes(  

    x = as.Date("12/12/2019", format = "%d/%m/%Y"),  

    y = 49.2,  

    label = "General Election"  

  ),  

  color = "darkgrey",  

  fill = "white",  

  size = 2.5  

) +  

theme_minimal() +
theme(  


```

```
plot.margin = unit(c(0.5, 1, 0.5, 0.5), "cm"),
panel.grid.minor.y = element_blank(),
panel.grid.major.x = element_blank(),
panel.grid.minor.x = element_blank(),
panel.grid.major.y = element_line(size = 0.2),
axis.ticks.x = element_line(size = 0.2),
axis.title.x = element_blank(),
legend.text = element_text(size = 8),
legend.title = element_blank(),
legend.position = c(0.3, 0.99),
legend.direction = "horizontal",
legend.key.size = unit(0.9, "line"),
legend.spacing.x = unit(0.1, "cm"),
legend.background = element_rect(
  fill = "white",
  size = 0.5,
  colour = "white"
),
axis.text.y.right = element_text(
  margin = margin(0, 0, 0, 0) ,
  size = 8,
  colour = ukpoll.right.axis.colours
),
axis.ticks.y.right =element_line(size = 0.2)
)  +
guides(colour = guide_legend(nrow = 1)) +
coord_cartesian(clip = 'off')
```


Chapter 12

Tutorial: Bubble Charts over time (#bubbleseries)

In this section we are going to include all the time data in the data set – so we are going to visualise the GDP and life expectancy trends from 1990 to 2014 for each region in the data set

GDP is a simply a fairly crude economic measure of a country's wealth. However, it seems to be fact of life that higher GDP in a country entails relatively higher capacity to support the well being of the country's population.

Our visualisation should uncover that relationship.

I would expect to see some regions to have made relatively short journeys, while others have accelerated to catch up with the wealthier regions.

12.0.1 Data Processing using dplyr

I use `dplyr` functions to produce a view of the `nations_data` where each region has a `ave_gdp_perca` (average gdp) and `ave_life_expect` (average life expectancy).

In order to show the comparative populations per region, the population of each region is calculated by suming the populations of its member countries.

The `dplyr` library is very useful for processing and filtering data. It is part of tidyverse so you more than likely have it installed

Some of the most useful functions in `dplyr`:

- `select` Choose which columns to include.
- `filter` Filter the data.

- **arrange** Sort the data, by size for continuous variables, by date, or alphabetically.
- **group_by** Group the data by a categorical variable.
- **summarize** Summarise or aggregate (for each group if following **group_by**). Often used in conjunction with functions including:
 - **mean** Calculate the mean, or average.
 - **median** Calculate the median.
 - **max** Find the maximum value.
 - **min** Find the minimum value
 - **sum** Add all the values together.
 - **n** Count the number of records.
- **mutate** Create new column(s) in the data, or change existing column(s).
- **rename** Rename column(s).
- **bind_rows** Merge two data frames into one, combining data from columns with the same name.

These functions can be chained together using the operator `%>%` which makes the output of one line of code the input for the next. This allows you to run through a series of operations in logical order. I find it helpful to think of `%>%` as “then.”

12.0.2 Analysis

As there is high variation in population per region, a simple mean is not appropriate for `ave_gdp_perca` (average gdp) and `ave_life_expect`.

As such, I calculate a weighted.mean per region, where the *weight* is defined by the population proportions of the region’s countries.

```
library(tidyverse)
library(ggplot2 )
library(readr)
library(scales)

# load data
nations_data <- read_csv("../data/nations.csv")

## Parsed with column specification:
## cols(
##   iso2c = col_character(),
##   iso3c = col_character(),
##   country = col_character(),
##   year = col_double(),
##   gdp_percap = col_double(),
##   life_expect = col_double(),
##   population = col_double(),
```

```

##   birth_rate = col_double(),
##   neonat_mortal_rate = col_double(),
##   region = col_character(),
##   income = col_character()
## )

library(kableExtra)
# summarising the region data.
# For each region average life expectancy and average gdp per cap is calculated, using a weighted mean

regions<-nations_data %>%
  filter(!is.na(population))%>% # weighted.mean can't handle NAs in the weight vector
  select(year, region, population, gdp_per_cap, life_expect) %>%
  group_by(year, region) %>%
  summarize(totalPop = sum(population,na.rm = TRUE), ave_gdp_per_cap = weighted.mean(gdp_per_cap, population))

kable(regions, digits = 2, format = "html", row.names = TRUE) %>%
  kable_styling( bootstrap_options = c("striped"), full_width = F,
                font_size = 10) %>%
  scroll_box(height = "300px")

year
region
totalPop
ave_gdp_per_cap
ave_life_expect
1
1990
East Asia & Pacific
1780171581
3194.59
68.87
2
1990
Europe & Central Asia
841062141
11738.26

```

402CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

72.08
3
1990
Latin America & Caribbean
445998222
5914.43
67.76
4
1990
Middle East & North Africa
254531443
7140.30
65.80
5
1990
North America
277473326
23567.36
75.43
6
1990
South Asia
1132833314
1208.81
58.22
7
1990
Sub-Saharan Africa
508274783
1725.35
49.90

8
1991
East Asia & Pacific
1805445473
3440.61
69.06
9
1991
Europe & Central Asia
844342182
11968.39
71.99
10
1991
Latin America & Caribbean
454117634
6162.47
68.11
11
1991
Middle East & North Africa
261076814
7276.86
66.38
12
1991
North America
281211703
23968.45
75.58
13

404CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

1991
South Asia
1157752321
1247.18
58.71
14
1991
Sub-Saharan Africa
522658547
1669.15
49.90
15
1992
East Asia & Pacific
1828962351
3662.45
69.23
16
1992
Europe & Central Asia
847708966
11948.62
71.92
17
1992
Latin America & Caribbean
462201058
6312.24
68.48
18
1992

Middle East & North Africa

265336498

7621.09

66.84

19

1992

North America

285092192

24986.76

75.79

20

1992

South Asia

1182862267

1321.59

59.20

21

1992

Sub-Saharan Africa

537401088

1632.53

49.90

22

1993

East Asia & Pacific

1851643425

3896.80

69.42

23

1993

Europe & Central Asia

406CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

850676140
12009.25
71.59
24
1993
Latin America & Caribbean
470263697
6898.36
68.86
25
1993
Middle East & North Africa
271312554
7756.96
67.31
26
1993
North America
288811320
25942.95
75.65
27
1993
South Asia
1208123497
1382.74
59.70
28
1993
Sub-Saharan Africa
552474255

1629.91
49.89
29
1994
East Asia & Pacific
1874414310
4175.27
69.63
30
1994
Europe & Central Asia
852497017
12176.42
71.66
31
1994
Latin America & Caribbean
478310786
7240.19
69.24
32
1994
Middle East & North Africa
277344152
7856.33
67.74
33
1994
North America
292297226
27246.16

408CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

75.84
34
1994
South Asia
1233480616
1466.86
60.19
35
1994
Sub-Saharan Africa
567849593
1647.57
49.90
36
1995
East Asia & Pacific
1896917329
4478.35
69.83
37
1995
Europe & Central Asia
853987473
12473.23
71.85
38
1995
Latin America & Caribbean
486343677
7328.80
69.64

39
1995
Middle East & North Africa
284973163
8434.08
68.16
40
1995
North America
295691746
28242.37
75.86
41
1995
South Asia
1258903823
1569.10
60.68
42
1995
Sub-Saharan Africa
583503523
1688.46
49.91
43
1996
East Asia & Pacific
1919182682
4787.11
70.12
44

410CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

1996

Europe & Central Asia

855160755

12922.66

72.20

45

1996

Latin America & Caribbean

494384205

7564.03

70.03

46

1996

Middle East & North Africa

290950450

8854.14

68.53

47

1996

North America

299126029

29455.56

76.25

48

1996

South Asia

1284343918

1673.08

61.15

49

1996

Sub-Saharan Africa
599459137
1762.76
49.92
50
1997
East Asia & Pacific
1940987063
5042.05
70.40
51
1997
Europe & Central Asia
856510660
13504.74
72.59
52
1997
Latin America & Caribbean
502390020
7937.62
70.42
53
1997
Middle East & North Africa
296986652
9178.24
68.86
54
1997
North America

412CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

302704697

30928.98

76.63

55

1997

South Asia

1309774343

1730.67

61.61

56

1997

Sub-Saharan Africa

615751507

1808.09

49.96

57

1998

East Asia & Pacific

1962063288

5015.74

70.71

58

1998

Europe & Central Asia

857709897

14154.96

72.81

59

1998

Latin America & Caribbean

510356845

7938.73
70.80
60
1998
Middle East & North Africa
303006934
9546.82
69.16
61
1998
North America
306162843
32273.59
76.79
62
1998
South Asia
1335193652
1810.80
62.06
63
1998
Sub-Saharan Africa
632437265
1820.14
50.02
64
1999
East Asia & Pacific
1981853907
5247.43

414CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

71.04
65
1999
Europe & Central Asia
859084460
14686.41
72.92
66
1999
Latin America & Caribbean
518188225
8020.69
71.16
67
1999
Middle East & North Africa
309069358
9936.34
69.47
68
1999
North America
309600485
33935.26
76.81
69
1999
South Asia
1360594891
1942.57
62.49

70
1999
Sub-Saharan Africa
649600908
1840.48
50.14
71
2000
East Asia & Pacific
2000694556
5629.48
71.41
72
2000
Europe & Central Asia
860243183
15824.94
73.03
73
2000
Latin America & Caribbean
525886558
8436.09
71.51
74
2000
Middle East & North Africa
315122478
10652.85
69.74
75

416CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

2000
North America
312993944
35736.72
76.89
76
2000
South Asia
1385959570
2030.38
62.91
77
2000
Sub-Saharan Africa
667303361
1898.43
50.33
78
2001
East Asia & Pacific
2018871629
5938.54
71.77
79
2001
Europe & Central Asia
861744170
16652.04
73.35
80
2001

Latin America & Caribbean

533449671

8616.55

71.83

81

2001

Middle East & North Africa

321163368

10921.72

70.01

82

2001

North America

316113359

36568.27

77.10

83

2001

South Asia

1411288216

2127.69

63.30

84

2001

Sub-Saharan Africa

685347923

1967.45

50.61

85

2002

East Asia & Pacific

418CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

2036127122
6302.93
72.12
86
2002
Europe & Central Asia
863439681
17363.52
73.41
87
2002
Latin America & Caribbean
540884684
8748.34
72.13
88
2002
Middle East & North Africa
327207928
11170.33
70.26
89
2002
North America
319050105
37448.11
77.20
90
2002
South Asia
1436527869

2182.27
63.69
91
2002
Sub-Saharan Africa
703548709
2004.18
50.99
92
2003
East Asia & Pacific
2052580068
6717.21
72.46
93
2003
Europe & Central Asia
865721702
18039.50
73.52
94
2003
Latin America & Caribbean
548225528
8925.15
72.41
95
2003
Middle East & North Africa
333373436
11620.42

420CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

70.49
96
2003
North America
321847258
38941.15
77.31
97
2003
South Asia
1461601023
2344.62
64.07
98
2003
Sub-Saharan Africa
722369527
2091.49
51.47
99
2004
East Asia & Pacific
2068447258
7292.75
72.79
100
2004
Europe & Central Asia
868315097
19156.60
73.95

101
2004
Latin America & Caribbean
555515431
9573.22
72.67
102
2004
Middle East & North Africa
339823622
12694.15
70.73
103
2004
North America
324864038
41118.37
77.75
104
2004
South Asia
1486402311
2547.69
64.44
105
2004
Sub-Saharan Africa
741840464
2348.62
52.05
106

422CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

2005
East Asia & Pacific
2083988624
7927.07
73.07
107
2005
Europe & Central Asia
870966111
20382.78
74.10
108
2005
Latin America & Caribbean
562783235
10237.36
72.92
109
2005
Middle East & North Africa
346657300
13522.43
70.95
110
2005
North America
327892753
43503.58
77.76
111
2005

South Asia
1510864613
2814.01
64.80
112
2005
Sub-Saharan Africa
762002566
2489.81
52.71
113
2006
East Asia & Pacific
2099209976
8709.04
73.35
114
2006
Europe & Central Asia
873647564
22559.08
74.58
115
2006
Latin America & Caribbean
570029991
11080.95
73.15
116
2006
Middle East & North Africa

424CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

353917337

14537.90

71.19

117

2006

North America

331014940

45604.11

77.96

118

2006

South Asia

1534949611

3100.51

65.17

119

2006

Sub-Saharan Africa

782879376

2674.02

53.42

120

2007

East Asia & Pacific

2113722000

9647.46

73.59

121

2007

Europe & Central Asia

876646494

24145.43
74.93
122
2007
Latin America & Caribbean
577248307
11847.82
73.38
123
2007
Middle East & North Africa
361549578
15532.24
71.41
124
2007
North America
334184023
47212.52
78.25
125
2007
South Asia
1558685624
3384.91
65.55
126
2007
Sub-Saharan Africa
804469172
2858.93

426 *CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)*

54.17
127
2008
East Asia & Pacific
2128418789
10279.76
73.81
128
2008
Europe & Central Asia
880129893
25724.89
75.18
129
2008
Latin America & Caribbean
584435842
12429.16
73.60
130
2008
Middle East & North Africa
369452647
16276.55
71.64
131
2008
North America
337405012
47602.87
78.31

132
2008
South Asia
1582147481
3529.94
65.93
133
2008
Sub-Saharan Africa
826755847
2996.94
54.92
134
2009
East Asia & Pacific
2142725463
10685.27
74.03
135
2009
Europe & Central Asia
883770601
25014.61
75.56
136
2009
Latin America & Caribbean
591577623
12300.93
73.82
137

428CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

2009
Middle East & North Africa
377485338
16374.69
71.86
138
2009
North America
340465736
46187.76
78.66
139
2009
South Asia
1605443788
3768.17
66.31
140
2009
Sub-Saharan Africa
849703613
3034.71
55.64
141
2010
East Asia & Pacific
2156964851
11609.68
74.20
142
2010

Europe & Central Asia
887253276
25914.41
75.86
143
2010
Latin America & Caribbean
598662940
12992.65
74.04
144
2010
Middle East & North Africa
385417024
17046.69
72.07
145
2010
North America
343417261
47553.07
78.83
146
2010
South Asia
1628688562
4093.79
66.69
147
2010
Sub-Saharan Africa

430CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

873289791
3162.99
56.33
148
2011
East Asia & Pacific
2171288002
12489.94
74.36
149
2011
Europe & Central Asia
890976693
27502.65
76.39
150
2011
Latin America & Caribbean
605666731
13803.65
74.26
151
2011
Middle East & North Africa
393274154
17567.93
72.26
152
2011
North America
346126201

48967.52
78.91
153
2011
South Asia
1651888921
4376.24
67.07
154
2011
Sub-Saharan Africa
897503371
3299.59
56.96
155
2012
East Asia & Pacific
2185901380
13362.76
74.55
156
2012
Europe & Central Asia
893196694
28179.93
76.57
157
2012
Latin America & Caribbean
612609865
14306.19

432CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

74.49
158
2012
Middle East & North Africa
401032763
18184.56
72.45
159
2012
North America
348918897
50508.19
79.02
160
2012
South Asia
1675019307
4641.06
67.44
161
2012
Sub-Saharan Africa
917462006
3424.65
57.51
162
2013
East Asia & Pacific
2200670458
14239.15
74.74

163
2013
Europe & Central Asia
898996744
28703.22
76.80
164
2013
Latin America & Caribbean
619484449
14702.95
74.71
165
2013
Middle East & North Africa
408731939
18447.59
72.66
166
2013
North America
351647895
51822.57
79.13
167
2013
South Asia
1698093032
4942.42
67.79
168

434 *CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)*

2013

Sub-Saharan Africa

942816296

3555.31

58.04

169

2014

East Asia & Pacific

2215635403

15046.42

74.93

170

2014

Europe & Central Asia

902126822

29350.88

76.90

171

2014

Latin America & Caribbean

626256018

14721.38

74.94

172

2014

Middle East & North Africa

416401841

19838.18

72.85

173

2014

```

North America
354516198
53458.51
79.24
174
2014
South Asia
1721152580
5293.76
68.12
175
2014
Sub-Saharan Africa
968747541
3689.88
58.52

```

12.1 Bubble plot

We can now plot this new data using a variation of earlier single plot. The main differences are:

- aesthetics are applied to `ave_gdp_per_cap, ave_life_expect` and `totalPop` fields
- x and y labels refer to *Average GDP per Capita (USD)*, etc
- I added a `breaks` attribute to the `scale_x_log10` function so that more labelled units would show on the x axis (`trans_breaks`).
- The `geo_point` has a very small `alpha` value because many of these points are going to overlap. I want the overlap to be clearly perceived.
- I've explicitly set the `alpha` value of the colour in the legend to 0.5. Otherwise, it would have the same value as the `alpha` value of the `geom_point` - which was too faint

```

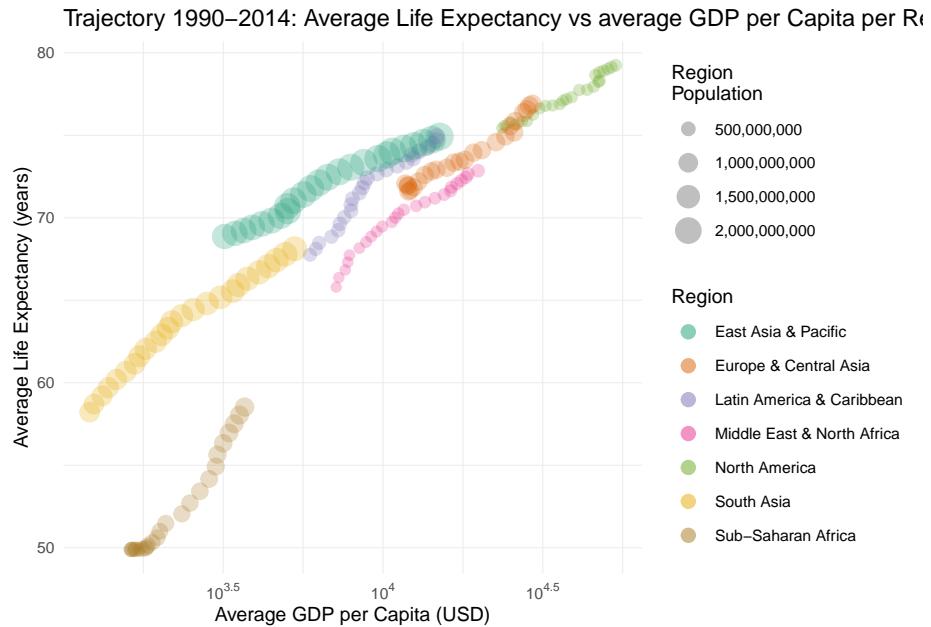
p.regions.ave <-
  ggplot(regions,
    aes(
      x = ave_gdp_percap,
      y = ave_life_expect,

```

436 CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

```
        size = totalPop,
        color = region
    )) +
  geom_point(na.rm = TRUE, alpha = 0.25) +
  ggtitle("Trajectory 1990-2014: Average Life Expectancy vs average GDP per Capita per"
  xlab("Average GDP per Capita (USD)") +
  ylab("Average Life Expectancy (years)") +
  # note how scale breaks are inserted
  scale_x_log10(
    labels = trans_format("log10", math_format(10 ^ .x)),
    breaks = scales::trans_breaks("log10", function(x)
      10 ^ x)
  ) +
  scale_color_brewer(palette = 'Dark2', name = "Region") +
  scale_size_area(max_size = 6,
    labels = comma,
    name = "Region \nPopulation") +
  #increase the size of the legend colour points, and their alpha value (too faint otherwise)
  guides(color = guide_legend(override.aes = list(size = 3, alpha = 0.5))) +
  theme_minimal(base_size = 10) +
  theme(panel.grid.major = element_line(size = 0.2),
    panel.grid.minor = element_line(size = 0.1))

plot(p.regions.ave)
```



12.1.1 Labelling the regions

Lets get rid of one of the legends. We will label one data point per region

Let's choose the data point in each region for the year 1990.

I will use the `ggrepel` library. Text labels repel away from each other, away from data points, and away from edges of the plotting area.

There is a trick to using `ggrepel` to make ensure that it doesn't place labels over *unlabelled* points : create a column called *label*. Give *every* point a blank label value ("") by default and then override this for the points that should actually have a label value.

`ggrepel` will not create a label for points with a blank label value, *but* will try to avoid overlapping them, treating as if they were in fact labelled.

```
regions$label <- "" # default is empty string

# my understanding of how ggrepel works is that it will repel labels away from empty labels as well
# just one label for each region in the year 1990
regions[regions$year==1990,]$label<- regions[regions$year==1990,]$region

library(ggrepel)

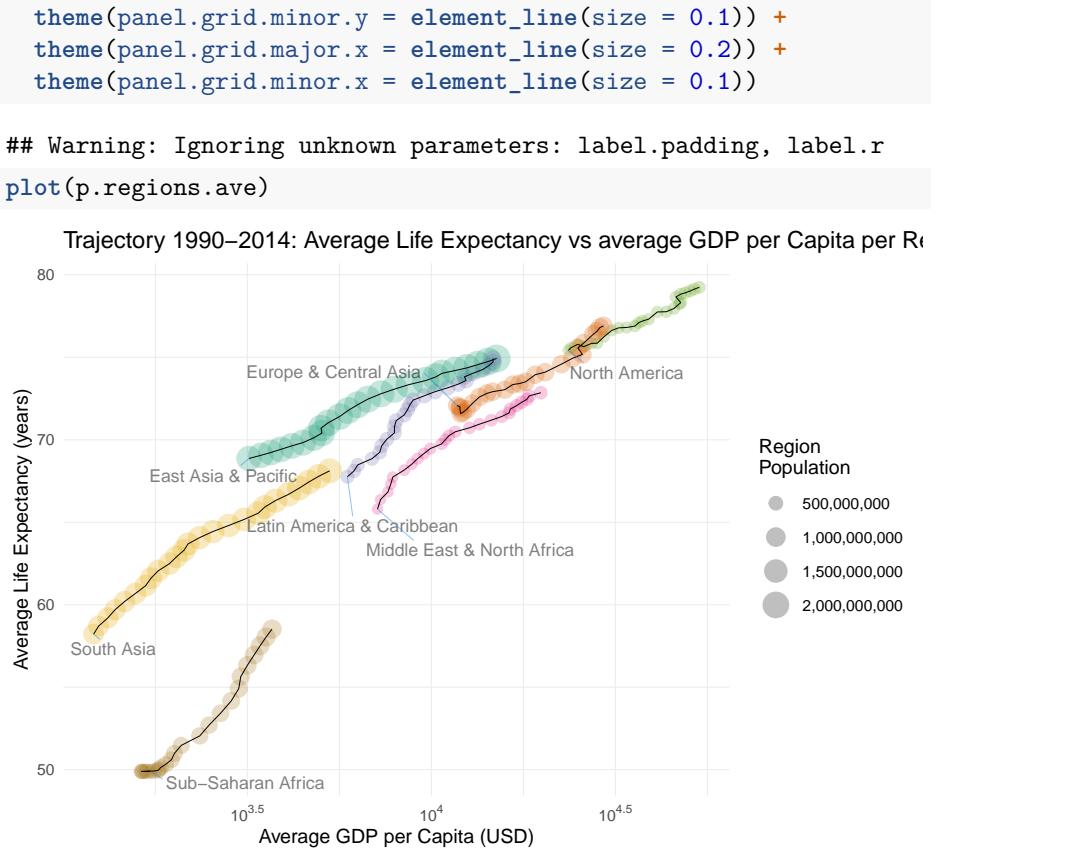
p.regions.ave <-
  ggplot(
```

```

regions,
aes(
  x = ave_gdp_percap,
  y = ave_life_expect,
  size = totalPop,
  color = region,
  group = region
)
) +
geom_point(na.rm = TRUE, alpha = 0.25) +
ggtitle("Trajectory 1990-2014: Average Life Expectancy vs average GDP per Capita per"
xlab("Average GDP per Capita (USD)") +
ylab("Average Life Expectancy (years)") +
# note how scale breaks are inserted
scale_x_log10(
  labels = trans_format("log10", math_format(10 ^ .x)),
  breaks = scales::trans_breaks("log10", function(x
    10 ^ x))
) +
scale_color_brewer(palette = 'Dark2', guide = FALSE) +
scale_size_area(max_size = 6,
  labels = comma,
  name = "Region \nPopulation") +

geom_path(size = 0.2,
  inherit.aes = T,
  colour = "black") +
#Using geom_label to add text labels
geom_text_repel(
  aes(label = label),
  size = 3,
  min.segment.length = unit(0, 'lines'),
  segment.size = 0.2,
  segment.color = "dodgerblue2",
  label.padding = unit(0.1, "lines"),
  force = 10,
  label.r = unit(0.15, "lines"),
  colour = "black",
  alpha = 0.5,
  na.rm = TRUE,
  show.legend = FALSE
) +
theme_minimal(base_size = 10) +
theme(panel.grid.major.y = element_line(size = 0.2)) +

```



12.1.2 Labelling time values

The labelling is mostly fine. However, There are a few problems with this plot

This is a time series plot and there is an implicit sequence of year values presented here - but not an indication of their starting or end values, or in fact the direction. In fact, each point does slightly more to the right from its predecessor - but there's no reason why GDP and Life Expectancy couldn't start to reverse.

In short, we need to label the time points - at least the start and end points.

```

# Create a label for the min and max year in each region

regions <- regions %>%
  filter(!is.na(ave_life_expect) & !is.na(ave_gdp_percap)) %>%
  group_by(region) %>% mutate(label = ifelse(year == min(year) |
                                             year == max(year) , year, ""))

```

p.regions.ave <-

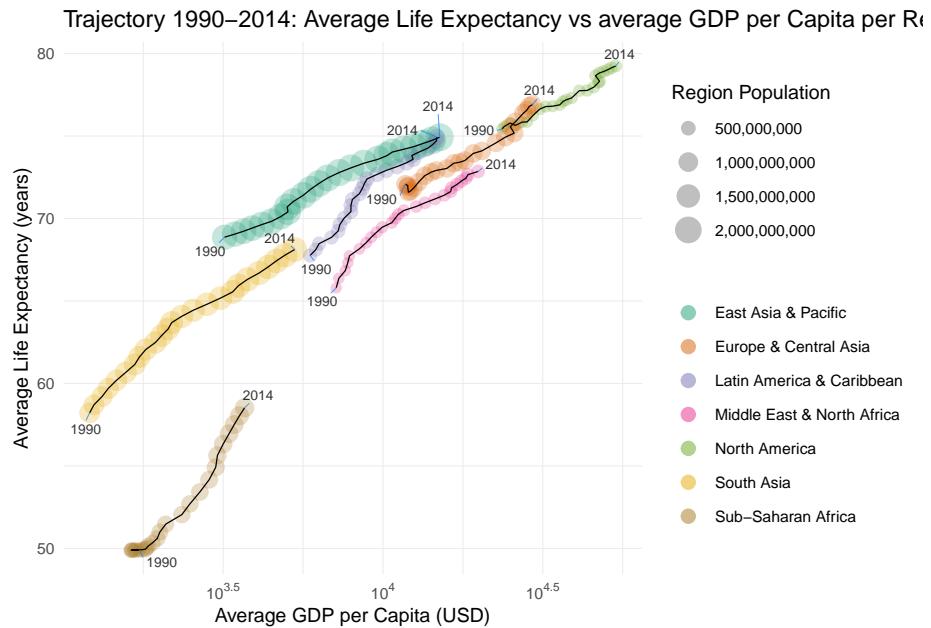
440CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

```

ggplot(
  regions,
  aes(
    x = ave_gdp_percap,
    y = ave_life_expect,
    size = totalPop,
    color = region,
    group = region
  )
) +
  geom_point(na.rm = TRUE, alpha = 0.25) +
  ggtitle("Trajectory 1990-2014: Average Life Expectancy vs average GDP per Capita per"
  xlab("Average GDP per Capita (USD)") +
  ylab("Average Life Expectancy (years)") +
  # note how scale breaks are inserted
  scale_x_log10(
    labels = trans_format("log10", math_format(10 ^ .x)),
    breaks = scales::trans_breaks("log10", function(x) 10 ^ x)) +
  scale_color_brewer(palette = 'Dark2', name = NULL) +
  scale_size_area(max_size = 6, labels = comma, name = "Region Population") +
  geom_path(size = 0.3, inherit.aes = T, colour="black") +
  geom_text_repel(
    aes(label = label),
    size = 2.5,
    min.segment.length = unit(0, 'lines'),
    segment.size = 0.2,
    segment.color = "dodgerblue2",
    colour = "black",
    force = 4,
    alpha = 0.8,
    na.rm = TRUE,
    show.legend = FALSE
  ) +
  theme_minimal(base_size = 10) +
  theme(panel.grid.major.y = element_line(size=0.2))++
  theme(panel.grid.minor.y = element_line(size=0.1))++
  theme(panel.grid.major.x = element_line(size=0.2))++
  theme(panel.grid.minor.x = element_line(size=0.1))++
  # increase the size of the legend colour points, and their alpha value (too faint otherwise)
  guides(color = guide_legend(override.aes = list(size = 3, alpha = 0.5)))

```

```
plot(p.regions.ave)
```



We've replaced the colour legend so as to allow the labelling of year values. I think that this is the preferred solution.

I have added a line through each set of regional points to emphasize the trends.

The geom I used for this is not `geom_line` which plots a line based on the sequence of x axis values. Instead I am using `geom_path` which plots a line according to the sequence of the points in the data set – which are ordered by year value ascending.

This is important because there may be situations that you plot where the point ordering is not clear

Here the line helps to emphasize the trend – and we can clearly see the drop in Life expectancy . In Europe in the early nineties – and the thickening of the line in the early 90s in Sub-Saharan Africa suggests the leftward direction towards lower GDP

12.2 A Faceted approach

However, there may be situations where you need to facet the graph. Because each panel represents a region, you no longer need colour to differentiate the series.

When faceting you inevitably lose some of the ease of comparison that a sin-

442CHAPTER 12. TUTORIAL: BUBBLE CHARTS OVER TIME (#BUBBLESERIES)

gle plot offers. However, the extra space per panel does allow you add extra information that might become too crowded in a single plot.

In the plot below, I've calculated the world mean per year for life expectancy and GDP. This acts as a reference line to explain the trends in each series.

For example, you can see that North America in 1990 was ahead of the 2014 world mean for GDP and Life Expectancy.

The GDP in South Asia in 2014 is about that of the 1990 world mean, though life expectancy in South Asia 2014 is higher than the world mean in 1990.

In this section of code, I calculate the world mean per year for life expectancy and GDP. I will plot this line and we will see what regions sit on either side of it

```
nations_mean_per_year<-nations_data %>%
  filter(!is.na(population))%>% # weighted.can't handle NAS in the weight vector
  select(country, region, year, population, gdp_per_cap, life_expect) %>%
  group_by(year) %>%
  summarize(totalPop = sum(population, na.rm = TRUE), ave_gdp_per_cap = weighted.mean(gdp_per_cap))

library(ggrepel)

p.regions.ave <-
  ggplot(
    regions,
    aes(
      x = ave_gdp_per_cap,
      y = ave_life_expect,
      size = totalPop,
      group = region
    )
  ) +
  geom_point(na.rm = TRUE, alpha = 0.15) +
  ggtitle("Trajectory 1990-2014: Average Life Expectancy vs average GDP per Capita per Region") +
  xlab("Average GDP per Capita (USD)") +
  ylab("Average Life Expectancy (years)") +
  # note how scale breaks are inserted
  scale_x_log10(
    labels = trans_format("log10", math_format(10 ^ .x)),
    breaks = scales::trans_breaks("log10", function(x) 10 ^ x)
  ) +
  scale_size_area(max_size = 4, labels = comma, name = "Region Population") +
  geom_path(data=nations_mean_per_year, aes(x = ave_gdp_per_cap,
                                             y = ave_life_expect), size = 0.2, colour = "red", inherit.aes = F) +
  geom_path(size = 0.2, inherit.aes = T, colour="black") +
```

```

geom_text_repel(
  aes(label = label),
  size = 2.5,
  min.segment.length = unit(0, 'lines'),
  segment.size = 0.15,
  colour = "black",
  segment.color = "dodgerblue2",
  force = 15,
  alpha = 0.8,
  na.rm = TRUE,
  show.legend = FALSE
) +
  theme_minimal(base_size = 10) +
  theme(panel.grid.major.y = element_line(size=0.2))+
  theme(panel.grid.minor.y = element_line(size=0.15))+
  theme(panel.grid.major.x = element_line(size=0.2))+
  theme(panel.grid.minor.x = element_line(size=0.15))+
  theme(axis.title.x = element_text(hjust = -0.01))+
  theme(legend.title = element_text(size = 8))+
  theme(legend.box.background = element_rect(colour = "grey", size=0.2))+  

  # increase the size of the legend colour points, and their alpha value (too faint otherwise)
  guides(color = guide_legend(override.aes = list(size = 3, alpha = 0.5))) +
  facet_wrap(vars(region))

```

Somebody posted a nice solution on Stack exchange to positioning the legend in a faceted plot.

```

https://stackoverflow.com/questions/54438495/shift-legend-into-empty-facets-of-a-faceted-plot-in-ggplot2
shift_legend2 <- function(p, position) {
  # ...
  # to grob
  gp <- ggplotGrob(p)
  facet.panels <- grep("^panel", gp[["layout"]][["name"]])
  empty.facet.panels <- sapply(facet.panels, function(i) "zeroGrob" %in% class(gp[["grobs"]][[i]]))
  empty.facet.panels <- facet.panels[empty.facet.panels]

  # establish name of empty panels
  empty.facet.panels <- gp[["layout"]][empty.facet.panels, ]
  names <- empty.facet.panels$name
  # example of names:
  #[1] "panel-3-2" "panel-3-3"

  # now we just need a simple call to reposition the legend

```

```

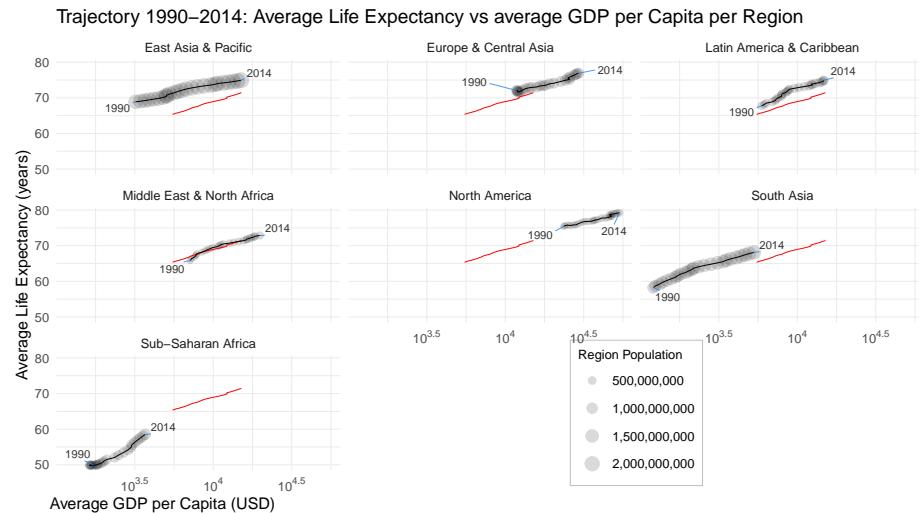
    reposition_legend(p, position, panel=names)
}

library(ggplot2)
library(gtable)
library(lemon)

## 
## Attaching package: 'lemon'

## The following object is masked from 'package:purrr':
## 
##     %||%
shift_legend2(p.regions.ave, "center")

```



12.3 Observations

This plot summarises many of the aspects of the data set we are analysing.

- Each region leaves a trail as it progresses across the GDP vs Life Expectancy plane.
- The destination for each region naturally is high GDP per capita and high life expectancy
- The trails show the starting point and end points, and the relative speed of movement of each region
- Each point has a low `alpha` value which means that when points from the same region overlap, the points are darker. This is a visual metaphor for a region that is moving slowly or has stalled. For example, South Asia leaves a long, uniformly light trail indicating steady growth. Europe & Central

Asia and North America have a journey that is shorter and has periods of stagnation, indicated by the dark colours of multiple point overlaps

- The size of the points indicate very well that a small proportion of the world's population enjoys the twin benefits of high GDP and high life expectancy - but it is clear that other regions are rapidly accelerating in this direction.

Chapter 13

Time Data as Ordinal Values

13.1 Scottish Independence

The Brexit campaign in the UK over the past two years has re-awoken the geo-political debate in Scotland over independence. As the UK prepared to leave the EU, successive opinion polls in Scotland showed no clear majority for a yes or a no vote for independent.

The Scottish first minister, Nicola Sturgeon, is trying to convince the UK government to allow Scotland hold a referendum on independence. In discussing this, many news articles have referenced opinion polls where there is no clear trend one way or the other.

We will look at producing a visualisation that shows the closeness of the two positions on independence: yes vs no, based on several recent surveys.

13.2 Read the data

```
library(readr)
library(dplyr)
library(tidyr)
library(ggplot2)
library(lubridate)
library(kableExtra)

# read in two data sets - one for 2018 and one for 2019
scot.ind<-read_csv("../data/2020_scottish_independence.csv")
```

```
# make the date field into a date object and order the dataset by date
scot.ind<-scot.ind%>%mutate(date = dmy(date))%>%arrange(desc(date))

kable(scot.ind, digits = 2, format = "html", row.names = FALSE) %>%
  kable_styling(
    full_width = F,
    font_size = 12,
    position = "center")%>%
  scroll_box(height = "200px")
```

date
pollster
yes
no
2020-02-03
Survation
50
50
2020-01-31
Panelbase
52
48
2020-01-27
YouGov
51
49
2020-01-22
Survation
50
50
2019-12-11
Survation
49
51

2019-11-25

Ipsos Mori

50

50

2019-11-22

Panelbase

49

51

13.3 Convert to long format

Convert the data to long format and plot

```
scot.ind.l <- scot.ind%>%
  gather(vote, value, yes:no)%>%
  arrange(desc(date))

kable(
  scot.ind.l,
  digits = 2,
  format = "html",
  row.names = FALSE
) %>%
  kable_styling(full_width = F,
                font_size = 12,
                position = "center")%>%
  scroll_box(height = "400px")
```

date

pollster

vote

value

2020-02-03

Survation

yes

50

2020-02-03

Survation

no

50

2020-01-31

Panelbase

yes

52

2020-01-31

Panelbase

no

48

2020-01-27

YouGov

yes

51

2020-01-27

YouGov

no

49

2020-01-22

Survation

yes

50

2020-01-22

Survation

no

50

2019-12-11

Survation

yes

49

2019-12-11

Survation

no

51

2019-11-25

Ipsos Mori

yes

50

2019-11-25

Ipsos Mori

no

50

2019-11-22

Panelbase

yes

49

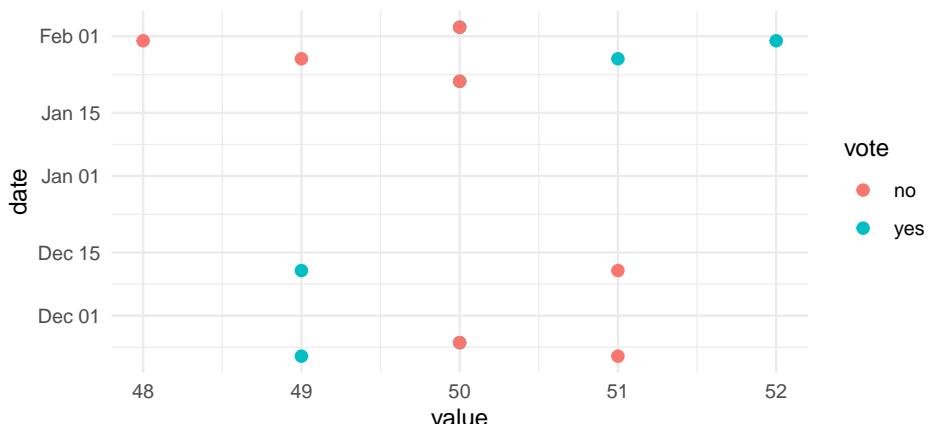
2019-11-22

Panelbase

no

51

```
ggplot(scot.ind.l, aes(x=value, y=date, colour=vote)) +  
  geom_point(size = 2.2) +  
  theme_minimal()
```



ggplot treated the data as it would time series data by spacing the points according to the interval between timestamps. However, the time stamps are not evenly spaced. E.g. there is a big gap between the Survation poll on Dec 11th and the following Survation poll on Jan 22nd. This is followed by a gap of just 5 days to the YouGov poll on January 27th. This is typical of polling data. It is carried out by independent organisations - and the polls are generally not synchronised.

This data might be called ordinal time data instead of Time Series data. The order of the data matters is of primary importance; the time stamp and the interval between time stamps are of less important.

To achieve this we have to reassign the variable assigned to the y-axis. We have to create a dummy ordinal variable that reflects the order of the dates. The y-axis scale will represent the values of this variable. The dates will be used as labels.

In the original wide form data set, I create a new variable called *seq*, which has sequence of evenly spaced dummy values. The variable is an ordered factor.

Then I transform the data to long format as before for plotting with ggplot.

Now we can attempt a basic plot again

```
# insert sequence of values - this is what we wil use on the y axis
scot.ind$seq <- factor(nrow(scot.ind):1, levels=1:nrow(scot.ind))
scot.ind <- scot.ind %>%
  select(seq, everything()) %>%
  arrange(seq) # brings seq to front and orders by seq

kable(scot.ind, digits = 2, format = "html", row.names = FALSE, caption = "The data wi
```

The data with the seq dummy variable

```
seq
date
pollster
yes
no
1
2019-11-22
Panelbase
```

49
51
2
2019-11-25

Ipsos Mori

50
50
3
2019-12-11

Survation

49
51
4
2020-01-22

Survation

50
50
5
2020-01-27

YouGov

51
49
6
2020-01-31

Panelbase

52
48
7
2020-02-03

Survation

50

```
50
```

```
# convert to long format
scot.ind.l <- scot.ind %>%
  gather(vote, value, yes:no) %>%
  arrange(seq)

kable(head(scot.ind.l), digits = 2, format = "html", row.names = FALSE, caption = "The
kable_styling(
  full_width = F,
  font_size = 12,
  position = "center")
```

The data in long format

```
seq
date
pollster
vote
value
1
2019-11-22
Panelbase
yes
49
1
2019-11-22
Panelbase
no
51
2
2019-11-25
Ipsos Mori
yes
50
2
```

2019-11-25

Ipsos Mori

no

50

3

2019-12-11

Survation

yes

49

3

2019-12-11

Survation

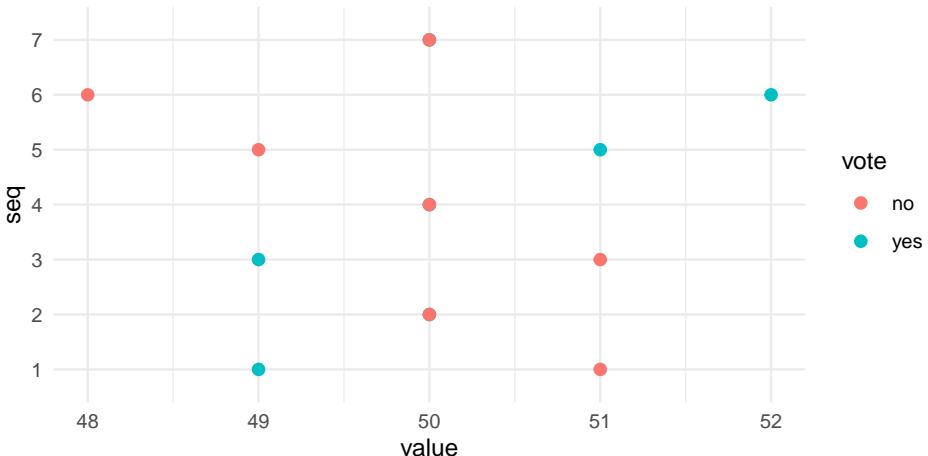
no

51

Now we can attempt a basic plot again. We can see now that the points are evenly distributed according to the y-axis values.

```
g<- ggplot(scot.ind.1, aes(x=value, y=seq, colour=vote)) +
  geom_point(size = 2.2) +
  theme_minimal()
```

g

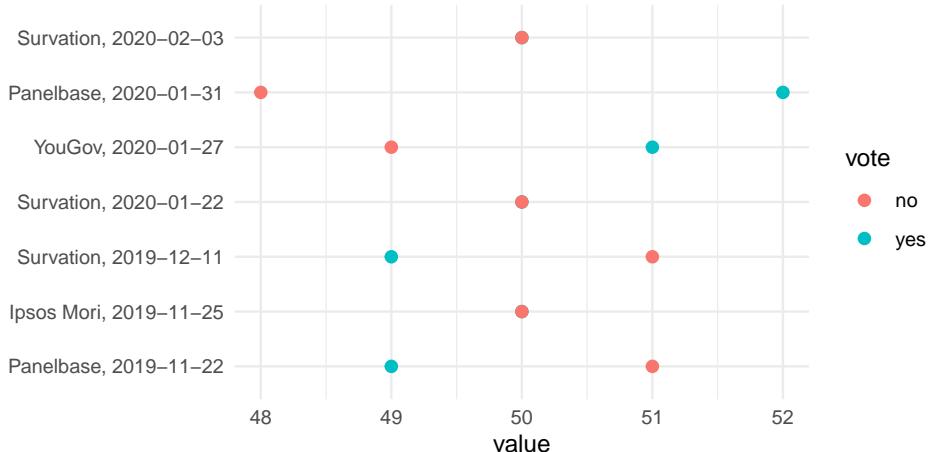


However The y-axis shows the sequence values and it should show the date and pollster.

We can set the labels we want using the `scale_y_discrete` function or override the axis settings for the the y axis. We use `scale_y_discrete` because you may recall that the `seq` variable was created as a ordered factor, not an integer.

The labels we will show are the data and pollster values in the data in its wide format. When you are imposing labels on a plot from an external data set, you need to make sure that the labels are ordered correctly. You may have noticed that I used the `arrange` function on the data sets in both wide and long form - this was to make sure that the instances were in the same ordering by date.

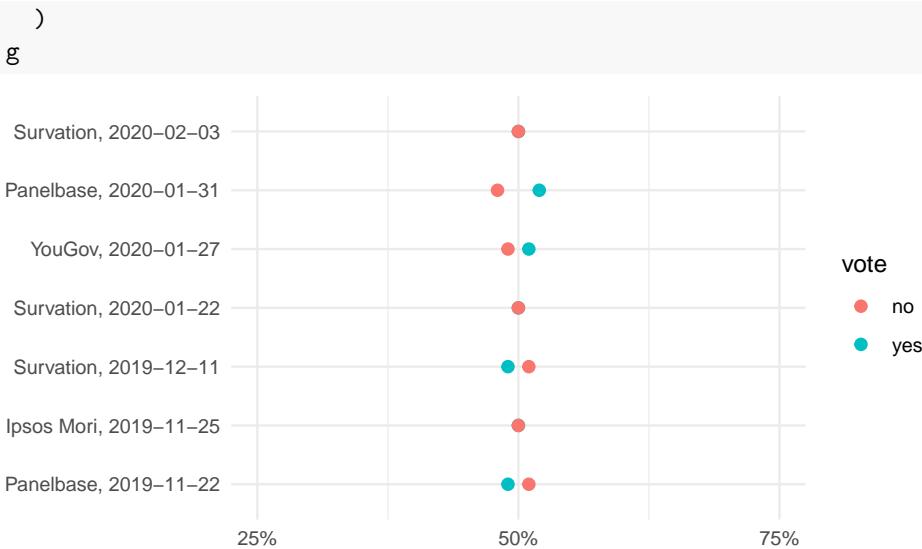
```
g <- g +
  scale_y_discrete(labels = paste0(scot.ind$pollster, ", ", scot.ind$date),
                    name = NULL)
g
```



The x axis by default has take the min and max values assigned to it and set these as the axis limits. However, it gives the impression of a much larger spread between *yes* and *no* values than is the case.

We are going to set the limit to between 25% and 75% to emphasise how close the polling values are. We override the x axis defaults by calling `scale_x_continuous` and setting the limits to between 25 and 75; We set the breaks to c(25,50 and 75) and the labels to include a % sign after the break values

```
g <- g +
  scale_x_continuous(
    breaks = c(25, 50, 75),
    limits = c(25, 75),
    labels = paste0(c(25, 50, 75), "%"),
    name = NULL)
```



This brings us very close to what we wanted to achieve. There are still a few modifications still.

- I want the date names to be represented in abbreviated form
- I want to override the default colours
- Where yes and no votes are tied at 50%, I want to show two points at that position. Currently, the no vote covers the yes vote completely at that 50% mark.

The lubridate library is a very useful library for manipulating date data in R,l

I use the `month` and `mday` functions from the `lubridate` library to extract out the abbreviated month string and the day of the month from the data object.

I specify two colours for the *no* and *yes* votes – and I override the default colours using the `scale_fill_manual` function

The final modification requires us to slightly nudge apart the values at the 50% value so that the reader can see that there are two points represented – a *yes* and a *no* value.

I could use `geom_jitter` which introduces some positional variation – but this is really only useful when we have a lot of points close together. Also , a slight variation in this plot could indicate a value of 51% or 49% to the reader

Instead, the strategy is:

- Plot all points not equal to 50% as before
- For yes points =50% change their y value position upwards slightly
- For no points = 50% change their y value position downwards by the same amount

```

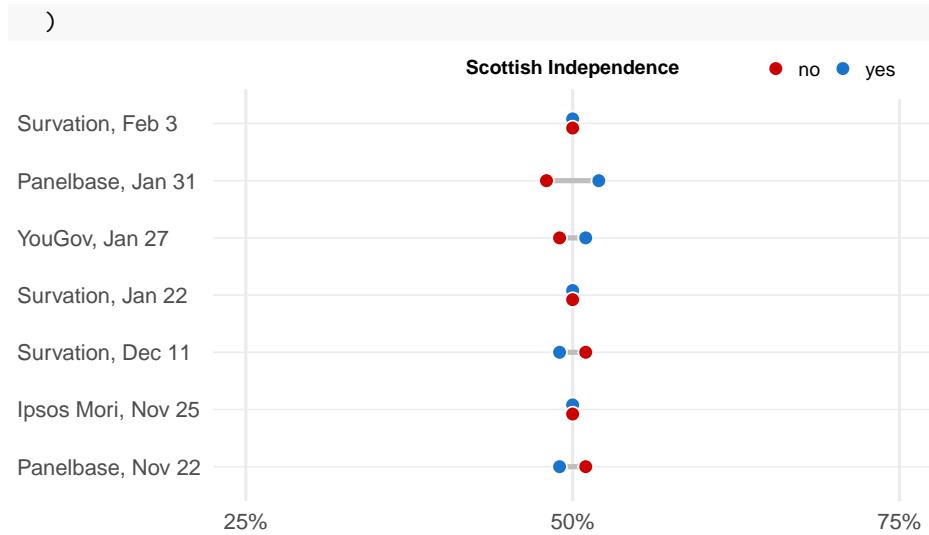
library(dplyr)

colours <- c("red3", "dodgerblue3")

ggplot(scot.ind.l, aes(
  x = value,
  y = seq,
  fill = vote,
  group = date
)) +
  geom_line(aes(group = seq), colour = "grey", size = 1) +
  # for values not equal to 50% plot as before
  geom_point(
    data = scot.ind.l %>% filter(value != 50),
    size = 2.5,
    shape = 21,
    colour = "white"
  ) +
  # for yes votes at 50%, nudge upwards
  geom_point(
    data = scot.ind.l %>% filter(vote == "yes" &
                                    value == 50),
    fill = colours[2],
    position = position_nudge(x = 0, y = 0.08),
    size = 2.5,
    shape = 21,
    colour = "white"
  ) +
  # for no votes at 50%, nudge upwards
  geom_point(
    data = scot.ind.l %>% filter(vote == "no" &
                                    value == 50),
    fill = colours[1],
    position = position_nudge(x = 0, y = -0.08),
    size = 2.5,
    shape = 21,
    colour = "white"
  ) +
  scale_y_discrete(labels = paste0(
    scot.ind$pollster,
    ", ",
    month(scot.ind$date, label = TRUE, abbr = TRUE),
    " "
  ))

```

```
" " ,
  mday(scot.ind$date)
),
name = NULL) +
scale_x_continuous(
  breaks = c(25, 50, 75),
  limits = c(25, 75),
  labels = paste0(c(25, 50, 75), "%"),
  name = NULL
) +
# add custom colours
scale_fill_manual(values = colours, name = NULL) +
ggtitle("Scottish Independence") +
theme_minimal() +
theme(
  axis.text.y = element_text(hjust = 0),
  panel.grid.major.y = element_line(size = 0.3),
  panel.grid.major.x = element_line(size = 0.6),
  panel.grid.minor.x = element_blank(),
  plot.margin = unit(c(1, 1, 1, 1), "cm"),
  plot.title = element_text(
    size = 8,
    face = "bold",
    vjust = 0.4,
    hjust = 0.5
  )
) +
theme(
  legend.text = element_text(size = 8),
  legend.title = element_blank(),
  legend.position = c(.85, 1.05),
  # move to the top
  legend.direction = "horizontal",
  # make it horizontal
  legend.key.size = unit(0.9, "line"),
  legend.spacing.x = unit(0.1, 'cm'),
  legend.background = element_rect(
    fill = "white",
    size = 0.5,
    colour = "white"
)
```



Chapter 14

Slope Charts

The Slope chart was introduced by Edward Tufte in his 1983 book The Visual Display of Quantitative Information.

He didn't give the chart a name then and it has only fairly recently acquired the name *slope chart*.

Tufte suggests that this type of chart is good for seeing

- The ranking of categories between time stamps
- The specific numbers associated with category in each time stamp
- How each categories numbers changed over time (the slope)
- How each categories 's rate of change compares to the other categories' rates of change (how the slopes compare)
- Notable deviations in terms of slope

To examine slope graphs we are going to visualise some data collected by an organisation called Transparency International (TI) Ireland, which is the Irish chapter of the Transparency International, a global organisation dedicated to exposing corruption worldwide.

14.1 Reading in the two data sets

```
library(readr)
library(dplyr)
library(tidyr)
library(kableExtra)

# read in two data sets - one for 2018 and one for 2019
ire.councils.2018<-read_csv("../data/ire-transparency-Ireland-councils-2018.csv")
ire.councils.2019<-read_csv("../data/ire-transparency-Ireland-councils-2019.csv")
```

```
kable(ire.councils.2018, digits = 2, format = "html", row.names = FALSE, caption = '<b>2018 National Integrity Index</b>')
  kable_styling(
    full_width = F,
    font_size = 12,
    position = "center")%>%
  scroll_box(height = "300px")
```

National Integrity Index 2018

rank

council

score-out-of-30

short

1

Galway City Council

21

Galway City

2

Fingal County Council

19

Fingal

2

South Dublin County Council

19

South Dublin

4

Dublin City Council

18

Dublin City

4

Monaghan County Council

18

Monaghan

6

Dún Laoghaire–Rathdown County Council

17

Dún Laoghaire–Rathdown

6

Laois County Council

17

Laois

6

Meath County Council

17

Meath

9

Kerry County Council

16

Kerry

9

Kildare County Council

16

Kildare

11

Clare County Council

15

Clare

11

Cork City Council

15

Cork City

11

Limerick City and County Council

15

Limerick City/County
11

Roscommon County Council
15

Roscommon
11

Tipperary County Council
15

Tipperary
16

Leitrim County Council
14

Leitrim
17

Donegal County Council
13

Donegal
17

Longford County Council
13

Longford
19

Cavan County Council
12

Cavan
19

Louth County Council
12

Louth
19

Mayo County Council

12	
Mayo	
19	
Sligo County Council	
12	
Sligo	
19	
Wicklow County Council	
12	
Wicklow	
24	
Cork County Council	
11	
Cork	
24	
Kilkenny County Council	
11	
Kilkenny	
24	
Westmeath County Council	
11	
Westmeath	
27	
Carlow County Council	
10	
Carlow	
28	
Offaly County Council	
9	
Offaly	
28	

Waterford City and County Council

9

Waterford City/County

30

Wexford County Council

7

Wexford

31

Galway County Council

5

Galway

```
kable(ire.councils.2019, digits = 2, format = "html", row.names = FALSE, caption = 'National Integrity Index 2019' ) %>%  
  kable_styling(  
    full_width = F,  
    font_size = 12,  
    position = "center") %>%  
  scroll_box(height = "300px")
```

National Integrity Index 2019

rank

council

score-out-of-30

short

1

Fingal County Council

22

Fingal

1

South Dublin County Council

22

South Dublin

3

Monaghan County Council

21	
Monaghan	
4	
Kildare County Council	
20	
Kildare	
5	
Dublin City Council	
19	
Dublin City	
5	
Kilkenny County Council	
19	
Kilkenny	
5	
Meath County Council	
19	
Meath	
5	
Wexford County Council	
19	
Wexford	
5	
Wicklow County Council	
19	
Wicklow	
10	
Clare County Council	
18	
Clare	
10	

- Donegal County Council
18
- Donegal
10
- Galway City Council
18
- Galway City
10
- Tipperary County Council
18
- Tipperary
14
- Cavan County Council
17
- Cavan
14
- Dún Laoghaire–Rathdown County Council
17
- Dún Laoghaire–Rathdown
14
- Limerick City and County Council
17
- Limerick City/County
17
- Sligo County Council
16
- Sligo
18
- Cork City Council
15
- Cork City

18

Cork County Council

15

Cork

18

Galway County Council

15

Galway

18

Roscommon County Council

15

Roscommon

18

Waterford City and County Council

15

Waterford City/County

23

Carlow County Council

14

Carlow

23

Laois County Council

14

Laois

23

Leitrim County Council

14

Leitrim

23

Louth County Council

14

Louth
27
Longford County Council
13
Longford
27
Mayo County Council
13
Mayo
27
Offaly County Council
13
Offaly
30
Kerry County Council
12
Kerry
30
Westmeath County Council
12
Westmeath

14.2 Merging the two data sets

The easiest way to do this is to first add a *year* field to each data set and then append one after the other using `rbind`.

I will also rename the *score-out-of-30* field to *score* and will calculate a *percent* field.

I make *year* into factor, with the default levels (2018, 2019)

```
ire.councils.2018$year <- '2018'
ire.councils.2019$year <- '2019'

ire.councils.18.19<-rbind(ire.councils.2018, ire.councils.2019)
```

```
ire.councils.18.19<-ire.councils.18.19%>%
  rename(
    score = `score-out-of-30`%
  )%>%
  mutate(percent= round(100*score/30,0))%>%
  mutate(year=factor(year))

kable(ire.councils.18.19, digits = 2, format = "html", row.names = FALSE, caption = '<b>National
  kable_styling(
    full_width = F,
    font_size = 12,
    position = "center")%>%
  scroll_box(height = "300px")
```

National Integrity Index 2018 - 2019

rank

council

score

short

year

percent

1

Galway City Council

21

Galway City

2018

70

2

Fingal County Council

19

Fingal

2018

63

2

South Dublin County Council
19
South Dublin
2018
63
4
Dublin City Council
18
Dublin City
2018
60
4
Monaghan County Council
18
Monaghan
2018
60
6
Dún Laoghaire–Rathdown County Council
17
Dún Laoghaire–Rathdown
2018
57
6
Laois County Council
17
Laois
2018
57
6
Meath County Council

17
Meath
2018
57
9
Kerry County Council
16
Kerry
2018
53
9
Kildare County Council
16
Kildare
2018
53
11
Clare County Council
15
Clare
2018
50
11
Cork City Council
15
Cork City
2018
50
11
Limerick City and County Council
15

Limerick City/County
2018
50
11
Roscommon County Council
15
Roscommon
2018
50
11
Tipperary County Council
15
Tipperary
2018
50
16
Leitrim County Council
14
Leitrim
2018
47
17
Donegal County Council
13
Donegal
2018
43
17
Longford County Council
13
Longford

2018
43
19
Cavan County Council
12
Cavan
2018
40
19
Louth County Council
12
Louth
2018
40
19
Mayo County Council
12
Mayo
2018
40
19
Sligo County Council
12
Sligo
2018
40
19
Wicklow County Council
12
Wicklow
2018

40

24

Cork County Council

11

Cork

2018

37

24

Kilkenny County Council

11

Kilkenny

2018

37

24

Westmeath County Council

11

Westmeath

2018

37

27

Carlow County Council

10

Carlow

2018

33

28

Offaly County Council

9

Offaly

2018

30

28	
Waterford City and County Council	
9	
Waterford City/County	
2018	
30	
30	
Wexford County Council	
7	
Wexford	
2018	
23	
31	
Galway County Council	
5	
Galway	
2018	
17	
1	
Fingal County Council	
22	
Fingal	
2019	
73	
1	
South Dublin County Council	
22	
South Dublin	
2019	
73	
3	

Monaghan County Council
21
Monaghan
2019
70
4
Kildare County Council
20
Kildare
2019
67
5
Dublin City Council
19
Dublin City
2019
63
5
Kilkenny County Council
19
Kilkenny
2019
63
5
Meath County Council
19
Meath
2019
63
5
Wexford County Council

19

Wexford

2019

63

5

Wicklow County Council

19

Wicklow

2019

63

10

Clare County Council

18

Clare

2019

60

10

Donegal County Council

18

Donegal

2019

60

10

Galway City Council

18

Galway City

2019

60

10

Tipperary County Council

18

Tipperary
2019
60
14
Cavan County Council
17
Cavan
2019
57
14
Dún Laoghaire–Rathdown County Council
17
Dún Laoghaire–Rathdown
2019
57
14
Limerick City and County Council
17
Limerick City/County
2019
57
17
Sligo County Council
16
Sligo
2019
53
18
Cork City Council
15
Cork City

2019
50
18
Cork County Council
15
Cork
2019
50
18
Galway County Council
15
Galway
2019
50
18
Roscommon County Council
15
Roscommon
2019
50
18
Waterford City and County Council
15
Waterford City/County
2019
50
23
Carlow County Council
14
Carlow
2019

47

23

Laois County Council

14

Laois

2019

47

23

Leitrim County Council

14

Leitrim

2019

47

23

Louth County Council

14

Louth

2019

47

27

Longford County Council

13

Longford

2019

43

27

Mayo County Council

13

Mayo

2019

43

27
Offaly County Council
13
Offaly
2019
43
30
Kerry County Council
12
Kerry
2019
40
30
Westmeath County Council
12
Westmeath
2019
40

14.3 Selecting the data

Including all councils may make this plot look a little cluttered

What councils do we want to compare? Obviously Galway City Council and Galway County Council.

We will compare them to the top 5 councils in 2018 (which includes Galway City) and in 2019. And we will include the bottom five councils in 2018 (which includes Galway county) and in 2019.

This means we have to create a subset of our data.

```
# top 5 councils in 2018
ire.councils.18.19%>%
  filter(year=='2018')%>%
  arrange(desc(score)) %>%
  top_n(n = 5, wt = score)%>%
  select(short) -> top2018
```

```

# bottom 5 councils in 2018
ire.councils.18.19%>%
  filter(year=='2018')%>%
  arrange(desc(score)) %>%
  top_n(n = -5, wt = score)%>%
  select(short) -> bottom2018

# top 5 councils in 2019
ire.councils.18.19%>%
  filter(year=='2019')%>%
  arrange(desc(score)) %>%
  top_n(n = 5, wt = score)%>%
  select(short) -> top2019

# bottom 5 councils in 2019
ire.councils.18.19%>%
  filter(year=='2019')%>%
  arrange(desc(score)) %>%
  top_n(n = -5, wt = score)%>%
  select(short) -> bottom2019

# bind these data frames with the list of councils we want
councils <- rbind(top2018,bottom2018, top2019, bottom2019)
# remove duplicates (e.g some councils are in the top 5 in 2018 and 2019)
councils <-unique(unlist(as.list(councils)))

# create a subset of councils
ire.councils.18.19.subset<- ire.councils.18.19%>%
  filter(short %in% councils )

# show as a table
kable(ire.councils.18.19.subset, digits = 2, format = "html", row.names = FALSE, caption =
  kable_styling(
    full_width = F,
    font_size = 12,
    position = "center")%>%
  scroll_box(height = "300px")

```

Subset of councils we will show in the slope chart

rank

council

score

short
year
percent
1
Galway City Council
21
Galway City
2018
70
2
Fingal County Council
19
Fingal
2018
63
2
South Dublin County Council
19
South Dublin
2018
63
4
Dublin City Council
18
Dublin City
2018
60
4
Monaghan County Council
18
Monaghan

2018
60
6
Meath County Council
17
Meath
2018
57
9
Kerry County Council
16
Kerry
2018
53
9
Kildare County Council
16
Kildare
2018
53
17
Longford County Council
13
Longford
2018
43
19
Mayo County Council
12
Mayo
2018

40
19
Wicklow County Council
12
Wicklow
2018
40
24
Kilkenny County Council
11
Kilkenny
2018
37
24
Westmeath County Council
11
Westmeath
2018
37
27
Carlow County Council
10
Carlow
2018
33
28
Offaly County Council
9
Offaly
2018
30

28

Waterford City and County Council

9

Waterford City/County

2018

30

30

Wexford County Council

7

Wexford

2018

23

31

Galway County Council

5

Galway

2018

17

1

Fingal County Council

22

Fingal

2019

73

1

South Dublin County Council

22

South Dublin

2019

73

3

Monaghan County Council

21

Monaghan

2019

70

4

Kildare County Council

20

Kildare

2019

67

5

Dublin City Council

19

Dublin City

2019

63

5

Kilkenny County Council

19

Kilkenny

2019

63

5

Meath County Council

19

Meath

2019

63

5

Wexford County Council

19

Wexford

2019

63

5

Wicklow County Council

19

Wicklow

2019

63

10

Galway City Council

18

Galway City

2019

60

18

Galway County Council

15

Galway

2019

50

18

Waterford City and County Council

15

Waterford City/County

2019

50

23

Carlow County Council

14

Carlow
2019
47
27
Longford County Council
13
Longford
2019
43
27
Mayo County Council
13
Mayo
2019
43
27
Offaly County Council
13
Offaly
2019
43
30
Kerry County Council
12
Kerry
2019
40
30
Westmeath County Council
12
Westmeath

2019

40

14.4 Building a basic slope chart

Now we can create a basic slope chart, plotting the percentage values per year for our subset of councils.

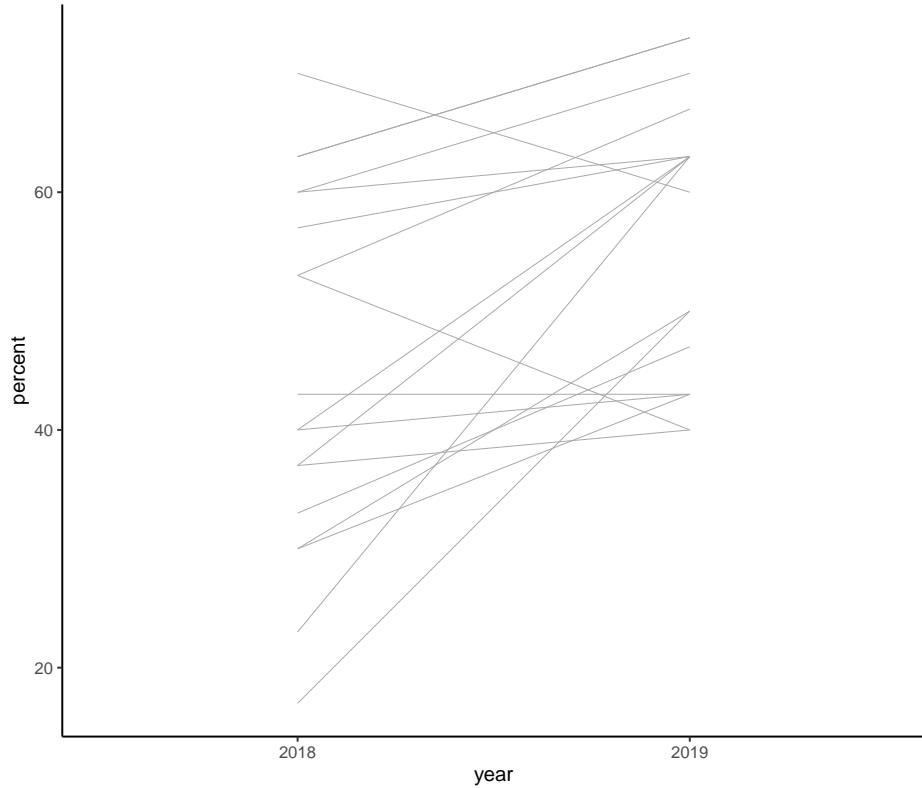
Two lines of ggplot code will show the basic anatomy of the plot

```
library(ggplot2)
library(ggrepel)

slope_data <- ire.councils.18.19.subset

g<-ggplot(slope_data, aes(x = year, y = percent, group = short)) +
  geom_line(size = 0.25, colour = "darkgrey")

g
```



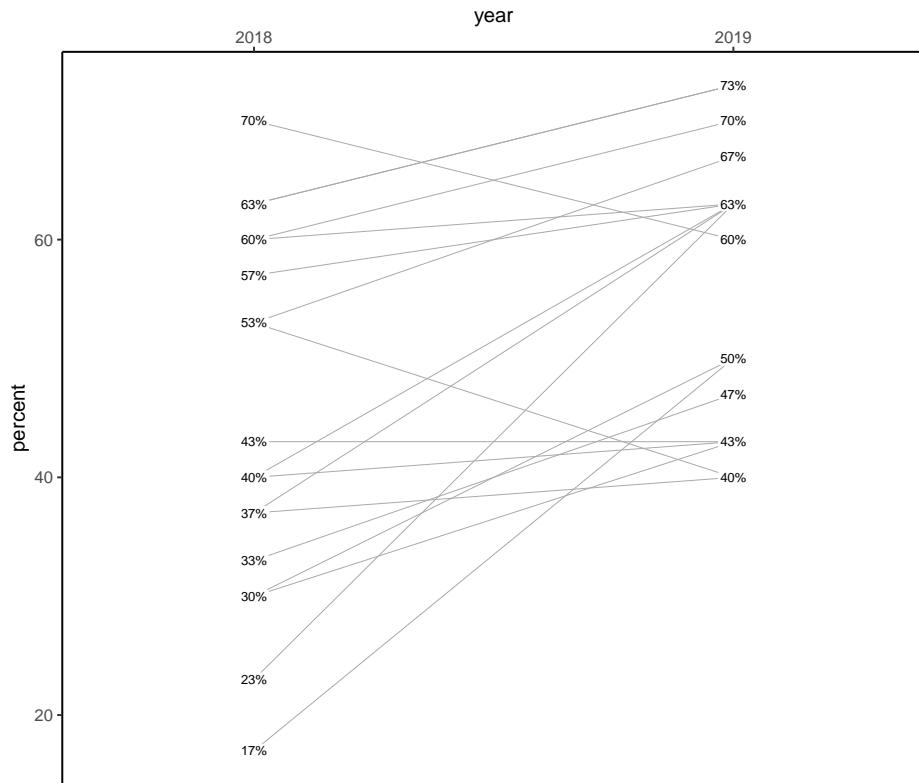
The x-axis uses a discrete scale to represent the *year* variable (a factor). It has two values : *2018* and *2019*

Here we also specify that the x axis is on top of the plot. This will move the axis text (for 2018 and 2019) to the top also.

The expand function expands distance between the first and last x values and the left and right edges of the graph. This leaves room for the labels for the council names

First we will display the line of values for each year. These are simply `geom_label` objects positioned on the plot according to the (year, percentage) value of each point

```
g <- g+
  scale_x_discrete(position = "top", expand=c(0.2, 0.2)) +
  geom_label(aes(label = paste0(percent, "%")),
             size = 2.5,
             label.padding = unit(0.085, "lines"),
             label.size = 0.0)
g
```

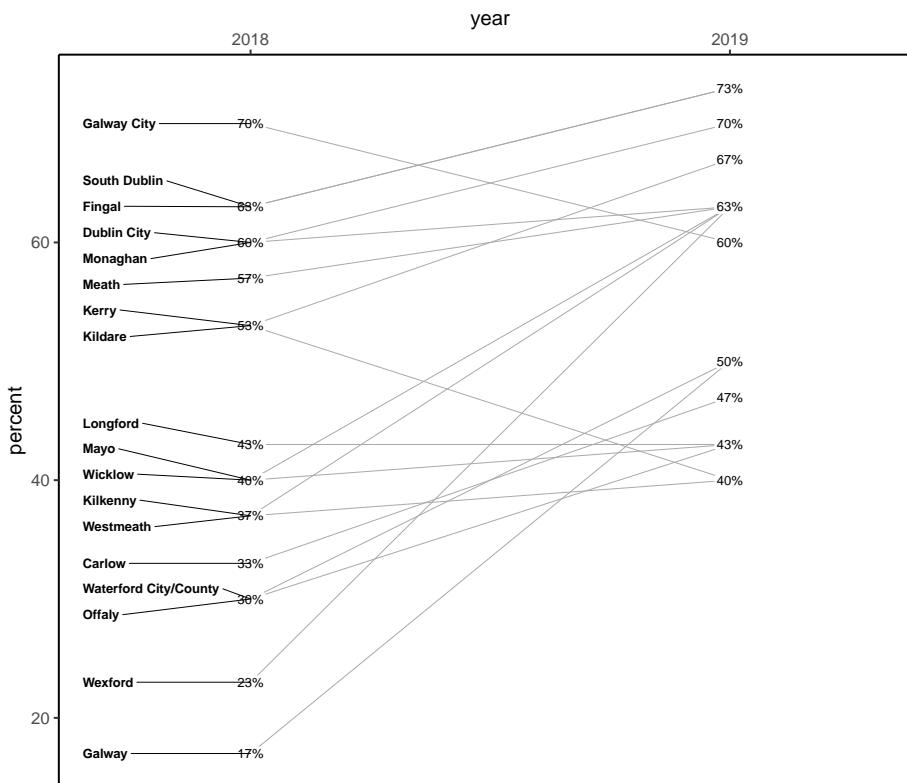


Now add the council name labels on the left (for the points with year ==‘2018’). These will be `geom_text_repel` objects because ordinary labels `geom_label` objects will overlap where councils have similar percentage values. The `geom_text_repel` objects will avoid overlapping each other and will automatically include a line segment connecting the label to the point where necessary. We’ve left justified these labels for readability.

```
library(ggplot2)
library(ggrepel)

g <- g +
  geom_text_repel(
    data = ire.councils.18.19.subset %>% filter(year == "2018"),
    aes(label = short) ,
    hjust = "left",
    fontface = "bold",
    size = 2.5,
    nudge_x = -0.35,
    segment.size = 0.05,
    direction = "y"
  )
```

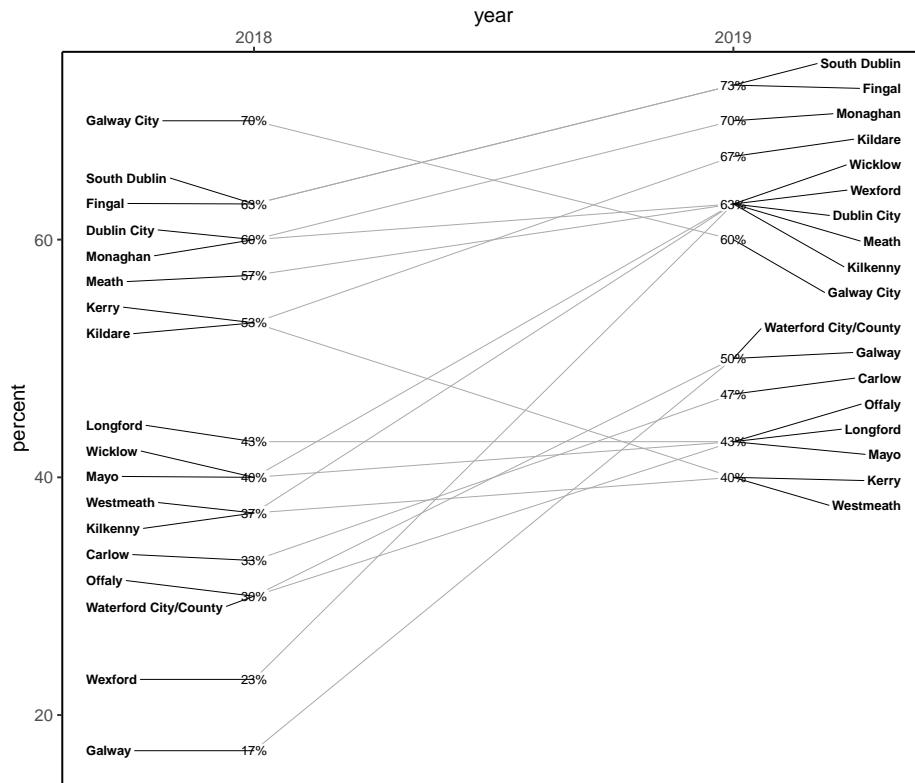
g



There needs to be two sets of `geom_text_repel` labels - one for the `2018` line and one for the `2019` line of values on the right. Now we add the right labels

```
g <- g +
  geom_text_repel(
    data = ire.councils.18.19.subset %>% filter(year == "2019"),
    aes(label = short) ,
    hjust = "right",
    fontface = "bold",
    size = 2.5,
    nudge_x = 0.35,
    segment.size = 0.05,
    direction = "y"
  )
```

g



We can see that we have all the elements of the slope plot.

There is a long list of theme tweaks- such as removing the y axis text and title, removing gridlines, changing the background colour.

```
g <- g +
  theme_bw() +
  theme(panel.grid.minor.y = element_blank()) +
  theme(panel.background = element_blank()) +
  theme(panel.grid = element_blank()) +
  theme(axis.ticks = element_blank()) +
  theme(panel.border = element_blank()) +
  theme(legend.position = "none") +
  theme(axis.title.y = element_blank()) +
  theme(axis.text.y = element_blank())

# Remove a few things from the x axis and increase font size
theme(axis.title.x = element_blank()) +
  theme(panel.grid.major.x = element_blank()) +
  theme(axis.text.x.top = element_text(size = 10, face = "bold")) +
  # Remove x & y tick marks
```

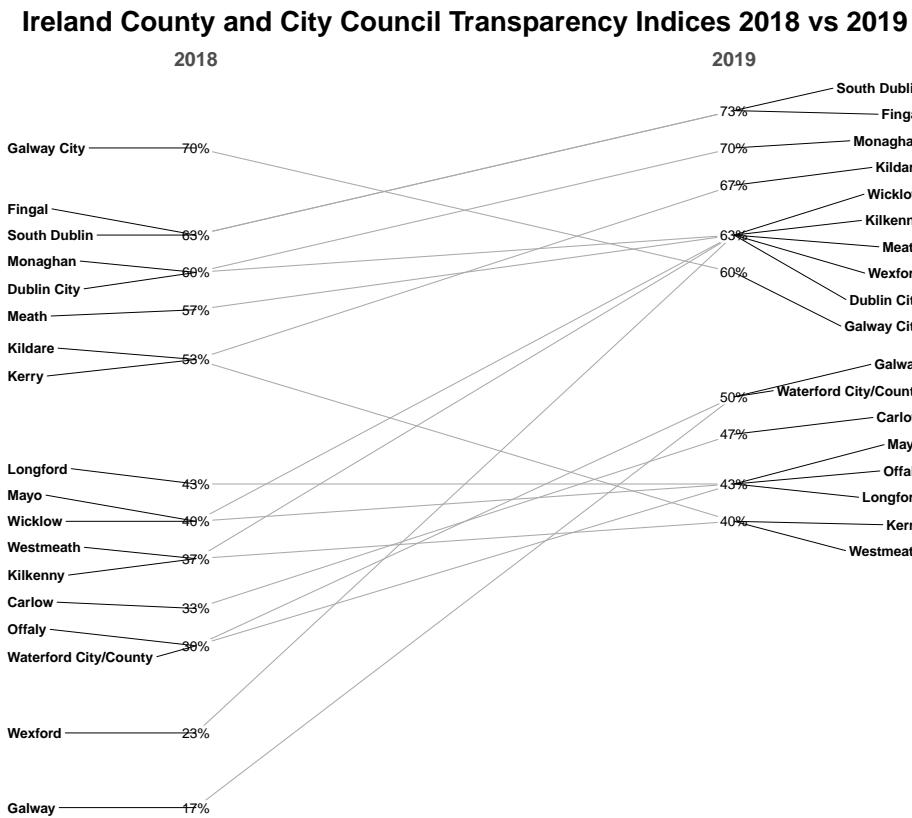
```

theme(axis.ticks = element_blank()) +
# Format title & subtitle
theme(plot.title = element_text(size = 14, face = "bold", hjust = 0.5)) +
theme(plot.subtitle = element_text(hjust = 0.5)) +

labs(title = "Ireland County and City Council Transparency Indices 2018 vs 2019")

g

```



Here is the full code

```

library(ggplot2)
library(ggrepel)

g <-
ggplot(ire.councils.18.19.subset,
      aes(x = year, y = percent, group = short)) +

geom_line(size = 0.25, colour = "darkgrey") +

```

```
scale_x_discrete(position = "top", expand = c(0.2, 0.2)) +  
  
  geom_label(  
    aes(label = paste0(percent, "%")),  
    size = 2.5,  
    label.padding = unit(0.085, "lines"),  
    label.size = 0.0  
  ) +  
  
  geom_text_repel(  
    data = ire.councils.18.19.subset %>% filter(year == "2018"),  
    aes(label = short) ,  
    hjust = "left",  
    fontface = "bold",  
    size = 2.5,  
    nudge_x = -0.35,  
    segment.size = 0.05,  
    direction = "y"  
  ) +  
  
  geom_text_repel(  
    data = ire.councils.18.19.subset %>% filter(year == "2019"),  
    aes(label = short) ,  
    hjust = "right",  
    fontface = "bold",  
    size = 2.5,  
    nudge_x = 0.35,  
    segment.size = 0.05,  
    direction = "y"  
  ) +  
  
  theme_bw() +  
  theme(panel.grid.minor.y = element_blank()) +  
  theme(panel.background = element_blank()) +  
  theme(panel.grid = element_blank()) +  
  theme(axis.ticks = element_blank()) +  
  theme(panel.border = element_blank()) +  
  theme(legend.position = "none") +  
  theme(axis.title.y = element_blank()) +  
  theme(axis.text.y = element_blank()) +  
  
  theme(axis.title.x = element_blank()) +  
  theme(panel.grid.major.x = element_blank()) +  
  theme(axis.text.x.top = element_text(size = 10,
```

```

        face = "bold")) +
# Remove x & y tick marks
theme(axis.ticks = element_blank()) +
# Format title & subtitle
theme(plot.title = element_text(size = 14,
                                 face = "bold", hjust = 0.5)) +
theme(plot.subtitle = element_text(hjust = 0.5)) +
labs(title = "Ireland County and City
      Council Transparency Indices 2018 vs 2019")

g

```

We can immediately see several pieces of information. Most councils have improved their scores in 2019. All but two have positive slopes. Overall, the scores for 2019 are much better than for 2018. We can see that the last ranked council in 2019, Westmeath, scores much higher than the last ranked council in 2018.

Two councils have noticeably strong positive slopes Galway County Council and Wexford County Council.

Only two councils have lower scores in 2019, Galway City and Kerry County Council.

14.5 Comparing Rankings

However, the slope of these lines represent the performance of each council without taking into account how other councils performed. In other words it doesn't really say whether a council has improved its rank.

While the ranking of each council is implicitly represented by its place on the 2018 and 2019 line, it would be useful to prioritise *rank* while also showing the percentage score. That way, the reader has a clear idea of how much a council has gained or dropped in rank in the interval

A key point to notice here is that I will change the y-axis scale from *percentage* to *rank*.

That may not seem like a great change - but bear in mind that a council might have a slight increase in percentage but a significant drop in rank. For example, Westmeath County Council increased its score from 37% in 2018 to 40% in 2019. However, while the council was ranked 24th in 2018, it was ranked in joint last place in 2019.

So changing the scale to rank is going to significantly change the message from this graph.

The only change I need to make to the plot is to substitute the aesthetic mapping for the y-axis from *percent* to *-rank*. It needs to be *minus rank* as we want the rank values to be ordered in descending order from the bottom of the y axis. E.g. rank 1 value should be the highest value on the axis, rather than the lowest.

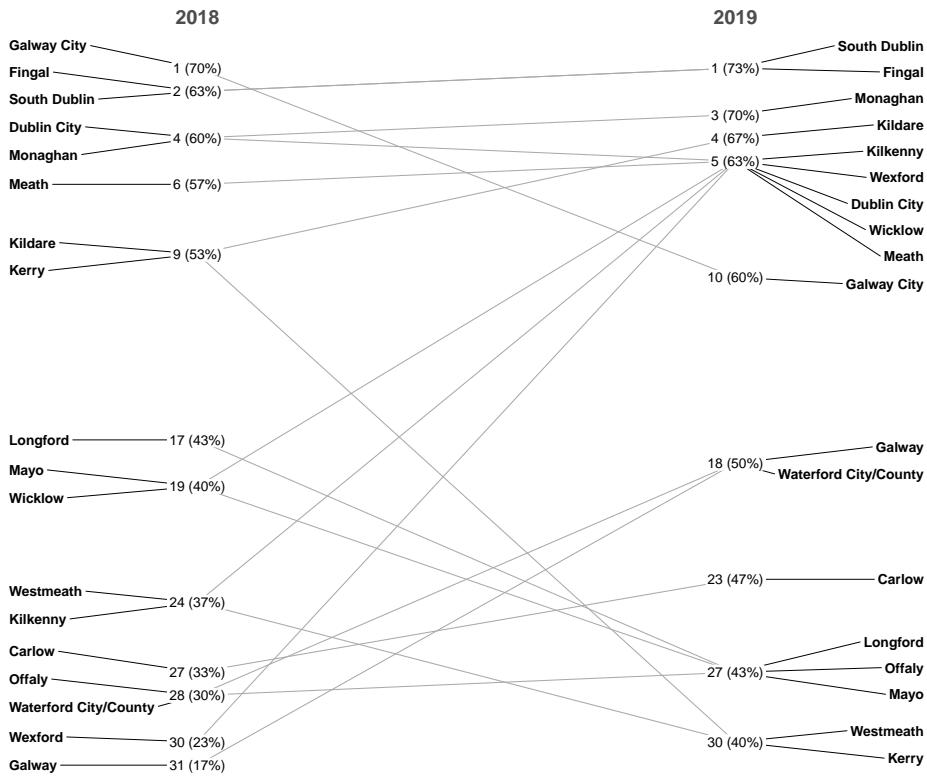
```
library(ggplot2)
library(ggrepel)

g <-
  ggplot(ire.councils.18.19.subset, aes(x = year, y = -rank, group = short)) +
    geom_line(size = 0.25, colour = "darkgrey") +
    scale_x_discrete(position = "top", expand = c(0.2, 0.2)) +
    geom_text_repel(
      data = ire.councils.18.19.subset %>% filter(year == "2018"),
      aes(label = short) ,
      hjust = "left",
      fontface = "bold",
      size = 2.5,
      nudge_x = -0.35,
      segment.size = 0.05,
      direction = "y"
    ) +
    geom_text_repel(
      data = ire.councils.18.19.subset %>% filter(year == "2019"),
      aes(label = short) ,
      hjust = "right",
      fontface = "bold",
      size = 2.5,
      nudge_x = 0.35,
      segment.size = 0.05,
      direction = "y"
    ) +
    geom_label(
      aes(label = paste0(rank, " (", percent, "%)")),
      size = 2.5,
      label.padding = unit(0.085, "lines"),
      label.size = 0.0
    ) +
    theme_bw() +
    theme(panel.grid.minor.y = element_blank()) +
```

```
theme(panel.background = element_blank()) +
  theme(panel.grid = element_blank()) +
  theme(axis.ticks = element_blank()) +
  theme(panel.border = element_blank()) +
  theme(legend.position = "none") +
  theme(axis.title.y = element_blank()) +
  theme(axis.text.y = element_blank()) +
  theme(axis.title.x = element_blank()) +
  theme(panel.grid.major.x = element_blank()) +
  theme(axis.text.x.top = element_text(size = 10, face = "bold")) +
  # Remove x & y tick marks
  theme(axis.ticks = element_blank()) +
  # Format title & subtitle
  theme(plot.title = element_text(size = 14, face = "bold", hjust = 0.5)) +
  theme(plot.subtitle = element_text(hjust = 0.5)) +
  labs(title = "Ireland County and City Council Transparency Indices 2018 vs 2019")
```

g

Ireland County and City Council Transparency Indices 2018 vs 2019



This graph is far more interpretable. We can make statements like Galway County council has climbed 13 places to 18th place. Or more dramatically, Kerry County council has dropped from 9th place in 2018 to joint last place in 2019.

14.6 Colouring Slope lines

A common approach in slope graphs is to colour lines depending on whether they have positive or negative slope. This allows the reader to immediately perceive the lines with negative slope and to get a sense of the proportion of positive to negative lines.

To colour the lines, we have to first designate which lines are positive or negative. We create a new field in the data set, *slope*, which has a positive score if the slope is positive and negative score if it is negative .

```
## Calculate difference so as to colour slope lines
ire.councils.18.19.subset <- ire.councils.18.19.subset %>%
  group_by(short) %>%
  mutate(slope = (rank[year == '2019'] - rank[year == '2018']))
```

The only change I've made here is to define an `aes` function in the `geom_line` that assigns a colour depending on whether `slope` is greater than zero or not.

I override the default ggplot colours with two colour blind friendly colours.

```
library(ggplot2)
library(ggrepel)

slope_data <- ire.councils.18.19.subset

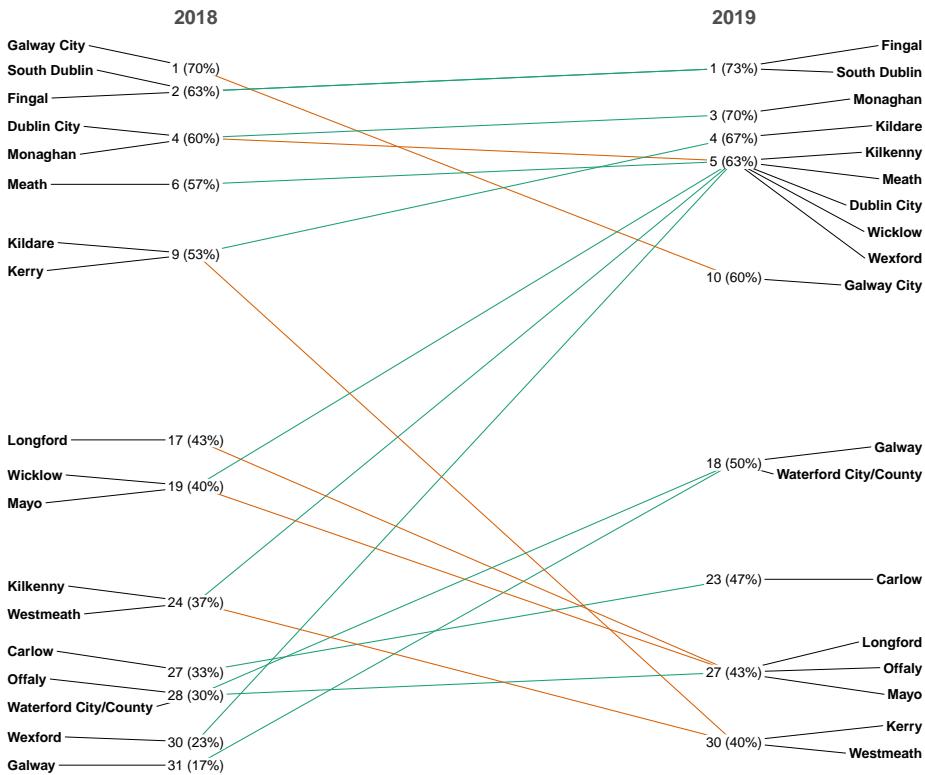
colrs <- c('#1b9e77', '#d95f02')

g <- ggplot(slope_data, aes(x = year, y = -rank, group = short)) +
  # colour the line based on whether slope > 0
  geom_line(size = 0.25, aes(colour = slope > 0)) +
  scale_colour_manual(values = colrs) +
  scale_x_discrete(position = "top", expand = c(0.2, 0.2)) +
  geom_text_repel(
    data = slope_data %>% filter(year == "2018"),
    aes(label = short) ,
    hjust = "left",
    fontface = "bold",
    size = 2.5,
    nudge_x = -0.35,
    segment.size = 0.05,
    direction = "y"
  ) +
  geom_text_repel(
    data = slope_data %>% filter(year == "2019"),
    aes(label = short) ,
    hjust = "right",
    fontface = "bold",
    size = 2.5,
    nudge_x = 0.35,
    segment.size = 0.05,
    direction = "y"
  ) +
  geom_label(
    aes(label = paste0(rank, " (", percent, "%)")),
    size = 2.5,
    label.padding = unit(0.085, "lines"),
```

```
    label.size = 0.0
) +
theme_bw() +
theme(panel.grid.minor.y = element_blank()) +
theme(panel.background = element_blank()) +
theme(panel.grid = element_blank()) +
theme(axis.ticks = element_blank()) +
theme(panel.border = element_blank()) +
theme(legend.position = "none") +
theme(axis.title.y = element_blank()) +
theme(axis.text.y = element_blank()) +

# Remove a few things from the x axis and increase font size
theme(axis.title.x = element_blank()) +
theme(panel.grid.major.x = element_blank()) +
theme(axis.text.x.top = element_text(size = 10, face = "bold")) +
# Remove x & y tick marks
theme(axis.ticks = element_blank()) +
# Format title & subtitle
theme(plot.title = element_text(size = 14, face = "bold", hjust = 0.5)) +
theme(plot.subtitle = element_text(hjust = 0.5)) +
labs(title = "Ireland County and City Council Transparency Indices 2018 vs 2019")
g
```

Ireland County and City Council Transparency Indices 2018 vs 2019



The plot allows us to see very quickly which councils have made gains and which have lost ground.

14.7 Foregrounding a subset of slope lines

Now, let's assume that this visualisation has been designed as part of a discussion piece around the proposed merger of Galway City and County council. A point of contention in any merger might be the contrasting results of the the two councils.

As such, let's revise this graph to foreground the Galway councils, while keeping the background slopes visible.

To create the background colours, colour the lines with a duplicate palette of colours that have have been slightly desaturated and have alpha reduced.

Then add a `geom_line` for *Galway* and for *Galway City*. Colour these lines with the the original full alpha colours.

```
library(ggplot2)
library(ggrepel)
```

```
library(grDevices)
library(colorspace)

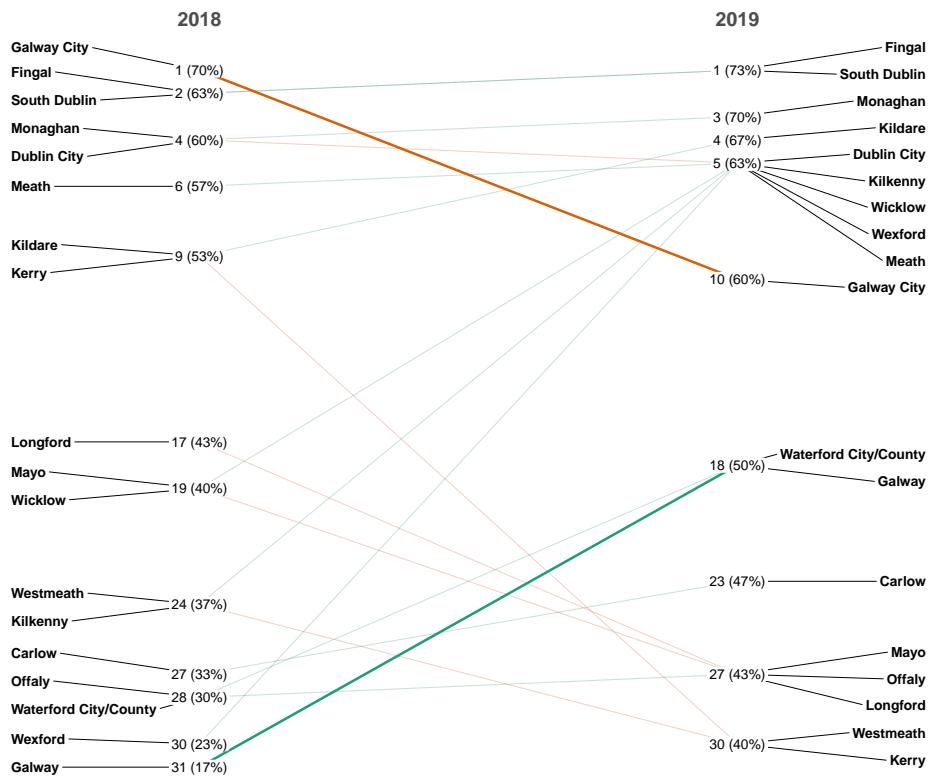
slope_data <- ire.councils.18.19.subset

colrs <- c('#1b9e77', '#d95f02')
colrs2 <- desaturate(colrs, 0.3)
colrs_faded <- adjustcolor(colrs2, alpha.f = 0.3)

g <- ggplot(slope_data, aes(x = year, y = -rank, group = short)) +
  # faded background
  geom_line(size = 0.25, aes(colour = slope > 0)) +
  scale_colour_manual(values = colrs_faded) +
  # foreground has non-faded colours and slightly thicker lines
  geom_line(
    data = slope_data %>%
      filter(short == "Galway"),
    size = 0.6,
    colour = colrs[1]
  ) +
  geom_line(
    data = slope_data %>%
      filter(short == "Galway City"),
    size = 0.6,
    colour = colrs[2]
  ) +
  scale_x_discrete(position = "top", expand = c(0.2, 0.2)) +
  geom_text_repel(
    data = slope_data %>% filter(year == "2018"),
    aes(label = short) ,
    hjust = "left",
    fontface = "bold",
    size = 2.5,
    nudge_x = -0.35,
    segment.size = 0.05,
    direction = "y"
  ) +
  geom_text_repel(
    data = slope_data %>% filter(year == "2019"),
    aes(label = short) ,
```

```
    hjust = "right",
    fontface = "bold",
    size = 2.5,
    nudge_x = 0.35,
    segment.size = 0.05,
    direction = "y"
) +
  geom_label(
    aes(label = paste0(rank, " (", percent, "%)")),
    size = 2.5,
    label.padding = unit(0.085, "lines"),
    label.size = 0.0
) +
  theme_bw() +
  theme(panel.grid.minor.y = element_blank()) +
  theme(panel.background = element_blank()) +
  theme(panel.grid = element_blank()) +
  theme(axis.ticks = element_blank()) +
  theme(panel.border = element_blank()) +
  theme(legend.position = "none") +
  theme(axis.title.y = element_blank()) +
  theme(axis.text.y = element_blank()) +
  # Remove a few things from the x axis and increase font size
  theme(axis.title.x = element_blank()) +
  theme(panel.grid.major.x = element_blank()) +
  theme(axis.text.x.top = element_text(size = 10, face = "bold")) +
  # Remove x & y tick marks
  theme(axis.ticks = element_blank()) +
  # Format title & subtitle
  theme(plot.title = element_text(size = 14, face = "bold", hjust = 0.5)) +
  theme(plot.subtitle = element_text(hjust = 0.5)) +
  labs(title = "Ireland County and City Council Transparency Indices 2018 vs 2019")
g
```

Ireland County and City Council Transparency Indices 2018 vs 2019



This plot brings to the foreground that message that both councils were at opposite ends of the scale in 2018 - but it could be argued that they are converging. The City council dropped 9 places in 2019 and the County Council rose from last place in 2018 to 18th place in 2019. The City council's lost of places is more dramatic - if you compare it to the performance of its peers in 2018. Of the top 5 in 2018, only Dublin City Council dropped a place from 4th to 5th. While Galway County Council increased its ranking from last place to 18th place, we can also see that the second last ranked county council in 2018, Wexford, increased its rank to joint 5th place in 2019 - a much better improvement than Galway's county council's.

The slope graph is a very simple but powerful graph for showing how rankings change over time. It is not just limited to two axis. When transparency Ireland publish their figures for 2020, I will update this graph with a third axis to represent 2020 values.

14.8 Labelling issues

However, there are some drawbacks. Labelling is an issue where there are multiple categories with the same value. Here I've used the `ggrepel` library to labels points like this. This introduces label line segments on the left and on the right of the graph. And this is not as directly readable as labelling the position on the axis directly. For example, look at the point shared by the county councils of *Wicklow* and *Mayo* in 2018. We can see that there is a positive and negative slope line leaving this point - but which county council does either line refer to? We can only find out by following the lines. In this case, we can see that the negative slopped line applies belongs to *Mayo* and the positive slopped line to *Wicklow*

Edward Tufte addressed this problem by placing points with the same value one below the other on the axis. You can see an example in the lectures notes. However this really only works for two points with equal values.

If you have several points with the same value, such as the 5th ranked councils in 2019, there may not be enough room on the y-axis scale to accommodate a list of points with the same value As such, the `ggrepel` labelling approach is probably the best approach.

14.9 Visualising all councils

At the top of this lesson, I suggested that including all the councils might have resulted in a too cluttered look. I've included a plot here with all the council data. I think that this version works because we've decided to foreground a few lines. The background is interpretable if you care to follow it. On the whole it tells me that there were slightly more rank gains to rank losses between the 2018 and 2019 indices.

I think the full plot also highlights the multiple labelling issue I mentioned earlier. In short, while slope graphs offer a simple powerful way to visualise how ranks or values change between times, they work best when there are not too many categories with the same values (to avoid the labelling problem), and when there are a few clear changes in rank or value that you want to highlight

```
library(ggplot2)
library(ggrepel)
library(grDevices)
library(colorspace)

slope_data <- ire.councils.18.19

## Calculate difference so as to colour slope lines
slope_data <- slope_data %>%
  group_by(short) %>%
```

```
mutate(slope = (rank[year == '2019'] - rank[year == '2018']))  
  
colrs <- c('#1b9e77', '#d95f02')  
colrs2 <- desaturate(colrs, 0.3)  
colrs_faded <- adjustcolor(colrs2, alpha.f = 0.3)  
  
g <- ggplot(slope_data, aes(x = year, y = -rank, group = short)) +  
  # faded background  
  geom_line(size = 0.25, aes(colour = slope > 0)) +  
  
  scale_colour_manual(values = colrs_faded) +  
  
  # foreground has non-faded colours and slightly thicker lines  
  geom_line(  
    data = slope_data %>%  
      filter(short == "Galway"),  
    size = 0.6,  
    colour = colrs[1]  
  ) +  
  geom_line(  
    data = slope_data %>%  
      filter(short == "Galway City"),  
    size = 0.6,  
    colour = colrs[2]  
  ) +  
  
  scale_x_discrete(position = "top", expand = c(0.2, 0.2)) +  
  
  geom_text_repel(  
    data = slope_data %>% filter(year == "2018"),  
    aes(label = short) ,  
    hjust = "left",  
    fontface = "bold",  
    size = 2.5,  
    nudge_x = -0.45,  
    segment.size = 0.05,  
    direction = "y"  
  ) +  
  
  geom_text_repel(  
    data = slope_data %>% filter(year == "2019"),  
    aes(label = short) ,
```

```

      hjust = "right",
      fontface = "bold",
      size = 2.5,
      nudge_x = 0.45,
      segment.size = 0.05,
      direction = "y"
    ) +
  geom_label(
    aes(label = paste0(rank, " (", percent, "%)")),
    size = 2.5,
    label.padding = unit(0.085, "lines"),
    label.size = 0.0
  ) +
  theme_bw() +
  theme(panel.grid.minor.y = element_blank()) +
  theme(panel.background = element_blank()) +
  theme(panel.grid = element_blank()) +
  theme(axis.ticks = element_blank()) +
  theme(panel.border = element_blank()) +
  theme(legend.position = "none") +
  theme(axis.title.y = element_blank()) +
  theme(axis.text.y = element_blank()) +
  # Remove a few things from the x axis and increase font size
  theme(axis.title.x = element_blank()) +
  theme(panel.grid.major.x = element_blank()) +
  theme(axis.text.x.top = element_text(size = 10, face = "bold")) +
  # Remove x & y tick marks
  theme(axis.ticks = element_blank()) +
  # Format title & subtitle
  theme(plot.title = element_text(size = 14, face = "bold", hjust = 0.5)) +
  theme(plot.subtitle = element_text(hjust = 0.5)) +
  labs(title = "Ireland County and City Council Transparency Indices 2018 vs 2019")
g

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted

```

```
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
```

```
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>
```

```
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <93>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

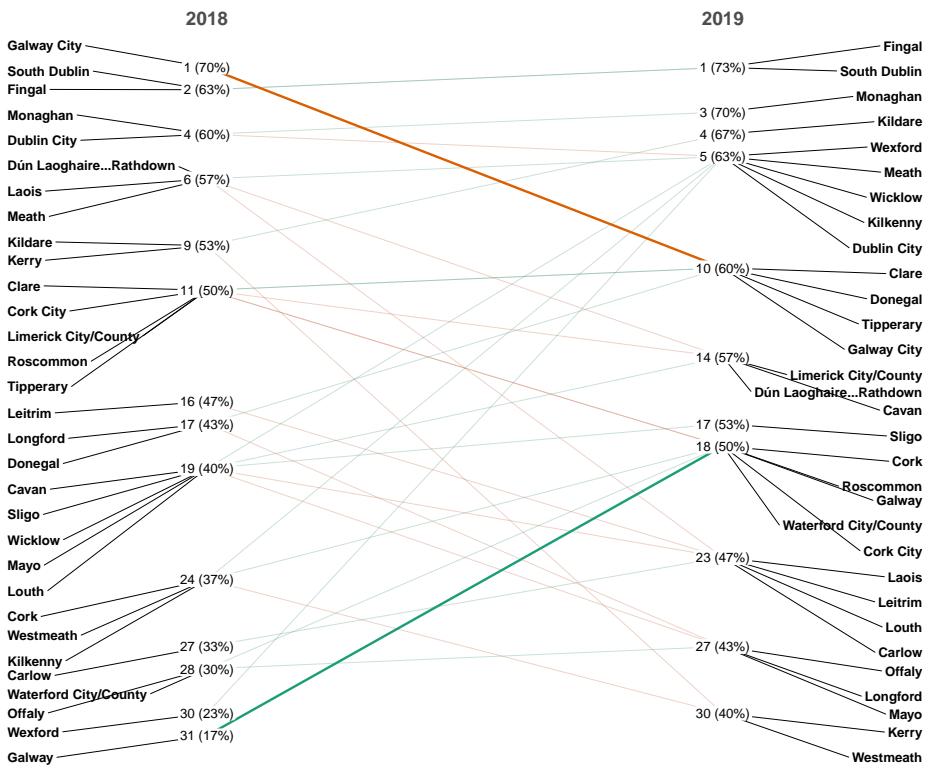
## Warning in grid.Call(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <e2>

## Warning in grid.Call(graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
## for <80>

## Warning in grid.Call(graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Dún Laoghaire-Rathdown' in 'mbcsToSbcs': dot substituted
```

```
## for <93>
```

Ireland County and City Council Transparency Indices 2018 vs 2019



Chapter 15

Parallel Sets

A Parallel Set chart shows the proportional relationship between variables as a flow between parallel axes. It shows the breakdown of the data values by each individual categorical variable; and it draws shaded bands to show how the category proportions relate to each other.

As there are potentially many overlapping bands in a parallel set, you need careful choice of a set of qualitative colours that are well separable from each other and a low alpha value to allow visual overlap of the bands.

The variables are standard categorical variables. They do not have to include a time variable, but in the example that follows I will show how a parallel set chart can be used to show the proportional relationship between variables where one variable representing time intervals is treated as an ordered factor, an ordered category.

15.1 Parallel plot 1: Visualising Departure times

I will visualise some data extracted from the 2016 Irish census. It shows the distribution of daily departure times from home for Galway people. Just under 120,000 Galwegians answered this question on the census.

The aesthetics used here are *length* – the length of the grey category bar measured against the *y axis* scale, which indicates the proportion of the categorical variable; and *colour* which indicates the value of a categorical variable - in this case either *Galway City* or *Galway County*.

First, as always, the data needs some preprocessing before visualisation. To start with, I will visualise just two variables - the *Area* variable and the *Departure* time variable.

Note that the `gather_set_data` function is a helper function from the `ggforce` library makes it easy to change tidy data into a `tidy(er)` format that can be used by `geom_parallel_sets`.

```
library(ggplot2)
library(dplyr)
library(ggforce)

leaving_home <- read.csv("../data/2016_time_leaving_home.csv")

leaving_home<-leaving_home%>%
  mutate(
    Status = case_when(
      Status == "Primary School" ~ "primary",
      Status == "Secondary School" ~ "secondary",
      Status == "College" ~ "college",
      Status == "Work" ~ "work"
    ))%>%
  mutate(Status=factor(Status, levels = c("primary", "secondary", "college", "work")))%>%
  mutate(
    Departure = case_when(
      Departure == "Before 06:30" ~ "6:30 -",
      Departure == "06:30 - 07:00" ~ "6:30-7",
      Departure == "07:01 - 07:30" ~ "7-7:30",
      Departure == "07:31 - 08:00" ~ "7:30-8",
      Departure == "08:01 - 08:30" ~ "8-8:30",
      Departure == "08:31 - 09:00" ~ "8:30-9",
      Departure == "09:01 - 09:30" ~ "9-9:30",
      Departure == "After 09:30" ~ "9:30 +",
    ))%>%
  mutate(Departure=factor(Departure, levels = c("6:30 -", "6:30-7", "7-7:30", "7:30-8", "9-9:30", "9:30 +")))

leaving_home%>%
  mutate(Departure =case_when(
    Departure == "6:30 -" ~ "6:30 -",
    Departure == "6:30-7" ~ "6:30-7:30",
    Departure == "7-7:30" ~ "6:30-7:30",
    Departure == "7:30-8" ~ "7:30-8:30",
    Departure == "8-8:30" ~ "7:30-8:30",
    Departure == "8:30-9" ~ "8:30-9:30",
    Departure == "9-9:30" ~ "8:30-9:30",
    Departure == "9:30 +" ~ "9:30 +")
  ))%>%
  group_by(Area, Status, Departure)%>%
```

```

summarize(Count=sum(Count)) %>%mutate(Departure=factor(Departure, levels = c("6:30 -", "6:30-7:30"))

# just visualising two variables
# gather_set_data is a function supplied by the geom_parallel_sets developer to get the data into
leaving_home_ps <- gather_set_data(leaving_home_hourly, c(1,3))

leaving_home_ps$x <- factor(leaving_home_ps$x, levels = c("Area", "Departure"))

```

The code for the plot. It relies on the `ggforce` library, which you will first have to install.

We can easily perceive the flows from the Galway city/county labels along the various pathways to their intersection with the different Departure times, even though the paths curve and intersect with each other.

We can see straightaway that only small fraction of people have to leave their house before 6.30 every day.

We can also see why this data could not be easily visualised as a tree map. The category relationships are non-disjoint.

Whether in the city or the County Galway, people leave their house at various times, whereas in the Tree map example each county belonged to one province only.

```

library(ggforce)

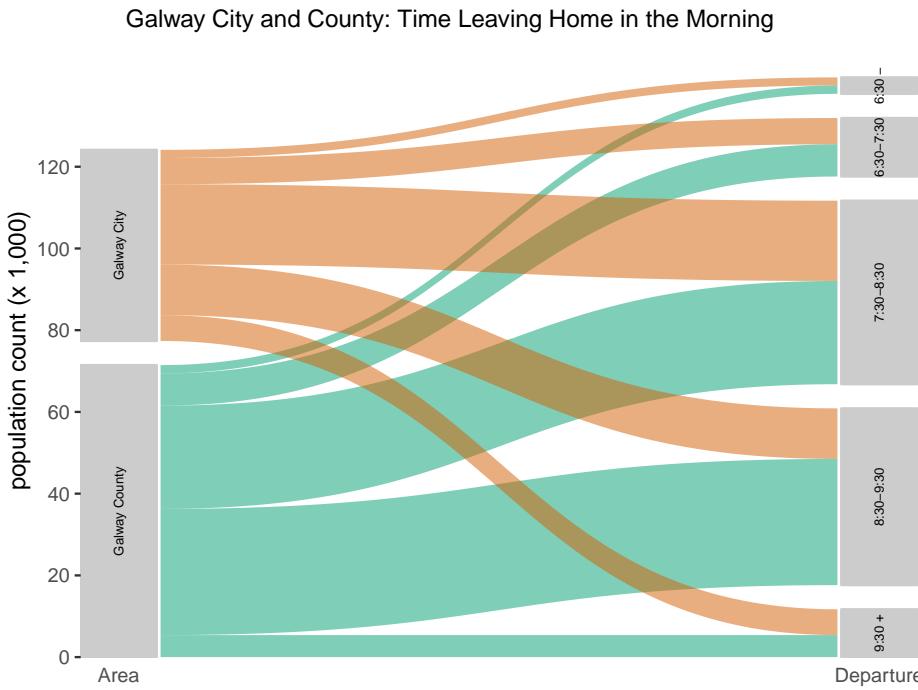
ggplot(leaving_home_ps, aes(x, id = id, split = y, value = Count)) +
  geom_parallel_sets(aes(fill = Area), alpha = 0.5, axis.width = 0.11) +
  geom_parallel_sets_axes(axis.width = 0.1, fill = "grey80", color = "grey80") +
  geom_parallel_sets_labels(
    color = 'black',
    size = 6/.pt,
    angle = 90
  ) +
  scale_x_discrete(
    name = NULL,
    expand = c(0.0, 0.0)
  ) +
  scale_y_continuous(
    #breaks = NULL,
    expand = c(0, 0),
    limits = c(0, 1.5e5),
    breaks = seq(0,1.2e5, by =2e4),

```

```
labels = seq(0,120, by =20),
  name = "population count (x 1,000)"
    )+
  scale_fill_manual(
    values = c("Galway City" = "#D55E00D0", "Galway County" = "#009E73D0"),
    guide = "none"
  ) +
#theme_minimal() +

theme(
  axis.line = element_blank(),
  plot.margin = margin(14, 2, 2, 2),
  axis.ticks.x      = element_blank(),
  legend.background = element_blank(),
  legend.key        = element_blank(),
  panel.background  = element_blank(),
  panel.border       = element_blank(),
  strip.background   = element_blank(),
  plot.background   = element_blank(),
  panel.grid= element_blank(),
  plot.title = element_text(size = 11, hjust = 0.24)

) +
ggttitle("Galway City and County: Time Leaving Home in the Morning")
```



15.2 Parallel plot 2: adding an extra ‘axis’

Now I add an extra ‘axis’ to represent a third variable, the *Status* of each person leaving the house. This requires simply selecting the column (number) representing this variable.

```
# Now visualise three variables
# gather_set_data is a function supplied by the geom_parallel_sets developer to get the data into
leaving_home_ps <- gather_set_data(leaving_home_hourly, c(1,2,3))

leaving_home_ps$x <- factor(leaving_home_ps$x, levels = c("Area", "Status", "Departure"))
```

And the plot code is much the same as before.

We can see that the flows split at each vertical axis into sub-flows representing the proportion of data assigned to values in the next axis.

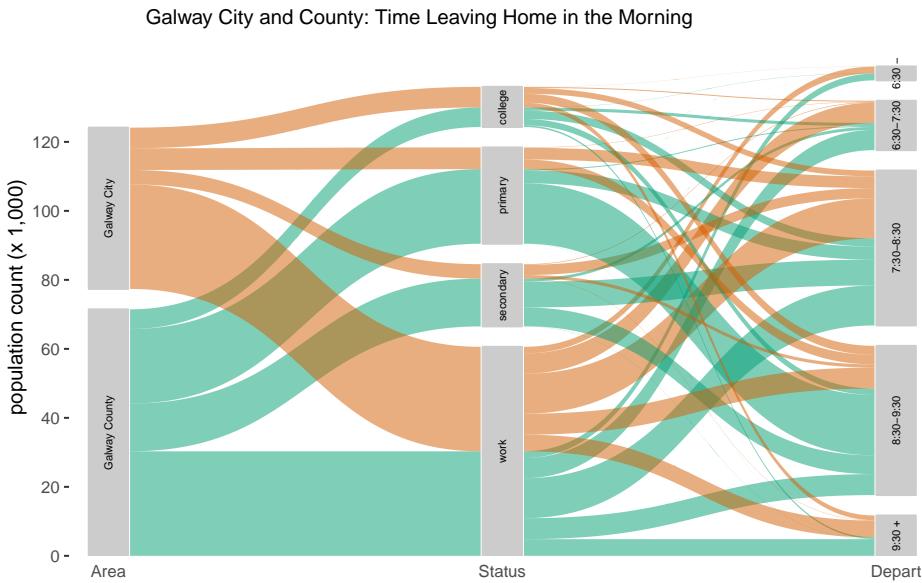
While the addition of another variable with four values has made the graphic more complex to look at, we can still easily follow paths through it.

For example we can say:

- the majority of people leaving the house in Galway city are going to work, and of those about 1/3 leave the house between after 8.30 am

- We can say that the majority of children leaving the house in Galway county are primary school children and by far most of these children leave the house between 8.30 and 9.30 am.

```
ggplot(leaving_home_ps, aes(x, id = id, split = y, value = Count)) +
  geom_parallel_sets(aes(fill = Area), alpha = 0.5, axis.width = 0.11) +
  geom_parallel_sets_axes(axis.width = 0.1, fill = "grey80", color = "grey80") +
  geom_parallel_sets_labels(
    color = 'black',
    size = 6/.pt,
    angle = 90
  ) +
  scale_x_discrete(
    name = NULL,
    expand = c(0.0, 0.1),
    labels=c("Area", "Status", "Depart")
  ) +
  scale_y_continuous(
    #breaks = NULL,
    expand = c(0, 0),
    limits = c(0, 1.5e5),
    breaks = seq(0,1.2e5, by =2e4),
    labels = seq(0,120, by =20),
    name = "population count (x 1,000)"
  ) +
  scale_fill_manual(
    values = c("Galway City" = "#D55E00D0", "Galway County" = "#009E73D0"),
    guide = "none"
  ) +
  theme(axis.line = element_blank(),
        plot.margin = margin(14, 2, 2, 2),
        axis.ticks.x = element_blank(),
        legend.background = element_blank(),
        legend.key = element_blank(),
        panel.background = element_blank(),
        panel.border = element_blank(),
        strip.background = element_blank(),
        plot.background = element_blank(),
        panel.grid= element_blank(),
        plot.title = element_text(size = 11, hjust = 0.24)) +
  ggtitle("Galway City and County: Time Leaving Home in the Morning")
```



15.3 Parallel plot 3: reordering the ‘axes’

We can change the order of the axis which can give us a different view of the data

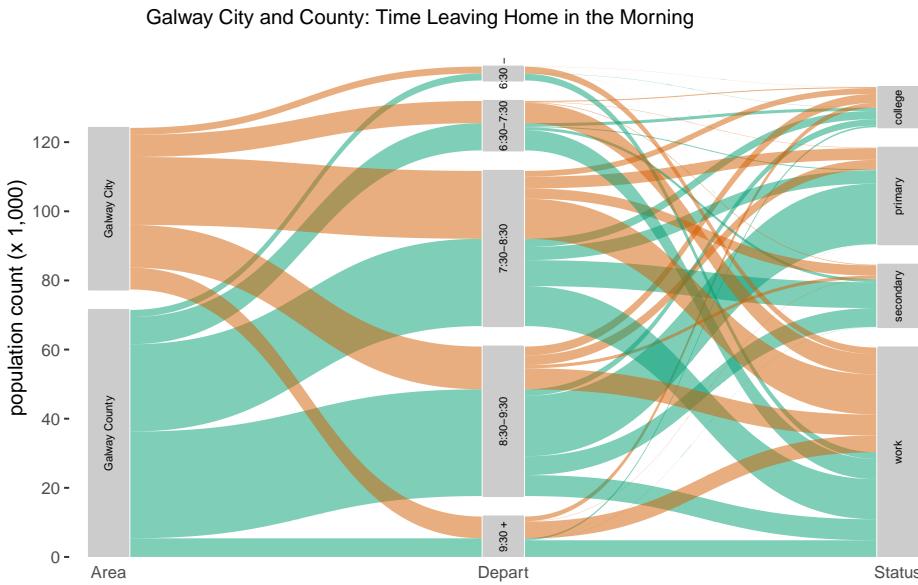
Now we can easily see that the

- The majority of Galway city people leaving the house between 7.30 and 8.30 are workers
- The majority of county people leaving the house between 8.30 and 9.30 are children going to primary school.

```
leaving_home_ps$x <- factor(leaving_home_ps$x, levels = c("Area", "Departure", "Status"))

ggplot(leaving_home_ps, aes(x, id = id, split = y, value = Count)) +
  geom_parallel_sets(aes(fill = Area), alpha = 0.5, axis.width = 0.11) +
  geom_parallel_sets_axes(axis.width = 0.1, fill = "grey80", color = "grey80") +
  geom_parallel_sets_labels(
    color = 'black',
    size = 6/.pt,
    angle = 90
  ) +
  scale_x_discrete(
```

```
name = NULL,
expand = c(0.0, 0.1),
labels=c("Area", "Depart","Status")
) +
scale_y_continuous(
  #breaks = NULL,
  expand = c(0, 0),
  limits = c(0, 1.5e5),
  breaks = seq(0,1.2e5, by =2e4),
  labels = seq(0,120, by =20),
  name = "population count (x 1,000)"
  )+
scale_fill_manual(
  values = c("Galway City" = "#D55E00D0", "Galway County" = "#009E73D0"),
  guide = "none"
) +
theme(axis.line = element_blank(),
plot.margin = margin(14, 2, 2, 2),
  axis.ticks.x      = element_blank(),
  legend.background = element_blank(),
  legend.key        = element_blank(),
  panel.background  = element_blank(),
  panel.border       = element_blank(),
  strip.background   = element_blank(),
  plot.background    = element_blank(),
  panel.grid= element_blank(),
  plot.title = element_text(size = 11, hjust = 0.24)) +
ggttitle("Galway City and County: Time Leaving Home in the Morning")
```



15.4 Alluvial plot

The parallel set plot, which was generated by the package *ggforce*, didn't allow me to order the values on the axis, which is a pretty important feature when dealing with an ordered factor representing series of time windows.

A related approach is called the *alluvial plot*. The library for this plot *ggalluvial* did allow me to re-order the time values on the axis. I was also able to change the alpha value of the axes, allowing the paths to be viewed underneath, which helps in tracing them from left to right.

An alluvial plot largely the same except that the split in data occurs at the starting axis and continues until terminating at the right most axis. This allows you to follow a path fairly easily from start to finish

```
library(ggalluvial)
library(forcats)

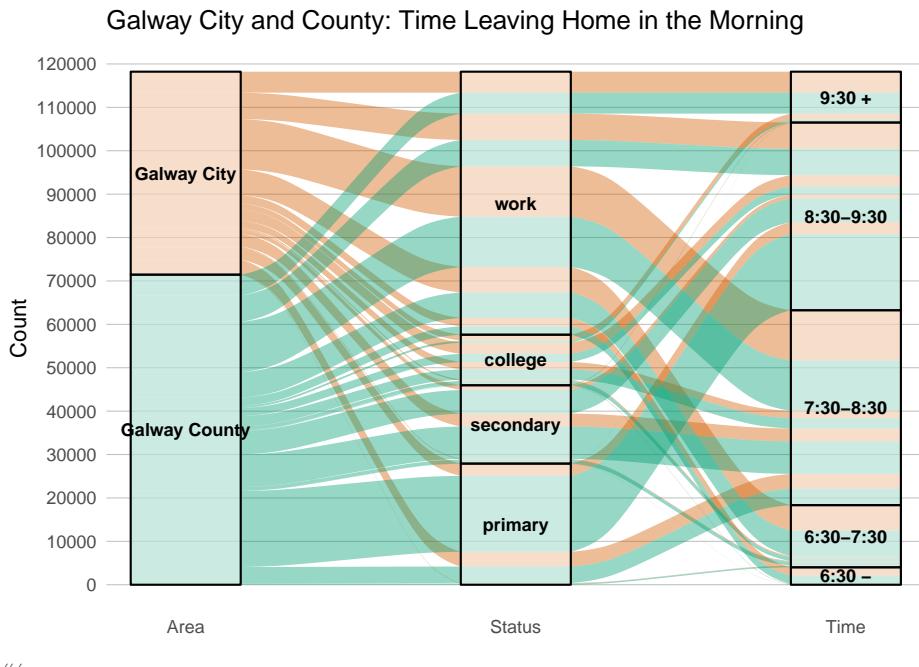
leaving_home_hourly$Departure <- fct_rev(leaving_home_hourly$Departure)
leaving_home_hourly>Status <- fct_rev(leaving_home_hourly>Status)

ggplot(data = leaving_home_hourly,
       aes(axis1 = Area, axis2 = Status, axis3 = Departure,
           y = Count)) +
  scale_x_discrete(limits = c("Area", "Status", "Time"), expand = c(.01, .05)) +
  scale_y_continuous(breaks=seq(0,120000, by =10000)) +
  #xlab("Leaving Home in the morning") +
```

```

scale_fill_manual(
  values = c("Galway City" = "#D55E00D0", "Galway County" = "#009E73D0"),
  guide = "none"
) +
  geom_alluvium(aes(fill = Area)) +
  geom_stratum(fill="white", alpha=0.5) + geom_text(stat = "stratum", label.strata = TRUE)
  theme_minimal() +
  theme(panel.grid.minor.y= element_blank(),
        panel.grid.major.y= element_line(size=0.1, colour = "grey"),
        panel.grid.major.x= element_blank()
      ) +
  ggtitle("Galway City and County: Time Leaving Home in the Morning")

```



““

Chapter 16

Time Series Heat Maps

A heatmap visualises data through variations in colouring. It is essentially a colour coding of a 2-variable table

Heatmaps are good for showing variance across multiple variables, revealing any patterns, displaying whether any variables are similar to each other, and for detecting if any correlations exist in-between them

Typically, all the rows are one category (labels displayed on the left or right side) and all the columns are another category (labels displayed on the top or bottom).

The cells contained within the table either contain colour-coded categorical data or numerical data, that is based on a colour scale.

The value of the colour coded cell can represent the raw value from the corresponding input matrix, or a z-score, which represents the number of standard deviations an observation is from the mean.

Because of their reliance on colour to communicate value, Heatmaps are ideal for presenting general patterns in the data rather than specific values

Heatmaps can also be used to show the changes in data over time if one of the rows or columns are set to time intervals. In this section, our example will be based on measles outbreak data from 1928 to 2002 in the US.

16.1 Read the data

```
library(readr)
library(dplyr)
library(tidyr)
library(ggplot2)
```

```

library(lubridate)
library(knitr)
library(kableExtra)

dis <- "MEASLES"

vaccine_introduced <- c("MUMPS"=1967, "MEASLES"=1963, "HEPATITIS A" = 1996, "PERTUSSIS"=1985)

# read in two data sets - one for 2018 and one for 2019
us_disease_data<-read_csv("../data/ProjectTycho_Level1_v1.0.0.csv")

disease<-us_disease_data%>%filter(disease==dis)

kable(head(disease),
      digits = 2,
      format = "html",
      row.names = TRUE) %>%
kable_styling(
  bootstrap_options = c("striped"),
  full_width = F,
  font_size = 12
) %>%
column_spec(column = 3, width = "8em")

```

epi_week
 state
 loc
 loc_type
 disease
 cases
 incidence_per_100000

1
 192801

LA
 LOUISIANA

STATE
 MEASLES

1.89

2

192801

KY

KENTUCKY

STATE

MEASLES

79

3.08

3

192801

NE

NEBRASKA

STATE

MEASLES

22

1.60

4

192801

CO

COLORADO

STATE

MEASLES

85

8.38

5

192801

WY

WYOMING

STATE

MEASLES

```

2
0.91
6
192801
WA
WASHINGTON
STATE
MEASLES
230
14.83

```

16.2 Convert epi_week to year and week

The snap shot of the data shows that the time value is actually a sequential set of values defined as *epi_week* values. This is a standard epidemiological time stamp format and it gives the year and week of the year (up to 52) for each row of values.

As we wish to plot yearly values, we have two pre-processing steps to take. First we need to convert the *epi_week* values to standard *year* and *week* values.

Using the `substring` function on the the *epi_week* values we create two new columns with year and week values respectively

```

# convert epidweek to standard year and week values
disease<-disease %>% mutate(year=as.integer(substring(as.character(epi_week), 1, 4)))
  mutate (week=as.integer(substring(as.character(epi_week), 5, 6)))

kable(head(disease),
      digits = 2,
      format = "html",
      row.names = TRUE) %>%
kable_styling(
  bootstrap_options = c("striped"),
  full_width = F,
  font_size = 12
)

epi_week
state
loc

```

loc_type
disease
cases
incidence_per_100000
year
week
1
192801
LA
LOUISIANA
STATE
MEASLES
39
1.89
1928
1
2
192801
KY
KENTUCKY
STATE
MEASLES
79
3.08
1928
1
3
192801
NE
NEBRASKA
STATE

MEASLES

22

1.60

1928

1

4

192801

CO

COLORADO

STATE

MEASLES

85

8.38

1928

1

5

192801

WY

WYOMING

STATE

MEASLES

2

0.91

1928

1

6

192801

WA

WASHINGTON

STATE

MEASLES

230

14.83

1928

1

16.3 Calculate yearly means

```
disease_year <- disease %>%
  group_by(year, loc) %>%
  summarise(ave_incidence_per_100000 = mean(incidence_per_100000))
```

```
kable(head(disease_year),
      digits = 2,
      format = "html",
      row.names = TRUE) %>%
kable_styling(
  bootstrap_options = c("striped"),
  full_width = F,
  font_size = 12
)
```

year

loc

ave_incidence_per_100000

1

1928

ALABAMA

6.44

2

1928

ARIZONA

4.78

3

1928

ARKANSAS

```

9.83
4
1928
CALIFORNIA
1.33
5
1928
COLORADO
4.31
6
1928
CONNECTICUT
12.21

```

16.4 Make an initial plot

```

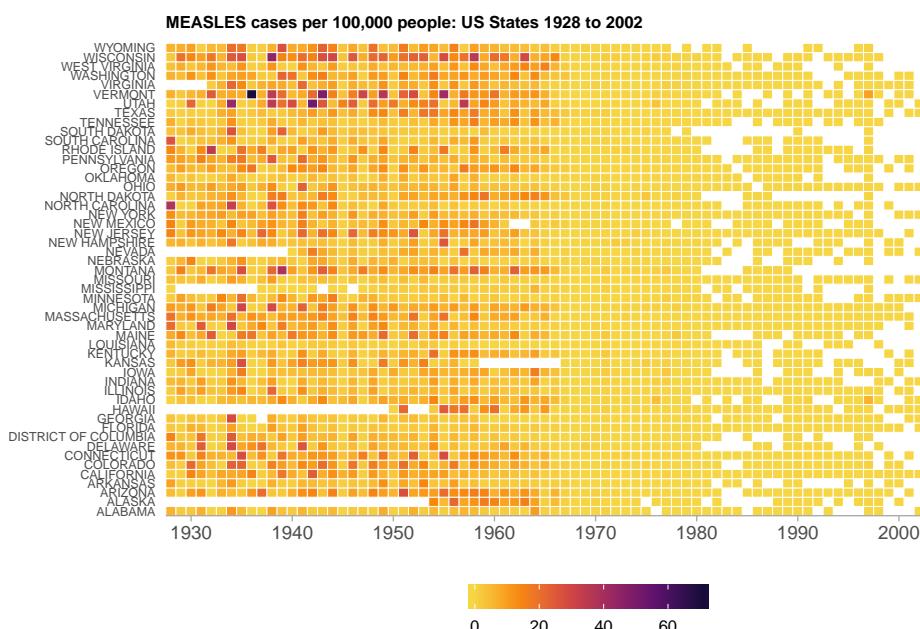
ggplot(disease_year, aes(x=year, y=loc, fill = ave_incidence_per_100000)) +
  geom_tile(colour = "white") +
  scale_y_discrete(name = NULL, expand = c(0,0)) +
  scale_x_continuous(name = NULL, expand = c(0,0), breaks = seq(1930,2000, by=10)) +
  scale_fill_viridis_c(option = "B", begin = 0.1, end = 0.9, direction = -1,
                       name = "incidence per 100,000",
                       guide = guide_colourbar(direction = "horizontal", barwidth = 8,
                                               title = "Ave incidence per 100,000"),
  #geom_vline(xintercept = vaccine_introduced, colour = "black", alpha=0.8) +
  ggttitle(paste0(dis, " cases per 100,000 people: US States 1928 to 2002")) +
  coord_cartesian(clip = 'off') +
  theme(axis.text.y = element_text(size=5.6),
        axis.ticks.x = element_line(size=0.3, colour = "darkgrey"),
        plot.title = element_text(size=10, color="black"))

```

```

axis.line.x = element_line(size=0.3, colour = "darkgrey"),
axis.ticks.y = element_blank(),
axis.line.y = element_blank(),
panel.background = element_blank(),
panel.grid = element_blank(),
plot.margin = unit(c(0.5, 0.5, 2, 0.5), "cm"),
plot.title = element_text(size=8, face="bold")) +
  theme(
    legend.text = element_text(size = 8),
    legend.position = c(0.55, -0.2), # move to the bottom
    legend.title = element_blank(),
    legend.key.size = unit(0.9, "line"),
    legend.spacing.x = unit(0.2, 'cm'),
    legend.background = element_rect(
      fill = "white",
      size = 0.5,
      colour = "white"
    )
  )
)

```



16.5 Order the rows

We want to order the rows by average incidence rate. As several states appear to have no values for some years (e.g Alaska), we will impute a mean value so that when we calculate the incidence rate these states do not have a disproportionately high/low incidence rate.

We have to transform the data to wide format and then for each year fill each NA with a mean value. This is the easiest way (that I know) to fill impute the missing values.

```
# wide format
disease_year_wide <- disease_year %>% spread(year, ave_incidence_per_100000)
disease_year_wide<-as.data.frame(disease_year_wide)

# showing the data in wide form before missing values imputed
kable(head(disease_year_wide) [,1:10],
      digits = 2,
      format = "html",
      row.names = TRUE, caption = "Snap shot of data in wide form BEFORE missing values")
kable_styling(
  bootstrap_options = c("striped"),
  full_width = F,
  font_size = 12
)
```

Snap shot of data in wide form BEFORE missing values imputed

loc

1928

1929

1930

1931

1932

1933

1934

1935

1936

1

A

6.44

2.28

3.02

6.88

0.25

1.28

11.35

5.42

0.52

2

ALASKA

NA

NA

NA

NA

NA

NA

NA

NA

NA

3

ARIZONA

4.78

1.57

9.92

11.85

0.59

6.88

4.78

3.21

11.42

4

ARKANSAS

9.83

1.46

1.37

0.94

0.14

5.64

10.99

1.91

0.24

5

CALIFORNIA

1.33

1.40

14.62

9.18

4.12

8.56

8.14

8.97

15.47

6

COLORADO

4.31

1.46

21.78

9.25

4.55

0.57

23.16

22.81

1.12

```
# impute each NA value with year's mean
for(i in 2:ncol(disease_year_wide)){
  disease_year_wide[is.na(disease_year_wide[,i]), i] <- mean(disease_year_wide[,i], na.rm = TRUE)
}

# showing the data in wide form AFTER missing values imputed
kable(head(disease_year_wide)[,1:10],
      digits = 2,
      format = "html",
      row.names = TRUE,caption = "Snapshot of data in wide form AFTER missing values imputed") %>%
kable_styling(
  bootstrap_options = c("striped"),
  full_width = F,
  font_size = 12
)
```

Snapshot of data in wide form AFTER missing values imputed

loc

1928

1929

1930

1931

1932

1933

1934

1935

1936

1

ALABAMA

6.44

2.28

3.02

6.88

0.25

1.28

11.35

5.42

0.52

2

ALASKA

7.56

5.29

6.49

6.92

6.40

5.05

13.24

10.49

5.71

3

ARIZONA

4.78

1.57

9.92

11.85

0.59

6.88

4.78

3.21

11.42

4

ARKANSAS

9.83

1.46

1.37

0.94

0.14

5.64

10.99

1.91

0.24

5

CALIFORNIA

1.33

1.40

14.62

9.18

4.12

8.56

8.14

8.97

15.47

6

COLORADO

4.31

1.46

21.78

9.25

4.55

0.57

23.16

22.81

1.12

Now convert this data to long format

```
disease_year2<- disease_year_wide%>%  
gather(year, ave_incidence_per_100000, 2:ncol(disease_year_wide) )
```

Now that we have replaced the missing years with average values, we calculate the average incidence per state

```
# get average incidence per location, order in ascending order
disease_year2%>%
  group_by( loc)%>%
  summarise(mean_incidence=mean(ave_incidence_per_100000))%>%
  arrange(mean_incidence)%>%
  select(loc,mean_incidence )->disease2_loc_ordered_incidence

# We set the y-axis ordering in the heat map to the order of the
# states in disease2_loc_ordered_incidence
# To do this we set the levels of the loc variable
# in the original disease_year data frame to the order of states in
# disease2_loc_ordered_incidence

disease_year<-disease_year %>%
  mutate(loc=factor(loc, levels =disease2_loc_ordered_incidence$loc ))

# showing the state ordering
kable(head(disease2_loc_ordered_incidence),
      digits = 2,
      format = "html",
      row.names = TRUE,caption = "Snap shot of ranked US states by mean yearly incidence")
kable_styling(
  bootstrap_options = c("striped"),
  full_width = F,
  font_size = 12
)
```

Snap shot of ranked US states by mean yearly incidence of measles outbreaks (per 100,000). 1928-2002 - Ascending order

```
loc
mean_incidence
1
LOUISIANA
0.74
2
OKLAHOMA
1.42
3
```

MISSOURI

1.53

4

GEORGIA

1.57

5

FLORIDA

1.57

6

MISSISSIPPI

1.59

16.6 Plot heatmap with ordered rows

```
vaccine <- as.integer(vaccine_introduced[dis])

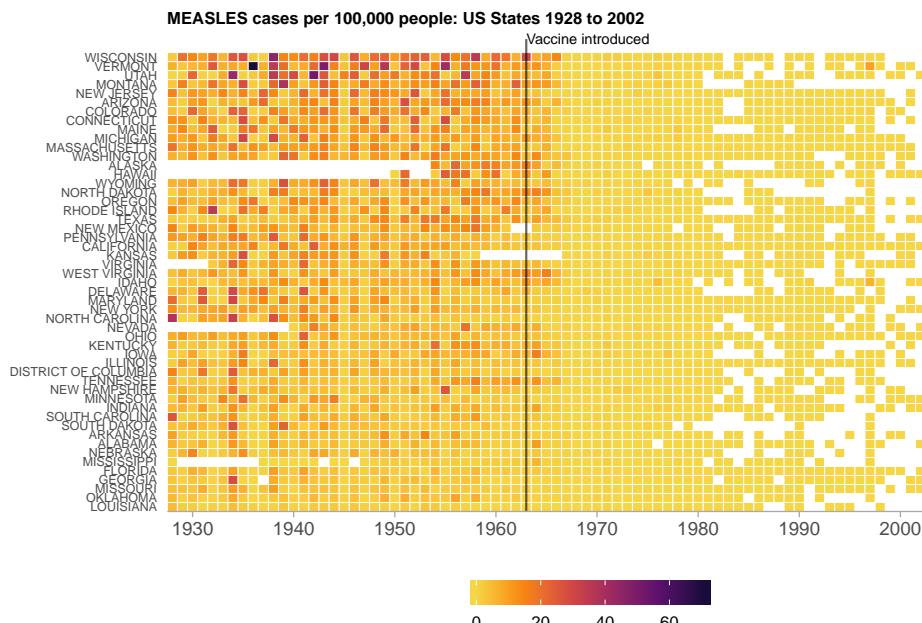
ggplot(disease_year, aes(x=year, y=loc, fill = ave_incidence_per_100000)) +
  geom_tile(colour = "white") +
  scale_y_discrete(name = NULL, expand = c(0,0)) +
  scale_x_continuous(name = NULL, expand = c(0,0), breaks = seq(1930,2000, by=10)) +
  scale_fill_viridis_c(option = "B", begin = 0.1, end = 0.9, direction = -1,
                        name = "incidence per 100,000",
                        guide = guide_colourbar(direction = "horizontal", barwidth = 8, barheight = 10))

  geom_vline(xintercept = vaccine, colour = "black", alpha=0.6, na.rm = T) +
  annotate("text", x = vaccine + 6.1, y = 53, label = "Vaccine introduced", size = 2.6) +
  ggtitle(paste0(dis, " cases per 100,000 people: US States 1928 to 2002")) +
  coord_cartesian(clip = 'off') +
  theme(axis.text.y = element_text(size=5.6),
```

```

axis.ticks.x = element_line(size=0.3, colour = "darkgrey"),
axis.line.x = element_line(size=0.3, colour = "darkgrey"),
axis.ticks.y = element_blank(),
axis.line.y = element_blank(),
panel.background = element_blank(),
panel.grid = element_blank(),
plot.margin = unit(c(0.5, 0.5, 2, 0.5), "cm"),
plot.title = element_text(size=8, face="bold")) +
  theme(
    legend.text = element_text(size = 8),
    legend.position = c(0.55, -0.2), # move to the bottom
    legend.title = element_blank(),
    legend.key.size = unit(0.9, "line"),
    legend.spacing.x = unit(0.2, 'cm'),
    legend.background = element_rect(
      fill = "white",
      size = 0.5,
      colour = "white"
    )
  )
)

```



We have arranged the rows in descending order of average_incidence.

While there is a visual pattern – we can see a cohort of STATES at the top had persistently high measles incidence. However, there doesn't seem to be

any geographic coherence to those states being a mixture of east coast and mid-western states.

That's because our ranking function was based on mean_incidence. We didn't take the time ordering of each outbreak into account. In many cases like a simple ranking like this is fine.

16.7 Clustering the rows

Very often though the rows in heat maps are not ranked - instead they are arranged according to similarity. The similarity function is usually very simple – very often it is the Pearson coefficient , which measures correlation. In our case, if two states were correlated in terms of measles outbreaks, then their incidence score per years would be similar.

If state A and state B are correlated, then if A had a high number of cases in 1928 we could expect state B to have also a high number of cases in 1928;

As diseases know no political boundaries, we might also expect bordering states to be correlated. If we used correlation as basis for grouping states on the y-axis we might expect to see together some groups of states that are in the same geographic region.

Let's test this idea out.

We are going to use a dedicated heatmap function called `heatmap.2`. It's in the `gplots` library so you will have to install that.

`heatmap.2` has quite a lot of options - and I've chosen a few to enable a visualisation configuration that is similar to what we have just done - but where the rows are clustered by Pearson coefficient similarity. The clustering algorithm is a agglomerative and like all such algorithms, it requires you to pre-calculate a distance matrix, giving the pairwise distance between the rows - according to some distance measure.

We set the names of the rows to be the location values (i.e.the state names). The clustering algorithm will present outputs in terms of row names.

```
library(gplots)

rownames(disease_year_wide)<-disease_year_wide$loc

disease_year_wide<-disease_year_wide[, -1]
```

We will correlate the pre-vaccine years. As the vaccine took some time to take hold we'll select the years 1928 to 1966.

We'll use the grouping based on this clustering for the heat map

```

library(viridis)

# select years 1928 to 1966
disease_year_pre_vac <- disease_year_wide[,1:39]

# As correlation (cor) is a similarity measure and dist requires a distance measure, we
# need to convert it. We use 1 - cor to get a distance matrix where lower values
# indicate higher similarity. We then divide by 2 because dist expects (1-r)/2, where r is pearson correlation
dist_mat <- dist((1-cor(t(disease_year_pre_vac)), method="pearson"))/2

# hclust does the hierarchical clustering
hr <- hclust(dist_mat, method="complete")

## Plot heatmap. The heatmap.2 only take matrix objects
mat<- as.matrix(disease_year_wide)

```

In the plot below, the rows are grouped by similarity and the dendrogram on the left hand side shows the structure of the clustering

There are two clear patches colour on the heatmap.

I am not an epidemiologist, so I can't explain these – but as data scientists, lets try to rule out random chance. Remember the hypothesis that the clustering should have uncovered measles outbreaks in neighbouring states. If we can find that to some extent – then we can say the row ordering is meaningful. We know the column ordering is fixed – so the patterns we are then viewing are non random.

We're going to do this informally by inspection by comparing the state groupings produced by the clustering to the the state's geographical location. Of course, this idea holds for states where there are significant outbreaks. The clustering will also cluster those states with consistently low outbreaks. There is no reason to believe that these states should be geographically close. There similarity in terms of low disease incidence may be due to other reasons such as a similar public health policy.

If you are not familiar with a dendrogram the idea is the branching structure indicates the closeness of the similarity of the instances. The higher the branch is the less similar the items belonging to it are.

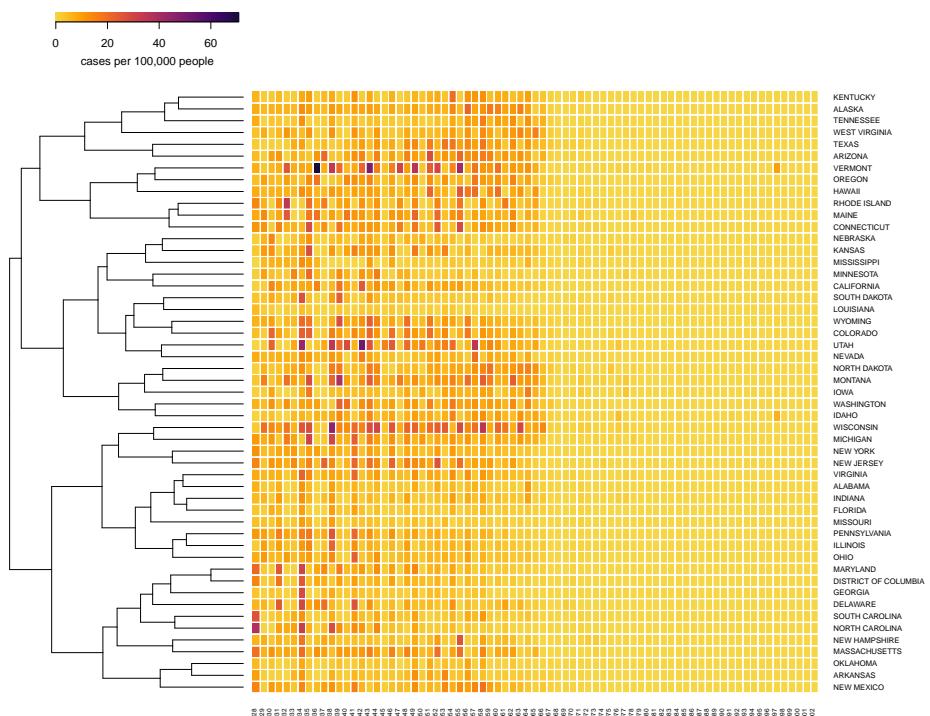
16.8 Plot the heat map with clustered rows

```

# This replicates the colouring we uses earlier
pal<-viridis(n=256,option = "B", begin = 0.1, end = 0.9, direction = -1,)

```

```
heatmap.2(mat, col=pal, Rowv=as.dendrogram(hr), Colv = F, scale="none", density.info="none", trace=TRUE, sepcolor="white", colsep=1:ncol(mat), rowsep=1:nrow(mat), margin=c(3, 7))
```



Chapter 17

Reference Materials

17.1 Colour Table

cornsilk3	dodgerblue4	gray45	grey3	grey69	lemonchiffon2	diumont	violetred4	slateblue
cornsilk2	dodgerblue3	gray44	grey2	grey68	lemonchiffon2	mediumblue	violetred3	skyblue4
cornsilk1	dodgerblue2	gray43	grey1	grey67	lemonchiffon2	mediumblue	violetred2	skyblue3
cornsilk	dodgerblue1	gray42	grey0	grey66	lawngreen	maroon	palevioletred1	skyblue2
mnflowerblue	dodgerblue	gray41	grey	grey65	lavenderblush	maroon3	palevioletred1	skyblue1
coral4	dimgray	gray40	greenyellow	grey64	lavenderblush	maroon3	slateurquoise4	yellow4
coral3	dimgray	gray39	green4	grey63	lavenderblush	maroon3	slateurquoise3	yellow3
coral2	deepskyblue4	gray38	green3	grey62	lavenderblush	maron3	slateurquoise2	sienna3
coral1	deepskyblue3	gray37	green2	grey61	lavenderblush	magenta1	slateurquoise1	sienna2
coral	deepskyblue2	gray36	green1	grey60	lavender	magenta3	slateurquoise1	sienna1
chocolate	deepskyblue1	gray35	green	grey59	khaki4	magenta2	palegreen4	sienna
chocolatedarkpink	gray35	green	grey58	khaki3	magenta1	palegreen3	seashell4	wheat4
chocolate3	deepskyblue	gray34	grey100	grey57	khaki2	magenta1	palegreen2	seashell3
chocolate2	deeppink4	gray33	grey99	grey56	khaki1	linen	palegreen1	seashell2
chocolate1	deeppink3	gray32	grey98	grey55	khaki	limegreen	palegreen	seashell1
chocolate	deeppink2	gray31	grey97	grey54	ivory	lightyellow	goldenrod	wheat
chartreuse4	deeppink1	gray30	grey96	grey53	ivory3	lightyellow3	orchid4	seagreen4
chartreuse3	deeppink	gray29	grey95	grey52	ivory2	lightyellow2	orchid3	seagreen3
chartreuse2	darkviolet	gray28	grey94	grey51	ivory1	lightyellow1	orchid2	seagreen2
chartreuse1	darkturquoise	gray27	grey93	grey50	ivory	lightyellow	orchid1	violetred1
chartreuse	darkslategray	gray26	grey92	grey49	indianred	lightsteelblue4	orchid	seagreen1
cadetblue1	darkslategray4	gray25	grey91	grey48	indianred	lightsteelblue4	orchid	seagreen
cadetblue1	darkslategray3	gray24	grey90	grey47	indianred	lightsteelblue4	dangere	sienna
cadetblue1	darkslategray2	gray23	grey89	grey46	indianred	lightsteelblue4	dangere3	turquoise4
cadetblue1	darkslategray1	gray22	grey88	grey45	indianred	lightsteelblue4	dangere2	salmon4
cadetblue1	darkslategray	gray21	grey87	grey44	indianred	lightsteelblue4	dangere1	turquoise3
burlywood4	darkslategray	gray20	grey86	grey43	hotpink4	lightslategray	orange4	salmon1
burlywood4	darkslategray4	gray20	grey85	grey42	hotpink3	lightslategray	orange4	turquoise
burlywood3	darkslategray4	gray19	grey84	grey41	hotpink2	lightslateblue	orange3	tomato4
burlywood3	darkslategray3	gray18	grey83	grey40	hotpink1	lightskyblue4	orange2	royalblue4
burlywood3	darkslategray2	gray17	grey82	grey39	hotpink1	lightskyblue3	orange1	tomato2
burlywood3	darkslategray1	gray16	grey81	grey38	hotpink1	lightskyblue2	orange	royalblue3
brown4	darkseagreen	gray15	grey80	grey37	honeydew8	lightskyblue1	olivedrab4	royalblue2
brown3	darksalmon	gray14	grey79	grey36	honeydew8	lightskyblue1	olivedrab3	thistle4
brown2	darkred	gray13	grey78	grey35	honeydew8	lightskyblue1	olivedrab2	rosybrown4
brown1	darkorchid4	gray12	grey77	grey34	honeydew8	lightskyblue1	olivedrab1	thistle3
brown	darkorchid3	gray11	grey76	grey33	honeydew8	lightskyblue1	olivedrab1	rosybrown3
blueviolet	darkorchid2	gray10	grey75	grey32	honeydew8	lightskyblue1	olivedrab1	thistle1
blue4	darkorchid1	gray9	grey74	grey31	honeydew8	lightskyblue1	olivedrab1	rosybrown2
blue3	darkorchid	gray8	grey73	grey30	honeydew8	lightskyblue1	olivedrab1	thistle
blue2	darkorange4	gray7	grey72	grey29	honeydew8	lightskyblue1	olivedrab1	rosybrown
blue1	darkorange3	gray6	grey71	grey28	honeydew8	lightskyblue1	olivedrab1	tan4
blue	darkorange2	gray5	grey70	grey27	honeydew8	lightskyblue1	olivedrab1	tan3
nchedalmond	darkorange1	gray4	grey69	grey26	honeydew8	lightskyblue1	olivedrab1	tan2
black	darkorange	gray3	grey68	grey25	honeydew8	lightskyblue1	olivedrab1	tan1
bisque4	darkolivegreen4	gray2	grey67	grey24	honeydew8	lightskyblue1	olivedrab1	tan
bisque3	darkolivegreen3	gray1	grey66	grey23	honeydew8	lightskyblue1	olivedrab1	steelblue4
bisque2	darkolivegreen2	gray0	grey65	grey22	honeydew8	lightskyblue1	olivedrab1	steelblue3
bisque1	darkolivegreen1	gray1	grey64	grey21	honeydew8	lightskyblue1	olivedrab1	steelblue2
bisque	darkolivegreen1	gray1	grey63	grey20	honeydew8	lightskyblue1	olivedrab1	steelblue1
beige	darkolivedero	gray2	grey62	grey19	honeydew8	lightskyblue1	olivedrab1	springgreen
azure4	darkkhaki	goldenrod2	grey61	grey18	honeydew8	lightskyblue1	olivedrab1	plum4
azure3	darkgray	goldenrod1	grey61	grey17	honeydew8	lightskyblue1	olivedrab1	springgreen
azure2	darkgreen	goldenrod	grey60	grey16	honeydew8	lightskyblue1	olivedrab1	snow4
azure1	darkgray	goldenrod4	grey59	grey15	honeydew8	lightskyblue1	olivedrab1	snow3
azure	darkolivedero4	goldenrod3	grey58	grey14	honeydew8	lightskyblue1	olivedrab1	snow2

0	1	2	3	4
□	○	△	+	×
5	6	7	8	9
◇	▽	⊗	✳	◇
10	11	12	13	14
⊕	⊗	田	⊗	□
15	16	17	18	19
■	●	▲	◆	●
20	21	22	23	24
●	●	■	◆	▲

Figure 17.1: The shapes available to ggplot

17.2 Shapes