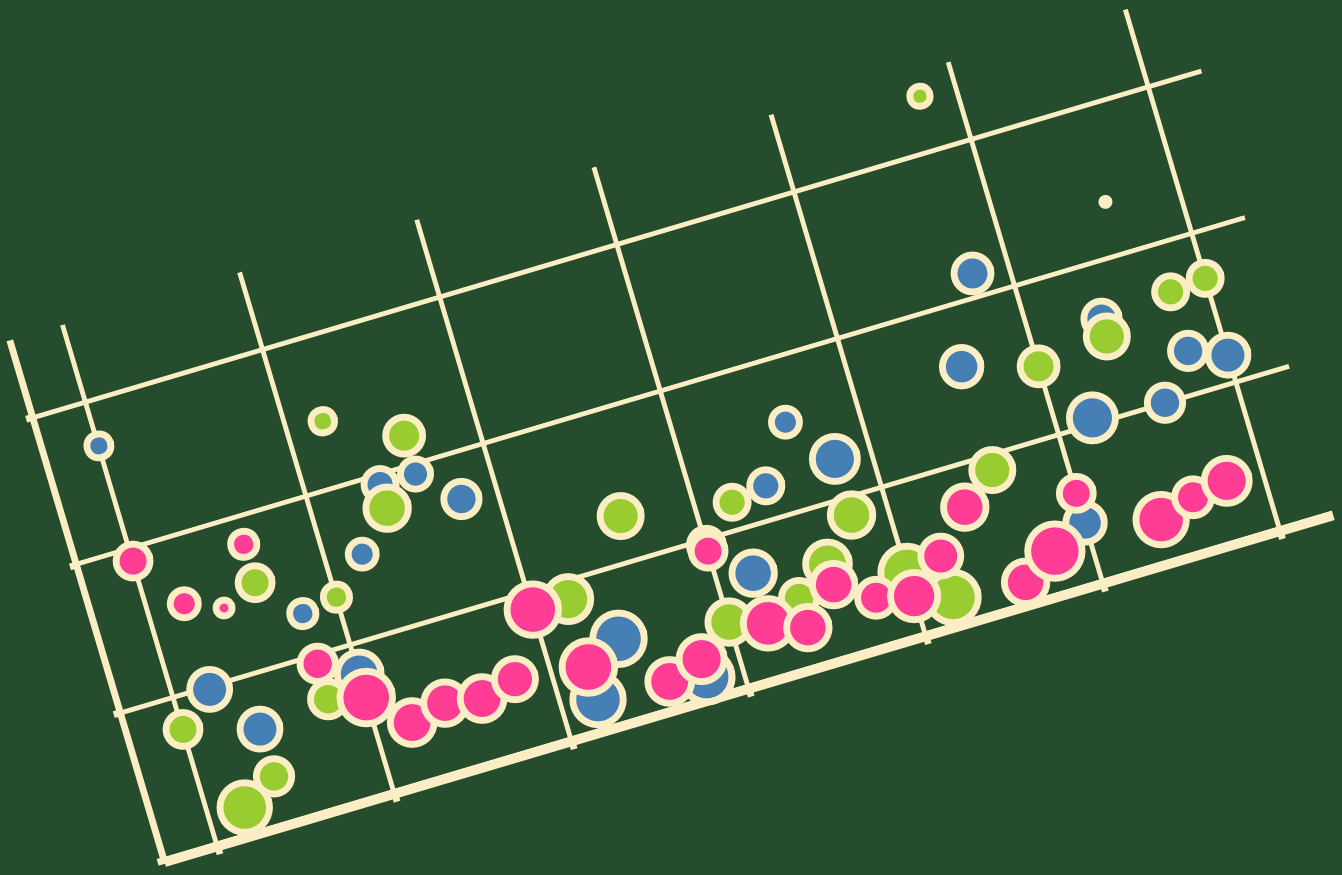


THE HITCHHIKER'S GUIDE TO GGPLOT2 IN R

JODIE BURCHELL & MAURICIO VARGAS

THE HITCHHIKER'S GUIDE TO GGPLOT2 IN R

JODIE BURCHELL & MAURICIO VARGAS

The Hitchhiker's Guide to Ggplot2 in R

Jodie Burchell & Mauricio Vargas

This book is for sale at https://leanpub.com/hitchhikers_ggplot2

This version was published on 11th June, 2016.

ISBN: 978-956-362-693-3

Contents

What to expect from this book

This is a technical book. The book aims to get straight to the point, and the writing style is similar to a recipe with detailed instructions. It is assumed that you know the basics of R and that you want to learn to create beautiful plots.

Each chapter will explain how to create a different type of plot, and will take you step-by-step from a basic plot to a highly customised graph. The chapters are ordered by degree of graph complexity.

Every chapter is independent from the others. You can read the whole book or go to a section of your interest, and we are sure that it will be easy to understand the instructions and reproduce our examples without reading the earlier chapters.

Chapters 1-4 are based in an international trade dataset created from different sources (Chile Customs, Central Bank of Chile and General Directorate of International Economic Relations) that was used in a post about China and Chile. The rest of the chapters are based in R's datasets and datasets obtained from Quandl's API.

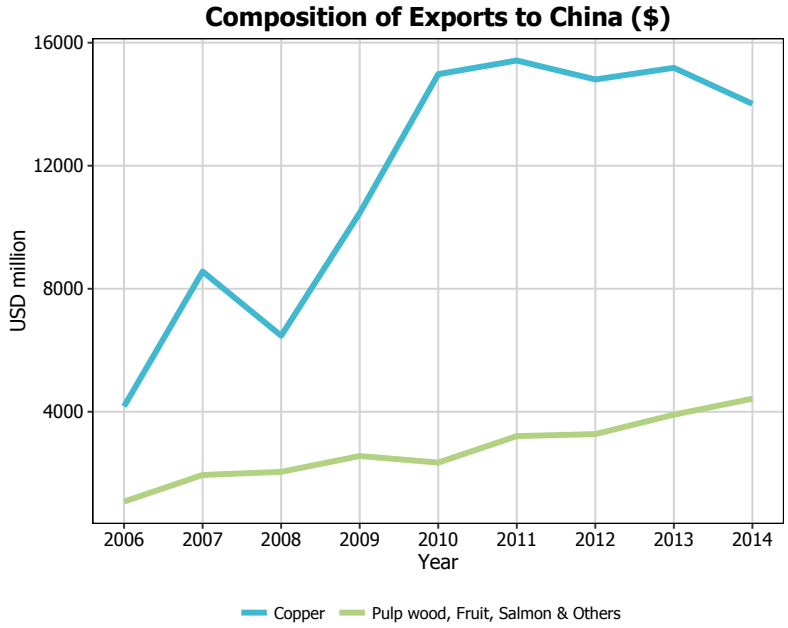
We invite you to stay in touch and read the authors' blogs where they publish articles about R and Statistics. Jodie's blog is Standard error and Mauricio's blog is Reimagined Invention.

Don't panic!

CHAPTER 1

Line plots

In this chapter, we will work towards creating the line plot below. We will take you from a basic line plot and explain all the customisations we add to the code step-by-step.



In this chapter we will be using a dataset about trade between Chile and China. You can find this dataset, as well as the rmarkdown file containing all of the code used in this chapter, in the book's Github repo. We encourage you to fork this repo to practise as you read this material.

The first thing to do is load in the data and libraries, as below:

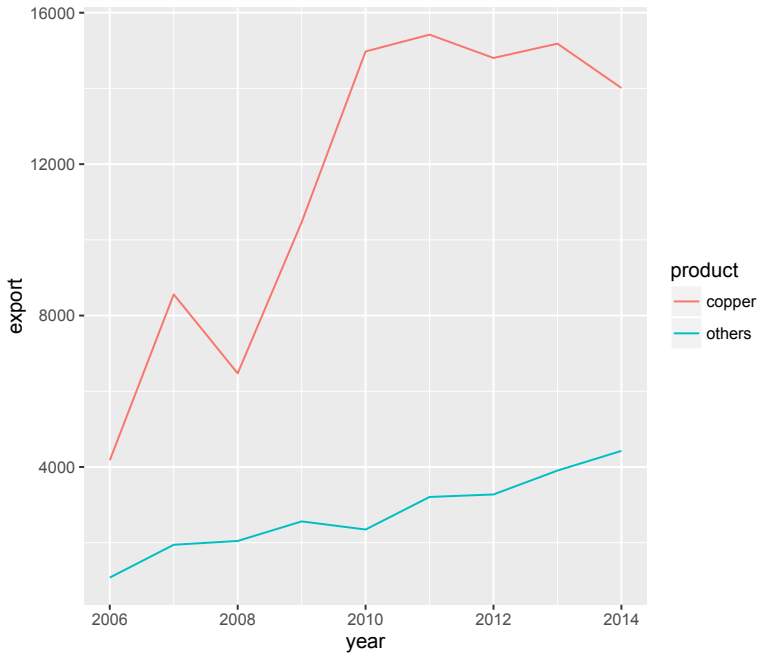
```
library(ggplot2)
library(ggthemes)
library(extrafont)
```

```
charts.data <- read.csv("copper-data-for-book.csv")
```

1.1. Basic graph

In order to initialise a plot we tell ggplot that `charts.data` is our data. We then instruct ggplot to render this as a line plot by adding the `geom_line` command. We can specify our x-axis, y-axis and grouping variables within `geom_line` using the `x`, `y` and `colour` arguments respectively.

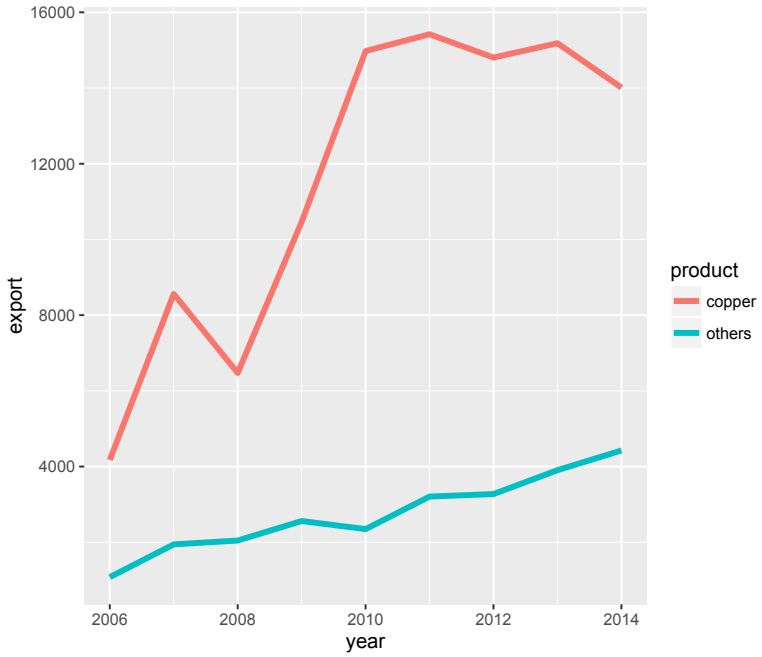
```
p1 <- ggplot() + geom_line(aes(y = export, x = year, colour = product),
  data = charts.data, stat="identity")
p1
```



1.2. Adjusting line width

To change the line width, we add a `size` argument to `geom_line`.

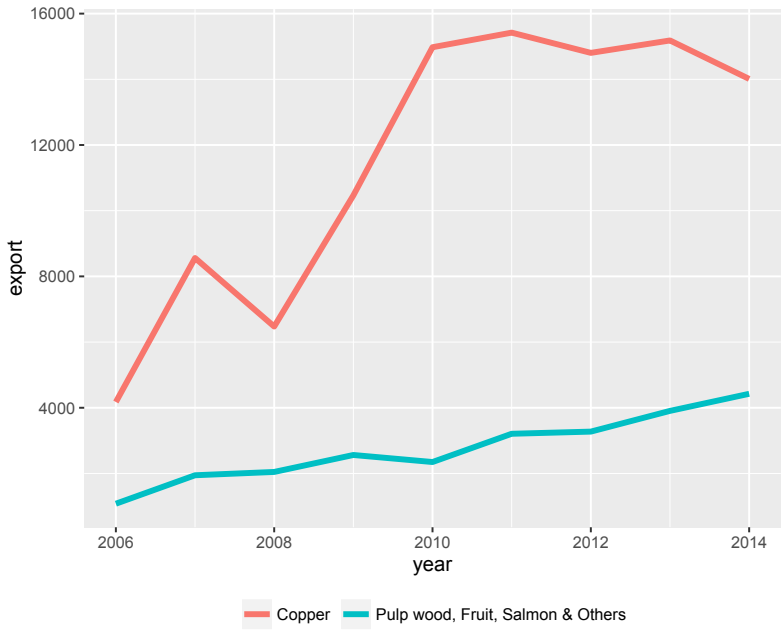
```
p1 <- ggplot() + geom_line(aes(y = export, x = year, colour = product), size=1.5,
  data = charts.data, stat="identity")
p1
```



1.3. Adjusting the legend

To change the name of the groups in the legend, we need to re-factor our data labels in our `charts.data` data frame. Then we move the legend to the bottom using the `theme` command.

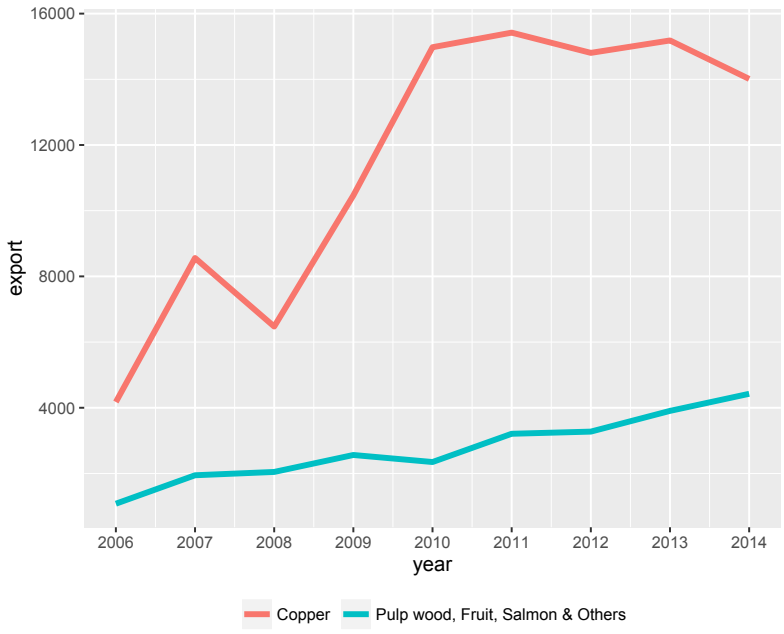
```
charts.data <- as.data.frame(charts.data)
charts.data$product <- factor(charts.data$product, levels = c("copper","others"),
  labels = c("Copper ","Pulp wood, Fruit, Salmon & Others"))

p1 <- ggplot() +
  geom_line(aes(y = export, x = year, colour = product), size=1.5,
    data = charts.data, stat="identity") +
  theme(legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank())
p1
```



1.4. Adjusting x-axis scale

To change the axis tick marks, we use the `scale_x_continuous` and/or `scale_y_continuous` commands. Let's convert the x-axis into yearly increments using the `breaks` argument. We'll make this more efficient by using the `seq` function, a base R function that indicates the start and endpoints and the units to increment by respectively. See `help(seq)` for more information.

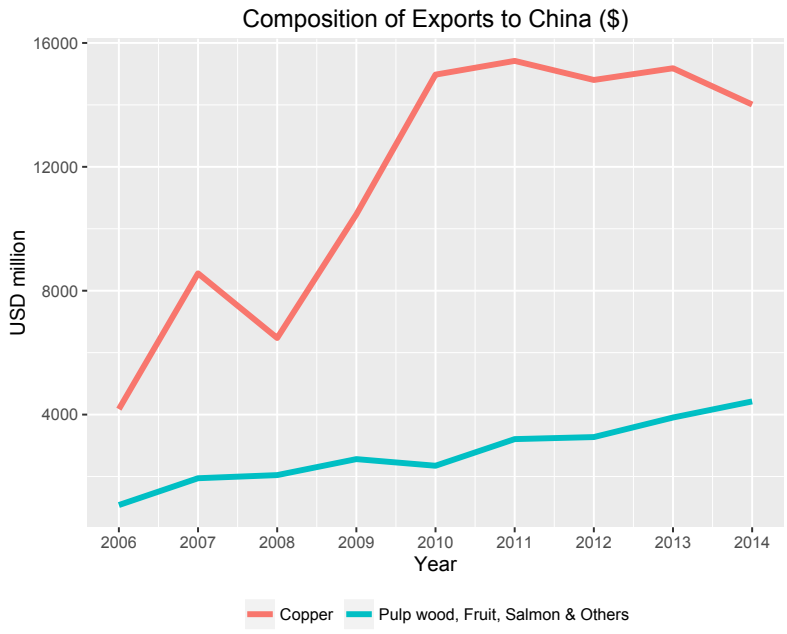
```
p1 <- p1 + scale_x_continuous(breaks=seq(2006,2014,1))
p1
```



1.5. Adjusting axis labels & adding title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument, and to change the axis names we use the `labs` command.

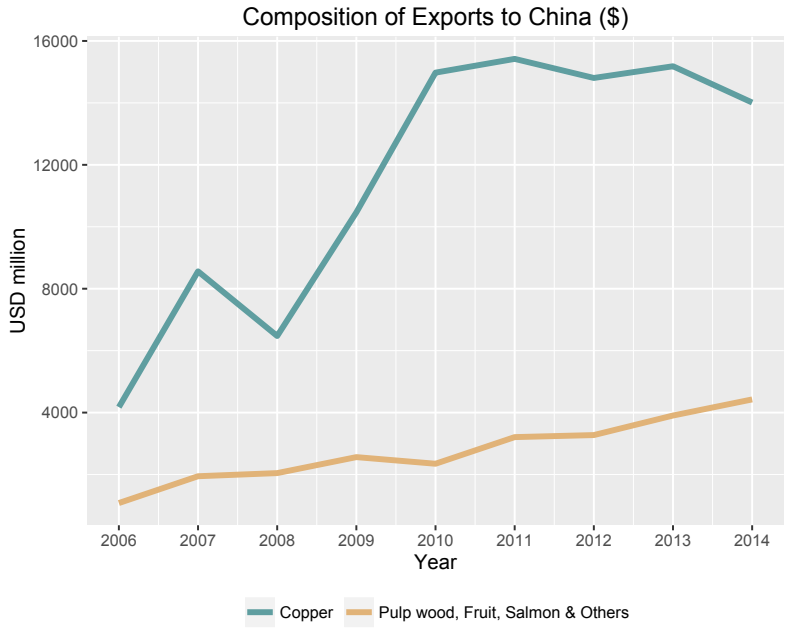
```
p1 <- p1 + ggtitle("Composition of Exports to China ($)") +
  labs(x="Year", y="USD million")
p1
```



1.6. Adjusting color palette

To change the colours, we use the `scale_colour_manual` command. You can either using exact HEX codes (as we have in the chart below), or a valid colour name. A full list of colours recognised by R is here.

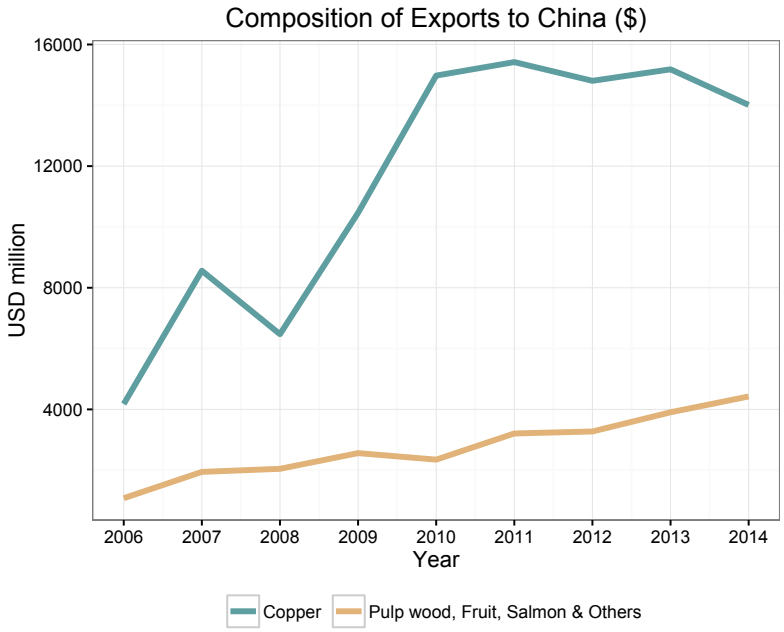
```
colour <- c("#5F9EA0", "#E1B378")
p1 <- p1 + scale_colour_manual(values=colour)
p1
```



1.7. Using the white theme

We'll start using a simple theme customisation made adding `theme_bw()` after `ggplot()`. That theme argument can be modified to use different themes.

```
p1 <- ggplot() +
  geom_line(aes(y = export, x = year, colour = product), size=1.5,
    data = charts.data, stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_colour_manual(values=colour) +
  theme_bw() +
  theme(legend.position="bottom",
    legend.direction="horizontal",
    legend.title = element_blank())
p1
```

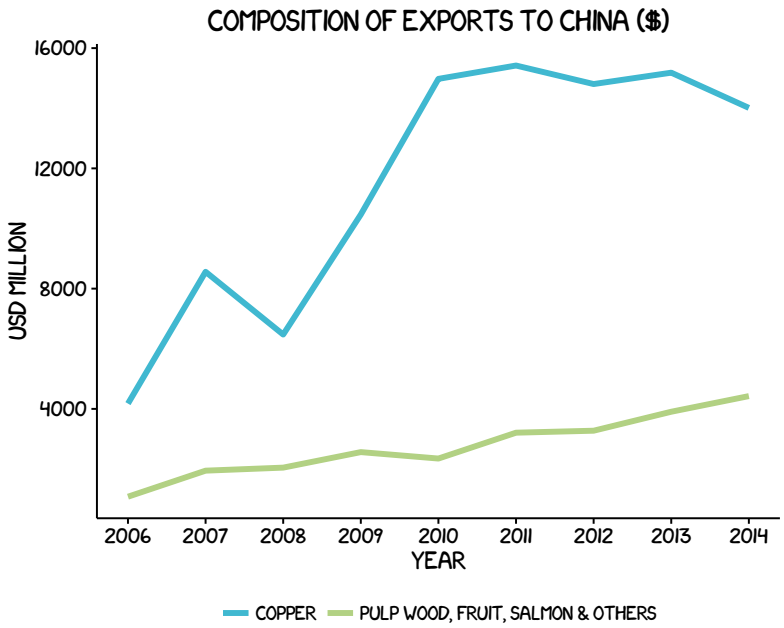


1.8. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
fill <- c("#40b8d0", "#b2d183")

p1 <- ggplot() +
  geom_line(aes(y = export, x = year, colour = product), size=1.5,
    data = charts.data, stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_color_manual(values=fill) +
  theme_bw() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.key=element_rect(fill="white", colour="white"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(), panel.border = element_blank(),
```

```
    panel.background = element_blank(),
    plot.title=element_text(family="xkcd-Regular"),
    text=element_text(family="xkcd-Regular"))
p1
```

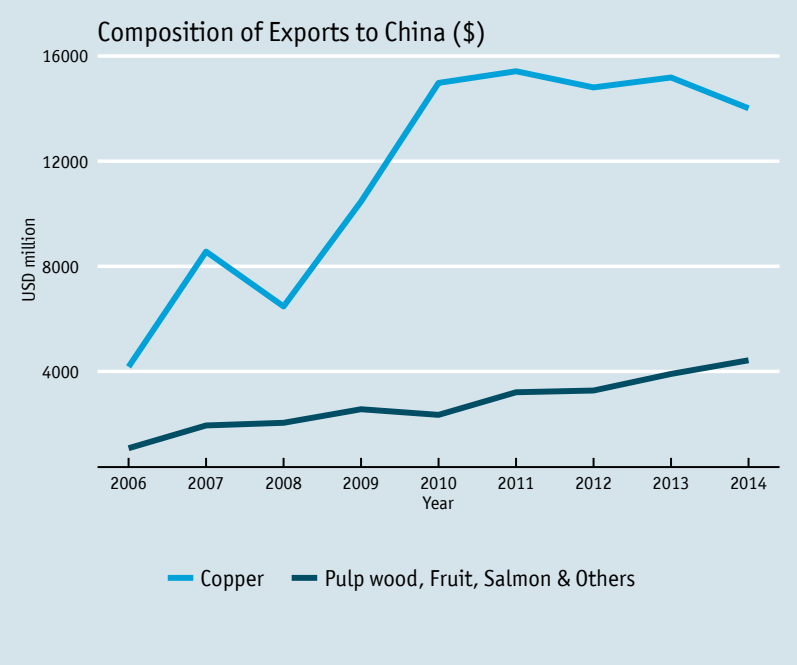


1.9. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these is here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important to note that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

```
p1 <- ggplot() +
  geom_line(aes(y = export, x = year, colour = product), size=1.5,
    data = charts.data, stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  theme_economist() + scale_colour_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.title = element_blank(),
    plot.title=element_text(family="OfficinaSanITC-Book"),
```

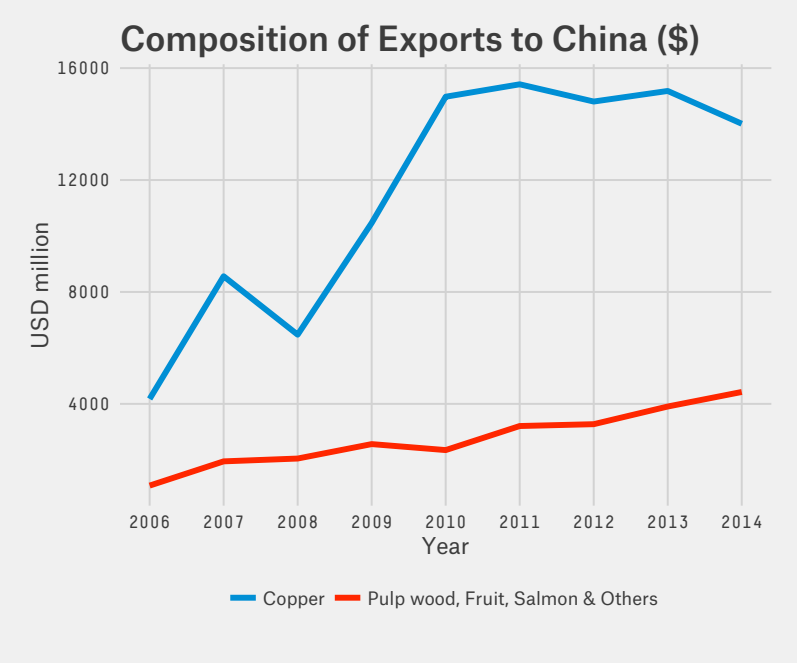
```
    text=element_text(family="OfficinaSanITC-Book"))
p1
```



1.10. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here respectively.

```
p1 <- ggplot() +
  geom_line(aes(y = export, x = year, colour = product), size=1.5,
    data = charts.data, stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  theme_fivethirtyeight() + scale_colour_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title=element_blank(),
    plot.title=element_text(family="Atlas Grotesk Medium"),
    legend.text=element_text(family="Atlas Grotesk Regular"),
    text=element_text(family="DecimaMonoPro"))
p1
```

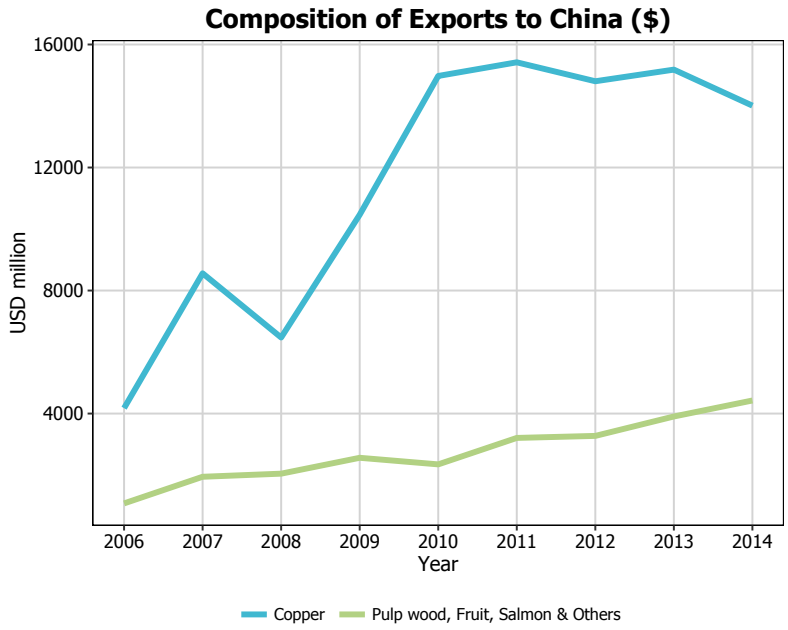


1.11. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here we present our original result shown at the beginning of the chapter.

```
colour <- c("#40b8d0", "#b2d183")

p1 <- ggplot() +
  geom_line(aes(y = export, x = year, colour = product), size=1.5,
    data = charts.data, stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_colour_manual(values=colour) +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.key=element_rect(fill="white", colour="white"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
```

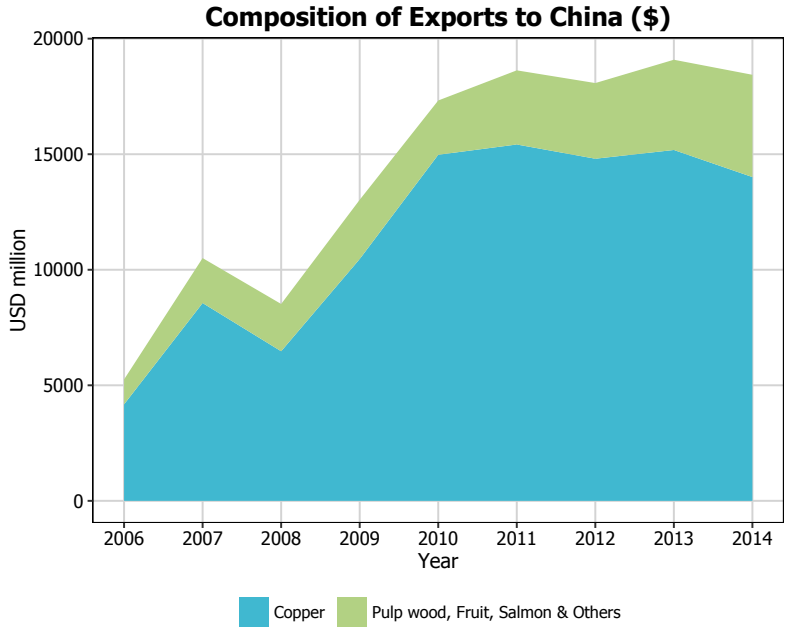
```
p1
```


Composition of Exports to China ($)
16000
12000
8000
4000
USD million
2006
2007
2008
2009
2010
2011
2012
2013
2014
Year
Copper
Pulp wood, Fruit, Salmon & Others

CHAPTER 2

Area plots

In this chapter, we will work towards creating the area plot below. We will take you from a basic area plot and explain all the customisations we add to the code step-by-step.



2.1. Basic graph

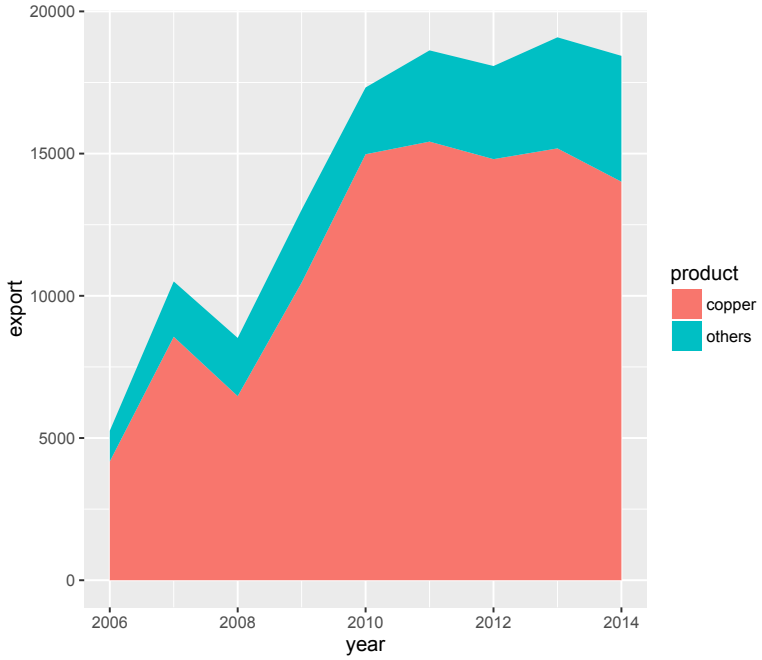
We will use the same data as in Chapter 1. The first thing to do is load in the data and libraries, as below:

```
library(ggplot2)
library(ggthemes)
library(extrafont)
library(plyr)
```

```
charts.data <- read.csv("copper-data-for-book.csv")
```

In order to initialise a plot we tell ggplot that `charts.data` is our data. We then instruct ggplot to render this as an area plot by adding the `geom_area` command. We can specify our x-axis, y-axis and grouping variables within `geom_area` using the `x`, `y` and `fill` arguments respectively.

```
charts.data <- read.csv("copper-data-for-book.csv")

p2 <- ggplot() + geom_area(aes(y = export, x = year, fill = product),
  data = charts.data, stat="identity")
p2
```



2.2. Adjusting legend position

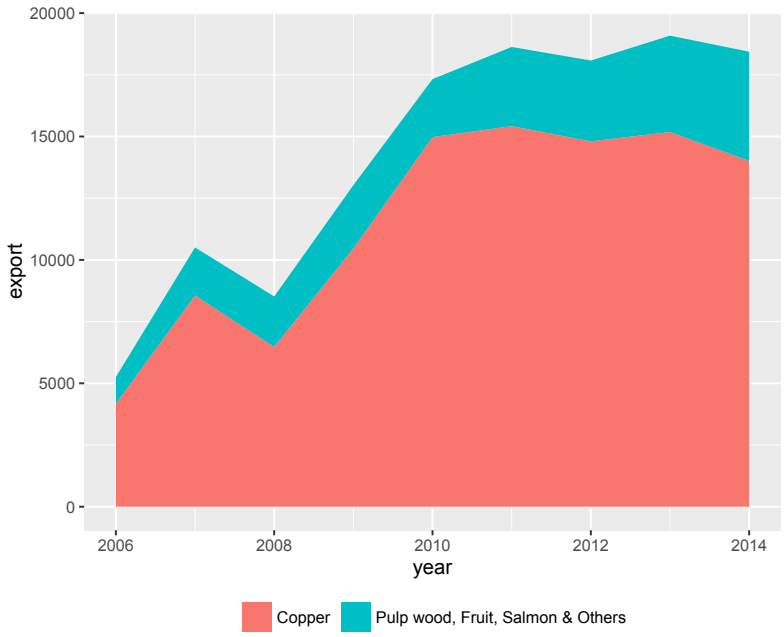
To adjust the position of the legend from the default spot of right of the graph, we add the `theme` option and specify the `legend.position="bottom"` argument. We can also change the title to blank using the `legend.title = element_blank()` argument and change the legend shape using the `legend.direction="horizontal"` argument.

```
charts.data <- ddply(charts.data, .(year), transform,
  pos = cumsum(export) - (0.5 * export))

p2 <- p2 + theme(legend.position="bottom", legend.direction="horizontal",
  legend.title = element_blank())
p2
```

2.3. Changing legend labels

To change the name of the groups in the legend, we need to re-factor our data labels in `charts.data` data frame.

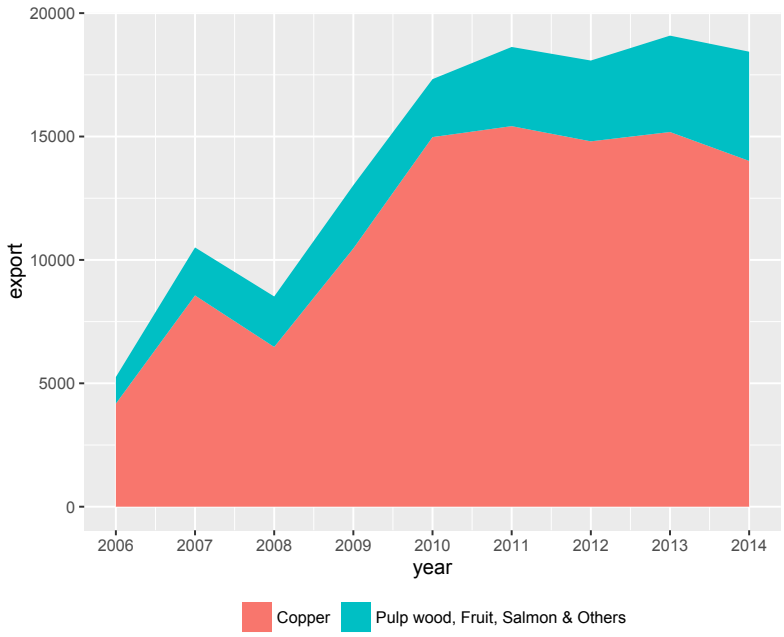
```
charts.data <- as.data.frame(charts.data)
charts.data$product <- factor(charts.data$product, levels = c("copper","others"),
  labels = c("Copper ","Pulp wood, Fruit, Salmon & Others"))

p2 <- ggplot() +
  geom_area(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  theme(legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank())
p2
```



2.4. Adjusting x-axis scale

To change the axis tick marks, we use the `scale_x_continuous` and/or `scale_y_continuous` commands. Let's convert the x-axis into yearly increments using the `breaks` argument. We'll make this more efficient by using the `seq` function, a base R function that indicates the start and endpoints and the units to increment by respectively. See `help(seq)` for more information.

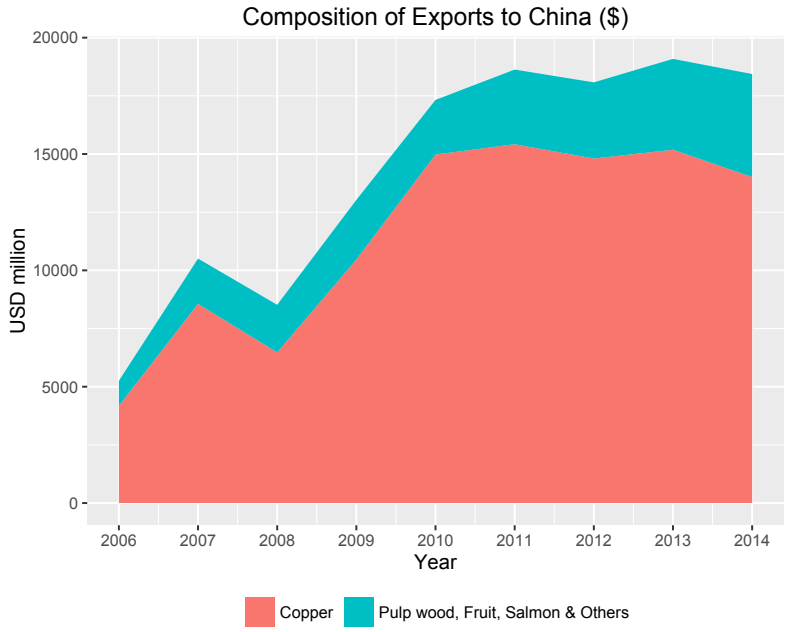
```
p2 <- p2 + scale_x_continuous(breaks=seq(2006,2014,1))
p2
```



2.5. Adjusting axis labels & adding title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument, and to change the axis names we use the `labs` command.

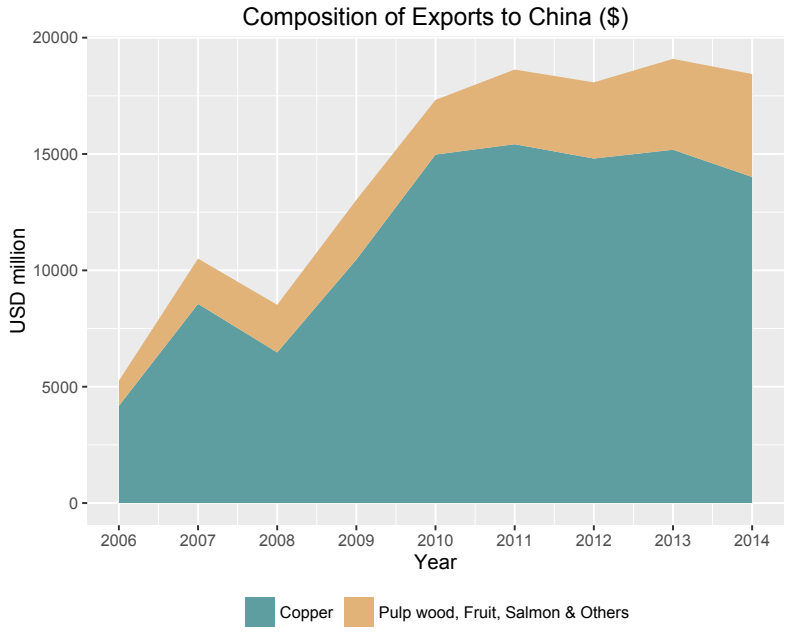
```
p2 <- p2 + ggtitle("Composition of Exports to China ($)") +
  labs(x="Year", y="USD million")
p2
```



2.6. Adjusting color palette

To change the colours, we use the `scale_colour_manual` command. Note that you can reference the specific colours you'd like to use with specific HEX codes. You can also reference colours by name, with the full list of colours recognised by R here.

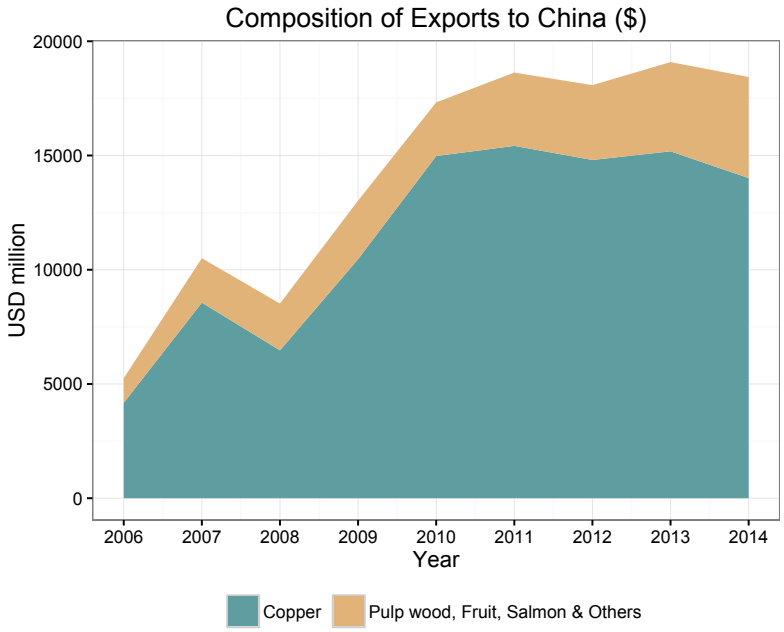
```
fill <- c("#5F9EA0", "#E1B378")
p2 <- p2 + scale_fill_manual(values=fill)
p2
```



2.7. Using the white theme

We can also change the overall look of the graph using themes. We'll start using a simple theme customisation by adding `theme_bw()` after `ggplot()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p2 <- ggplot() +
  geom_area(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_fill_manual(values=fill) +
  theme_bw() +
  theme(legend.position="bottom",
    legend.direction="horizontal",
    legend.title = element_blank())
p2
```

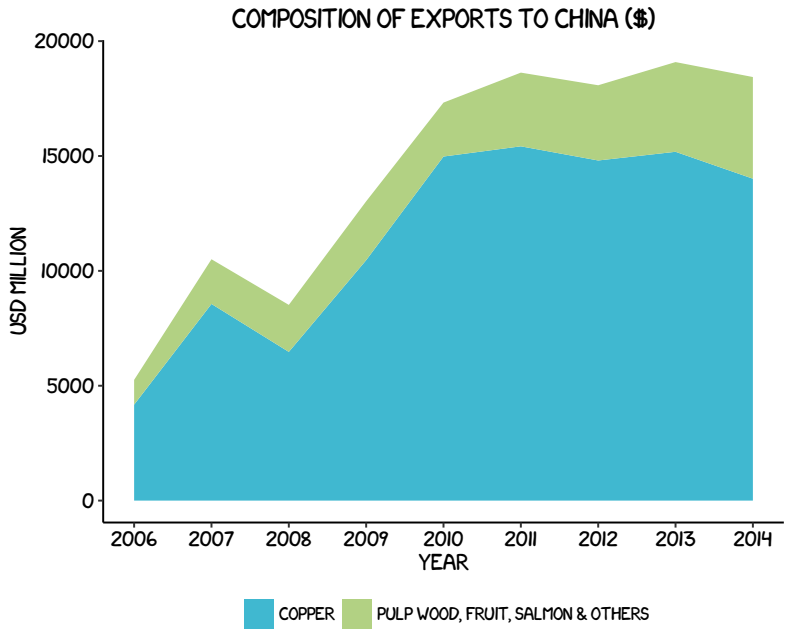


2.8. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
fill <- c("#40b8d0", "#b2d183")

p2 <- ggplot() +
  geom_area(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_fill_manual(values=fill) +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.key=element_rect(fill="white", colour="white"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(), panel.border = element_blank(),
    panel.background = element_blank(),
```

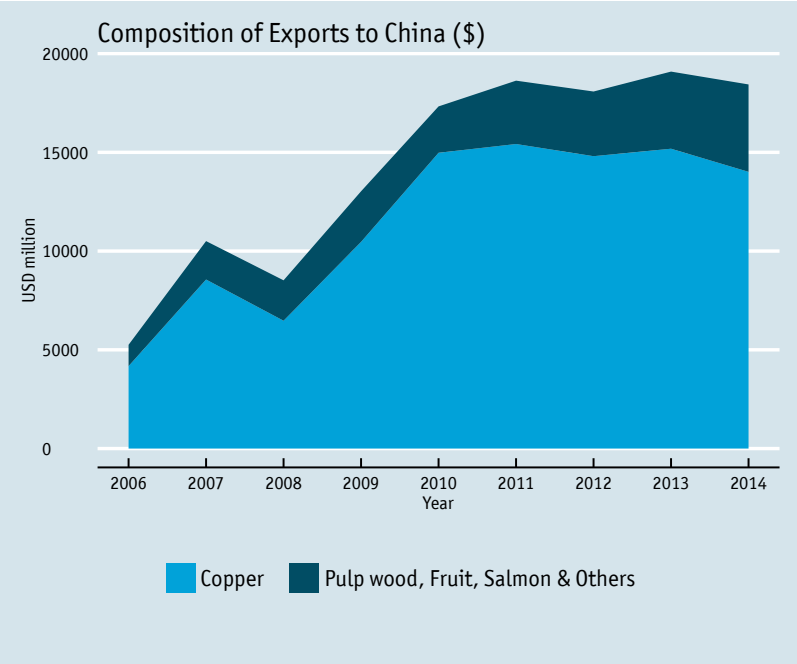
```
    plot.title=element_text(family="xkcd-Regular"),
    text=element_text(family="xkcd-Regular"))
p2
```



2.9. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

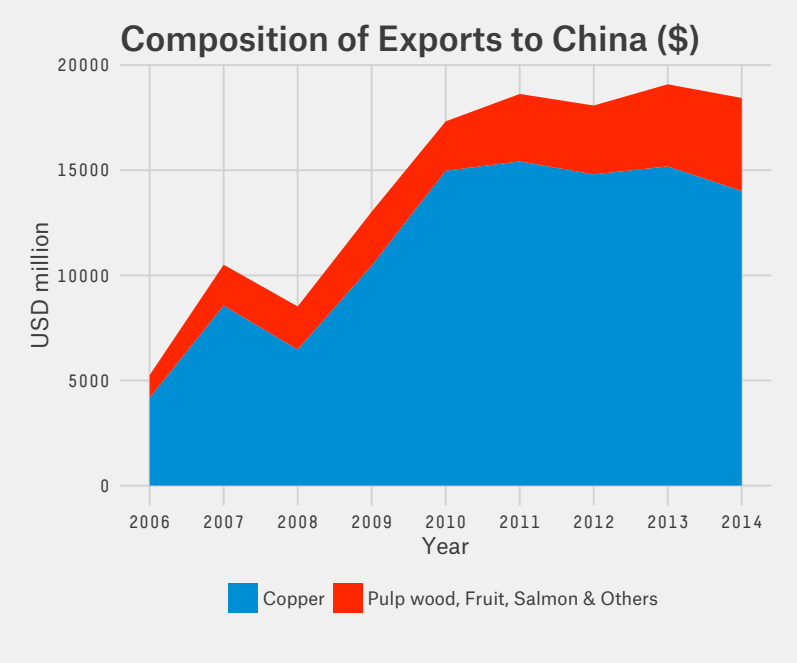
```
p2 <- ggplot() +
  geom_area(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.title = element_blank(),
    plot.title=element_text(family="OfficinaSanITC-Book"),
    text=element_text(family="OfficinaSanITC-Book"))
p2
```



2.10. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

```
p2 <- ggplot() +
  geom_area(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title=element_blank(),
    plot.title=element_text(family="Atlas Grotesk Medium"),
    legend.text=element_text(family="Atlas Grotesk Regular"),
    text=element_text(family="DecimaMonoPro"))
p2
```

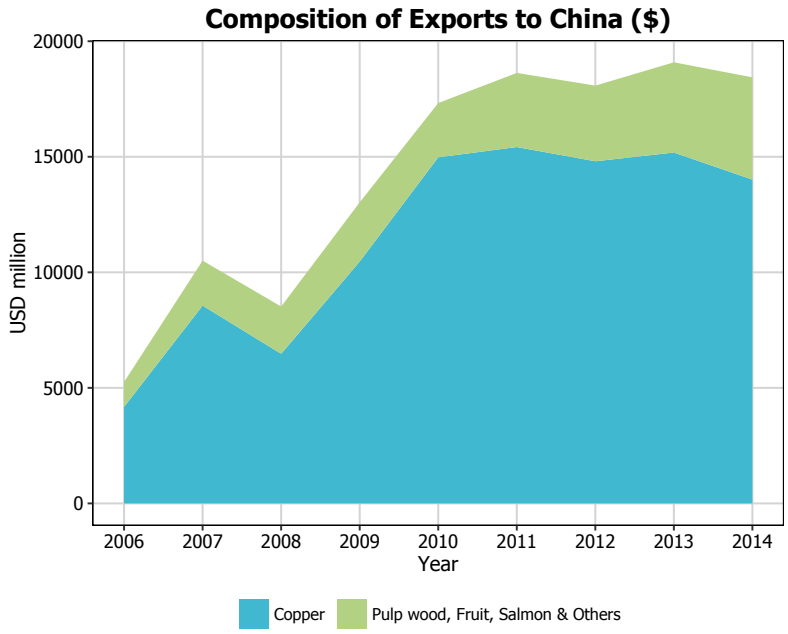


2.11. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here we present our original result shown at the beginning of the chapter.

```
fill <- c("#40b8d0", "#b2d183")

p2 <- ggplot() +
  geom_area(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_fill_manual(values=fill) +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.key=element_rect(fill="white", colour="white"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
```

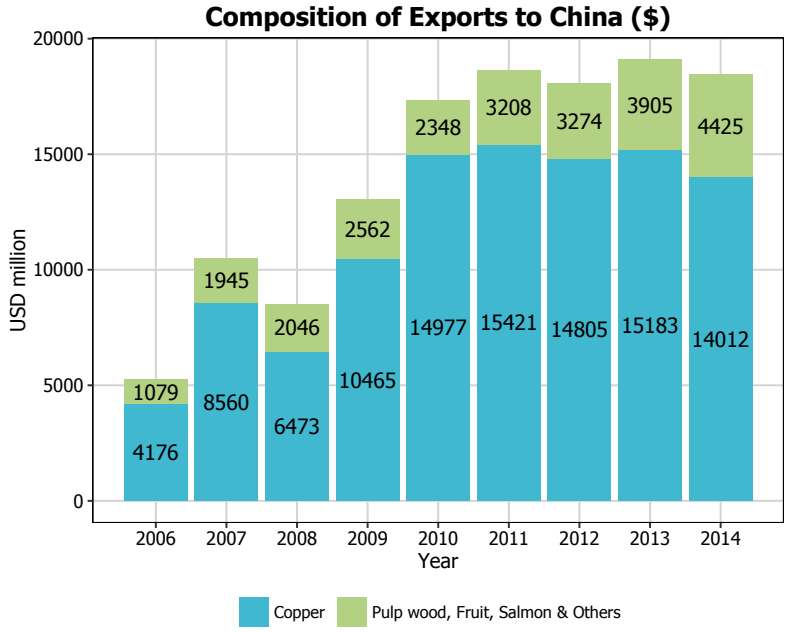
```
p2
```


Composition of Exports to China ($)
20000
15000
10000
5000
0
USD million
2006
2007
2008
2009
2010
2011
2012
2013
2014
Year
Copper
Pulp wood, Fruit, Salmon & Others

CHAPTER 3

Bar plots

In this chapter, we will work towards creating the area plot below. We will take you from a basic bar plot and explain all the customisations we add to the code step-by-step.



3.1. Basic graph

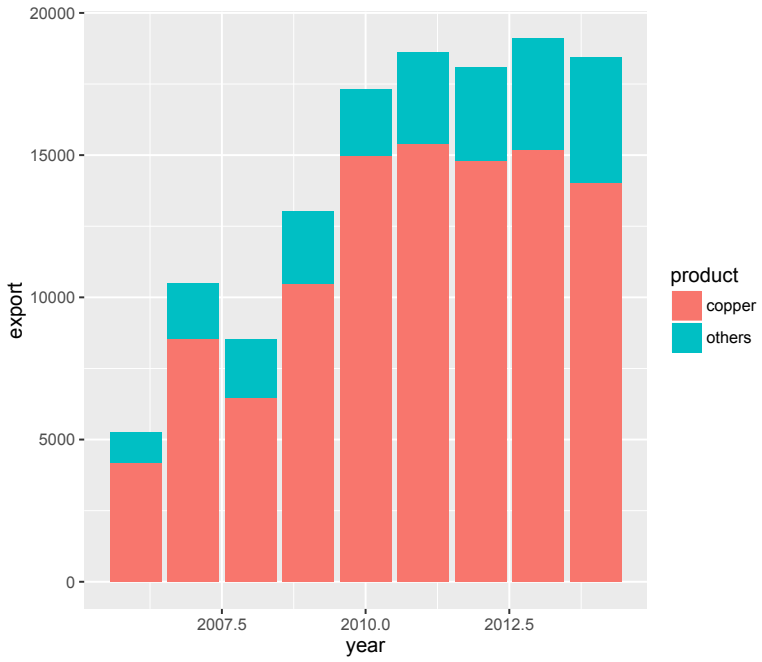
We will use the same data as in Chapter 1. The first thing to do is load in the data and libraries, as below:

```
library(ggplot2)
library(ggthemes)
library(extrafont)
library(plyr)
```

```
library(scales)

charts.data <- read.csv("copper-data-for-book.csv")
```

In order to initialise a plot we tell ggplot that `charts.data` is our data. We then instruct ggplot to render this as a bar plot by adding the `geom_bar` command. We can specify our x-axis, y-axis and grouping variables within `geom_bar` using the `x`, `y` and `fill` arguments respectively.

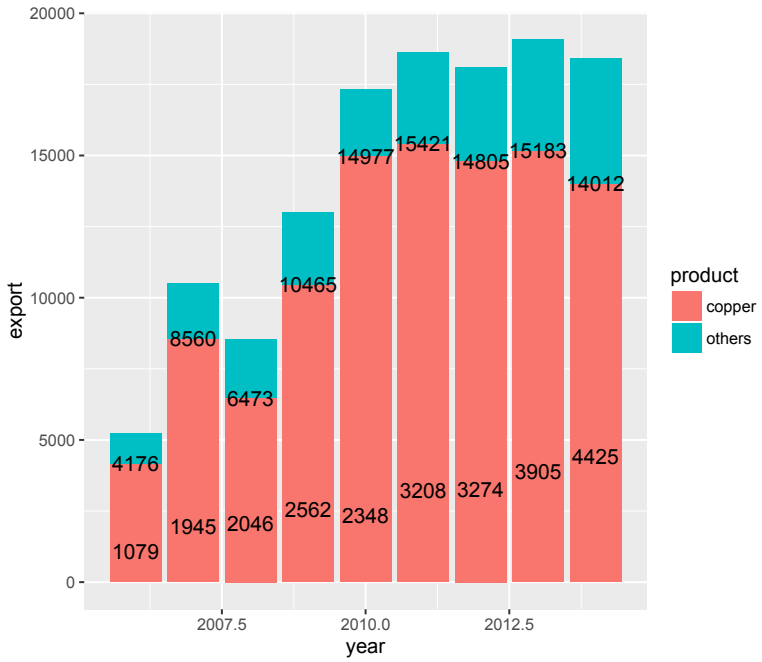
```
p3 <- ggplot() + geom_bar(aes(y = export, x = year, fill = product),
  data = charts.data, stat="identity")
p3
```



3.2. Adding data labels

To label the bars according to some variable in the data, we add the `label` argument to the `ggplot(aes())` option. In this case, we have labelled the bars with numbers from the `export` variable.

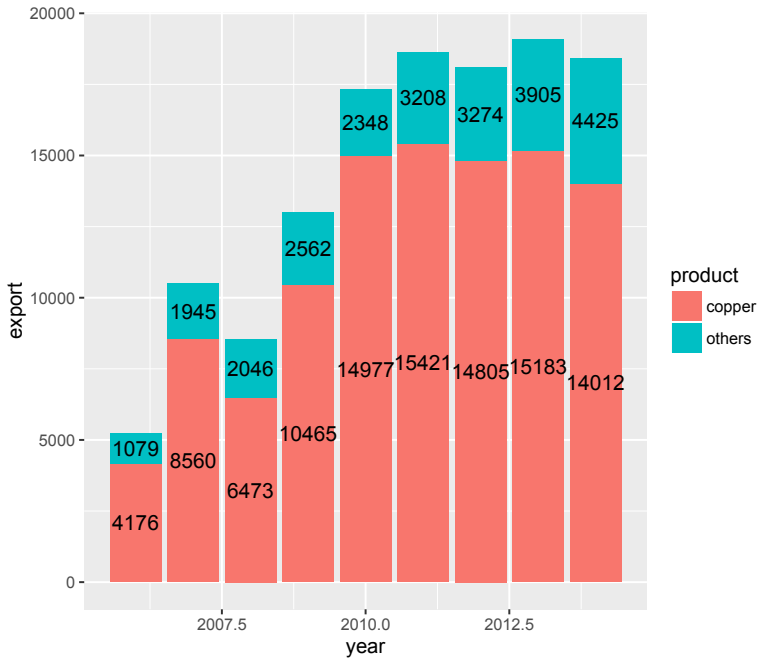
```
p3 <- p3 + geom_text(data=charts.data, aes(x = year, y = export, label = export),
  size=4)
p3
```



3.3. Adjusting data labels position

To adjust the position of the data labels from the default placement, we use the `ddply` function on the data, and add a new variable called `pos` to our data. This variable is at the centre of each bar and can be used to specify the position of the labels by assigning it to the `y` argument in `geom_text(aes())`.

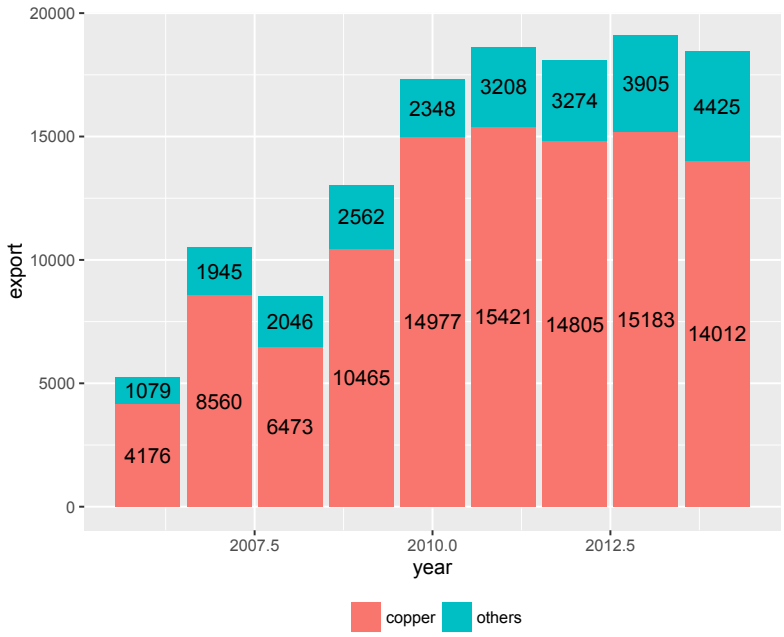
```
charts.data <- ddply(charts.data, .(year), transform,
  pos = cumsum(export) - (0.5 * export))

p3 <- ggplot() + geom_bar(aes(y = export, x = year, fill = product),
  data = charts.data, stat="identity")
p3 <- p3 + geom_text(data=charts.data, aes(x = year, y = pos, label = export),
  size=4)
p3
```



3.4. Adjusting legend position

To adjust the position of the legend from the default spot of right of the graph, we add the `theme` option and specify the `legend.position="bottom"` argument. We can also change the title to blank using the `legend.title = element_blank()` argument and change the legend shape using the `legend.direction="horizontal"` argument.

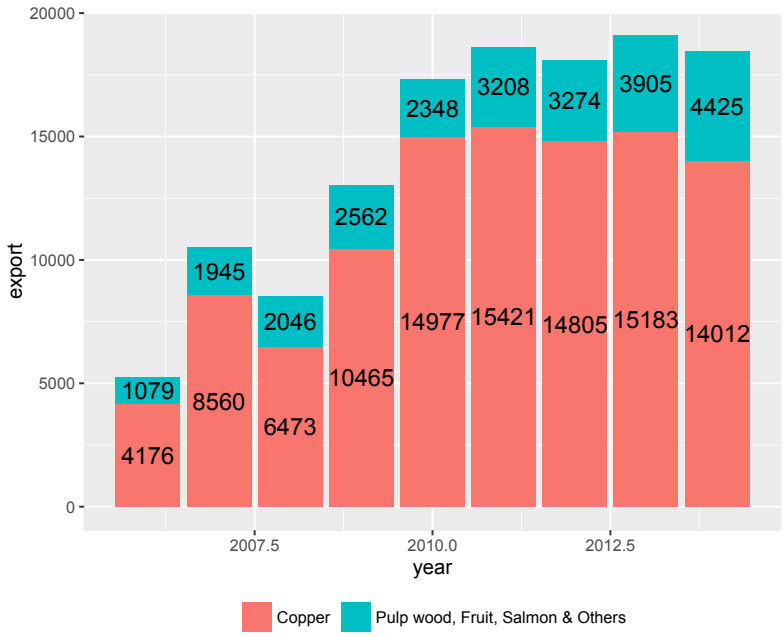
```
p3 <- p3 + theme(legend.position="bottom", legend.direction="horizontal",
  legend.title = element_blank())
p3
```



3.5. Changing legend labels

To change the name of the groups in the legend, we need to re-factor our data labels in `charts.data` data frame.

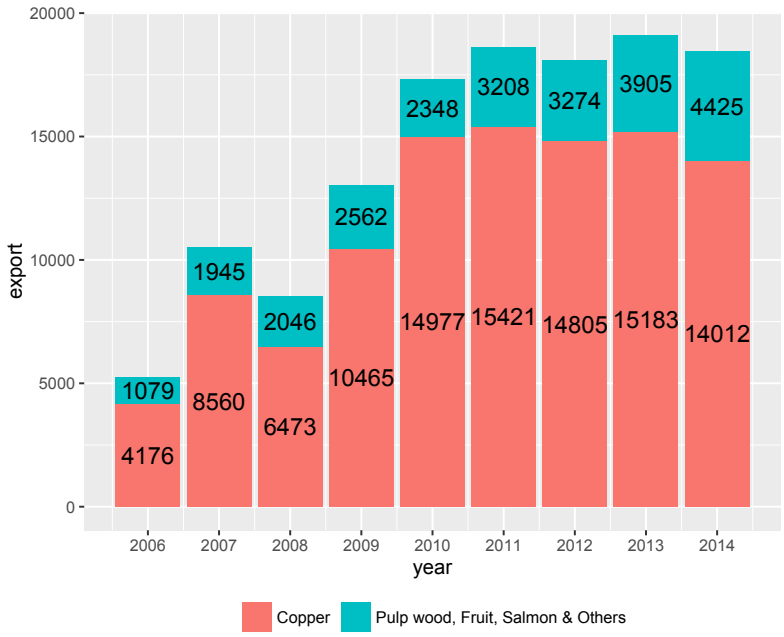
```
charts.data$product <- factor(charts.data$product, levels = c("copper","others"),
  labels = c("Copper ","Pulp wood, Fruit, Salmon & Others"))

p3 <- ggplot() + geom_bar(aes(y = export, x = year, fill = product),
    data = charts.data, stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = export, size=4),
    show.legend = F) +
  theme(legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank())
p3
```



3.6. Adjusting x-axis scale

To change the axis tick marks, we use the `scale_x_continuous` and/or `scale_y_continuous` commands. Let's convert the x-axis into yearly increments using the `breaks` argument. We'll make this more efficient by using the `seq` function, a base R function that indicates the start and endpoints and the units to increment by respectively. See `help(seq)` for more information.

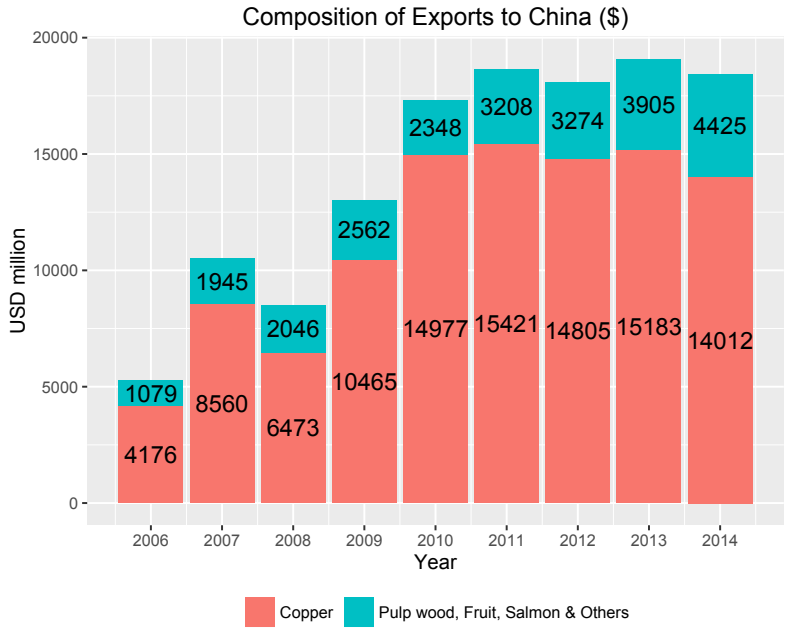
```
p3 <- p3 + scale_x_continuous(breaks=seq(2006,2014,1))
p3
```



3.7. Adjusting axis labels & adding title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument, and to change the axis names we use the `labs` command.

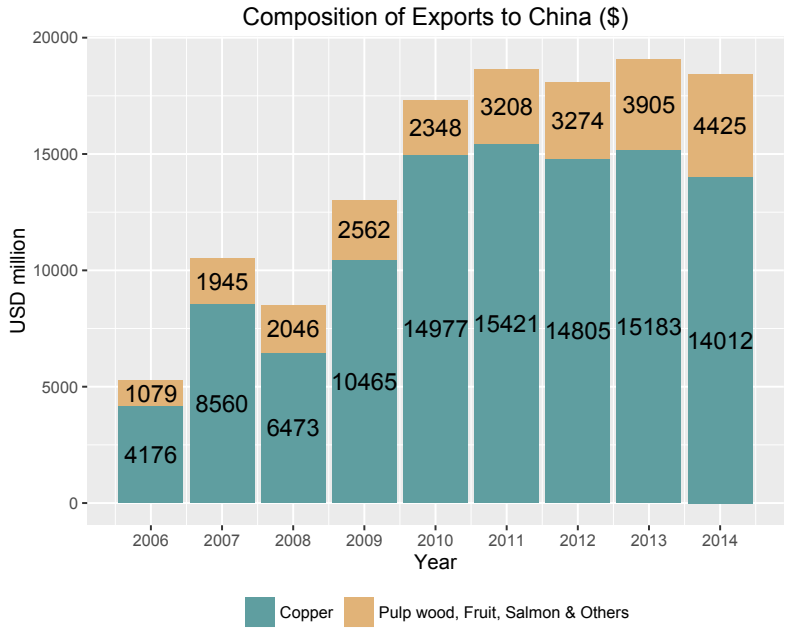
```
p3 <- p3 + ggtitle("Composition of Exports to China ($)") +
  labs(x="Year", y="USD million")
p3
```



3.8. Adjusting color palette

To change the colours, we use the `scale_colour_manual` command. Note that you can reference the specific colours you'd like to use with specific HEX codes. You can also reference colours by name, with the full list of colours recognised by R here.

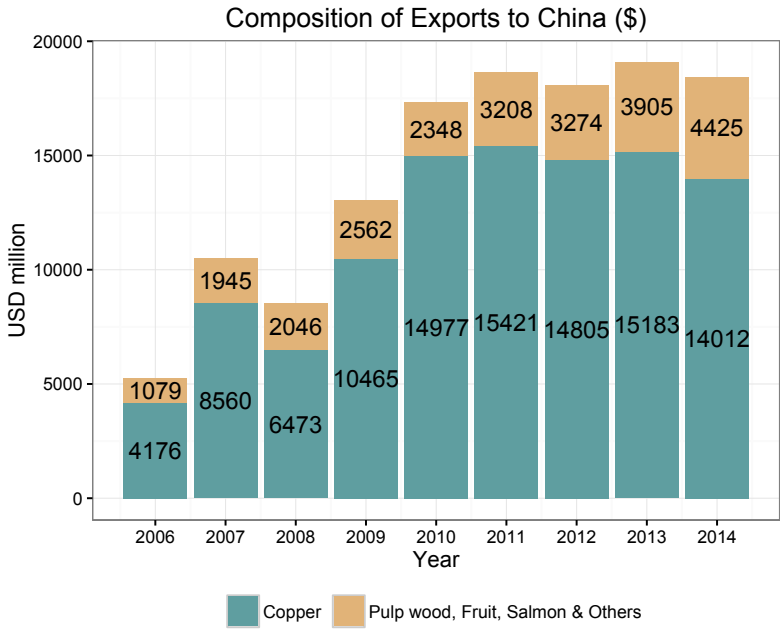
```
fill <- c("#5F9EA0", "#E1B378")
p3 <- p3 + scale_fill_manual(values=fill)
p3
```



3.9. Using the white theme

We can also change the overall look of the graph using themes. We'll start using a simple theme customisation by adding `theme_bw()` after `ggplot()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p3 <- ggplot() +
  geom_bar(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = export, size=4),
    show.legend = F) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_fill_manual(values=fill) +
  theme_bw() +
  theme(legend.position="bottom",
    legend.direction="horizontal",
    legend.title = element_blank())
p3
```

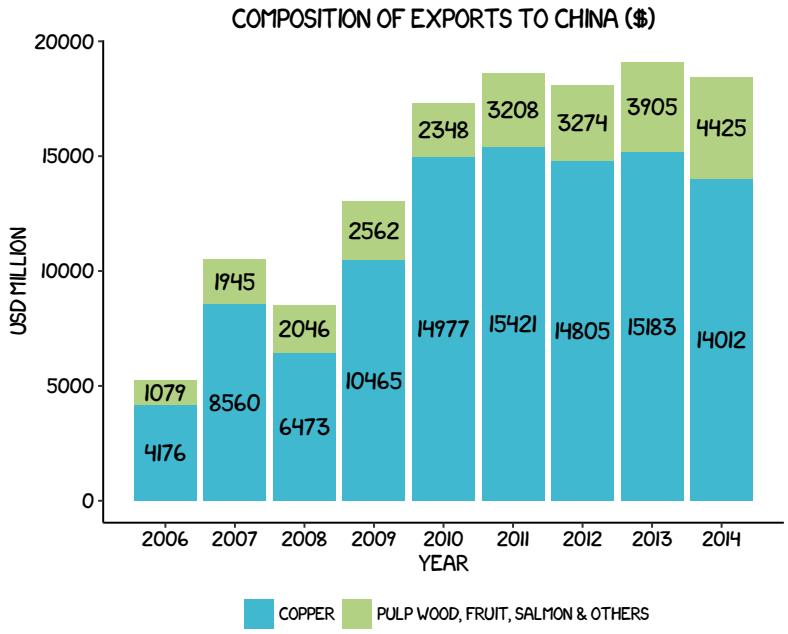


3.10. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
fill <- c("#40b8d0", "#b2d183")

p3 <- ggplot() +
  geom_bar(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = export), colour="black",
    family="xkcd-Regular", size = 4, show.legend = F) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_fill_manual(values=fill) +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.key=element_rect(fill="white", colour="white"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank(),
    panel.grid.major = element_blank(),
```

```
    panel.grid.minor = element_blank(), panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title=element_text(family="xkcd-Regular"),
    text=element_text(family="xkcd-Regular"))
p3
```

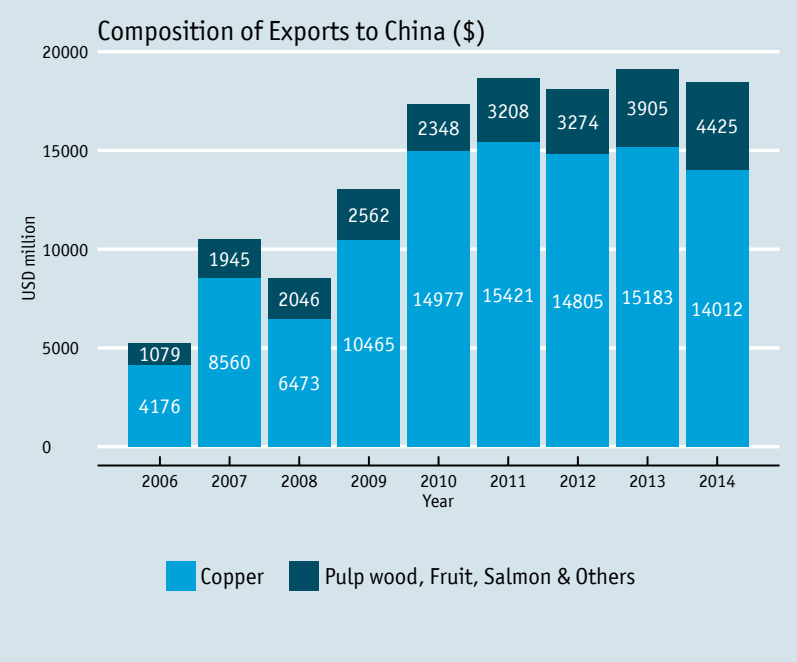


3.11. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

```
p3 <- ggplot() +
  geom_bar(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = export), colour="white",
    size = 4, family = "OfficinaSanITC-Book", show.legend = F) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    legend.position="bottom",
```

```
    legend.direction="horizontal",
    legend.title = element_blank(),
    plot.title=element_text(family="OfficinaSanITC-Book"),
    text=element_text(family="OfficinaSanITC-Book"))
p3
```

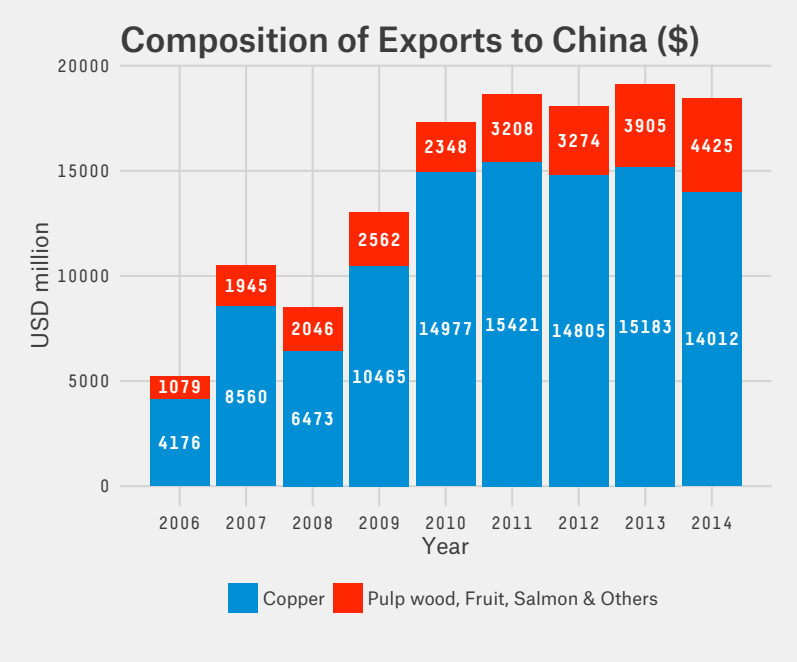


3.12. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

```
p3 <- ggplot() +
  geom_bar(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = export), colour="white",
    size = 3.5, family = "DecimaMonoPro-Bold", show.legend = F) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title=element_blank(),
```

```
    plot.title=element_text(family="Atlas Grotesk Medium"),
    legend.text=element_text(family="Atlas Grotesk Regular"),
    text=element_text(family="DecimaMonoPro"))
p3
```

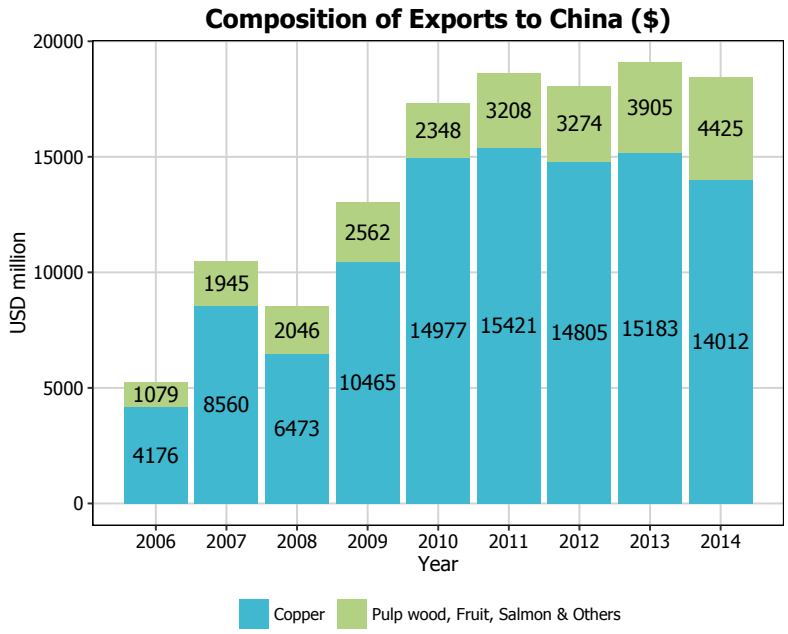


3.13. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here we present our original result shown at the beginning of the chapter.

```
fill <- c("#40b8d0", "#b2d183")

p3 <- ggplot() +
  geom_bar(aes(y = export, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = export), colour="black",
    family="Tahoma", size = 4, show.legend = F) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_fill_manual(values=fill) +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.key=element_rect(fill="white", colour="white"),
```

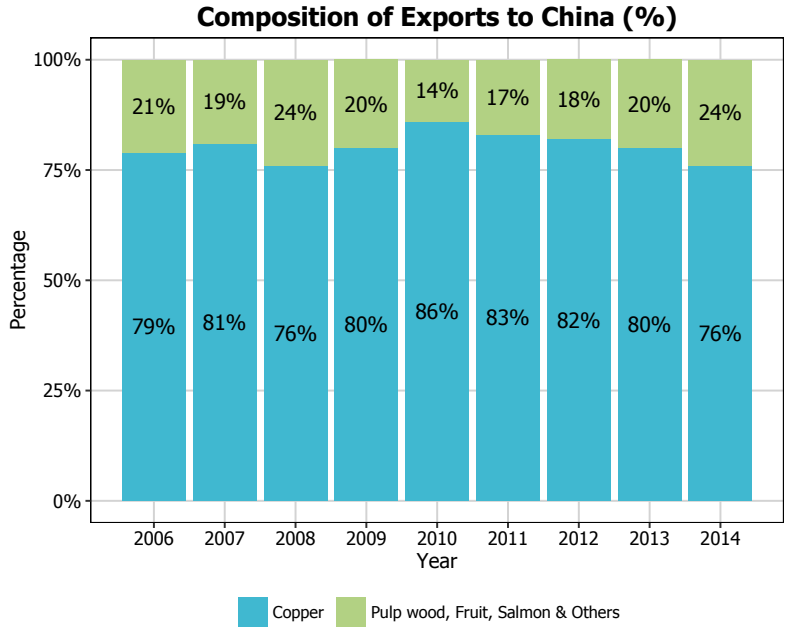
```
    legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p3
```



CHAPTER 4

Stacked bar plots

In this chapter, we will work towards creating the bar plot below. We will take you from a basic stacked bar plot and explain all the customisations we add to the code step-by-step.



4.1. Basic graph

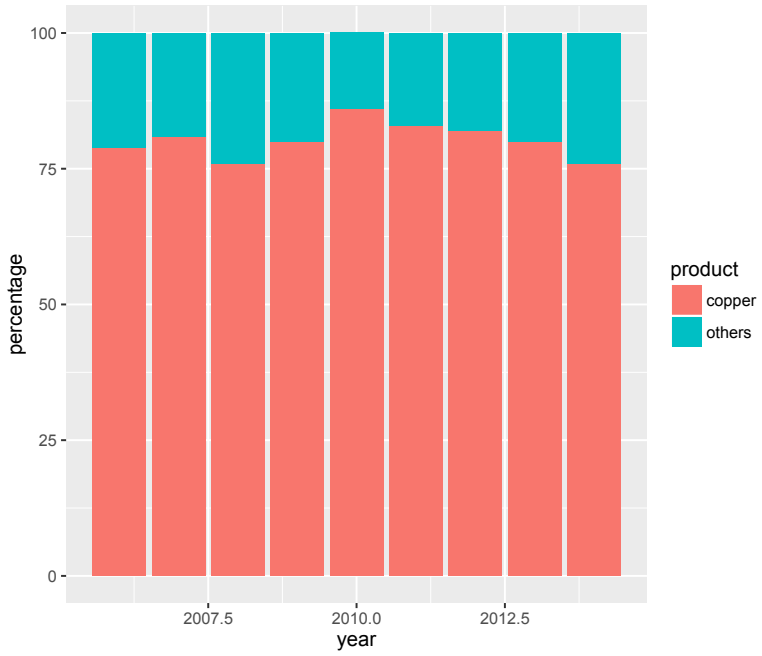
We will use the same data as in Chapter 1. The first thing to do is load in the data and libraries, as below:

```
library(ggplot2)
library(ggthemes)
library(extrafont)
library(plyr)
```

```
library(scales)

charts.data <- read.csv("copper-data-for-book.csv")
```

In order to initialise a plot we tell ggplot that `charts.data` is our data. We then instruct ggplot to render this as a bar plot by adding the `geom_bar` command. We can specify our x-axis, y-axis and grouping variables within `geom_bar` using the `x`, `y` and `fill` arguments respectively. Note that this has been rendered as a stacked bar graph because we have used the percentage variable, rather than raw counts.

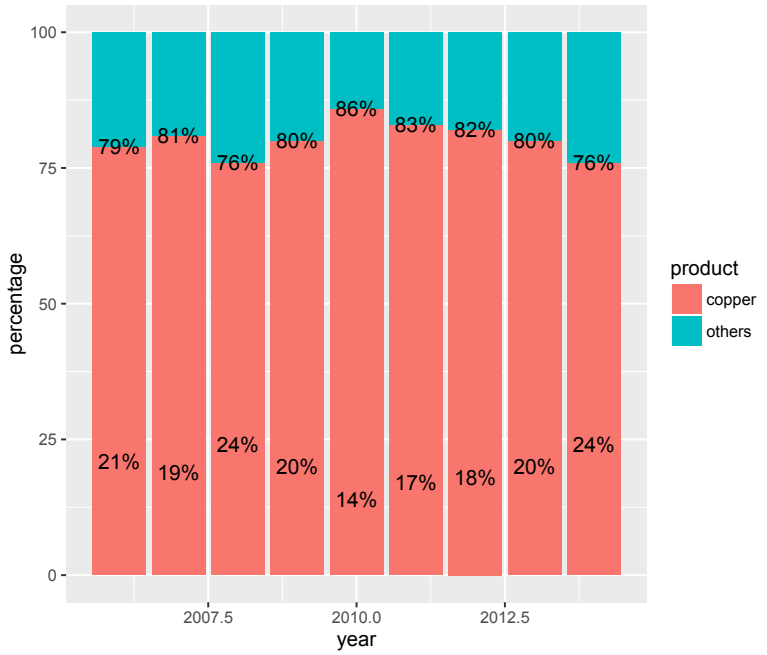
```
p4 <- ggplot() + geom_bar(aes(y = percentage, x = year, fill = product),
  data = charts.data, stat="identity")
p4
```



4.2. Adding data labels

To label the bars according to some variable in the data, we add the `label` argument to the `ggplot(aes())` option. In this case, we have labelled the bars with numbers from the `export` variable.

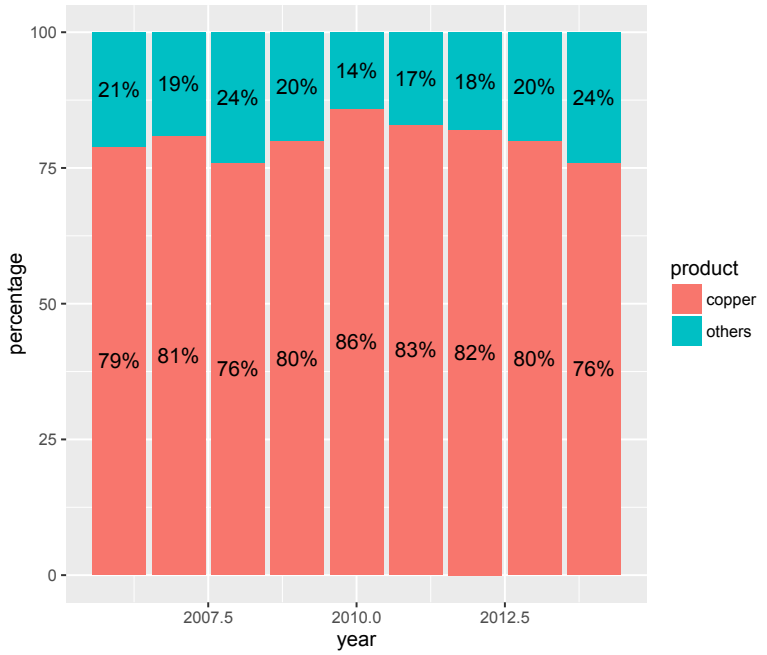
```
p4 <- p4 + geom_text(data=charts.data, aes(x = year, y = percentage,
  label = paste0(percentage,"%")), size=4)
p4
```



4.3. Adjusting data labels position

To adjust the position of the data labels from the default placement, we use the `ddply` function on the data, and add a new variable called `pos` to our data. This variable is at the centre of each bar and can be used to specify the position of the labels by assigning it to the `y` argument in `geom_text(aes())`.

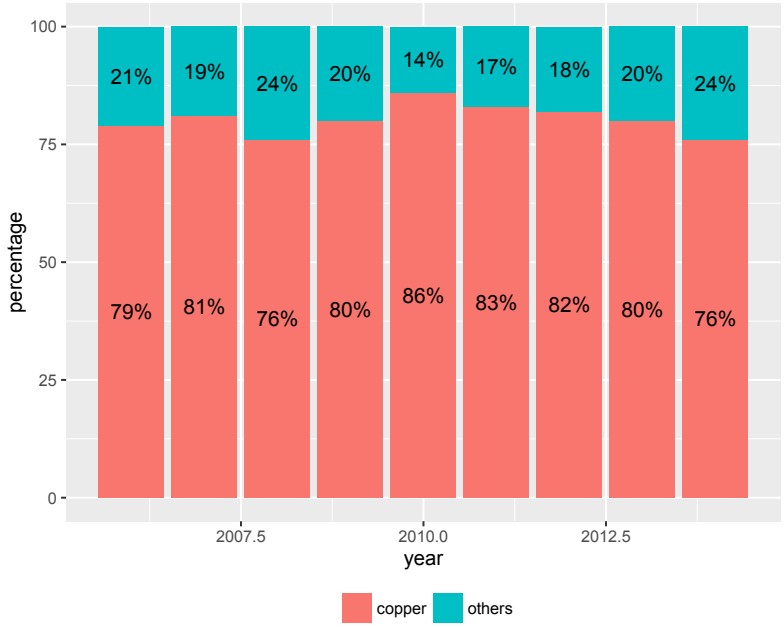
```
charts.data <- ddply(charts.data, .(year), transform,
    pos = cumsum(percentage) - (0.5 * percentage))

p4 <- ggplot() + geom_bar(aes(y = percentage, x = year, fill = product),
    data = charts.data, stat="identity")
p4 <- p4 + geom_text(data=charts.data, aes(x = year, y = pos, label =
  paste0(percentage,"%")), size=4)
p4
```



4.4. Adjusting legend position

To adjust the position of the legend from the default spot of right of the graph, we add the `theme` option and specify the `legend.position="bottom"` argument. We can also change the title to blank using the `legend.title = element_blank()` argument and change the legend shape using the `legend.direction="horizontal"` argument.

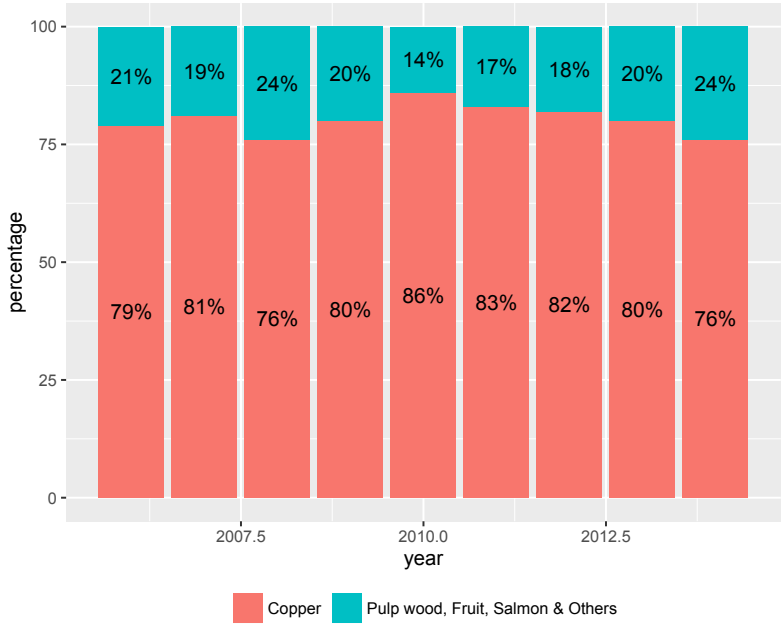
```
p4 <- p4 + theme(legend.position="bottom", legend.direction="horizontal",
  legend.title = element_blank())
p4
```



4.5. Changing legend labels

To change the name of the groups in the legend, we need to re-factor our data labels in `charts.data` data frame.

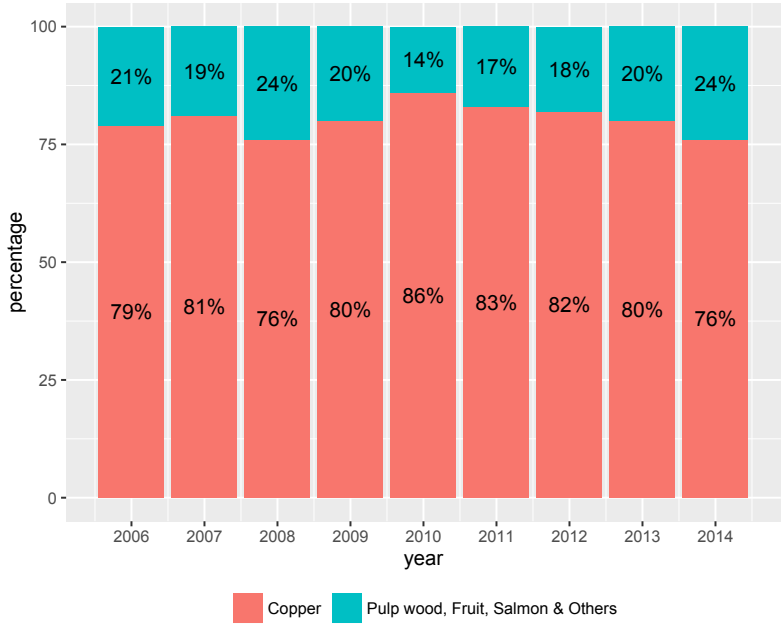
```
charts.data <- as.data.frame(charts.data)
charts.data$product <- factor(charts.data$product, levels = c("copper","others"),
  labels = c("Copper ","Pulp wood, Fruit, Salmon & Others"))

p4 <- ggplot() +
  geom_bar(aes(y = percentage, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = paste0(percentage,"%")),
    size=4) +
  theme(legend.position="bottom", legend.direction="horizontal", legend.title =
    element_blank())
p4
```



4.6. Adjusting x-axis scale

To change the axis tick marks, we use the `scale_x_continuous` and/or `scale_y_continuous` commands. Let's convert the x-axis into yearly increments using the `breaks` argument. We'll make this more efficient by using the `seq` function, a base R function that indicates the start and endpoints and the units to increment by respectively. See `help(seq)` for more information.

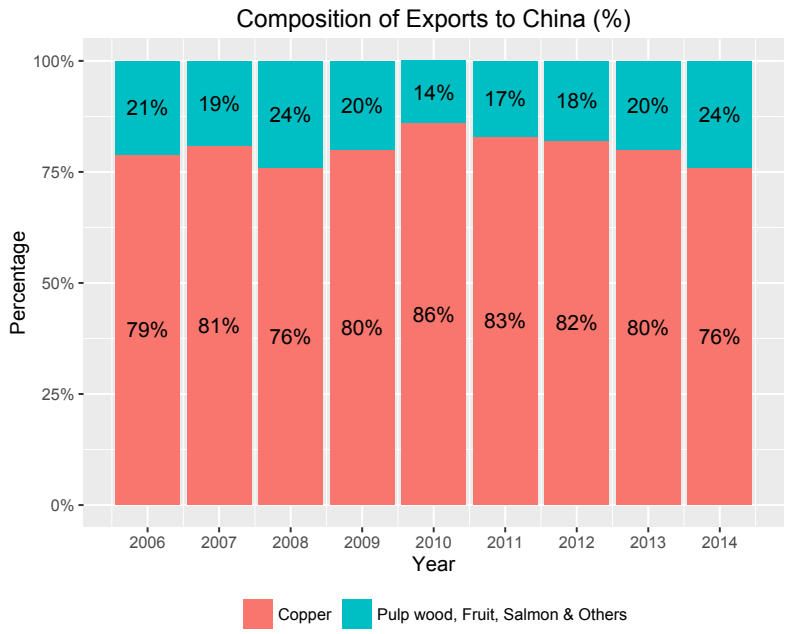
```
p4 <- p4 + scale_x_continuous(breaks=seq(2006,2014,1))
p4
```



4.7. Adjusting axis, title & units

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument, and to change the axis names we use the `labs` command. Note that we've rendered the labels on the y-axis as percentages by adding the `dollar_format` argument to `labels`.

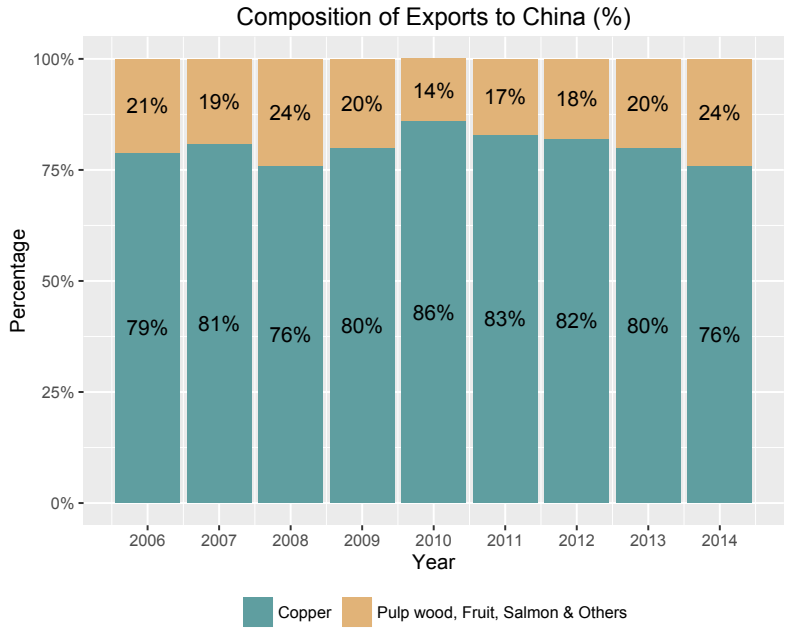
```
p4 <- p4 + labs(x="Year", y="Percentage") +
  scale_y_continuous(labels = dollar_format(suffix = "%", prefix = "")) +
  ggtitle("Composition of Exports to China (%)")
p4
```



4.8. Adjusting color palette

To change the colours, we use the `scale_colour_manual` command. Note that you can reference the specific colours you'd like to use with specific HEX codes. You can also reference colours by name, with the full list of colours recognised by R here.

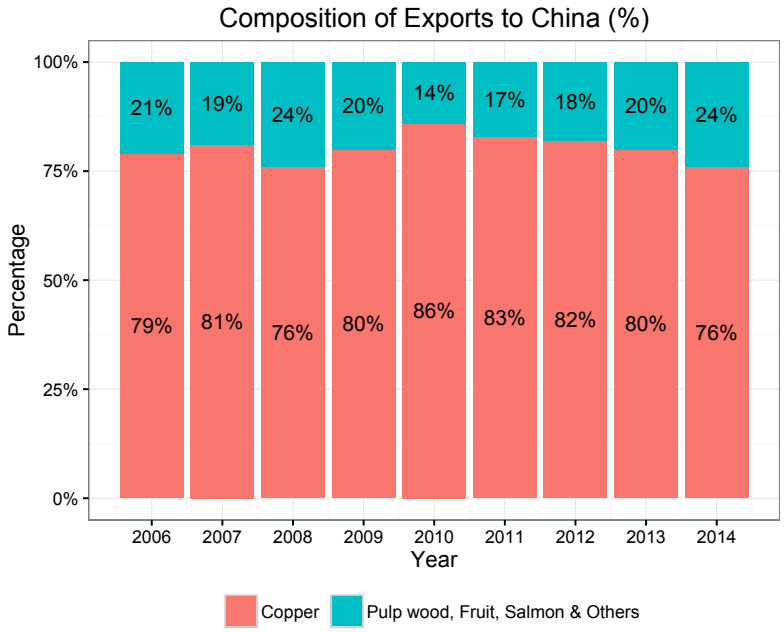
```
fill <- c("#5F9EA0", "#E1B378")
p4 <- p4 + scale_fill_manual(values=fill)
p4
```



4.9. Using the white theme

We can also change the overall look of the graph using themes. We'll start using a simple theme customisation by adding `theme_bw()` after `ggplot()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p4 <- ggplot() +
  geom_bar(aes(y = percentage, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = paste0(percentage,"%")),
    size=4) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  scale_y_continuous(labels = dollar_format(suffix = "%", prefix = "")) +
  labs(x="Year", y="Percentage") +
  ggtitle("Composition of Exports to China (%)") +
  theme_bw() +
  theme(legend.position="bottom",
    legend.direction="horizontal",
    legend.title = element_blank())
p4
```

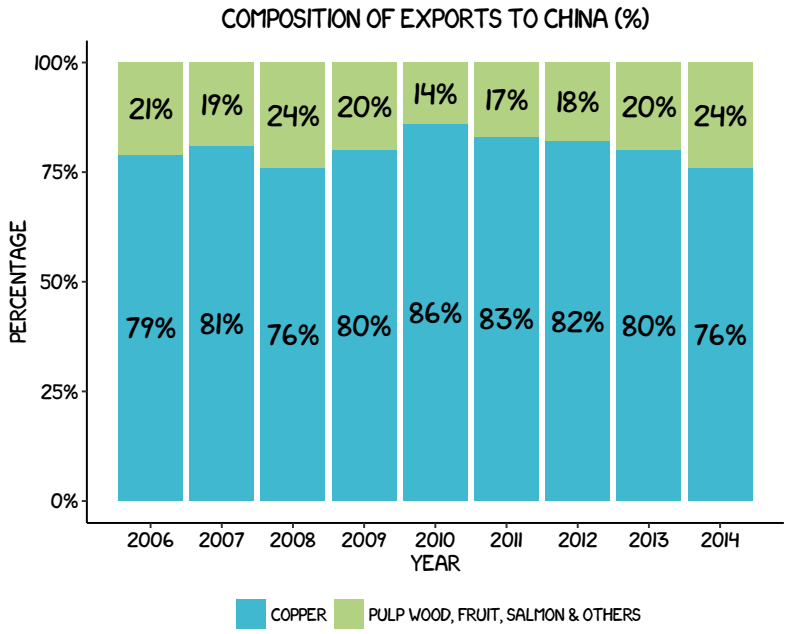


4.10. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
fill <- c("#40b8d0", "#b2d183")

p4 <- ggplot() +
  geom_bar(aes(y = percentage, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = paste0(percentage,"%")),
    colour="black", family="xkcd-Regular", size = 5, show.legend = F) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  scale_y_continuous(labels = dollar_format(suffix = "%", prefix = "")) +
  labs(x="Year", y="Percentage") +
  ggtitle("Composition of Exports to China (%)") +
  scale_fill_manual(values=fill) +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    axis.line.x = element_line(size=.5, colour = "black"),
    legend.key=element_rect(fill="white", colour="white"),
    legend.position="bottom", legend.direction="horizontal",
```

```
    legend.title = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(), panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title=element_text(family="xkcd-Regular"),
    text=element_text(family="xkcd-Regular"))
p4
```

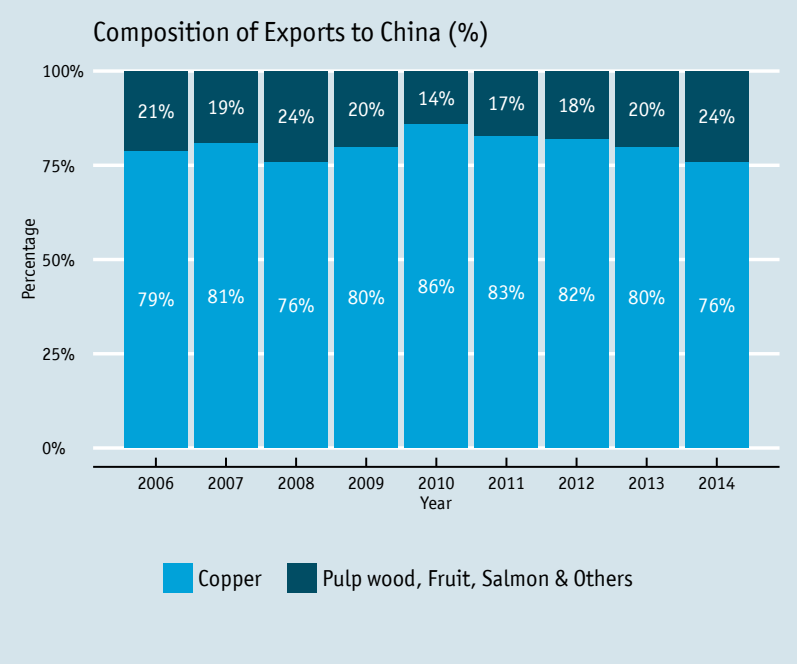


4.11. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these is here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

```
p4 <- ggplot() +
  geom_bar(aes(y = percentage, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = paste0(percentage,"%")),
    colour="white", family="OfficinaSanITC-Book", size=4) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  scale_y_continuous(labels = dollar_format(suffix = "%", prefix = "")) +
  labs(x="Year", y="Percentage") +
  ggtitle("Composition of Exports to China (%)") +
```

```
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.title = element_blank(),
    plot.title=element_text(family="OfficinaSanITC-Book"),
    text=element_text(family="OfficinaSanITC-Book"))
p4
```

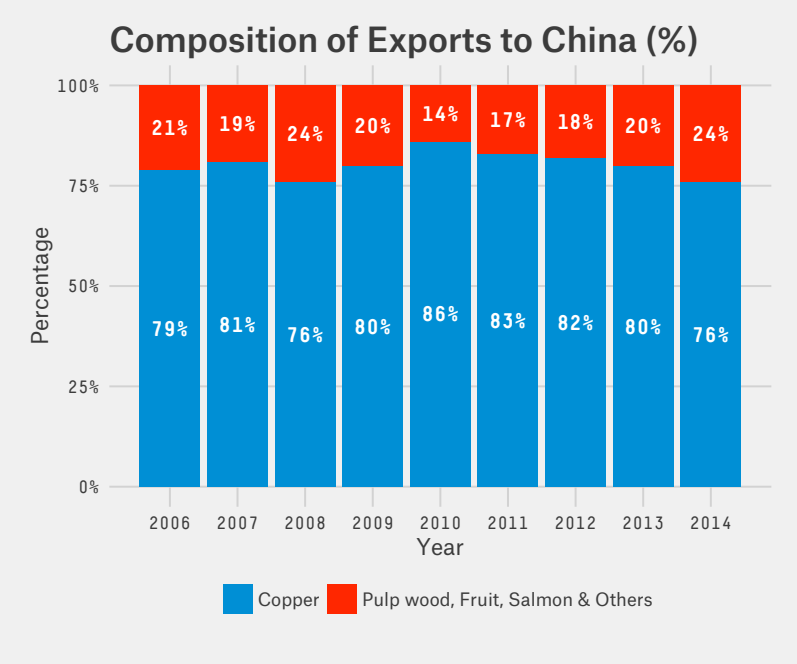


4.12. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

```
p4 <- ggplot() +
  geom_bar(aes(y = percentage, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = paste0(percentage,"%")),
    colour="white", family="DecimaMonoPro-Bold", size=4) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  scale_y_continuous(labels = dollar_format(suffix = "%", prefix = "")) +
  labs(x="Year", y="Percentage") +
  ggtitle("Composition of Exports to China (%)") +
```

```
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title=element_blank(),
    plot.title=element_text(family="Atlas Grotesk Medium"),
    legend.text=element_text(family="Atlas Grotesk Regular"),
    text=element_text(family="DecimaMonoPro"))
p4
```

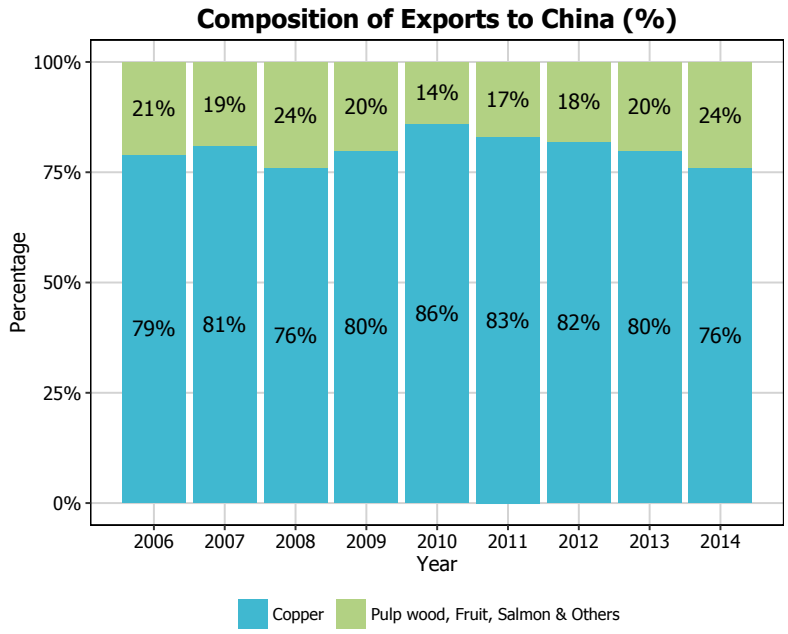


4.13. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here we present our original result shown at the beginning of the post.

```
fill <- c("#40b8d0", "#b2d183")

p4 <- ggplot() +
  geom_bar(aes(y = percentage, x = year, fill = product), data = charts.data,
    stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = paste0(percentage,"%")),
    colour="black", family="Tahoma", size=4) +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  scale_y_continuous(labels = dollar_format(suffix = "%", prefix = "")) +
  labs(x="Year", y="Percentage") +
  ggtitle("Composition of Exports to China (%)") +
```

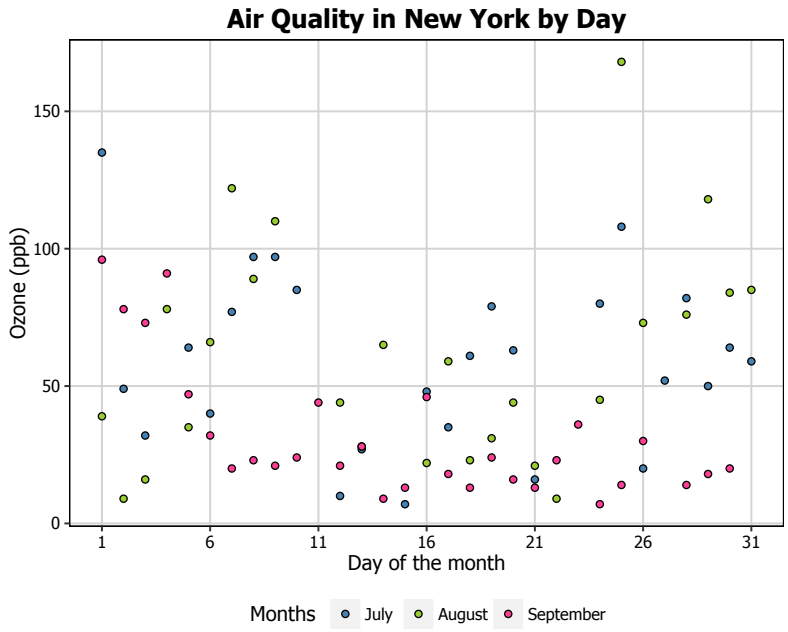
```
  scale_fill_manual(values=fill) +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.key=element_rect(fill="white", colour="white"),
    legend.position="bottom", legend.direction="horizontal",
    legend.title = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p4
```



CHAPTER 5

Scatterplots

In this chapter, we will work towards creating the scatterplot below. We will take you from a basic scatterplot and explain all the customisations we add to the code step-by-step.



In this chapter, we will be using R's airquality dataset, which is contained in the `datasets` package. The first thing to do is load in the data, as below:

```
library(ggplot2)
library(ggthemes)
library(extrafont)
library(datasets)
library(grid)
```

```
data(airquality)
```

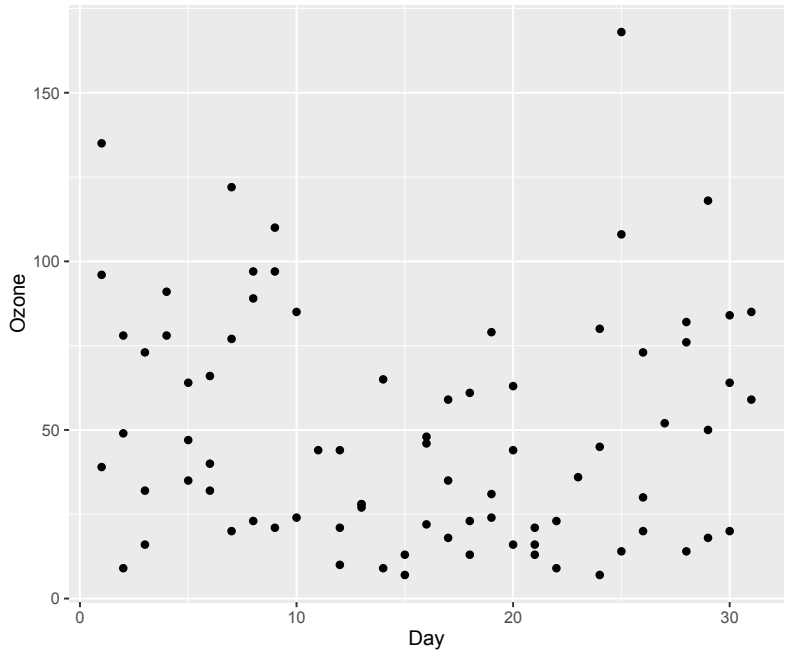
We will then trim the data down to the final three months and turn the `Month` variable into a labelled factor variable. We end up with a new dataset called `aq_trim`.

```
aq_trim <- airquality[which(airquality$Month == 7 |
airquality$Month == 8 |
airquality$Month == 9), ]
aq_trim$Month <- factor(aq_trim$Month,
labels = c("July", "August", "September"))
```

5.1. Basic scatterplot

In order to initialise a scatterplot we tell ggplot that `aq_trim` is our data, and specify that our x-axis plots the `Day` variable and our y-axis plots the `Ozone` variable. We then instruct ggplot to render this as a scatterplot by adding the `geom_point()` option.

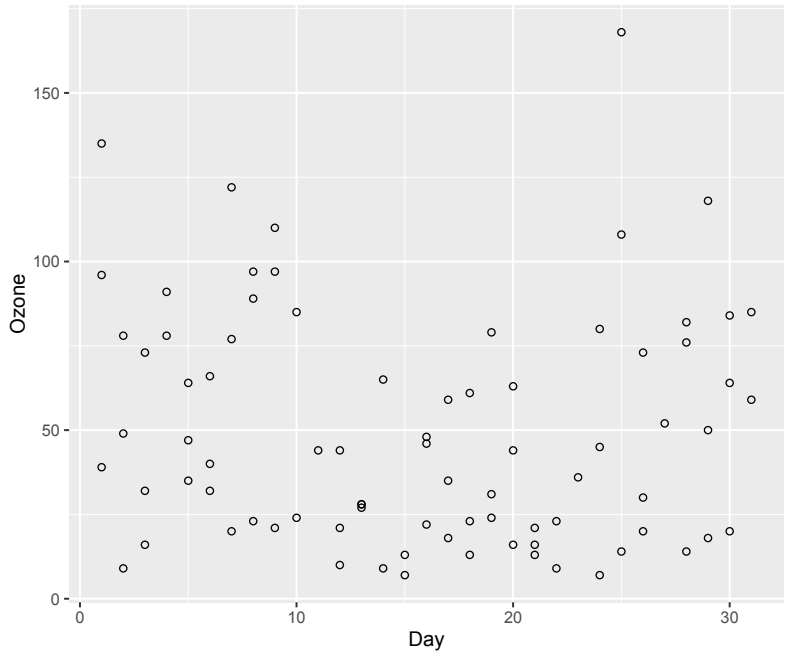
```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone)) +
  geom_point()
p5
```



5.2. Changing the shape of the data points

Perhaps we want the data points to be a different shape than a solid circle. We can change these by adding the `shape` argument to `geom_point`. An explanation of the allowed arguments for shape are described in this article. In this case, we will use shape 21, which is a circle that allows different colours for the outline and fill.

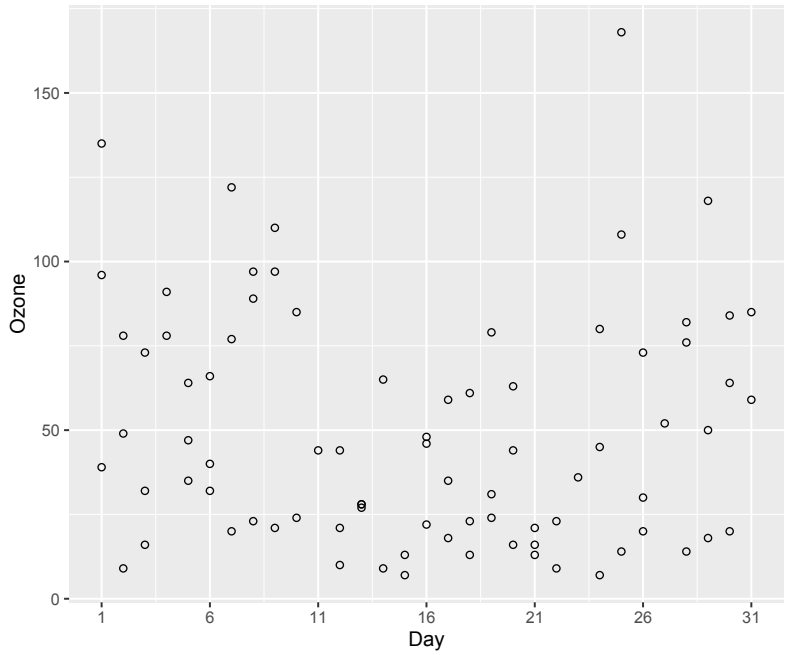
```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone)) + geom_point(shape = 21)
p5
```



5.3. Adjusting the axis scales

To change the x-axis tick marks, we use the `scale_x_continuous` option. Similarly, to change the y-axis we use the `scale_y_continuous` option. Here we will change the x-axis to every 5 days, rather than 10, and change the range from 1 to 31 (as 0 is not a valid value for this variable).

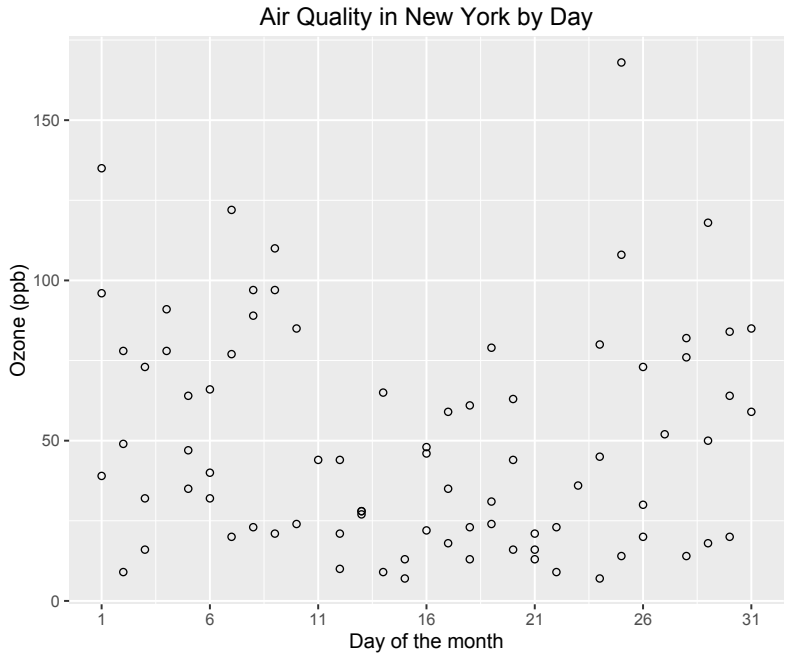
```
p5 <- p5 + scale_x_continuous(breaks = seq(1, 31, 5))
p5
```



5.4. Adjusting axis labels & adding title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument. To change the axis names we add `x` and `y` arguments to the `labs` command.

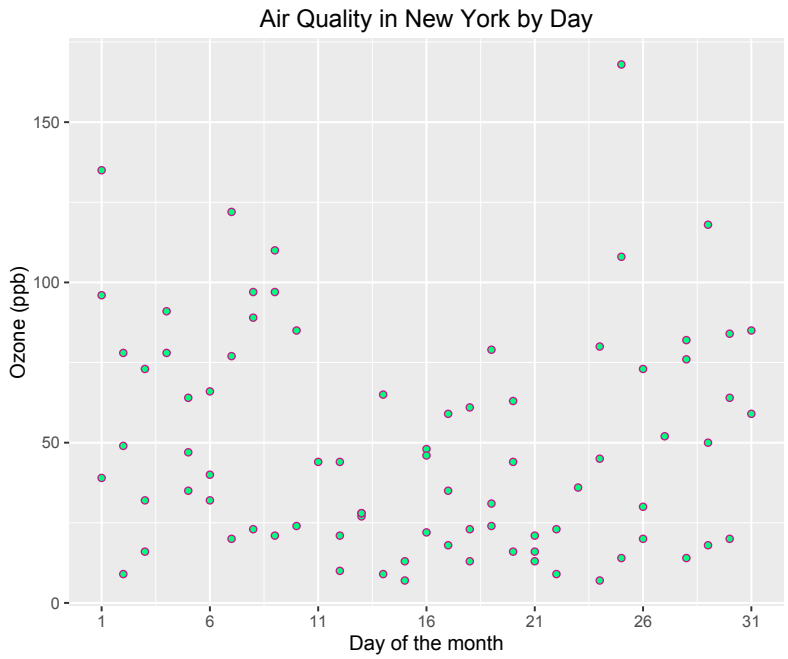
```
p5 <- p5 + ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)")
p5
```



5.5. Adjusting the colour palette

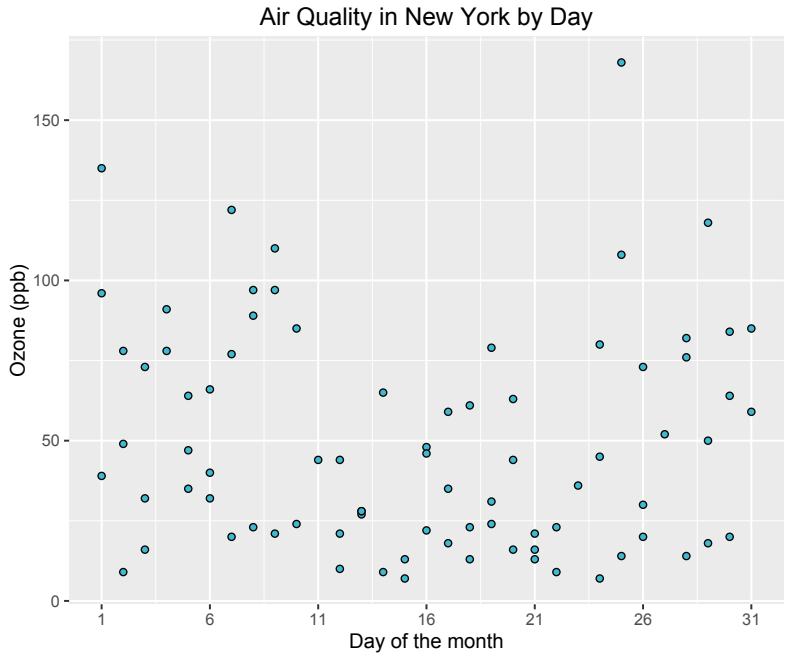
There are a few options for adjusting the colour. The most simple is to make every point one fixed colour. You can reference colours by name, with the full list of colours recognised by R here. Let's try making the outline `mediumvioletred` and the fill `springgreen`.

```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone)) +
  geom_point(shape = 21, colour = "mediumvioletred", fill = "springgreen") +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)") +
  scale_x_continuous(breaks = seq(1, 31, 5))
p5
```



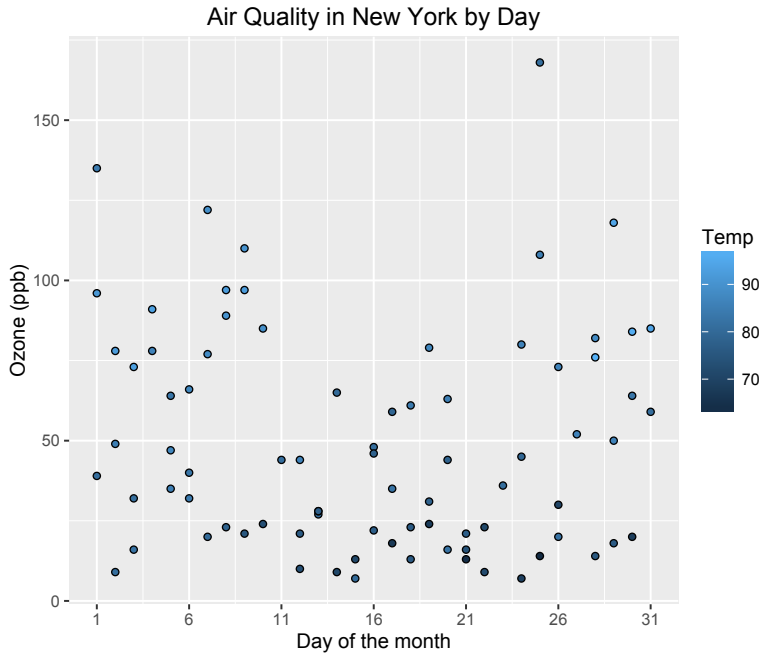
You can change the colours using specific HEX codes instead. Here we have made the outline #000000 (black) and the fill “#40b8d0 (vivid cyan).

```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone)) +
  geom_point(shape = 21, colour = "#000000", fill = "#40b8d0") +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)") +
  scale_x_continuous(breaks = seq(1, 31, 5))
p5
```



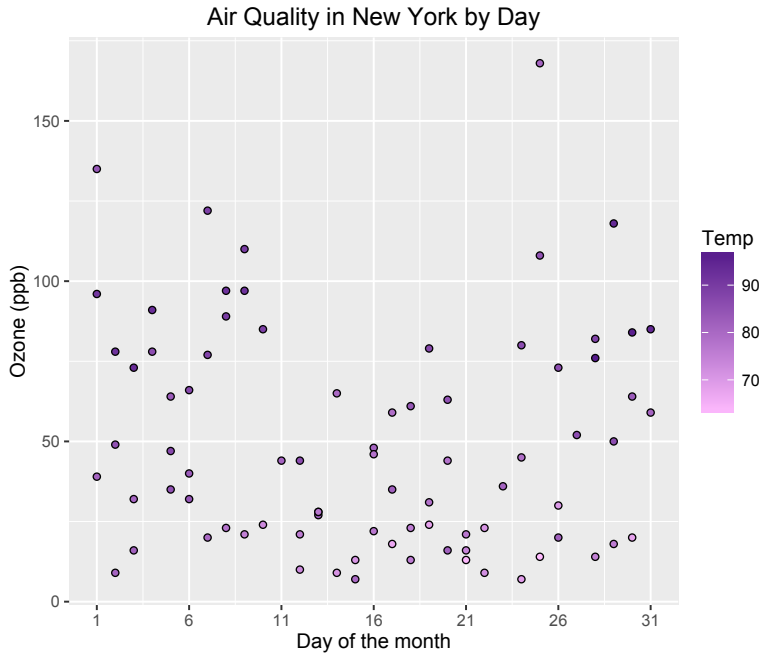
You can also change the colour of the data points according to the levels of another variable. This can be done either as a continuous gradient, or as a levels of a factor variable. Let's change the colour by the values of temperature:

```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone, fill = Temp)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)") +
  scale_x_continuous(breaks = seq(1, 31, 5))
p5
```



We can change the gradient's colours by adding the `scale_fill_continuous` option. The `low` and `high` arguments specify the range of colours the gradient should transition between.

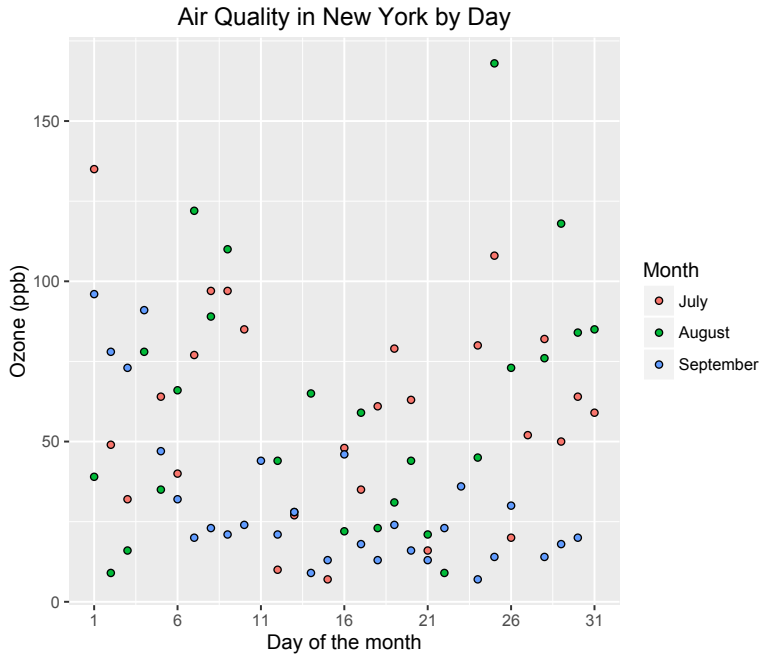
```
p5 <-  p5 + scale_fill_continuous(low = "plum1", high = "purple4")
p5
```



We can see that higher temperatures seem to have higher ozone levels.

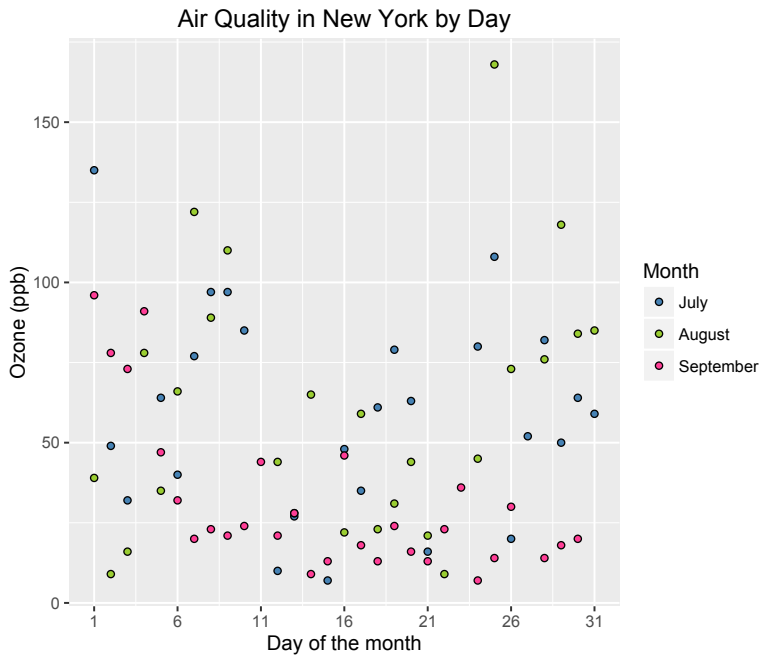
Let's now change the colours of the data points by a factor variable, `Month`.

```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)") +
  scale_x_continuous(breaks = seq(1, 31, 5))
p5
```



Again, we can change the colours of these data points, this time using `scale_fill_manual`.

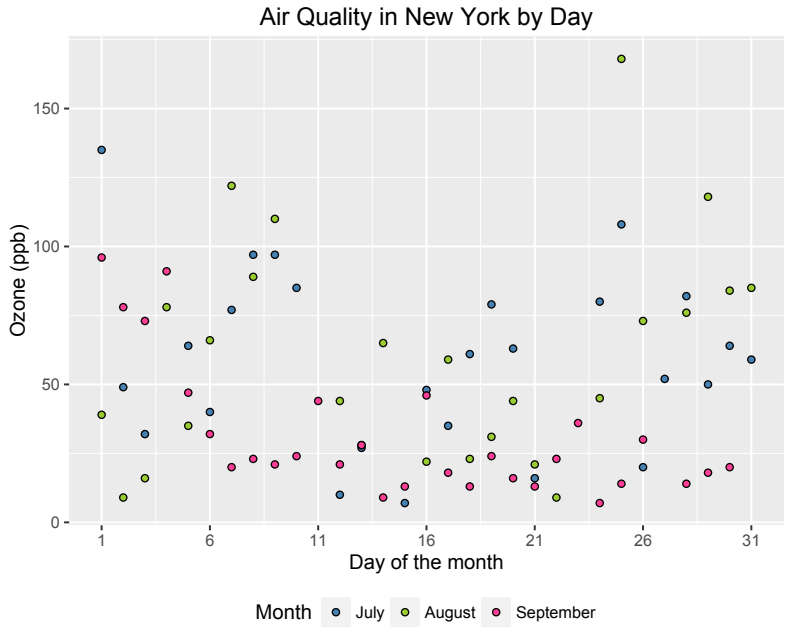
```
fill = c("steelblue", "yellowgreen", "violetred1")

p5 <- p5 + scale_fill_manual(values = fill)
p5
```



5.6. Adjusting legend position

To adjust the position of the legend from the default spot of right of the graph, we add the `theme` option and specify the `legend.position = "bottom"` argument. We can also change the legend shape using the `legend.direction = "horizontal"` argument.

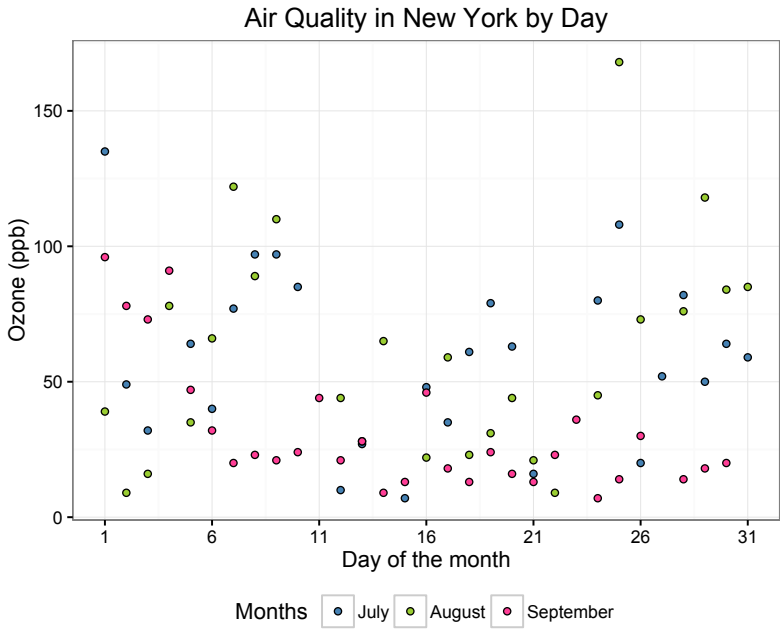
```
p5 <- p5 + theme(legend.position = "bottom", legend.direction = "horizontal")
p5
```



5.7. Using the white theme

As explained in the previous posts, we can also change the overall look of the plot using themes. We'll start using a simple theme customisation by adding `theme_bw()` after `ggplot()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)", fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_fill_manual(values = fill) +
  scale_size(range = c(1, 10)) +
  theme_bw() +
  theme(legend.position="bottom", legend.direction="horizontal")
p5
```

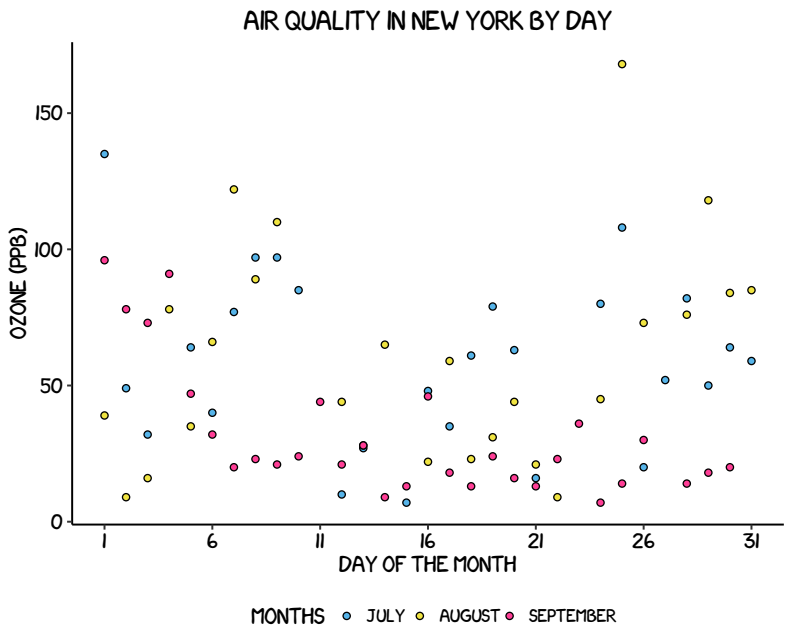


5.8. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
fill <- c("#56B4E9", "#F0E442", "violetred1")

p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)", fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_fill_manual(values = fill) +
  scale_size(range = c(1, 10)) +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.key = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
```

```
    panel.background = element_blank(),
    plot.title=element_text(family="xkcd-Regular"),
    text=element_text(family="xkcd-Regular"))
p5
```

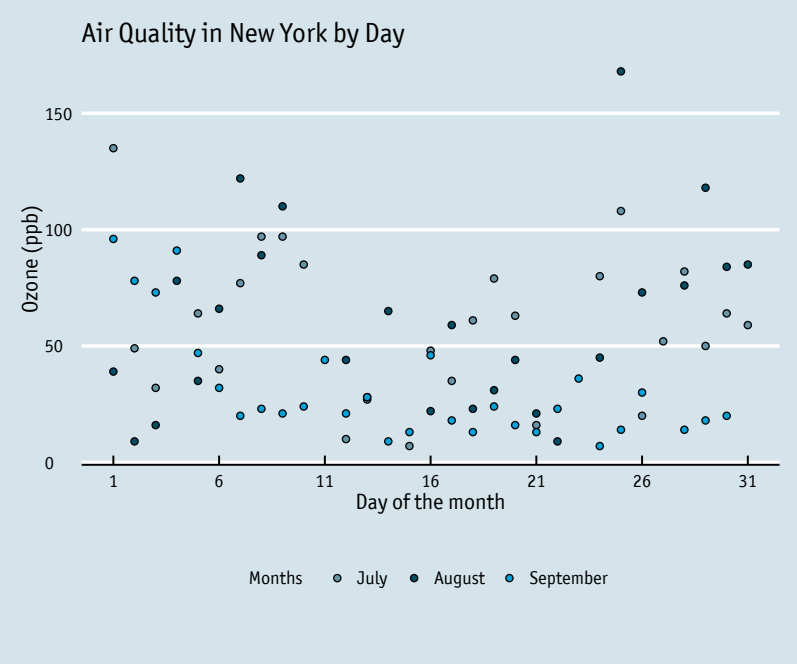


5.9. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these is here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)", fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_size(range = c(1, 10)) +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.title = element_text(size = 12),
    legend.position = "bottom", legend.direction = "horizontal",
    legend.text = element_text(size = 10),
    plot.title = element_text(family="OfficinaSanITC-Book"),
```

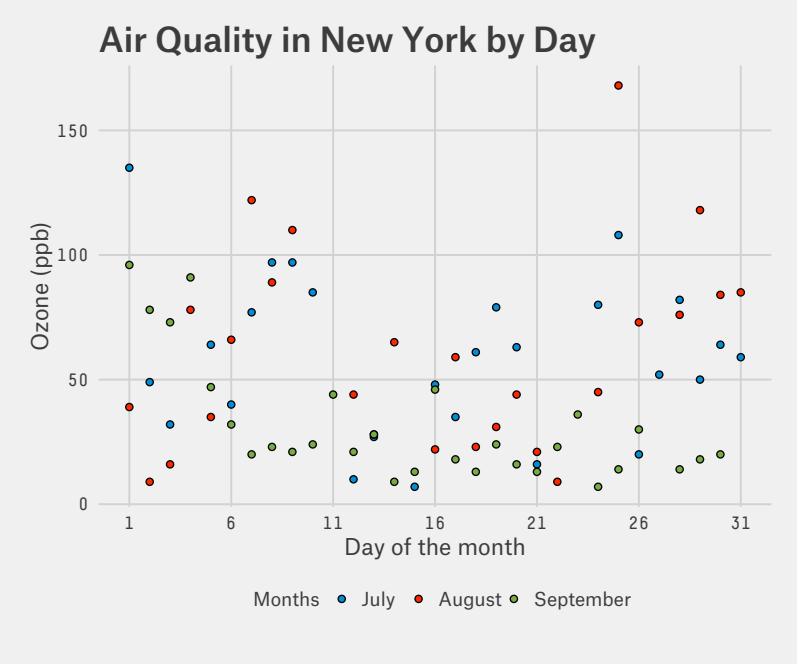
```
    text = element_text(family = "OfficinaSanITC-Book"))
p5
```



5.10. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

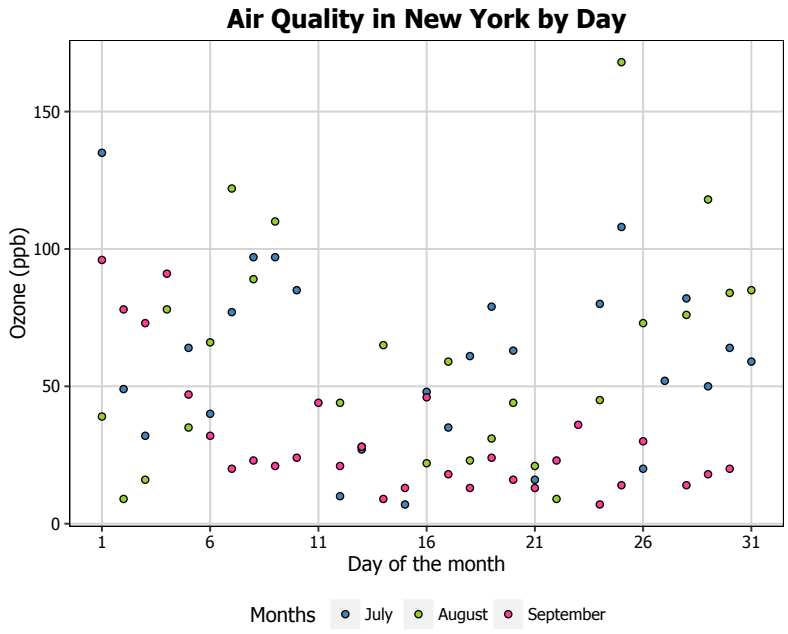
```
p4 <- ggplot(aq_trim, aes(x = Day, y = Ozone, fill = Month)) +
  scale_fill_economist() +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)", fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_size(range = c(1, 10)) +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.title=element_text(family="Atlas Grotesk Regular", size = 10),
    legend.text=element_text(family="Atlas Grotesk Regular", size = 10),
    plot.title=element_text(family="Atlas Grotesk Medium"),
    text=element_text(family="DecimaMonoPro"))
p4
```



5.11. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here we present our original result shown at the beginning of the chapter.

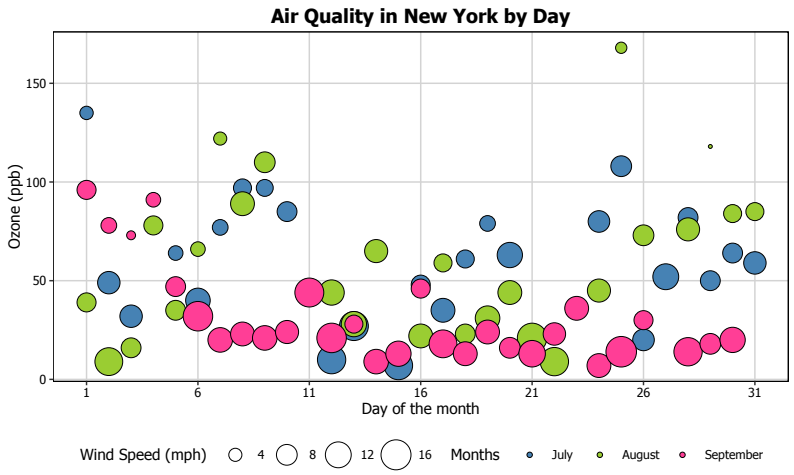
```
fill = c("steelblue", "yellowgreen", "violetred1")

p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)", fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_size(range = c(1, 10)) +
  scale_fill_manual(values = fill) +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    legend.position = "bottom", legend.direction = "horizontal",
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p5
```


Air Quality in New York by Day
Ozone (ppb)
150
100
50
0
1
6
11
16
21
26
31
Day of the month
Months
July
August
September

CHAPTER 6

Weighted scatterplots

In this chapter, we will work towards creating the weighted scatterplot below. We will take you from a basic scatterplot and explain all the customisations we add to the code step-by-step.



In this chapter, we will be using R's airquality dataset, which is contained in the `datasets` package. The first thing to do is load in the data, as below:

```
library(ggplot2)
library(ggthemes)
library(extrafont)
library(plyr)
library(scales)

data(airquality)
```

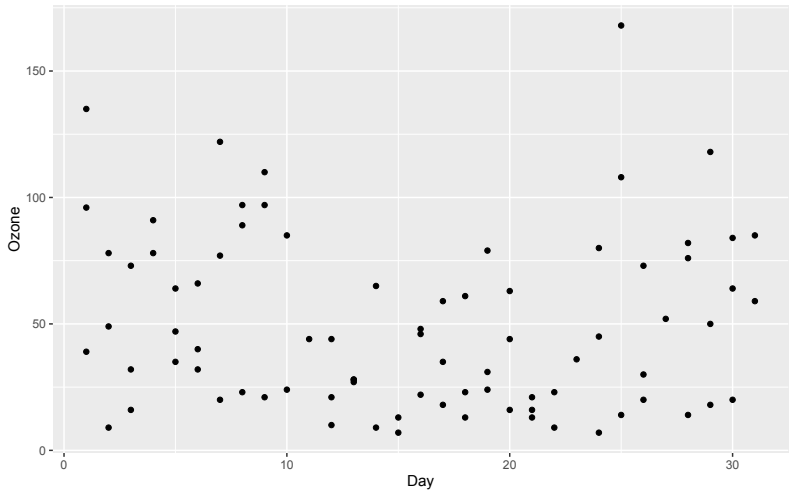
We will then trim the data down to the final three months and turn the `Month` variable into a labelled factor variable. We end up with a new dataset called `aq_trim`.

```
aq_trim <- airquality[which(airquality$Month == 7 |
airquality$Month == 8 |
airquality$Month == 9), ]
aq_trim$Month <- factor(aq_trim$Month,
labels = c("July", "August", "September"))
```

6.1. Basic weighted scatterplot

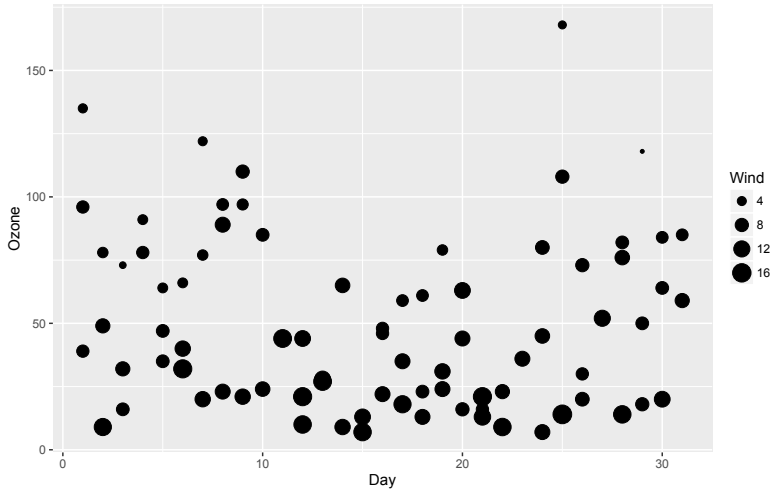
Let's start really slowly by revisiting how to create a basic scatterplot. In order to initialise this plot we tell ggplot that `aq_trim` is our data, and specify that our x-axis plots the `Day` variable and our y-axis plots the `Ozone` variable. We then instruct ggplot to render this as a scatterplot by adding the `geom_point()` option.

```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone)) +
      geom_point()
p6
```



In order to turn this into a weighted scatterplot, we simply add the `size` argument to `ggplot(aes())`. In this case, we want to weight the points by the `Wind` variable.

```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind)) +
  geom_point()
p6
```

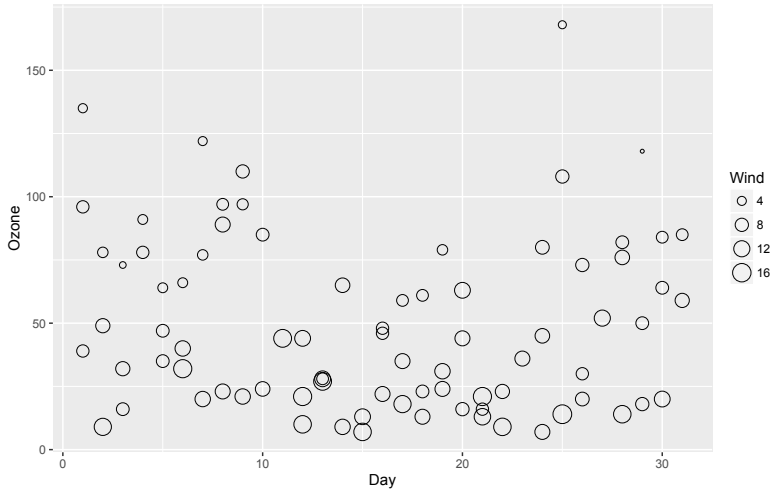


You can see we already have an interesting looking pattern, where days with higher wind speed tend to have lower ozone (or in other words, better air quality). Now let's make it beautiful!

6.2. Changing the shape of the data points

Perhaps we want the data points to be a different shape than a solid circle. We can change these by adding the `shape` argument to `geom_point`. An explanation of the allowed arguments for shape are described in this article. In this case, we will use shape 21, which is a circle that allows different colours for the outline and fill.

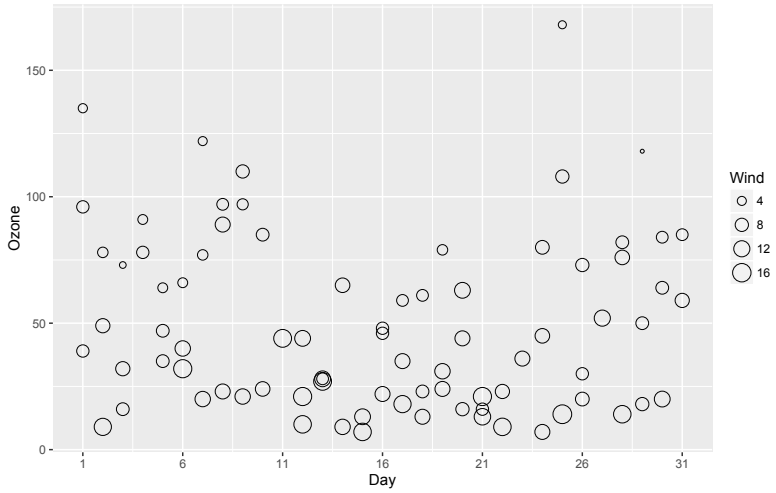
```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind)) +
  geom_point(shape = 21)
p6
```



6.3. Adjusting the axis scales

To change the x-axis tick marks, we use the `scale_x_continuous` option. Similarly, to change the y-axis we use the `scale_y_continuous` option. Here we will change the x-axis to every 5 days, rather than 10, and change the range from 1 to 31 (as 0 is not a valid value for this variable).

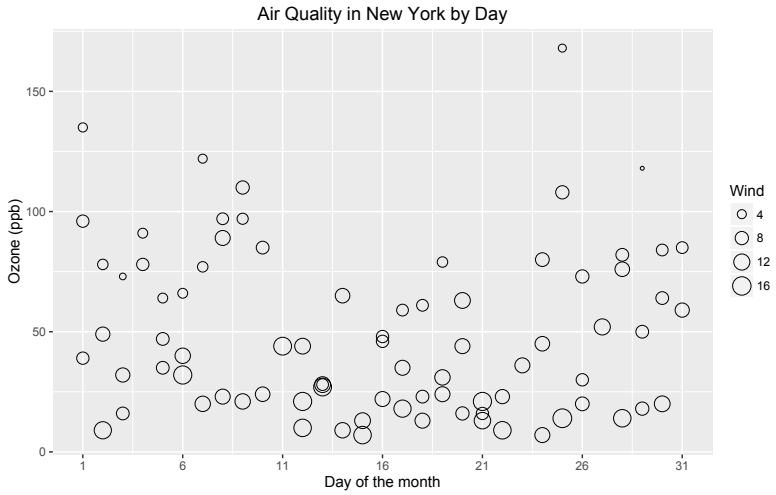
```
p6 <- p6 + scale_x_continuous(breaks = seq(1, 31, 5))
p6
```



6.4. Adjusting axis labels & adding title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument. To change the axis names we add `x` and `y` arguments to the `labs` command.

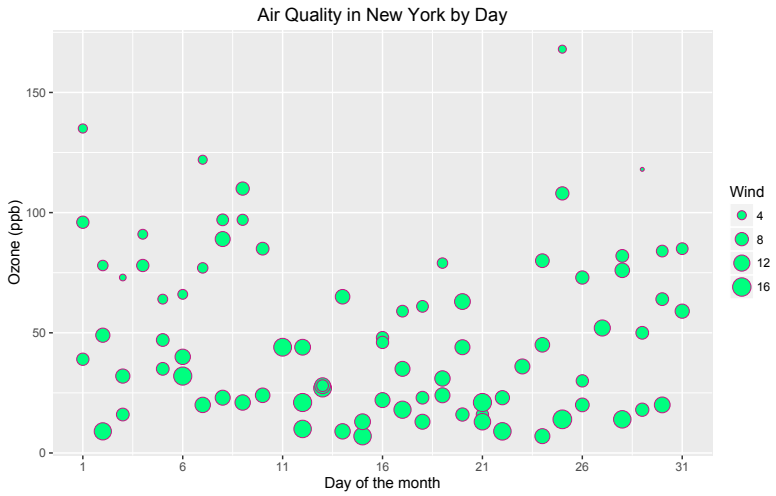
```
p6 <- p6 + ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)")
p6
```



6.5. Adjusting the colour palette

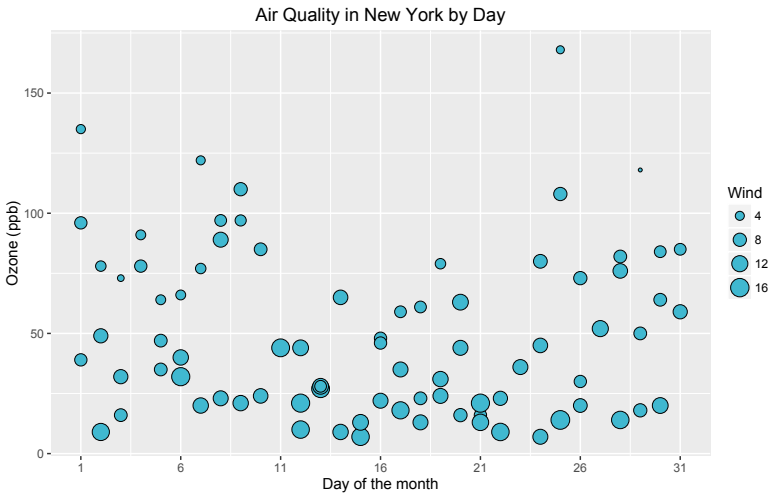
There are a few options for adjusting the colour. The most simple is to make every point one fixed colour. You can reference colours by name, with the full list of colours recognised by R here. Let's try making the outline `mediumvioletred` and the fill `springgreen`.

```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind)) +
  geom_point(shape = 21, colour = "mediumvioletred", fill = "springgreen") +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)") +
  scale_x_continuous(breaks = seq(1, 31, 5))
p6
```



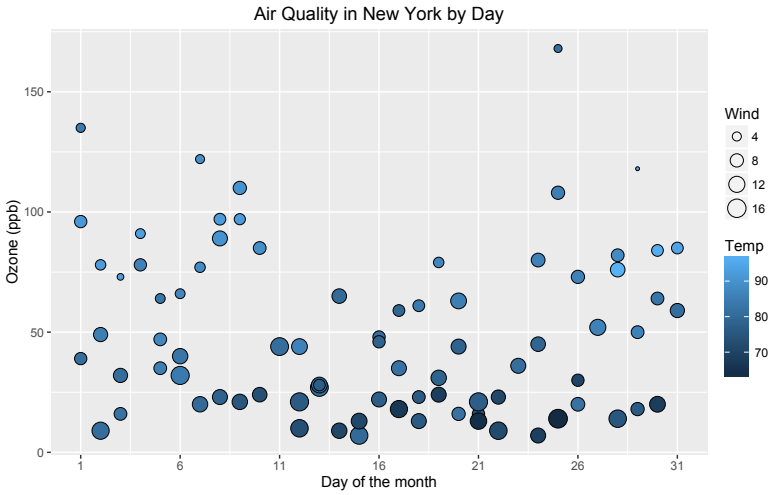
You can change the colours using specific HEX codes instead. Here we have made the outline #000000 (black) and the fill "#40b8d0 (vivid cyan).

```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind)) +
  geom_point(shape = 21, colour = "#000000", fill = "#40b8d0") +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)") +
  scale_x_continuous(breaks = seq(1, 31, 5))
p6
```



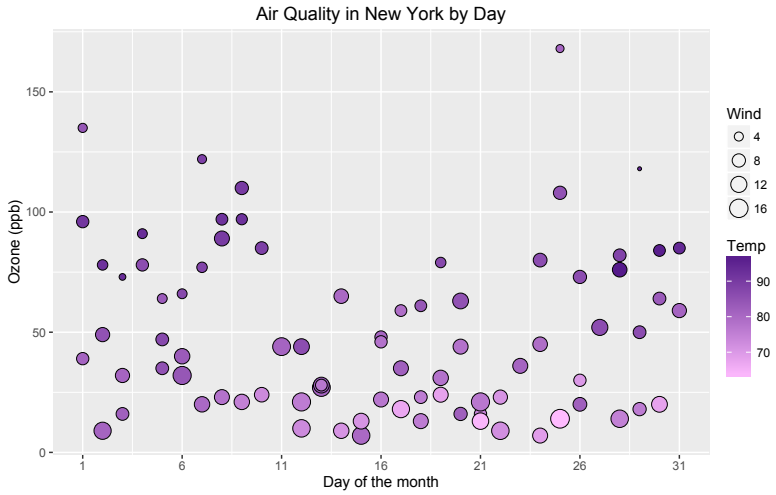
You can also change the colour of the data points according to the levels of another variable. This can be done either as a continuous gradient, or as a levels of a factor variable. Let's change the colour by the values of temperature:

```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Temp)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)") +
  scale_x_continuous(breaks = seq(1, 31, 5))
p6
```



We can change the gradient's colours by adding the `scale_fill_continuous` option. The `low` and `high` arguments specify the range of colours the gradient should transition between.

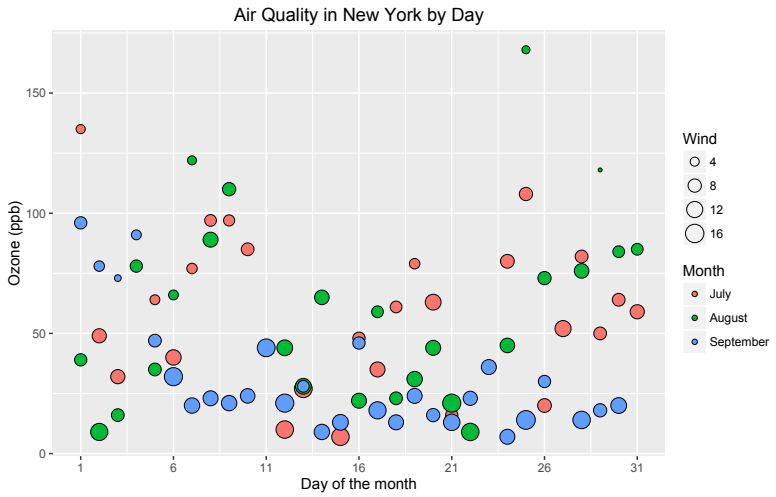
```
p6 <-p6 + scale_fill_continuous(low = "plum1", high = "purple4")
p6
```



We can see that higher temperatures seem to have higher ozone levels.

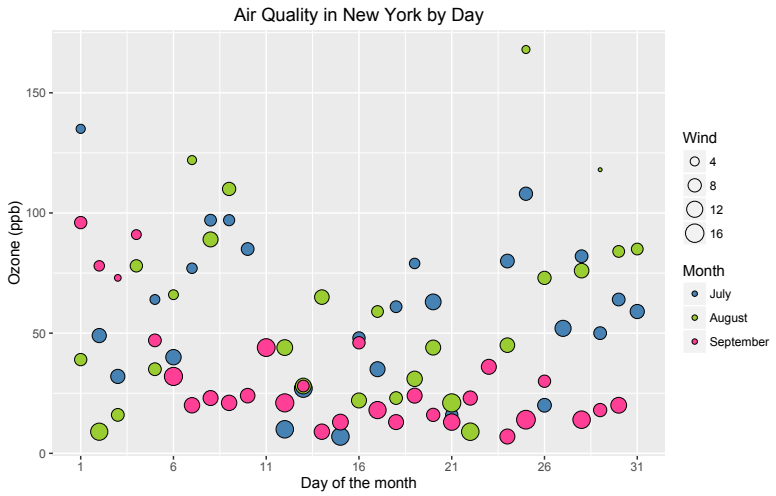
Let's now change the colours of the data points by a factor variable, `Month`.

```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)") +
  scale_x_continuous(breaks = seq(1, 31, 5))
p6
```



Again, we can change the colours of these data points, this time using `scale_fill_manual`.

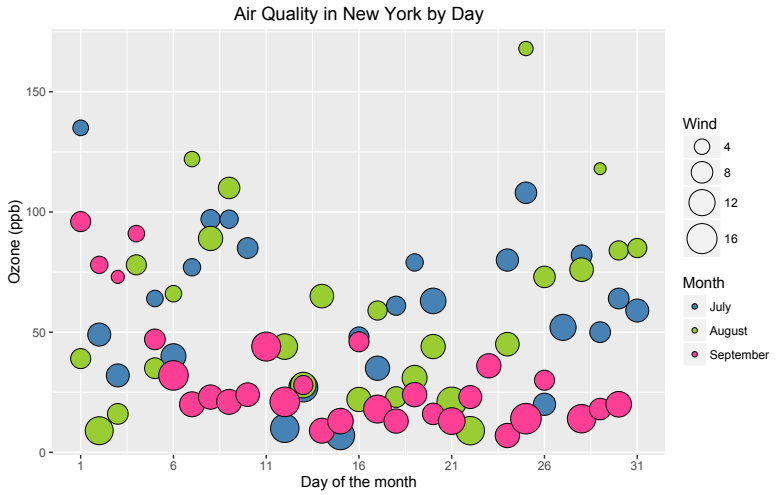
```
fill = c("steelblue", "yellowgreen", "violetred1")

p6 <- p6 + scale_fill_manual(values = fill)
p6
```



6.6. Adjusting the size of the data points

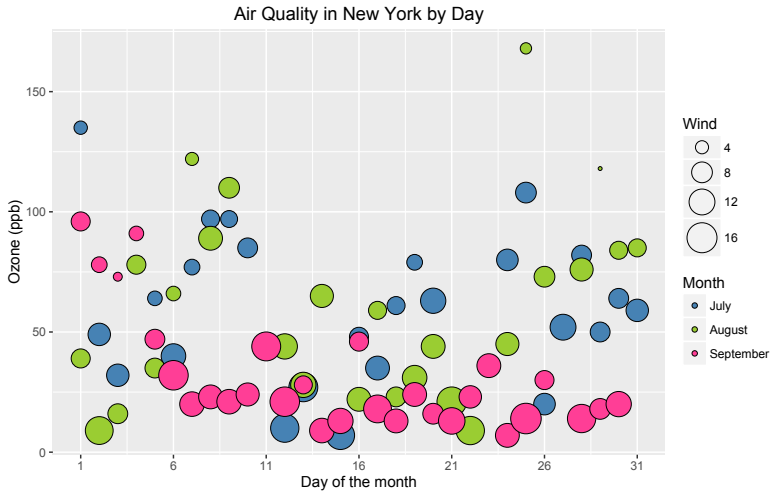
The default size of the the data points in a weighted scatterplot is mapped to the radius of the plots. If we want the data points to be proportional to the value of the weighting variable (e.g., a wind speed of 0 mph would have a value of 0), we need to use the `scale_size_area`.

```
p6 <- p6 + scale_size_area(max_size = 10)
p6
```



For our graph, this makes the pattern for `Wind` a little hard to see. Another way to adjust the size of the data points is to use `scale_size` and specify a desired range.

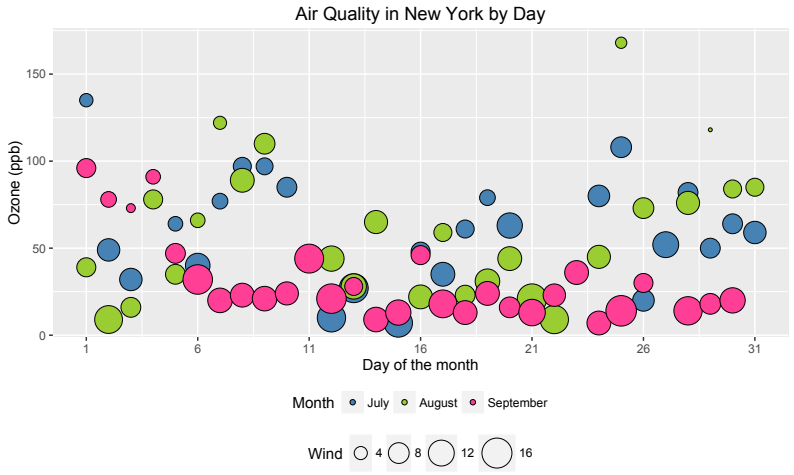
```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_fill_manual(values = fill) +
  scale_size(range = c(1, 10))
p6
```



6.7. Adjusting legend position

To adjust the position of the legend from the default spot of right of the graph, we add the `theme` option and specify the `legend.position = "bottom"` argument. We can also change the legend shape using the `legend.direction = "horizontal"` argument.

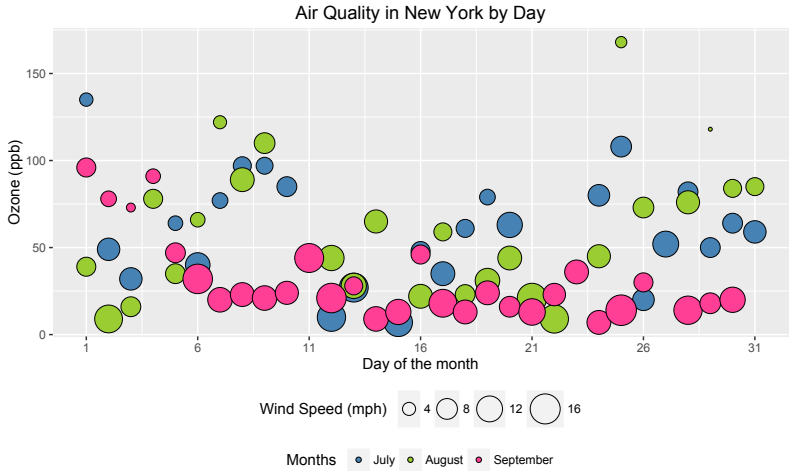
```
p6 <- p6 + theme(legend.position = "bottom", legend.direction = "horizontal")
p6
```



6.8. Changing the legend titles

To change the titles of the two legends, we use the `labs` option. In order to tell ggplot2 exactly what legend you're referring to, just have a look in the `ggplot` option and see what argument you used to create the legend in the first place. In this case we used the `size` argument for "Wind" and `fill` for "Month", so we pass these to `labs` with our new titles.

```
p6 <- p6 + labs(size = "Wind Speed (mph) ", fill = "Months ")
p6
```

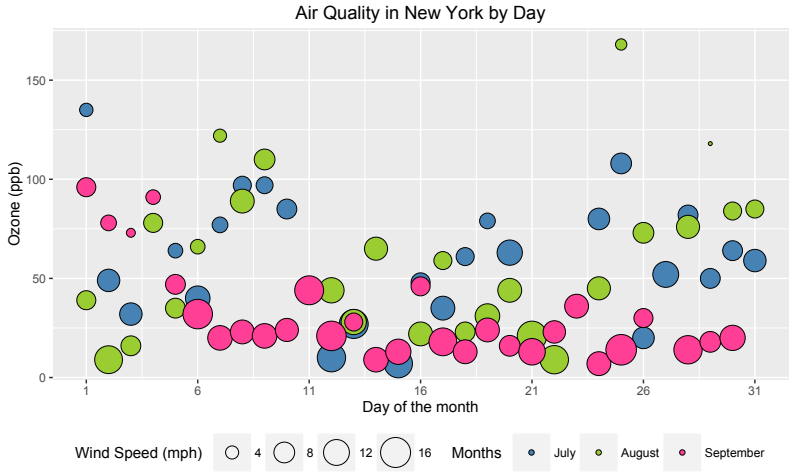


6.9. Creating horizontal legends

It looks a little awkward having the two titles sitting on top of each other, as well as taking up unnecessary space. To place the legends next to each other, we use the `legend.box = "horizontal"` argument in

`theme`. Because the boxes around the legend keys aren't even in each of the legends, this means the legends don't align properly. To fix this, we change the box size around the legend keys using `legend.key.size`. We need to load in the `grid` package to get this argument to work.

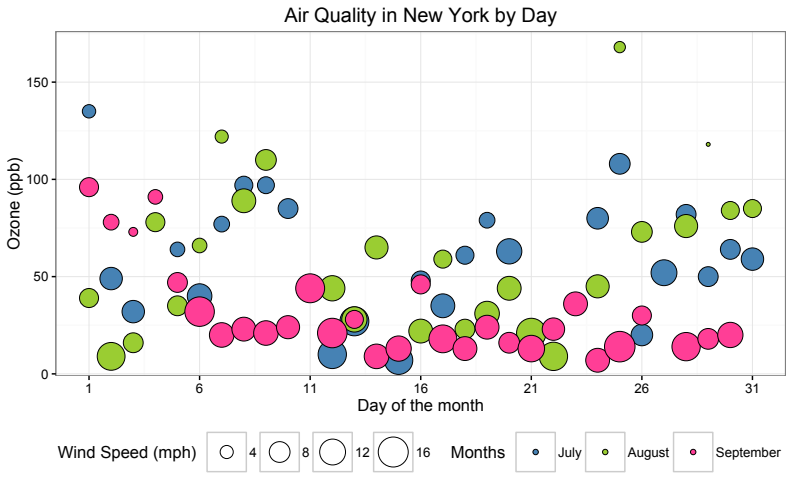
```
p6 <- p6 + theme(legend.box = "horizontal", legend.key.size = unit(1, "cm"))
p6
```



6.10. Using the white theme

We can also change the overall look of the plot using themes. We'll start using a simple theme customisation by adding `theme_bw()` after `ggplot()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)",
    size = "Wind Speed (mph) ", fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_fill_manual(values = fill) +
  scale_size(range = c(1, 10)) +
  theme_bw() +
  theme(legend.position="bottom", legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key.size = unit(1, "cm"))
p6
```

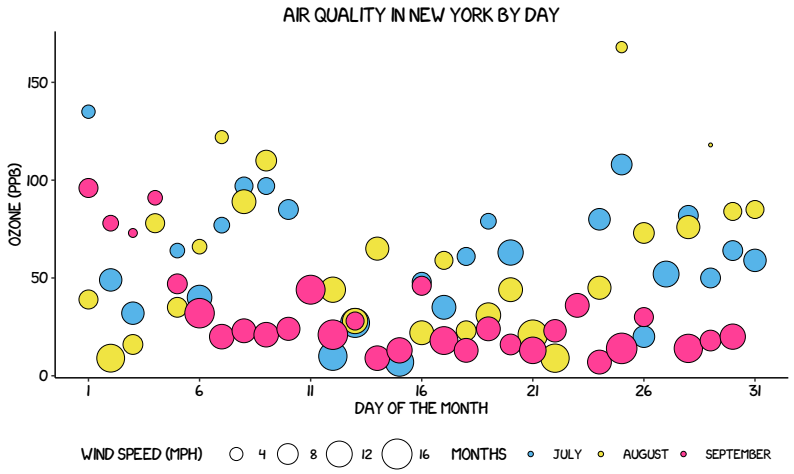


6.11. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
fill <- c("#56B4E9", "#F0E442", "violetred1")

p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)",
    size = "Wind Speed (mph) ", fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_fill_manual(values = fill) +
  scale_size(range = c(1, 10)) +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key.size = unit(1, "cm"),
    legend.key = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title=element_text(family="xkcd-Regular"),
```

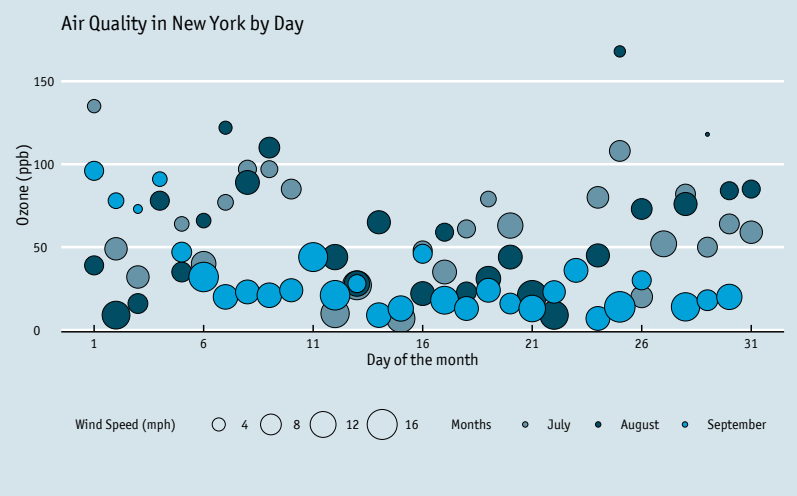
```
    text=element_text(family="xkcd-Regular"))
p6
```



6.12. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

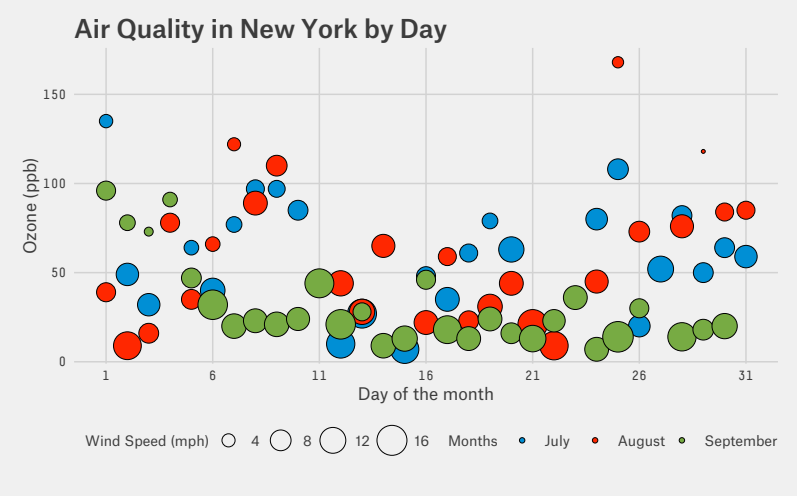
```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)", size = "Wind Speed (mph) ",
    fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_size(range = c(1, 10)) +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.title = element_text(size = 12),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key.size = unit(1, "cm"),
    legend.text = element_text(size = 10),
    text = element_text(family = "OfficinaSanITC-Book"),
    plot.title = element_text(family="OfficinaSanITC-Book"))
p6
```



6.13. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

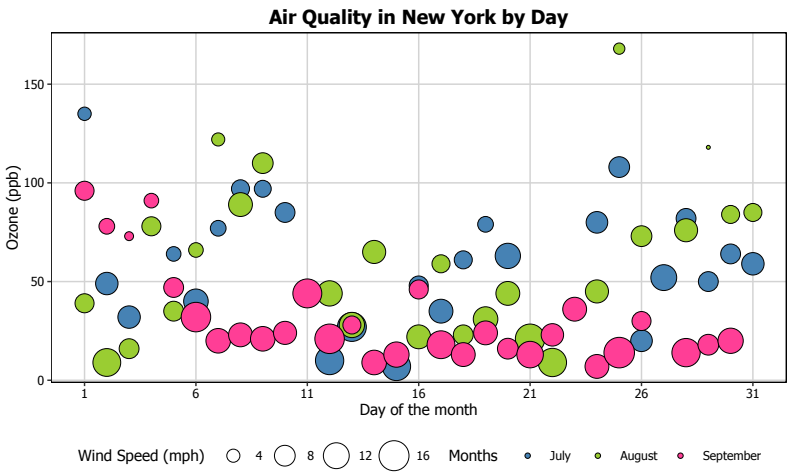
```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)", size = "Wind Speed (mph) ",
    fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_size(range = c(1, 10)) +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key.size = unit(1, "cm"),
    legend.title=element_text(family="Atlas Grotesk Regular", size = 10),
    legend.text=element_text(family="Atlas Grotesk Regular", size = 10),
    plot.title=element_text(family="Atlas Grotesk Medium"),
    text=element_text(family="DecimaMonoPro"))
p6
```



6.14. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here we present our original result shown at the beginning of the chapter.

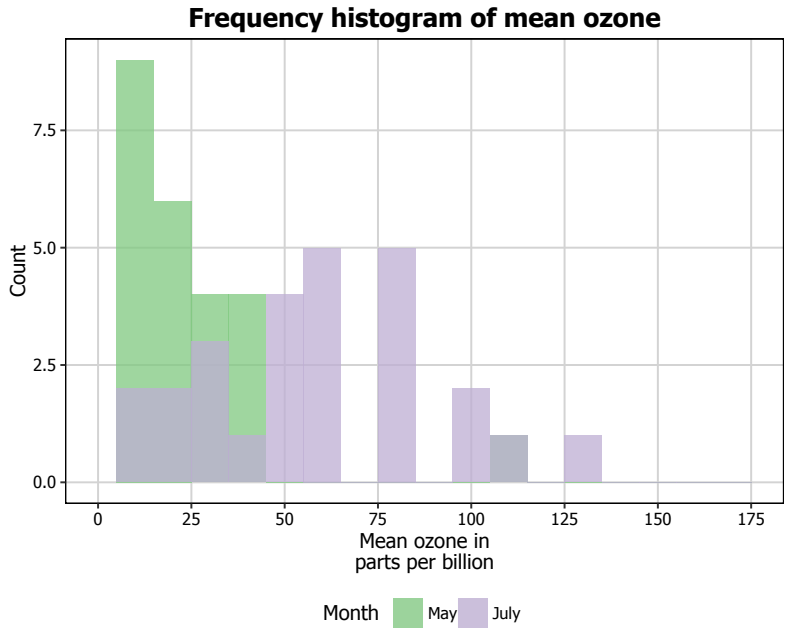
```
fill = c("steelblue", "yellowgreen", "violetred1")

p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
  geom_point(shape = 21) +
  ggtitle("Air Quality in New York by Day") +
  labs(x = "Day of the month", y = "Ozone (ppb)", size = "Wind Speed (mph) ",
    fill = "Months ") +
  scale_x_continuous(breaks = seq(1, 31, 5)) +
  scale_size(range = c(1, 10)) +
  scale_fill_manual(values = fill) +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key.size = unit(1, "cm"),
    legend.key = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p6
```


Air Quality in New York by Day
Ozone (ppb)
150
100
50
0
1
6
11
16
21
26
31
Day of the month
Wind Speed (mph)
4
8
12
16
Months
July
August
September

CHAPTER 7

Histograms

In this chapter, we will work towards creating the histogram below. We will take you from a basic histogram and explain all the customisations we add to the code step-by-step.



In this chapter, we will be using R's airquality dataset, which is contained in the `datasets` package. The first thing to do is load in the data, as below:

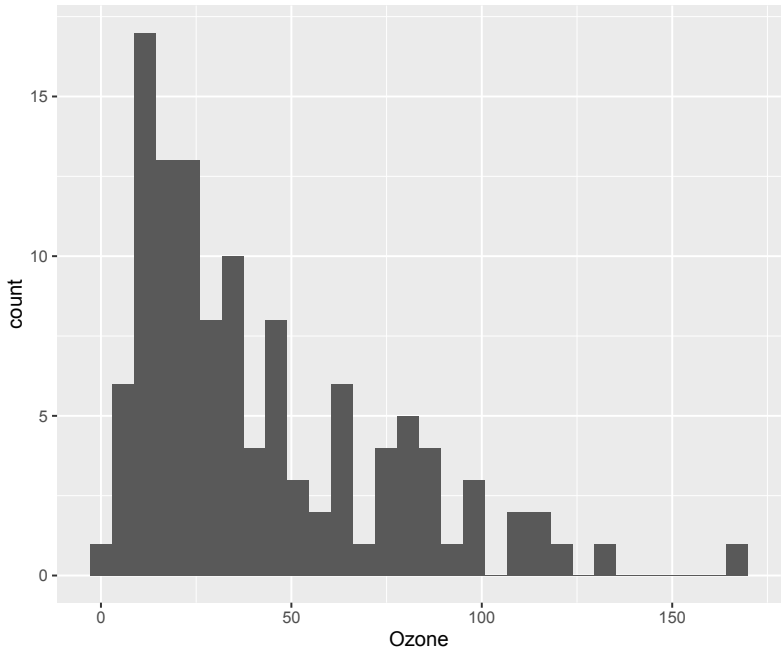
```
library(datasets)
library(ggplot2)
library(ggthemes)
library(extrafont)
library(grid)
library(RColorBrewer)
```

```
data(airquality)
```

7.1. Basic histogram

In order to initialise a plot we tell ggplot that `airquality` is our data, and specify that our x-axis plots the `Ozone` variable. We then instruct ggplot to render this as a histogram by adding the `geom_histogram()` option.

```
p7 <- ggplot(airquality, aes(x = Ozone)) + geom_histogram()
p7
```

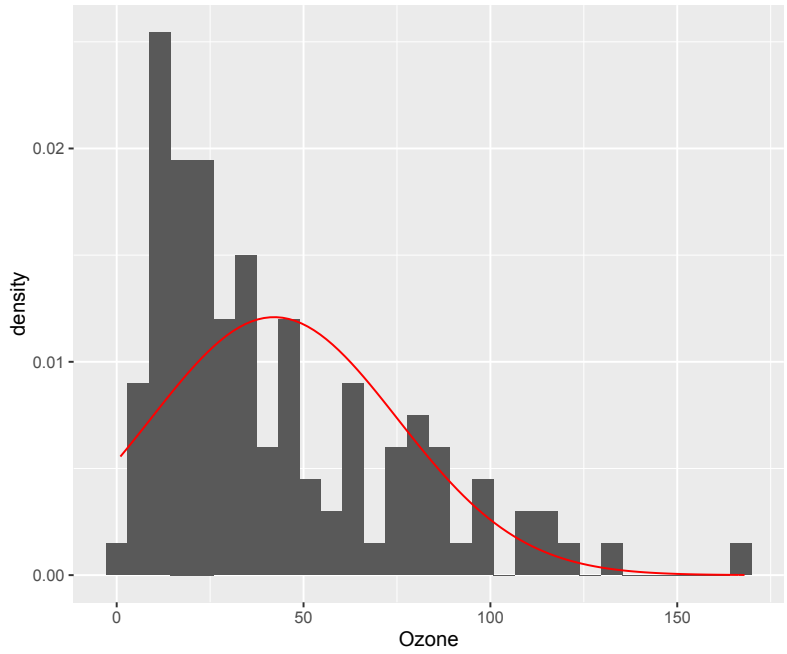


7.2. Adding a normal density curve

We can overlay a normal density function curve on top of our histogram to see how closely (or not) it fits a normal distribution. In this case, we can see it deviates from a normal distribution, showing marked positive skew. In order to overlay the function curve, we add the option `stat_function(fun = dnorm)`, and specify the shape using the `mean = mean(airquality$Ozone)` and `sd = sd(airquality$Ozone)` arguments. If you have missing data like we did, make sure you pass the `na.rm = TRUE` argument to the mean and sd parameters. Finally, you can change the colour using the `colour = "red"` argument. We will discuss how to customise colours further below.

One further change we must make to display the normal curve correctly is adding `aes(y = ..density..)` to the `geom_histogram` option. Note that the normal density curve will not work if you are using the frequency rather than the density, which we are changing in our next step.

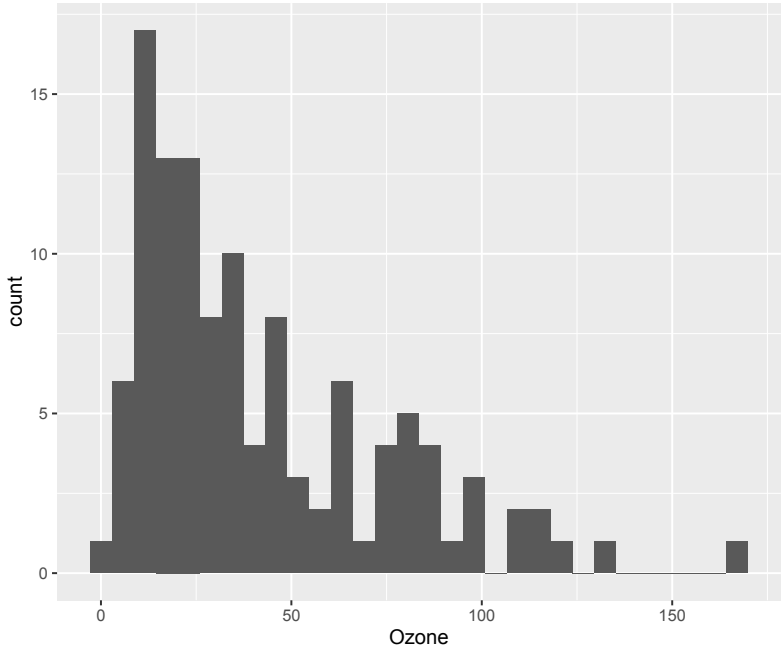
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..density..)) +
  stat_function(fun = dnorm, colour = "red",
    args = list(mean = mean(airquality$Ozone, na.rm = TRUE),
    sd = sd(airquality$Ozone, na.rm = TRUE)))
p7
```



7.3. Changing from density to frequency

Let's go back to the basic plot and lose the function curve. To change the y-axis from density to frequency, we add the `aes(y = ..count..)` option to `geom_histogram`.

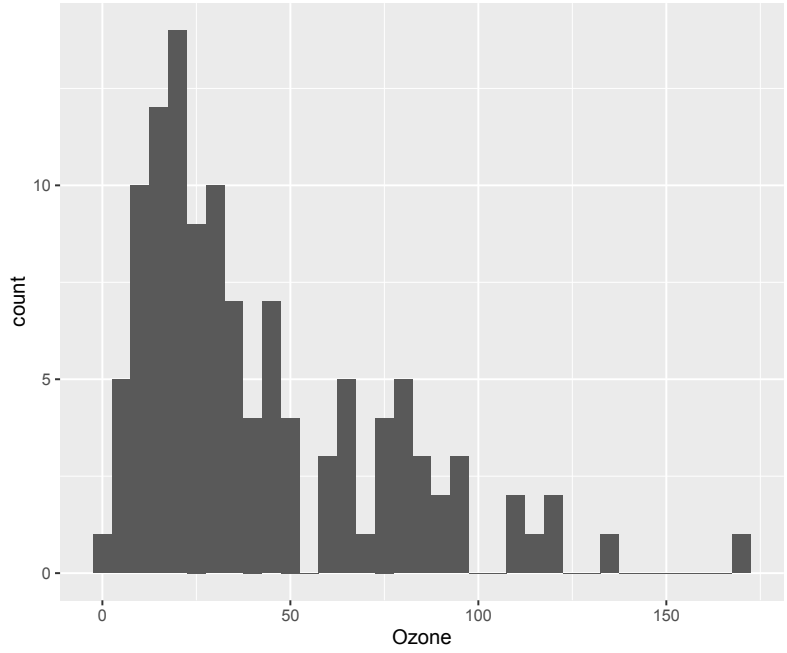
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..))
p7
```



7.4. Adjusting binwidth

To change the binwidth, we add a `binwidth` argument to `geom_histogram`. In this case, we will make binwidth 5 units of the `Ozone` variable.

```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5)
p7
```

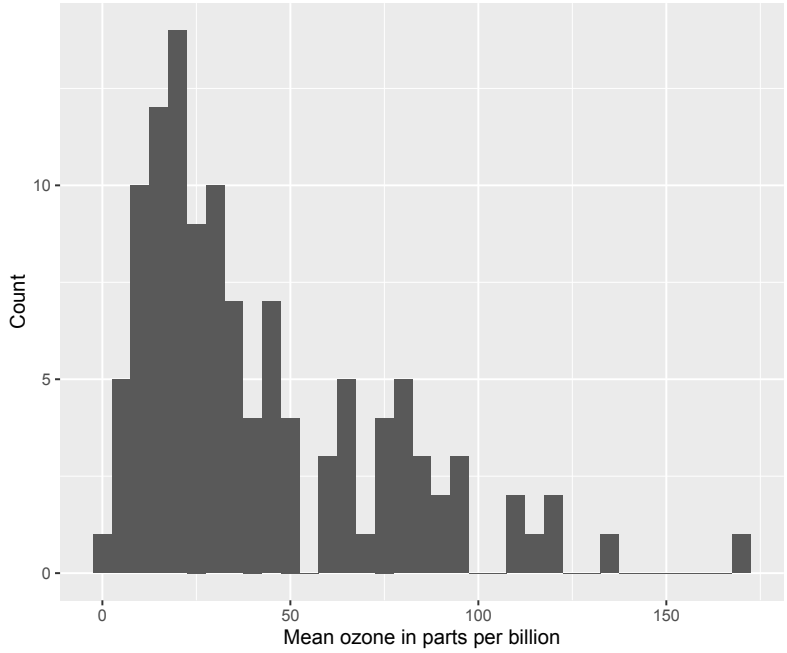


7.5. Customising axis labels

7.5.1. Single line labels

In order to change the axis labels, we have a couple of options. In this case, we have used the `scale_x_continuous` and `scale_y_continuous` options, as these have further customisation options for the axes we will use below. In each, we add the desired name to the `name` argument as a string.

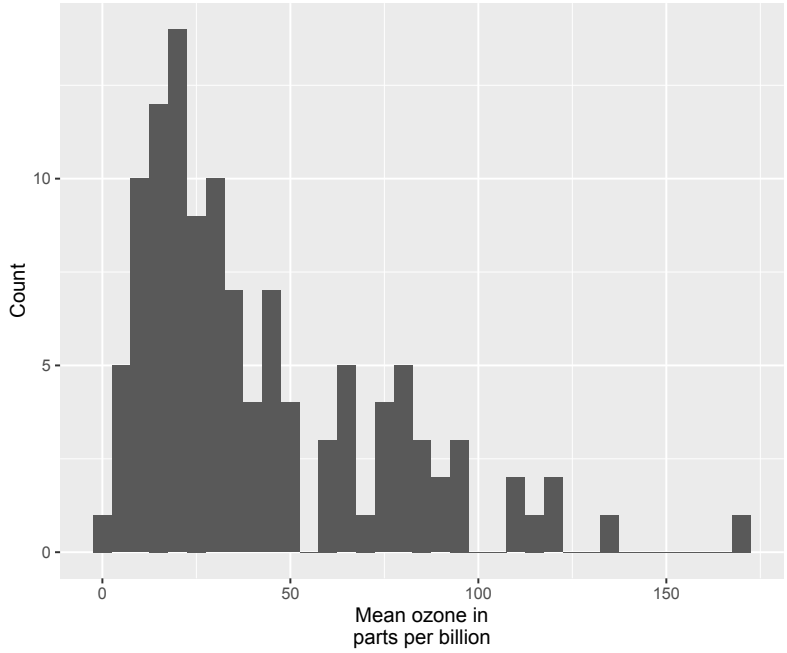
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5) +
  scale_x_continuous(name = "Mean ozone in parts per billion") +
  scale_y_continuous(name = "Count")
p7
```



7.5.2. Multiline labels

ggplot also allows for the use of multiline names (in both axes and titles). Here, we've changed the x-axis label so that it goes over two lines using the `\n` character to break the line.

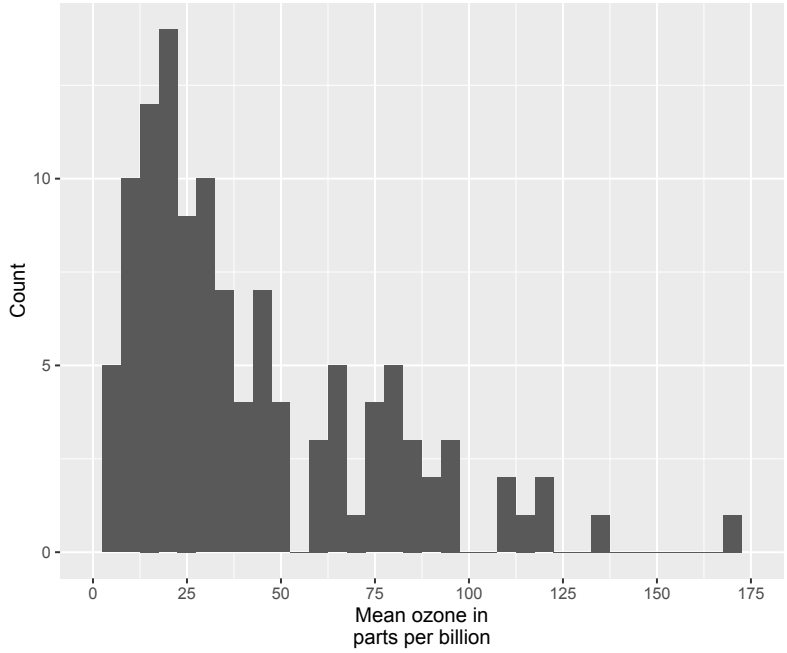
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion") +
  scale_y_continuous(name = "Count")
p7
```



7.6. Changing axis ticks

The next thing we will change is the axis ticks. Let's make the x-axis ticks appear at every 25 units rather than 50 using the `breaks = seq(0, 175, 25)` argument in `scale_x_continuous`. (The `seq` function is a base R function that indicates the start and endpoints and the units to increment by respectively. See `help(seq)` for more information.) We ensure that the x-axis begins and ends where we want by also adding the argument `limits = c(0, 175)` to `scale_x_continuous`.

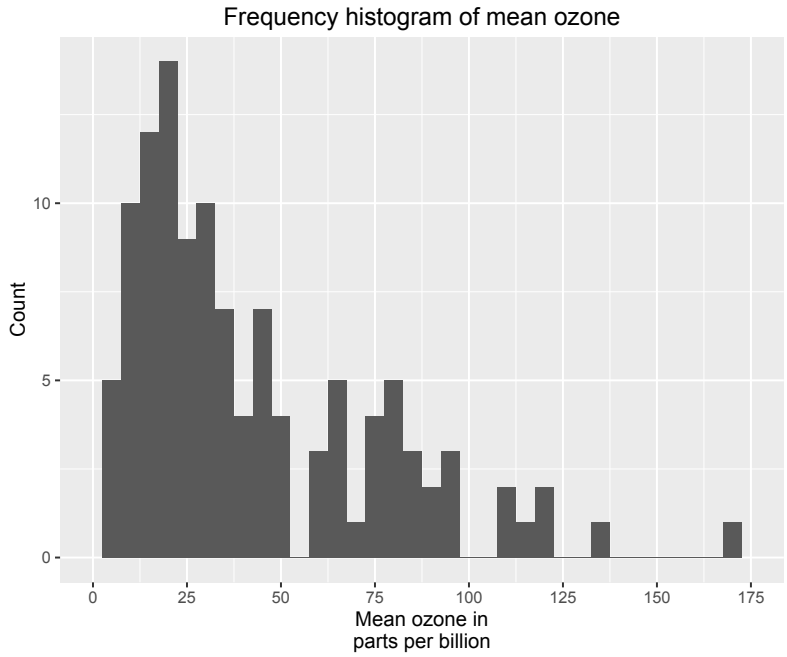
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count")
p7
```



7.7. Adding a title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument.

```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone")
p7
```

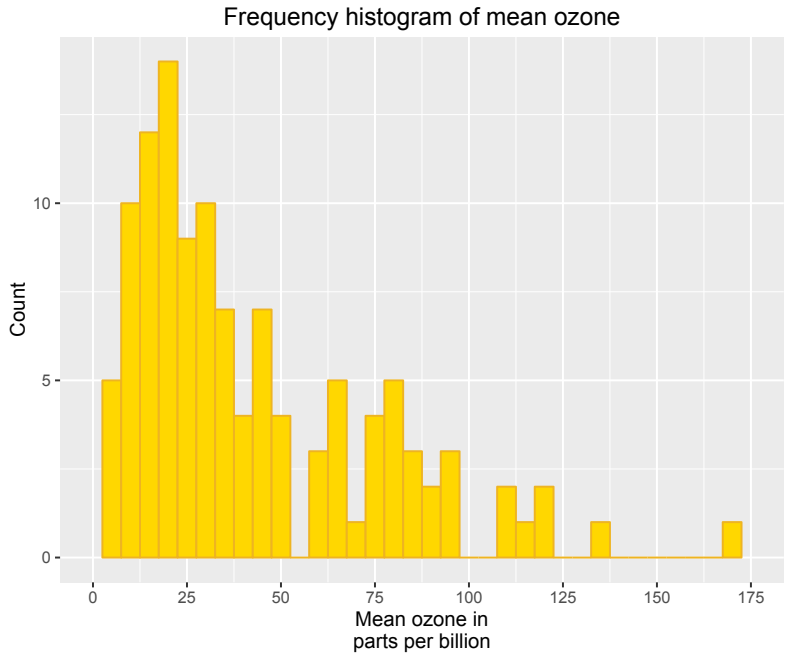


7.8. Changing the colour of the bars

7.8.1. By colour name

To change the line and fill colours of the bars, we add a valid colour to the `colour` and `fill` arguments in `geom_histogram` (note that we assigned these colours to variables outside of the plot to make it easier to change them). A list of valid colours is here.

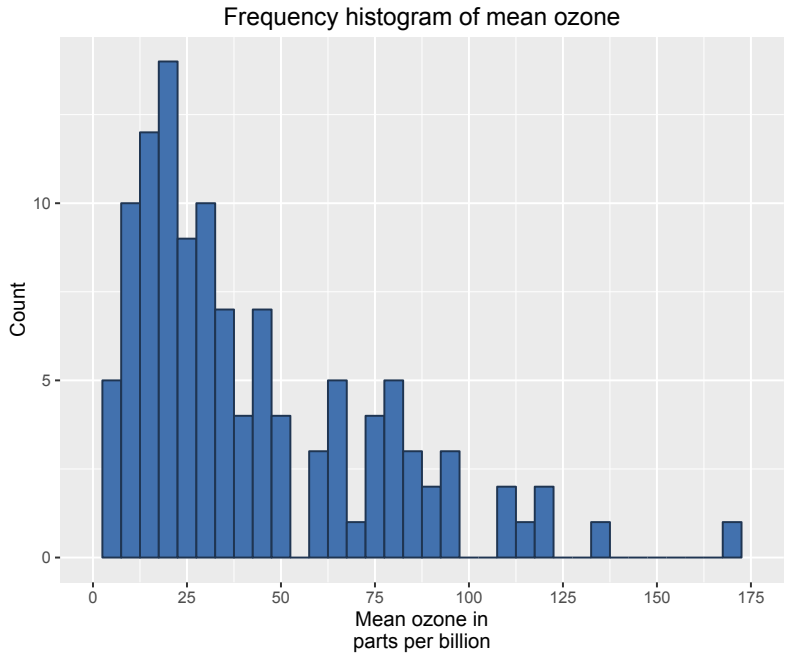
```
barfill <- "gold1"; barlines <- "goldenrod2"

p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5,
    colour = barlines, fill = barfill) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone")
p7
```



7.8.2. By HEX code

If you want to go beyond the options in the list above, you can also specify exact HEX colours by including them as a string preceded by a hash, e.g., "#FFFFFF". Below, we have called two shades of blue for the fill and lines using their HEX codes.

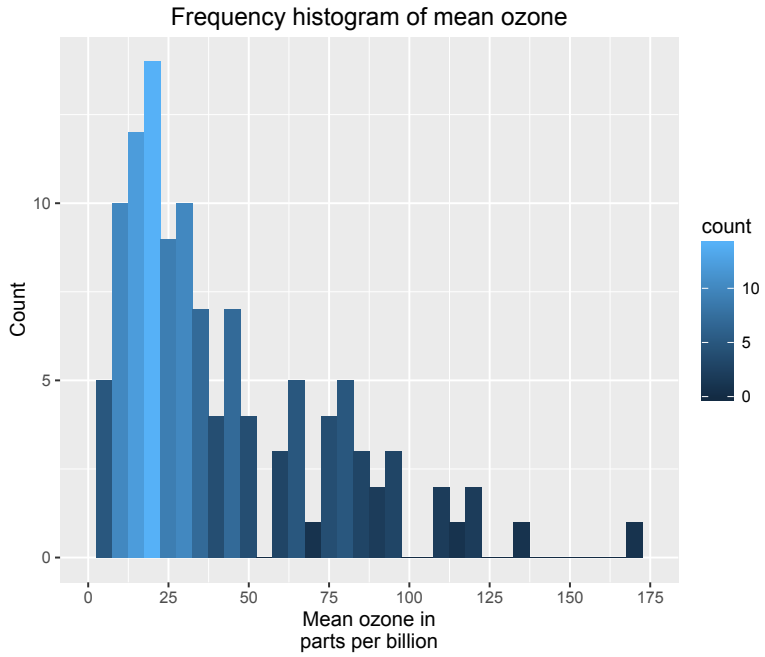
```
barfill <- "#4271AE"; barlines <- "#1F3552"

p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5,
    colour = barlines, fill = barfill) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone")
p7
```



7.8.3. Colour gradients

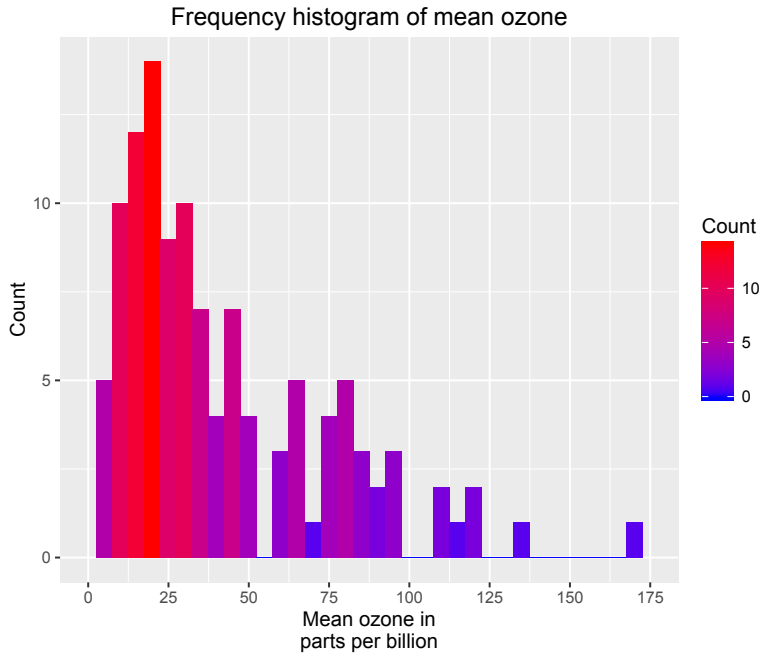
You can also add a gradient to your colour scheme that varies according to the frequency of the values. Below is the default gradient colour scheme. In order to do this, you can see we have changed the `aes(y = ..count..)` argument in `geom_histogram` to `aes(fill = ..count..)`.

```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(fill = ..count..), binwidth = 5) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone")
p7
```



You can customise the gradient by changing the anchoring colours for high and low. To do so, we have added the option `scale_fill_gradient` to the plot with the arguments `Count` (the name of the legend), `low` (the colour for the least frequent values) and `high` (the colour for the most frequent values).

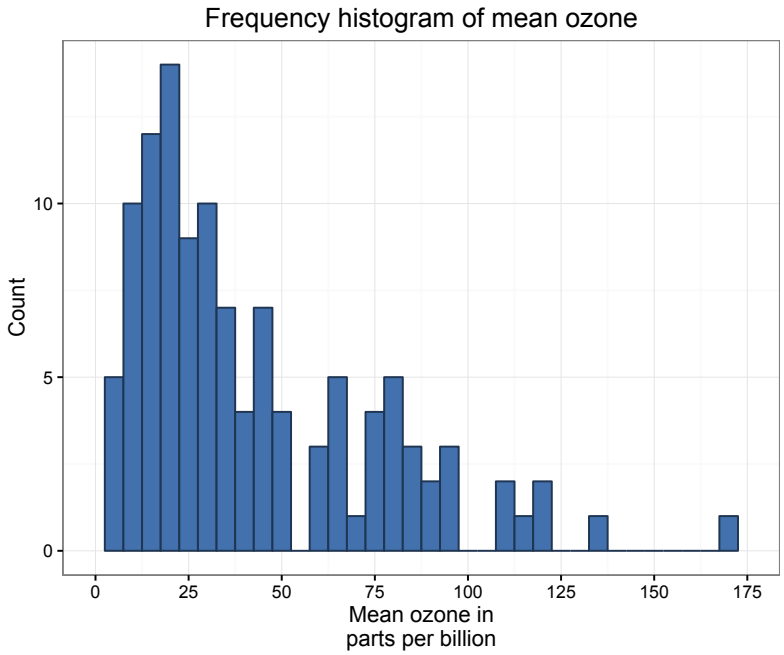
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(fill = ..count..), binwidth = 5) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  scale_fill_gradient("Count", low = "blue", high = "red")
p7
```



7.9. Using the white theme

We can also change the overall look of the plot using themes. We'll start using a simple theme customisation by adding `theme_bw()` after `ggplot()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5,
  colour = barlines, fill = barfill) +
    scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25),
    limits=c(0, 175)) +
    scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  theme_bw()
p7
```

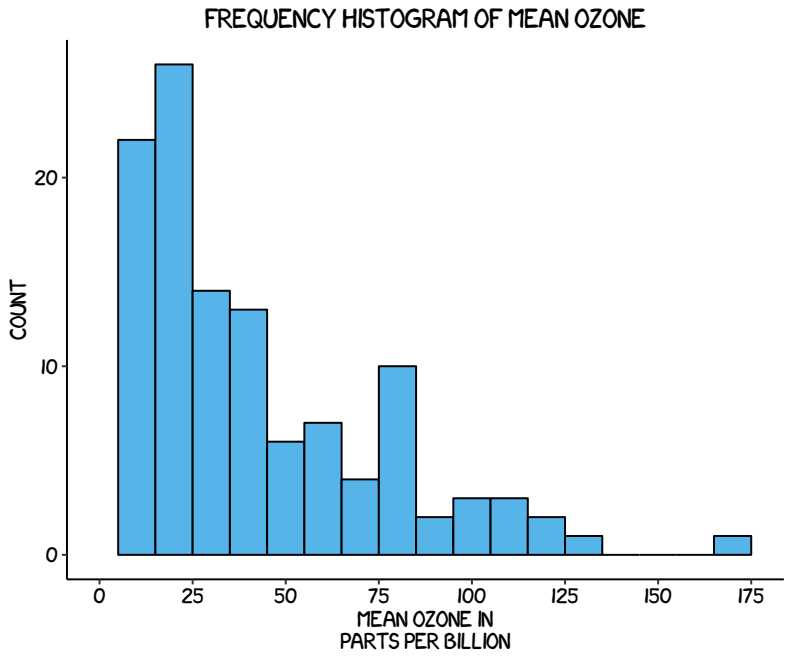


7.10. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 10,
    colour = "black", fill = "#56B4E9") +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
```

```
    plot.title=element_text(family="xkcd-Regular"),
    text=element_text(family="xkcd-Regular"))
p7
```

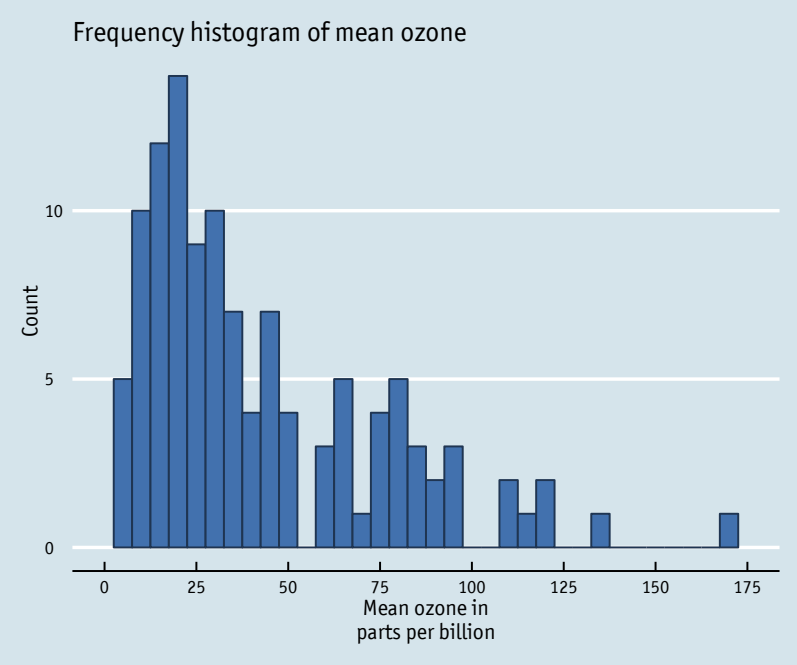


7.11. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine.

```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5,
    colour = barlines, fill = barfill) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.title = element_text(size = 12),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.text = element_text(size = 10),
    text = element_text(family = "OfficinaSanITC-Book"),
```

```
    plot.title = element_text(family="OfficinaSanITC-Book"))
p7
```

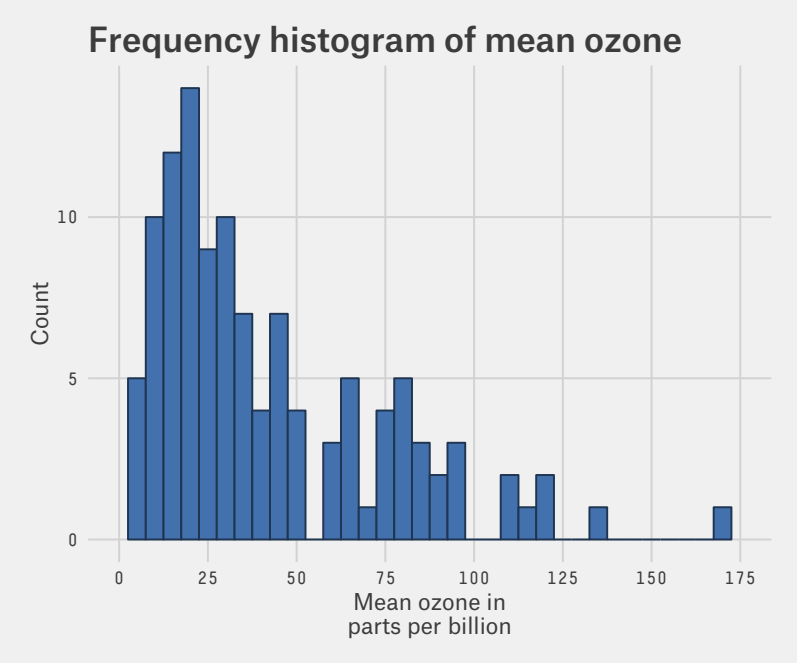


7.12. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

```
p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5,
    colour = barlines, fill = barfill) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.title=element_text(family="Atlas Grotesk Regular", size = 10),
    legend.text=element_text(family="Atlas Grotesk Regular", size = 10),
    plot.title=element_text(family="Atlas Grotesk Medium"),
```

```
    text=element_text(family="DecimaMonoPro"))
p7
```

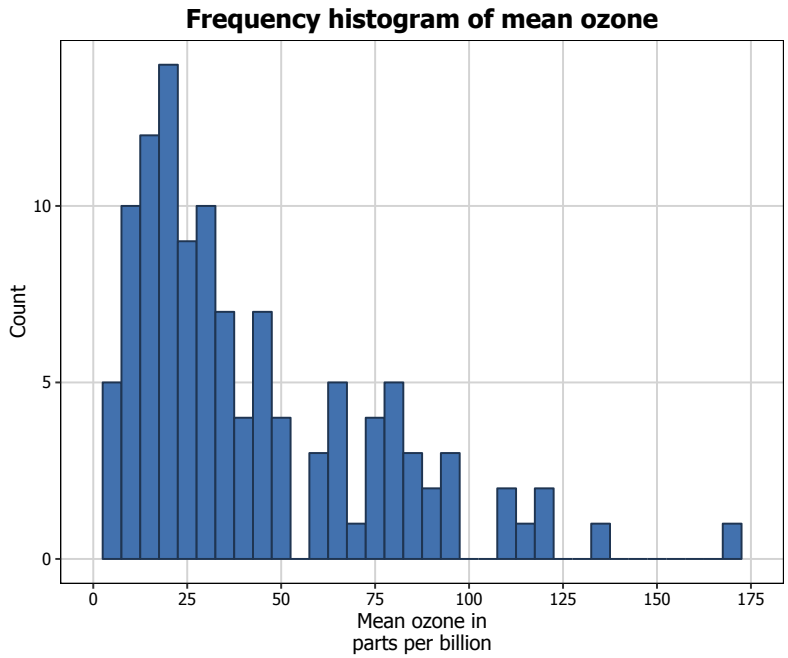


7.13. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here is a custom plot where we have modified the axes, background and font.

```
barfill <- "#4271AE"; barlines <- "#1F3552"

p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5,
    colour = barlines, fill = barfill) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25),
    limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key = element_blank(),
```

```
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p7
```

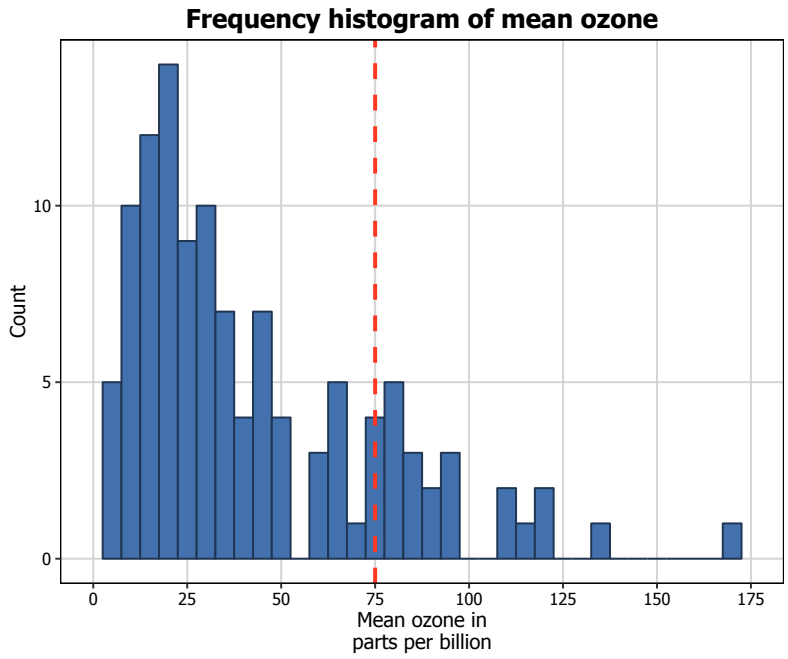


7.14. Adding lines

Let's say that we want to add a cutoff value to the chart (75 parts of ozone per billion). We add the `geom_vline` option to the chart, and specify where it goes on the x-axis using the `xintercept` argument. We can customise how it looks using the `colour` and `linetype` arguments in `geom_vline`. (In the the same way, horizontal lines can be added using the `geom_hline`.)

```
barfill <- "#4271AE"; barlines <- "#1F3552"

p7 <- ggplot(airquality, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 5,
    colour = barlines, fill = barfill) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  geom_vline(xintercept = 75, size = 1, colour = "#FF3721",
    linetype = "dashed") +
```

```
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p7
```



7.15. Multiple histograms

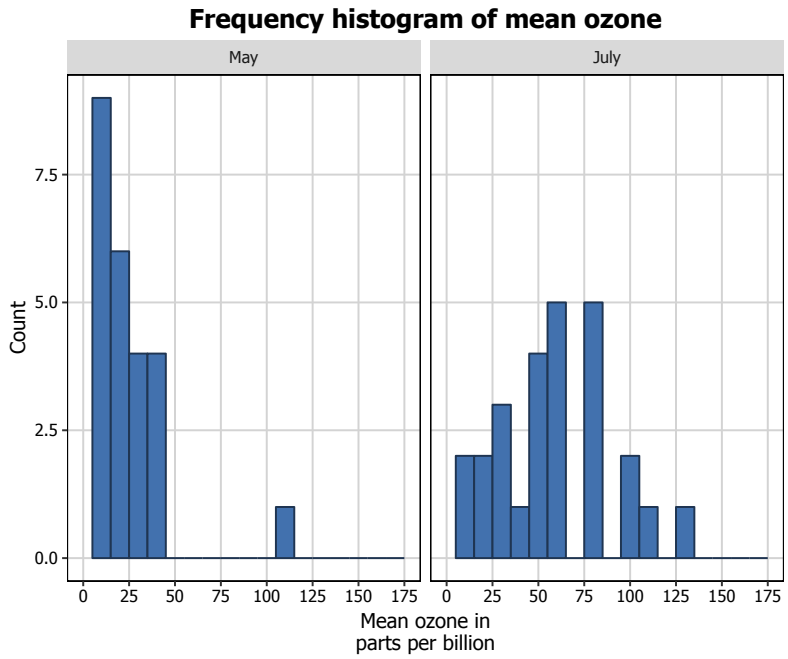
You can also easily create multiple histograms by the levels of another variable. There are two options, in separate (panel) plots, or in the same plot.

7.15.1. In panel plots

We first need to do a little data wrangling. In order to make the graphs a bit clearer, we've kept only months "5" (May) and "7" (July) in a new dataset `airquality_trimmed`. We also need to convert this variable into either a character or factor variable. We have created a new factor variable `Month.f`.

In order to produce a panel plot by month, we add the `facet_grid(. ~ Month.f)` option to the plot. The additional `scale = free` argument in `facet_grid` means that the y-axes of each plot do not need to be the same.

```
airquality_trimmed <- airquality[which(airquality$Month == 5 |
  airquality$Month == 7), ]
airquality_trimmed$Month.f <- factor(airquality_trimmed$Month,
  labels = c("May", "July"))

p7 <- ggplot(airquality_trimmed, aes(x = Ozone)) +
  geom_histogram(aes(y = ..count..), binwidth = 10,
    colour = barlines, fill = barfill) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  facet_grid(. ~ Month.f, scales = "free") +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p7
```

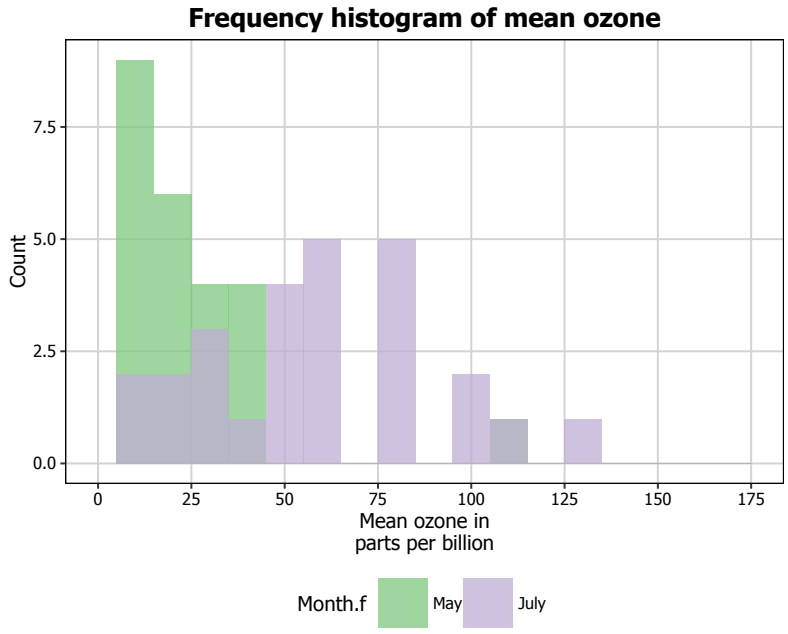


7.15.2. In the same plot

In order to plot the two months in the same plot, we add several things. Firstly, in the `ggplot` function, we add a `fill = Month.f` argument to `aes`. Secondly, in order to more clearly see the graph, we add two arguments to the `geom_histogram` option, `position = "identity"` and `alpha = 0.75`. This controls the position and transparency of the curves respectively. Finally, you can customise the colours of the histograms by adding the `scale_fill_brewer` to the plot from the `RColorBrewer` package. This blog post describes the available packages.

```
p7 <- ggplot(airquality_trimmed, aes(x = Ozone, fill = Month.f)) +
  geom_histogram(aes(y = ..count..), binwidth = 10,
    position="identity", alpha=0.75) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  scale_fill_brewer(palette="Accent") +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
```

```
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p7
```

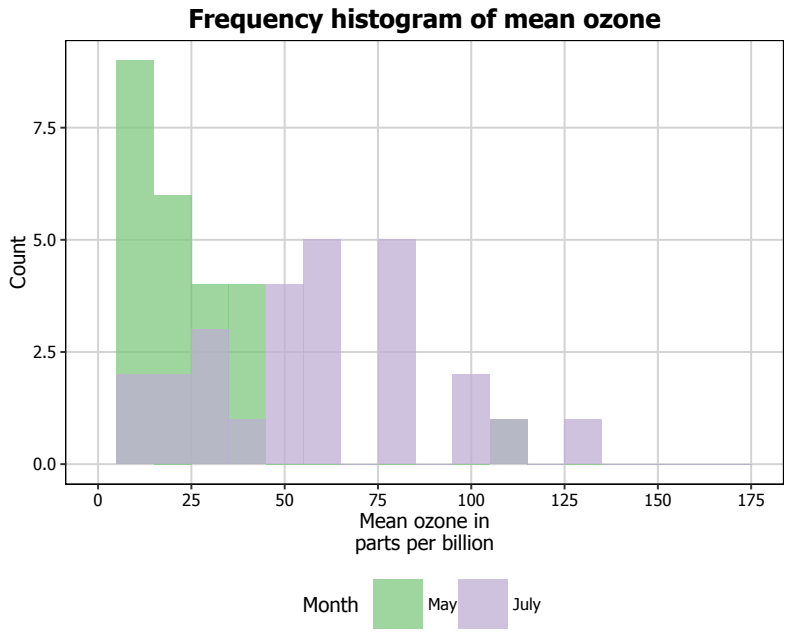


7.16. Formatting the legend

Finally, we can format the legend. Firstly, we can change the position by adding the `legend.position = "bottom"` argument to the `theme` option, which moves the legend under the plot. Secondly, we can fix the title by adding the `labs(fill="Month")` option to the plot. We now have our final graph that we showed at the beginning of this chapter.

```
p7 <- ggplot(airquality_trimmed, aes(x = Ozone, fill = Month.f)) +
  geom_histogram(aes(y = ..count..), binwidth = 10,
    position="identity", alpha=0.75) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_y_continuous(name = "Count") +
  ggtitle("Frequency histogram of mean ozone") +
  scale_fill_brewer(palette="Accent") +
  labs(fill="Month ") +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    legend.position="bottom",
    legend.direction="horizontal",
```

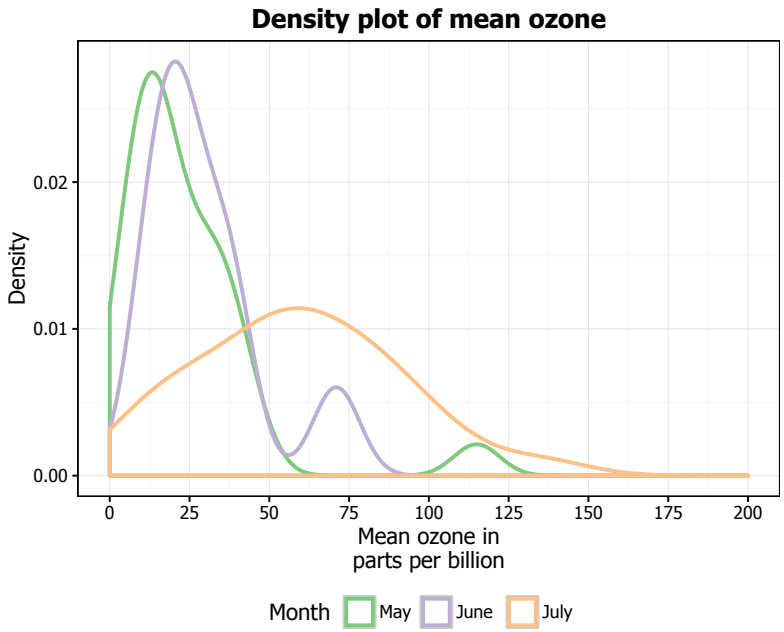
```
    legend.box = "horizontal",
    legend.key = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p7
```



CHAPTER 8

Density plots

In this chapter, we will work towards creating the density plot below. We will take you from a basic density plot and explain all the customisations we add to the code step-by-step.



In this chapter, we will be using R's airquality dataset, which is contained in the `datasets` package. The first thing to do is load in the data, as below:

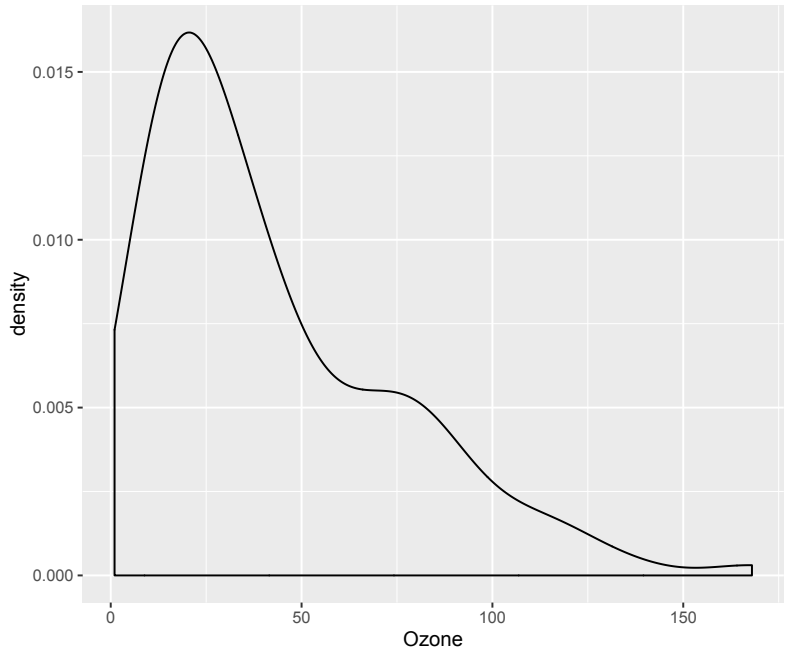
```
library(datasets)
library(ggplot2)
library(ggthemes)
library(RColorBrewer)
library(extrafont)
library(grid)
```

```
data(airquality)
```

8.1. Basic density plot

In order to initialise a plot we tell ggplot that `airquality` is our data, and specify that our x-axis plots the `Ozone` variable. We then instruct ggplot to render this as a density plot by adding the `geom_density()` option.

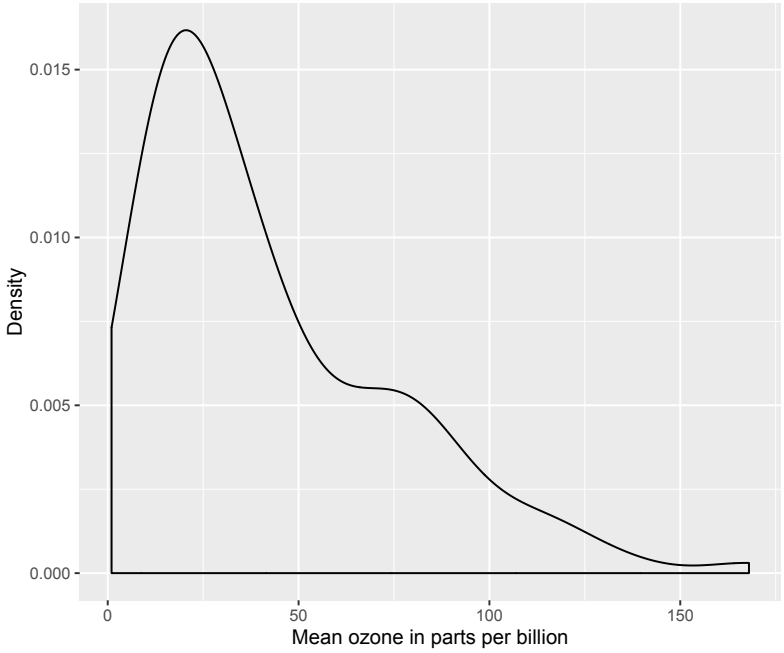
```
p8 <- ggplot(airquality, aes(x = Ozone)) + geom_density()
p8
```



8.2. Customising axis labels

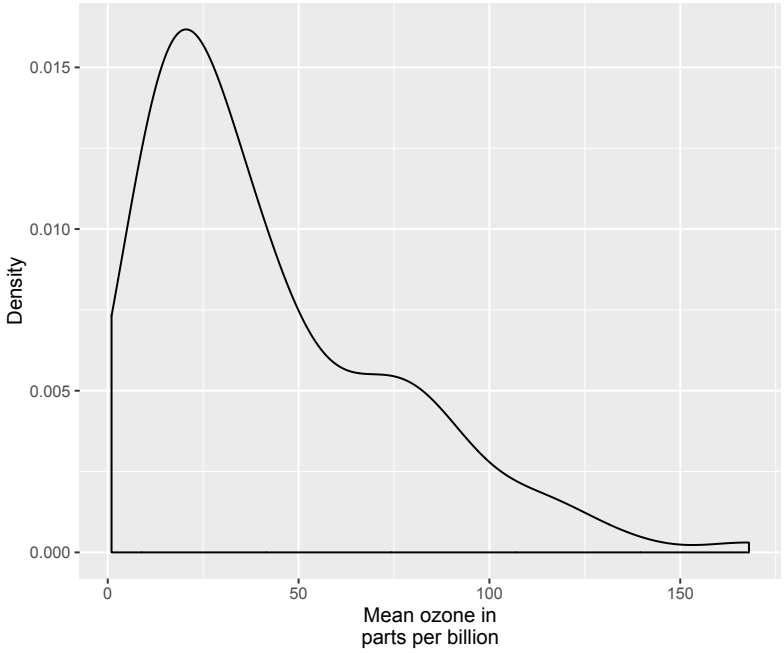
In order to change the axis labels, we have a couple of options. In this case, we have used the `scale_x_continuous` and `scale_y_continuous` options, as these have further customisation options for the axes we will use below. In each, we add the desired name to the `name` argument as a string.

```
p8 <- p8 + scale_x_continuous(name = "Mean ozone in parts per billion") +
    scale_y_continuous(name = "Density")
p8
```



ggplot also allows for the use of multiline names (in both axes and titles). Here, we've changed the x-axis label so that it goes over two lines using the `\n` character to break the line.

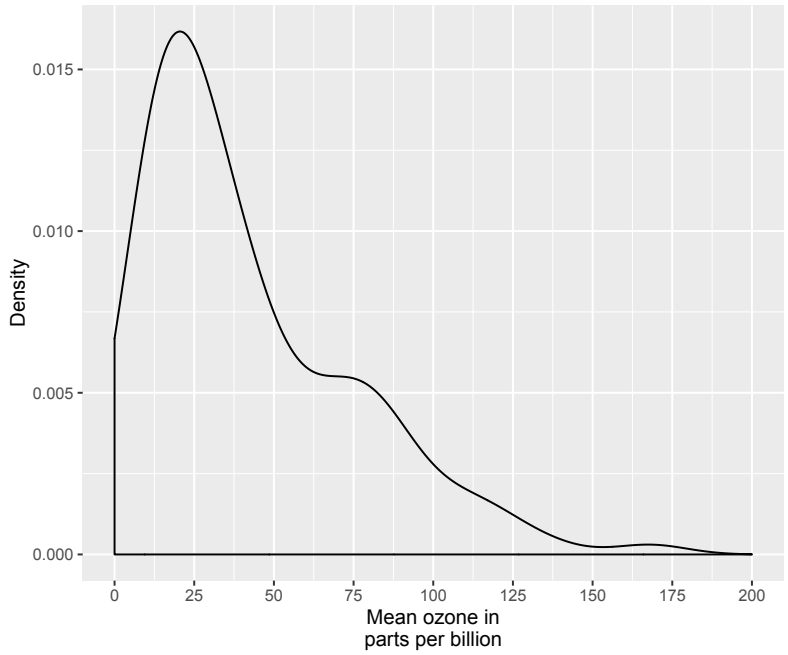
```
p8 <- p8 + scale_x_continuous(name = "Mean ozone in\nparts per billion")
p8
```



8.3. Changing axis ticks

The next thing we will change is the axis ticks. Let's make the x-axis ticks appear at every 25 units rather than 50 using the `breaks = seq(0, 200, 25)` argument in `scale_x_continuous`. (The `seq` function is a base R function that indicates the start and endpoints and the units to increment by respectively. See `help(seq)` for more information.) We ensure that the x-axis begins and ends where we want by also adding the argument `limits = c(0, 200)` to `scale_x_continuous`.

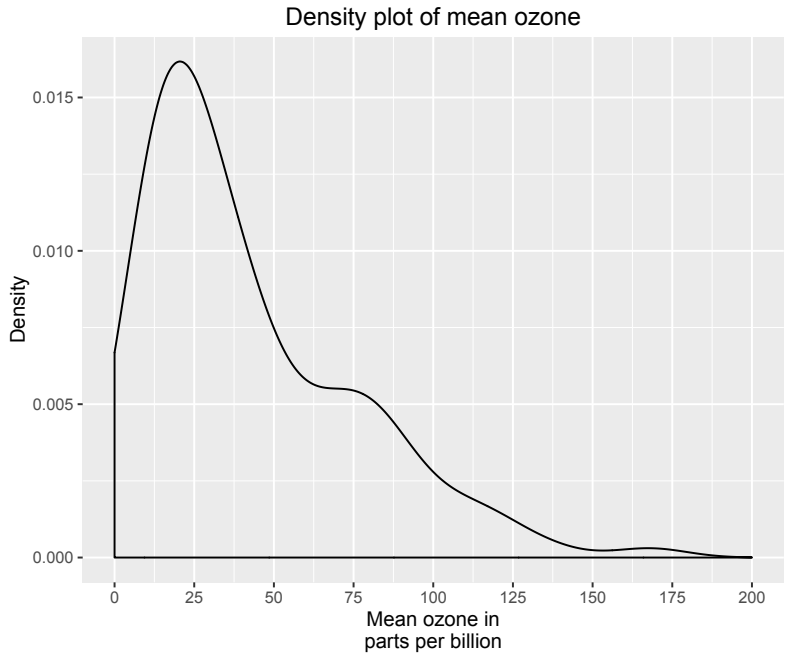
```
p8 <- p8 + scale_x_continuous(name = "Mean ozone in\nparts per billion",
     breaks = seq(0, 200, 25), limits=c(0, 200))
p8
```



8.4. Adding a title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument.

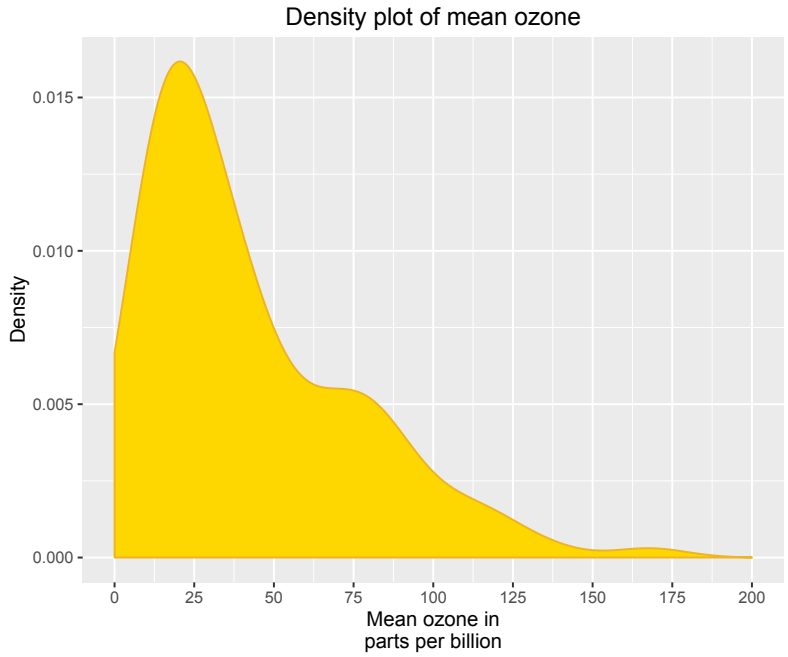
```
p8 <- p8 + ggtitle("Density plot of mean ozone")
p8
```



8.5. Changing the colour of the curves

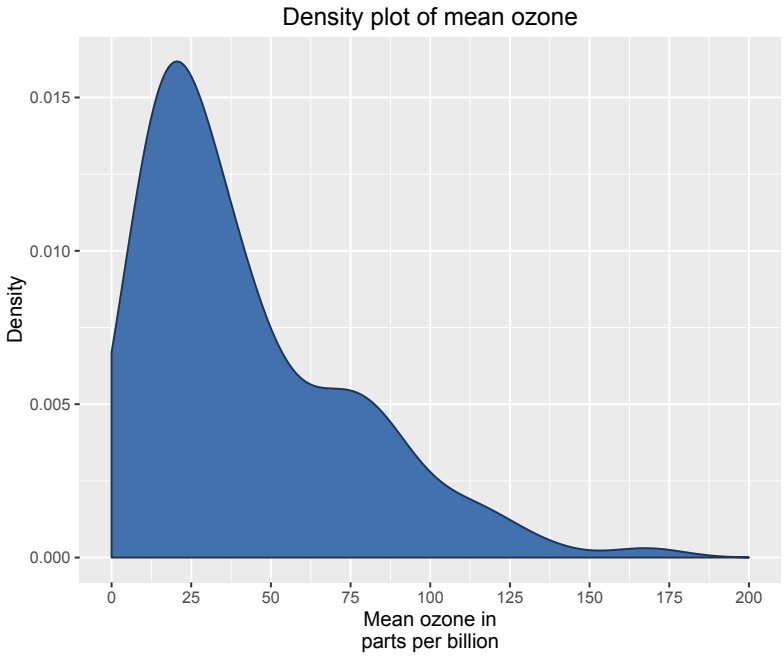
To change the line and fill colours of the density plot, we add a valid colour to the `colour` and `fill` arguments in `geom_density()` (note that we assigned these colours to variables outside of the plot to make it easier to change them). A list of valid colours is here.

```
fill <- "gold1"; line <- "goldenrod2"

p8 <- ggplot(airquality, aes(x = Ozone)) +
  geom_density(fill = fill, colour = line) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone")
p8
```



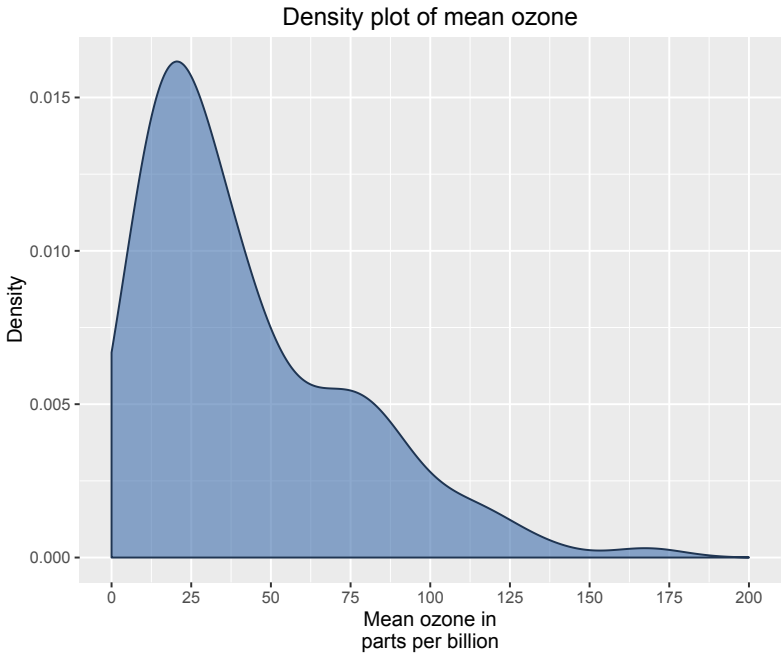
If you want to go beyond the options in the list above, you can also specify exact HEX colours by including them as a string preceded by a hash, e.g., "#FFFFFF". Below, we have called two shades of blue for the fill and lines using their HEX codes.

```
fill <- "#4271AE"; line <- "#1F3552"

p8 <- ggplot(airquality, aes(x = Ozone)) +
  geom_density(fill = fill, colour = line) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone")
p8
```



You can also specify the degree of transparency in the density fill area using the argument `alpha` in `geom_density`. This ranges from 0 to 1.

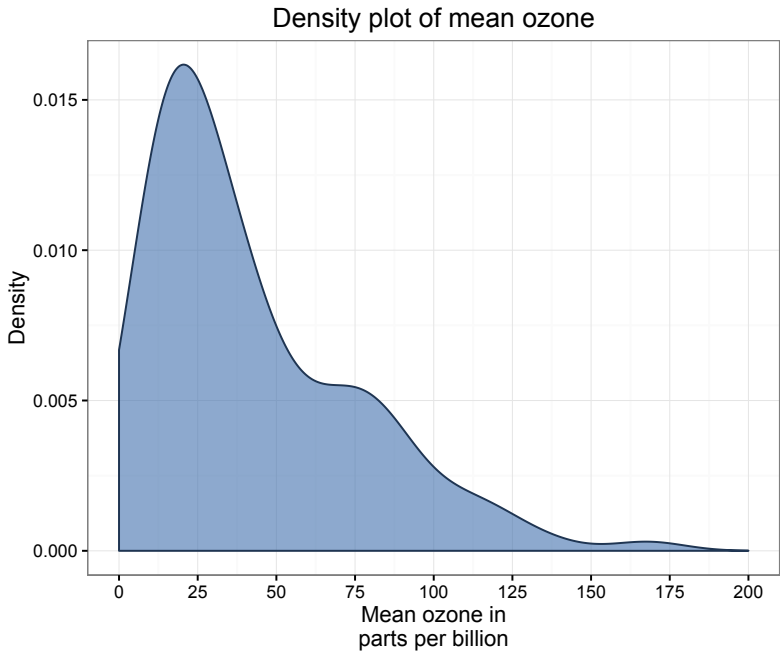
```
p8 <- ggplot(airquality, aes(x = Ozone)) +
  geom_density(fill = fill, colour = line, alpha = 0.6) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone")
p8
```



8.6. Using the white theme

We can also change the overall look of the plot using themes. We'll start using a simple theme customisation by adding `theme_bw()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p8 <- p8 + theme_bw()
p8
```

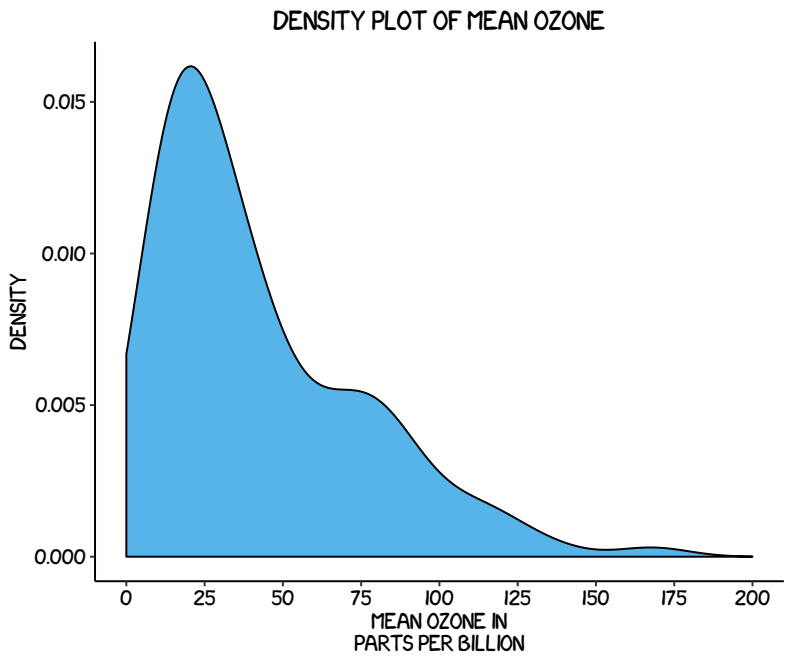


8.7. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
p8 <- ggplot(airquality, aes(x = Ozone)) +
  geom_density(colour = "black", fill = "#56B4E9") +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone") +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title=element_text(family="xkcd-Regular"),
```

```
    text=element_text(family="xkcd-Regular"))
p8
```

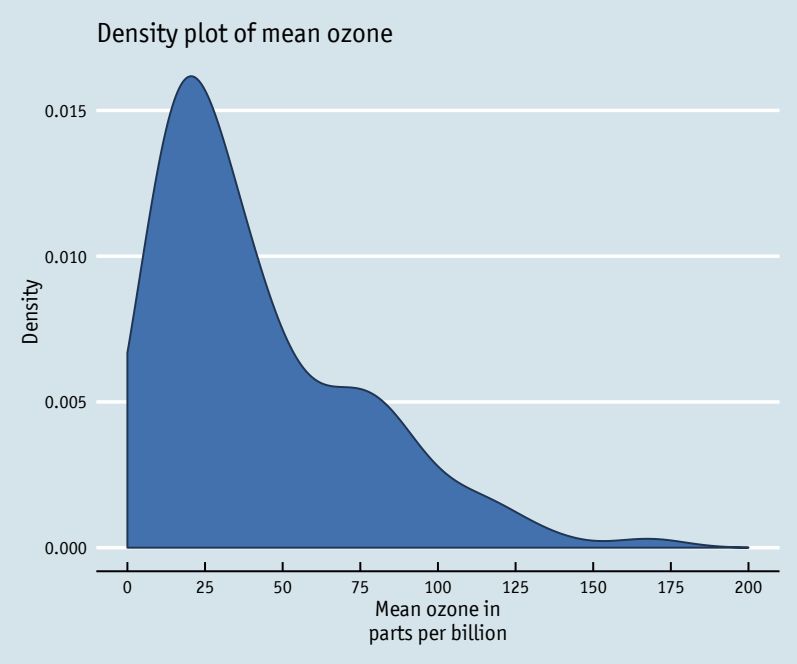


8.8. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

```
p8 <- ggplot(airquality, aes(x = Ozone)) +
  geom_density(fill = fill, colour = line) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone") +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.title = element_text(size = 12),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.text = element_text(size = 10),
    text = element_text(family = "OfficinaSanITC-Book"),
```

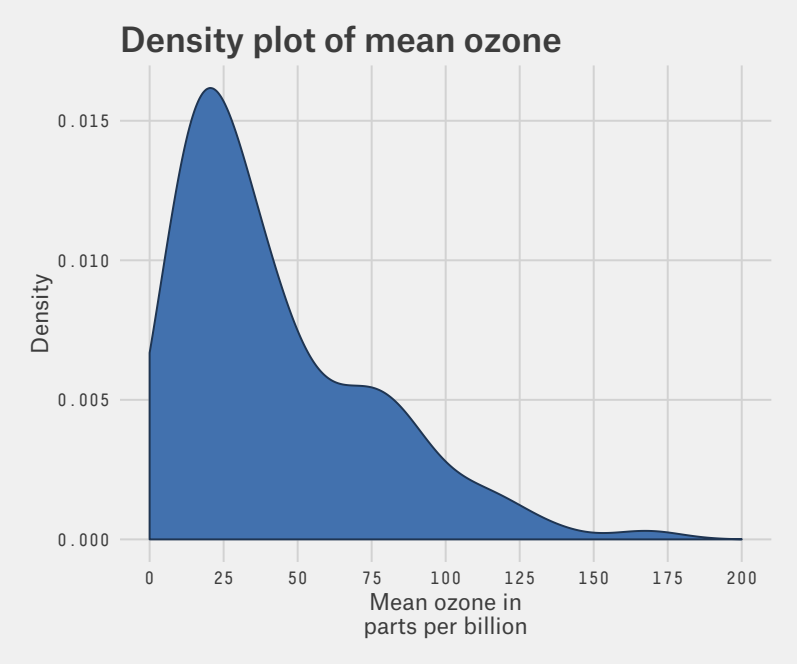
```
    plot.title = element_text(family="OfficinaSanITC-Book"))
p8
```



8.9. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

```
p8 <- ggplot(airquality, aes(x = Ozone)) +
  geom_density(fill = fill, colour = line) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone") +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.title=element_text(family="Atlas Grotesk Regular", size = 10),
    legend.text=element_text(family="Atlas Grotesk Regular", size = 10),
    plot.title=element_text(family="Atlas Grotesk Medium"),
    text=element_text(family="DecimaMonoPro"))
p8
```



8.10. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here is a custom plot where we have modified the axes, background and font.

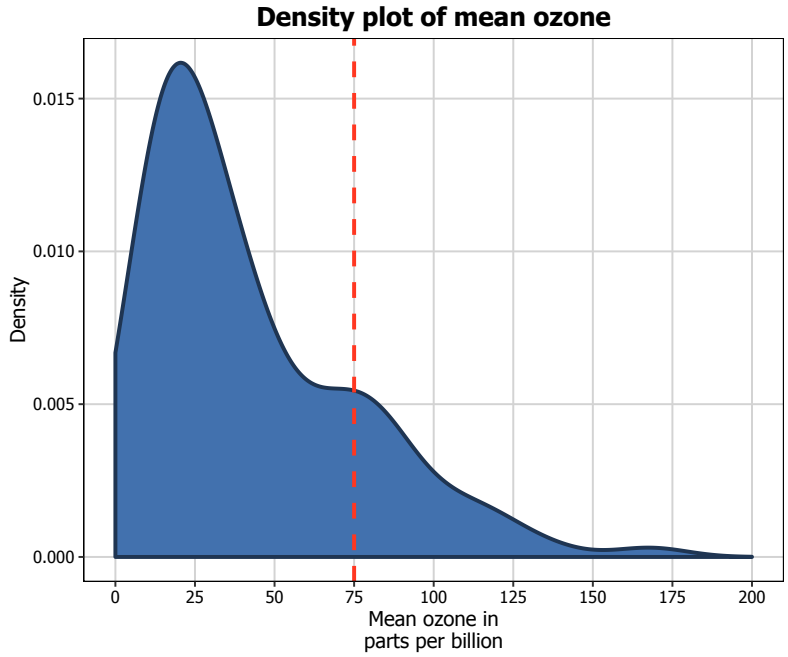
```
fill <- "#4271AE"; lines <- "#1F3552"

p8 <- ggplot(airquality, aes(x = Ozone)) +
  geom_density(colour = lines, fill = fill, size = 1) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone") +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key = element_blank(),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
```

```
    text=element_text(family="Tahoma"))
p8
```

8.11. Adding lines

Let's say that we want to add a cutoff value to the chart (75 parts of ozone per billion). We add the `geom_vline` option to the chart, and specify where it goes on the x-axis using the `xintercept` argument. We can customise how it looks using the `colour` and `linetype` arguments in `geom_vline`. (In the the same way, horizontal lines can be added using the `geom_hline`.)

```
p8 <- p8 + geom_vline(xintercept = 75, size = 1, colour = "#FF3721",
     linetype = "dashed")
p8
```



8.12. Multiple densities

You can also easily create multiple density plots by the levels of another variable. There are two options, in separate (panel) plots, or in the same plot. There are also a couple of variations on these we'll discuss below.

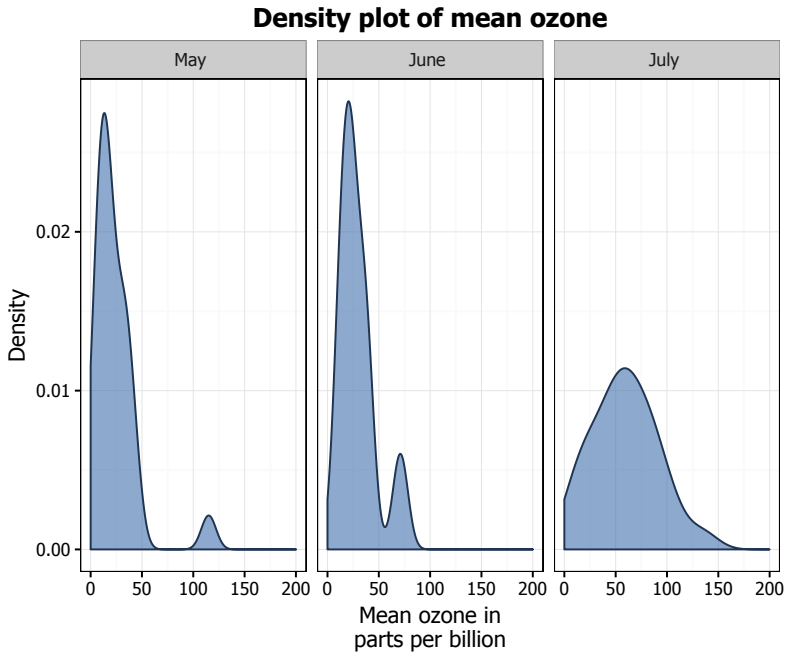
8.12.1. In panel plots

We first need to do a little data wrangling. In order to make the graphs a bit clearer, we've kept only months "5" (May), "6" (June) and "7" (July) in a new dataset `airquality_trimmed`. We also need to convert this variable into either a character or factor variable. We have created a new factor variable `Month.f`.

In order to produce a panel plot by month, we add the `facet_grid(. ~ Month.f)` option to the plot. Note that we've also changed the scale of the x-axis to make it fit a little more neatly in the panel format.

```
airquality_trimmed <- airquality[which(airquality$Month == 5 |
    airquality$Month == 6 |
    airquality$Month == 7), ]
airquality_trimmed$Month.f <- factor(airquality_trimmed$Month,
    labels = c("May ", "June ", "July "))

p8 <- ggplot(airquality_trimmed, aes(x = Ozone)) +
  geom_density(fill = fill, colour = line, alpha = 0.6) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 50), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
```

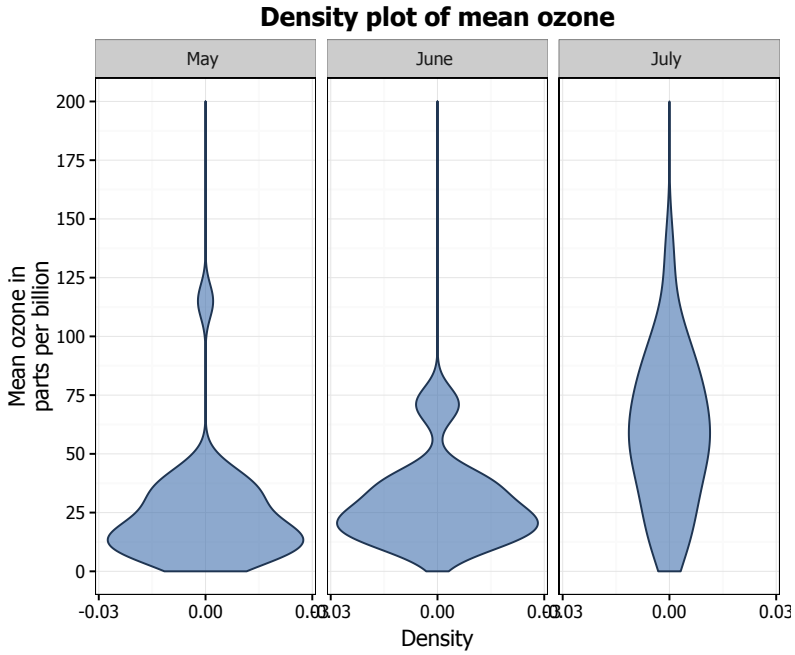
```
  ggtitle("Density plot of mean ozone") +
  facet_grid(. ~ Month.f) +
  theme_bw() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text = element_text(size = 12, family = "Tahoma"))
p8
```



An alternative to a panel plot is the *volcano plot*. This plot swaps the axes (so the variable of interest is on the y-axis and the density is on the x-axis), and reflects the density. In order to create this plot, we replace `geom_density` with `stat_density`, and include the arguments `aes(ymax = ..density.., ymin = -..density..)` and `geom = "ribbon"` to create a density plot, the usual `fill`, `colour` and `alpha` arguments, and `position = "identity"`. We also need to add a `coord_flip()` option to the plot.

```
p8 <- ggplot(airquality_trimmed, aes(x = Ozone)) +
  stat_density(aes(ymax = ..density..,  ymin = -..density..),
    geom = "ribbon",
    fill = fill, colour = line, alpha = 0.6,
    position = "identity") +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density",
    breaks = seq(-0.03, 0.03, 0.03)) +
  ggtitle("Density plot of mean ozone") +
  facet_grid(. ~ Month.f) +
  coord_flip() +
  theme_bw() +
```

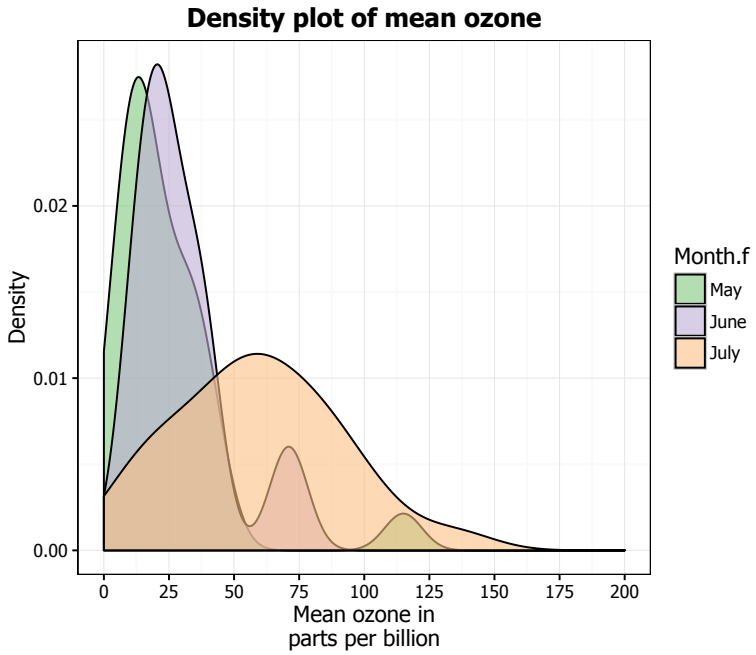
```
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text = element_text(size = 12, family = "Tahoma"))
p8
```



8.12.2. In the same plot

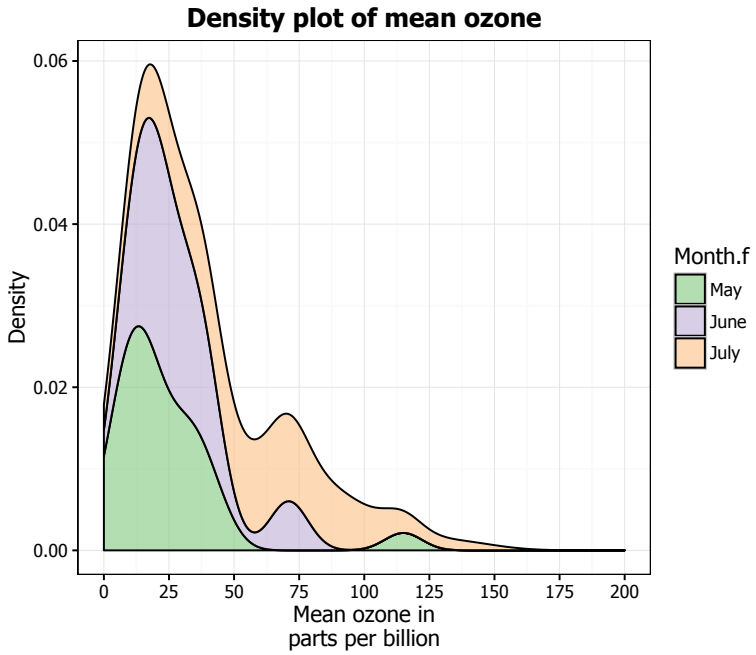
In order to plot the three months in the same plot, we add several things. Firstly, in the `ggplot` function, we add a `fill = Month.f` argument to `aes`. Secondly, in order to more clearly see the graph, we add the argument `position = "identity"` to the `geom_density` option. This controls the position of the curves respectively. Finally, you can customise the colours of the histograms by adding the `scale_fill_brewer` to the plot from the `RColorBrewer` package. This blog post describes the available palettes.

```
p8 <- ggplot(airquality_trimmed, aes(x = Ozone, fill = Month.f)) +
  geom_density(position="identity", alpha=0.6) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone") +
  scale_fill_brewer(palette="Accent") +
  theme_bw() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text = element_text(size = 12, family = "Tahoma"))
p8
```



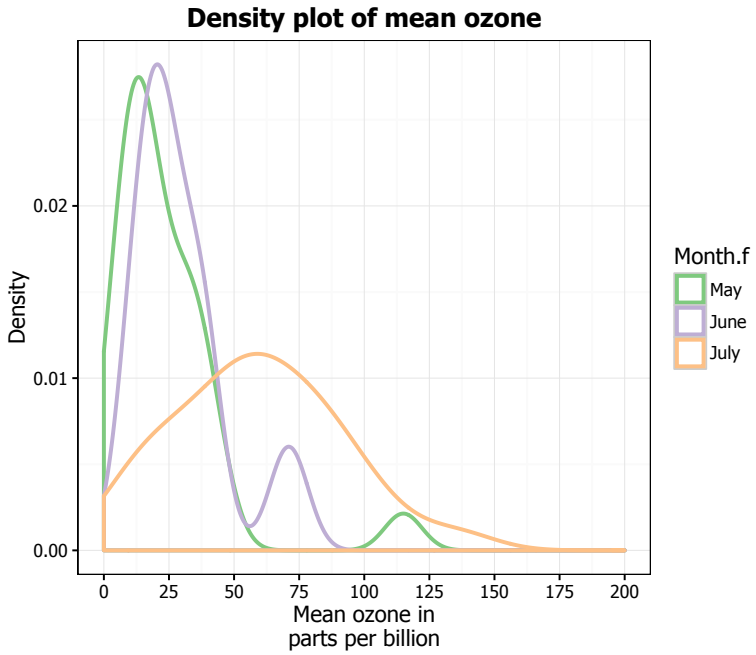
These densities are a little hard to see. One way we can make it easier to see them is to stack the densities on top of each other. To do so, we swap `position = "stack"` for `position = "identity"` in `geom_density`.

```
p8 <- ggplot(airquality_trimmed, aes(x = Ozone, fill = Month.f)) +
  geom_density(position = "stack", alpha = 0.6) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone") +
  scale_fill_brewer(palette="Accent") +
  theme_bw() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text = element_text(size = 12, family = "Tahoma"))
p8
```



Another way to make it a little easier to see the densities by dropping out the fill. To do this need a few changes. We need to swap the option `fill = Month.f` in `ggplot` for `colour = Month.f`. We add the `fill = NA` to `geom_density`, and we've also added `size = 1` to make it easier to see the lines. Finally, we change the `scale_fill_brewer()` option for `scale_colour_brewer()`.

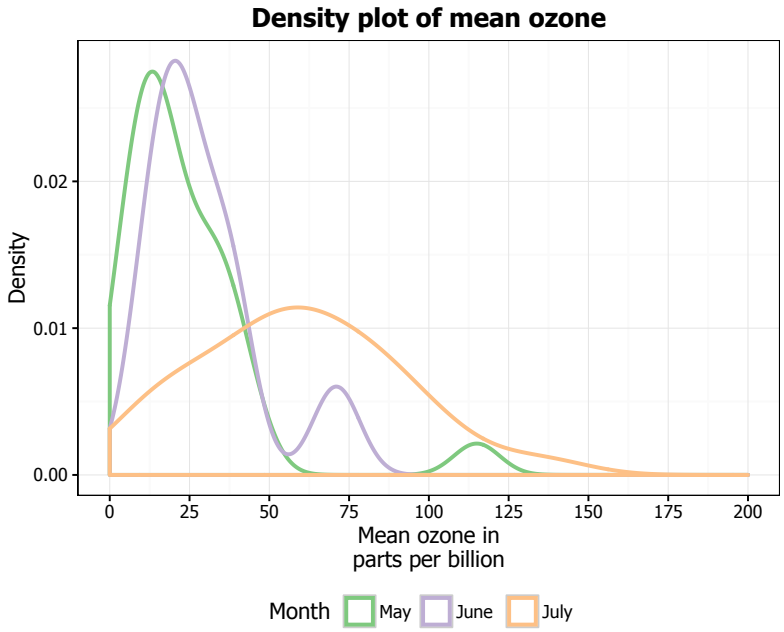
```
p8 <- ggplot(airquality_trimmed, aes(x = Ozone, colour = Month.f)) +
  geom_density(position="identity", fill = NA, size = 1) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone") +
  scale_colour_brewer(palette="Accent") +
  theme_bw() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text = element_text(size = 12, family = "Tahoma"))
p8
```



8.13. Formatting the legend

Finally, we can format the legend. Firstly, we can change the position by adding the `legend.position = "bottom"` argument to the `theme` option, which moves the legend under the plot. Secondly, we can fix the title by adding the `labs(fill="Month")` option to the plot. We now have our final graph that we showed at the beginning of this chapter.

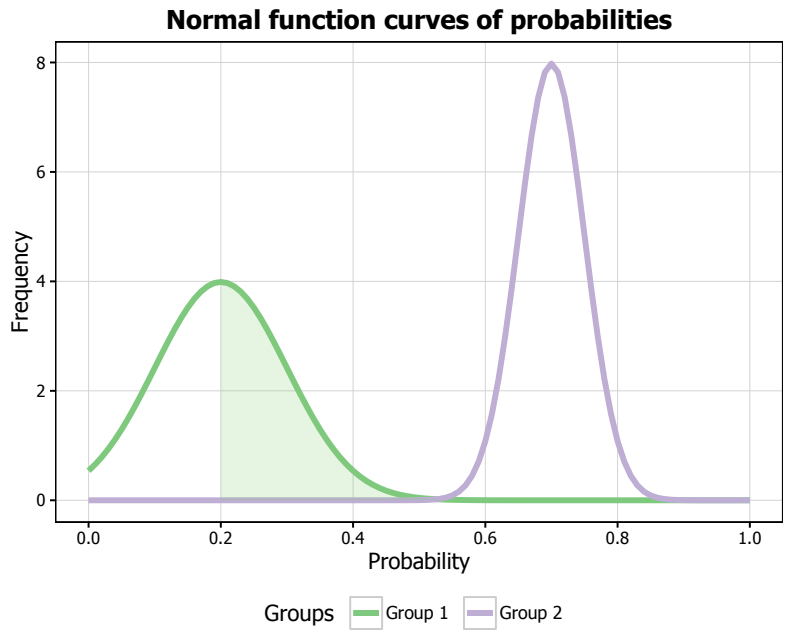
```
p8 <- ggplot(airquality_trimmed, aes(x = Ozone, colour = Month.f)) +
  geom_density(position="identity", fill = NA, size = 1) +
  scale_x_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 200, 25), limits=c(0, 200)) +
  scale_y_continuous(name = "Density") +
  ggtitle("Density plot of mean ozone") +
  scale_colour_brewer(palette="Accent") +
  labs(colour = "Month ") +
  theme_bw() +
  theme(legend.position = "bottom",
    panel.border = element_rect(colour = "black", fill=NA, size=.5),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text = element_text(size = 12, family = "Tahoma"))
p8
```


Density plot of mean ozone
Density
0.02
0.01
0.00
0
25
50
75
100
125
150
175
200
Mean ozone in
parts per billion
Month
May
June
July

CHAPTER 9

Function plots

In this chapter, we will work towards creating the function plot below. We will take you from a basic function plot and explain all the customisations we add to the code step-by-step.



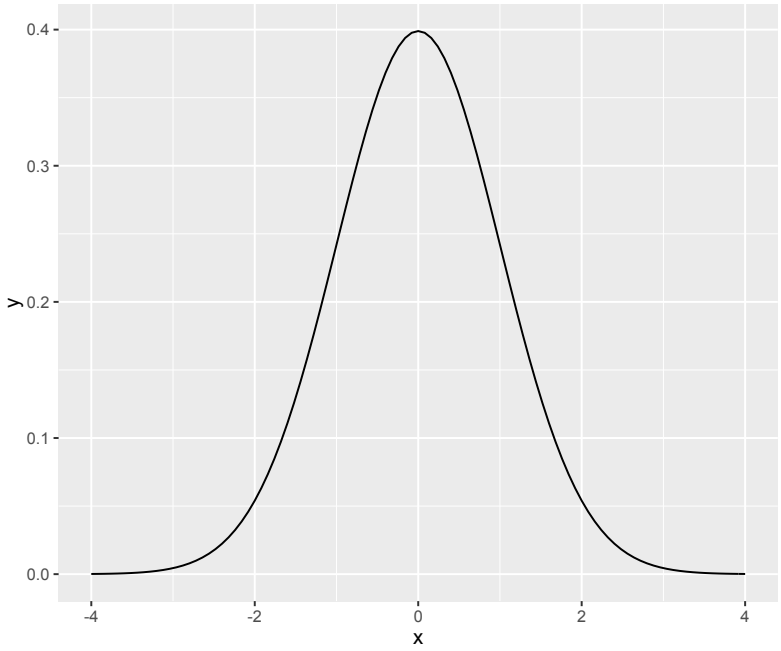
As these graphs are based on functions, there are no underlying data we are using. The first thing to do is load in the libraries, as below:

```
library(ggplot2)
library(ggthemes)
library(grid)
library(extrafont)
```

9.1. Basic normal curve

In order to create a normal curve, we create a ggplot base layer that has an x-axis range from -4 to 4 (or whatever range you want!), and assign the x-value aesthetic to this range (`aes(x = x)`). We then add the `stat_function` option and add `dnorm` to the function argument to make it a normal curve.

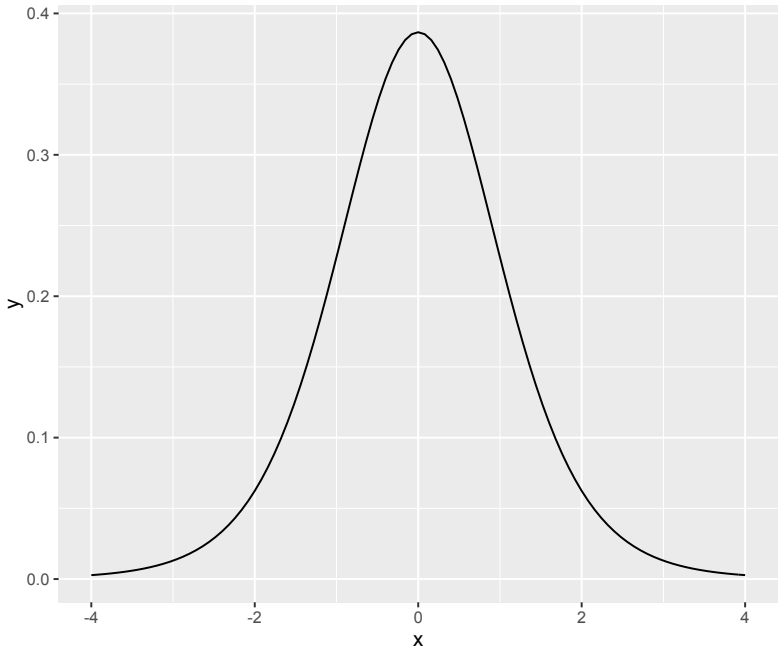
```
p9 <- ggplot(data.frame(x = c(-4, 4)), aes(x = x)) +
  stat_function(fun = dnorm)
p9
```



9.2. Basic t-curve

`stat_function` can draw a range of continuous probability density functions, including t (`dt`), F (`df`) and Chi-square (`dchisq`) PDFs. Here we will plot a t-distribution. As the shape of the t-distribution changes depending on the sample size (indicated by the degrees of freedom, or df), we need to specify our df value as part of defining our curve.

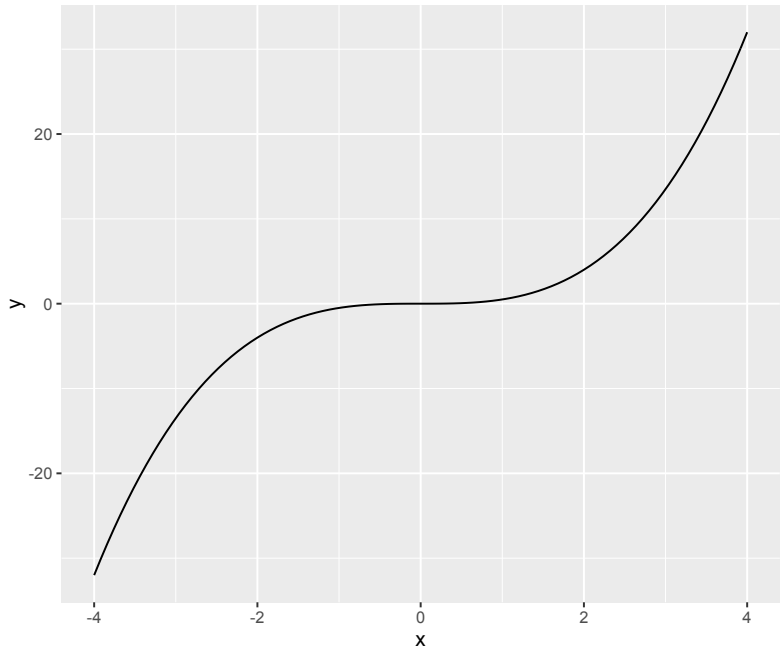
```
p9 <- ggplot(data.frame(x = c(-4, 4)), aes(x = x)) +
  stat_function(fun = dt, args = list(df = 8))
p9
```



9.3. Plotting your own function

You can also draw your own function, as long as it takes the form of a formula that converts an x-value into a y-value. Here we have plotted a curve that returns y-values that are the cube of x times a half:

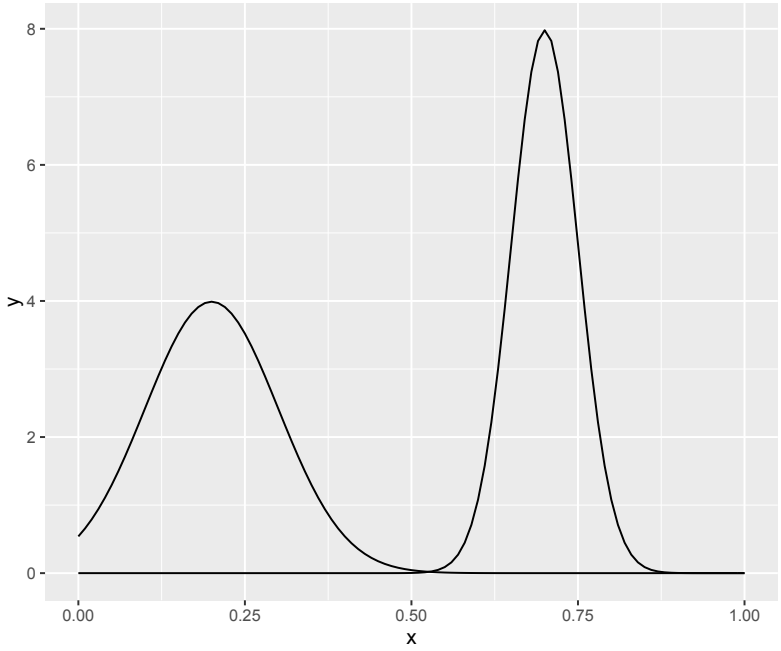
```
cubeFun <- function(x) {
    x^3 * 0.5
}

p9 <- ggplot(data.frame(x = c(-4, 4)), aes(x = x)) +
  stat_function(fun = cubeFun)
p9
```



9.4. Plotting multiple functions on the same graph

You can plot multiple functions on the same graph by simply adding another `stat_function()` for each curve. Here we have plotted two normal curves on the same graph, one with a mean of 0.2 and a standard deviation of 0.1, and one with a mean of 0.7 and a standard deviation of 0.05. (Note that the `dnorm` function has a default mean of 0 and a default standard deviation of 1, which is why we didn't need to explicitly define them in the first normal curve we plotted above.) You can also see we've changed the range of the x-axis to between 0 and 1.

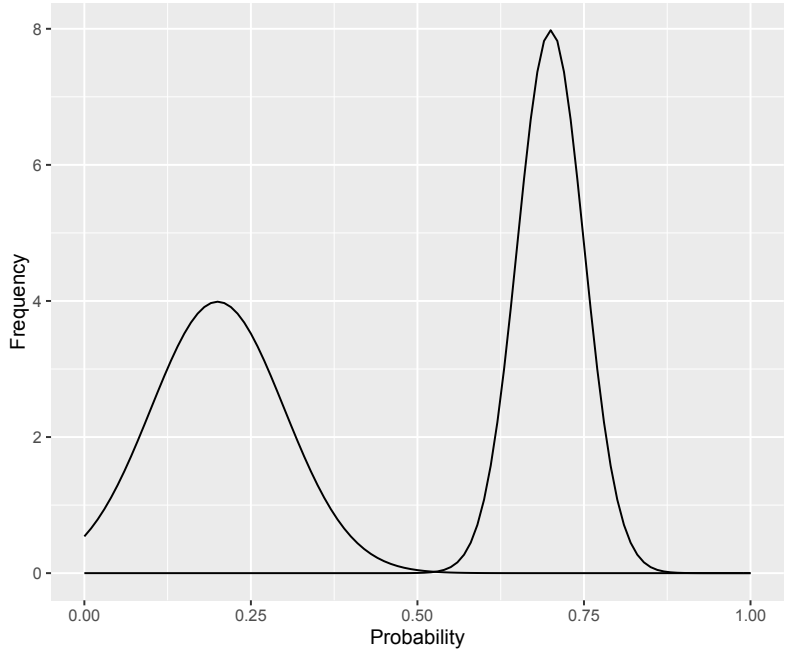
```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(0.2, 0.1)) +
  stat_function(fun = dnorm, args = list(0.7, 0.05))
p9
```



9.5. Customising axis labels

Let's move forward with this two function graph, and start tweaking the appearance. In order to change the axis labels, we have a couple of options. In this case, we have used the `scale_x_continuous` and `scale_y_continuous` options, as these have further customisation options for the axes we will use below. In each, we add the desired name to the `name` argument as a string.

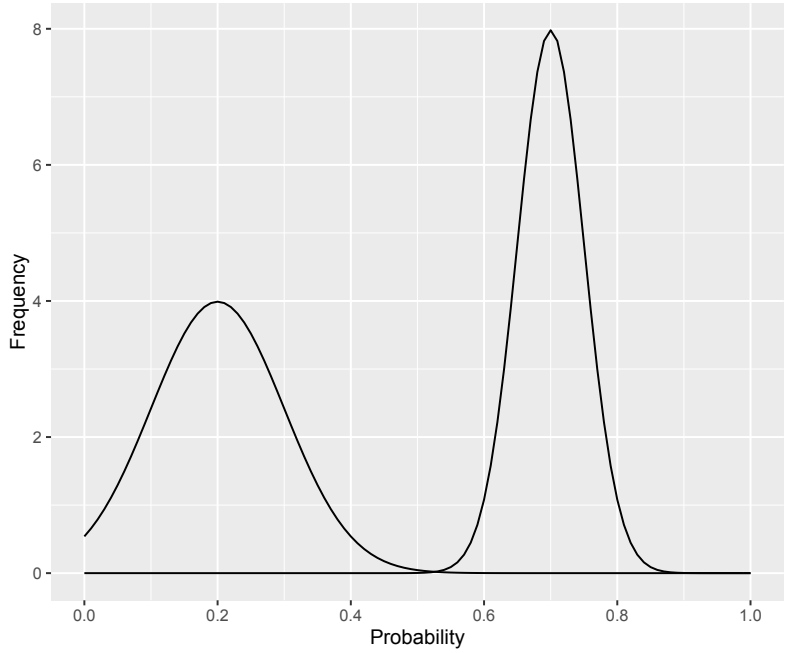
```
p9 <- p9 + scale_x_continuous(name = "Probability") +
  scale_y_continuous(name = "Frequency")
p9
```



9.6. Changing axis ticks

The next thing we will change is the axis ticks. Let's make the x-axis ticks appear at every 0.2 units rather than 0.25 using the `breaks = seq(0, 1, 0.2)` argument in `scale_x_continuous`. (The `seq` function is a base R function that indicates the start and endpoints and the units to increment by respectively. See `help(seq)` for more information.) We ensure that the x-axis begins and ends where we want by also adding the argument `limits = c(0, 1)` to `scale_x_continuous`.

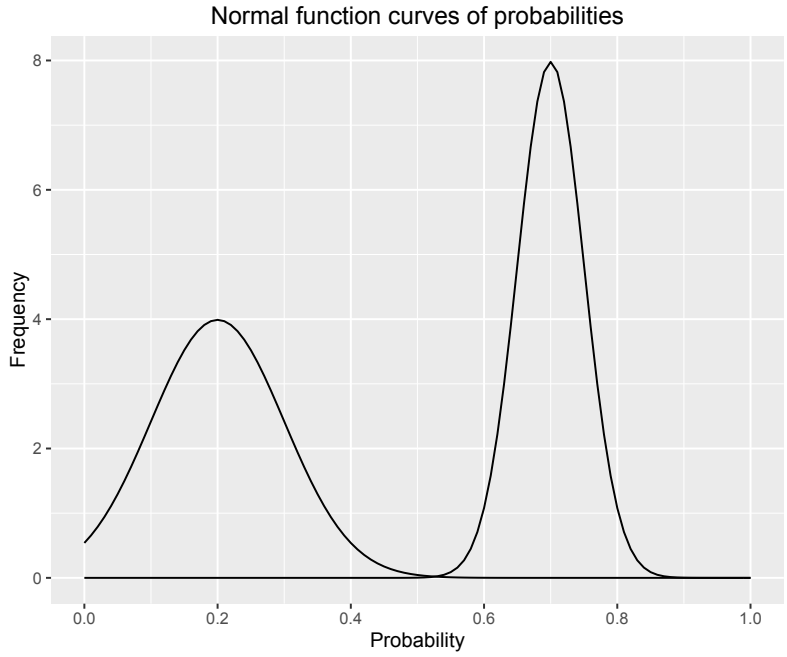
```
p9 <- p9 + scale_x_continuous(name = "Probability",
    breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
  scale_y_continuous(name = "Frequency")
p9
```



9.7. Adding a title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument.

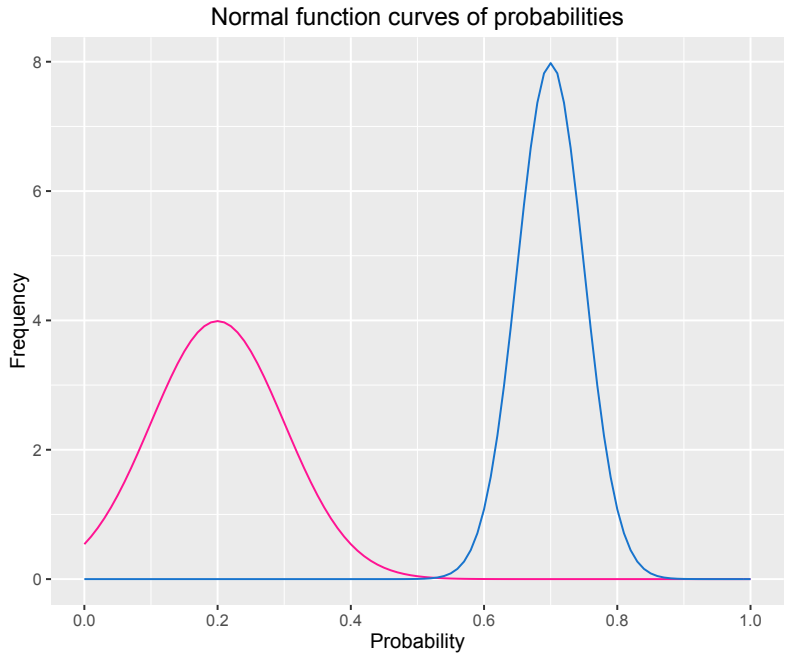
```
p9 <- p9 + ggtitle("Normal function curves of probabilities")
p9
```



9.8. Changing the colour of the curves

To change the line colours of the curves, we add a valid colour to the `colour` arguments in `stat_function`. A list of valid colours is here.

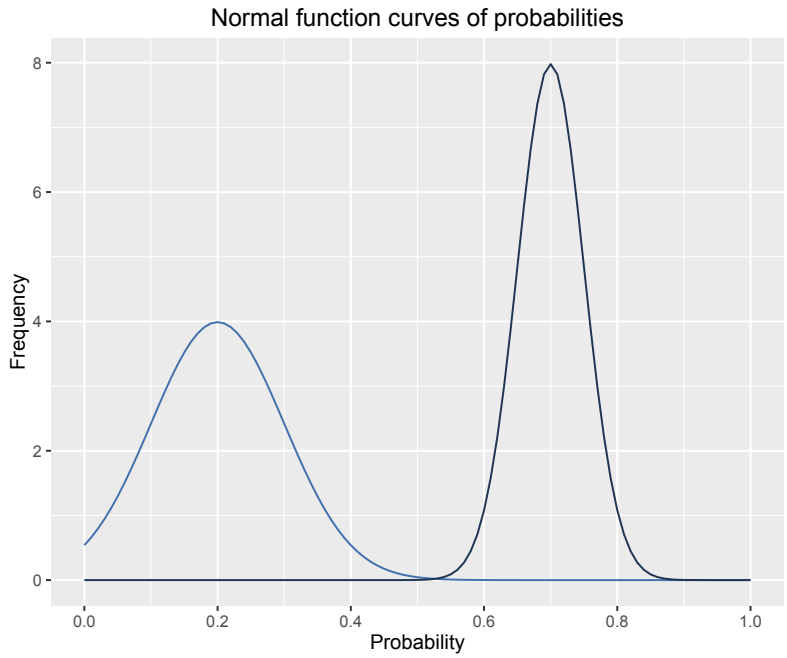
```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
    stat_function(fun = dnorm, args = list(0.2, 0.1),
      colour = "deeppink") +
    stat_function(fun = dnorm, args = list(0.7, 0.05),
      colour = "dodgerblue3") +
    scale_x_continuous(name = "Probability",
      breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
    scale_y_continuous(name = "Frequency") +
    ggtitle("Normal function curves of probabilities")
p9
```



If you want to go beyond the options in the list above, you can also specify exact HEX colours by including them as a string preceded by a hash, e.g., "#FFFFFF". Below, we have called two shades of blue for the lines using their HEX codes.

```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
    stat_function(fun = dnorm, args = list(0.2, 0.1),
      colour = "#4271AE") +
    stat_function(fun = dnorm, args = list(0.7, 0.05),
      colour = "#1F3552") +
    scale_x_continuous(name = "Probability",
```

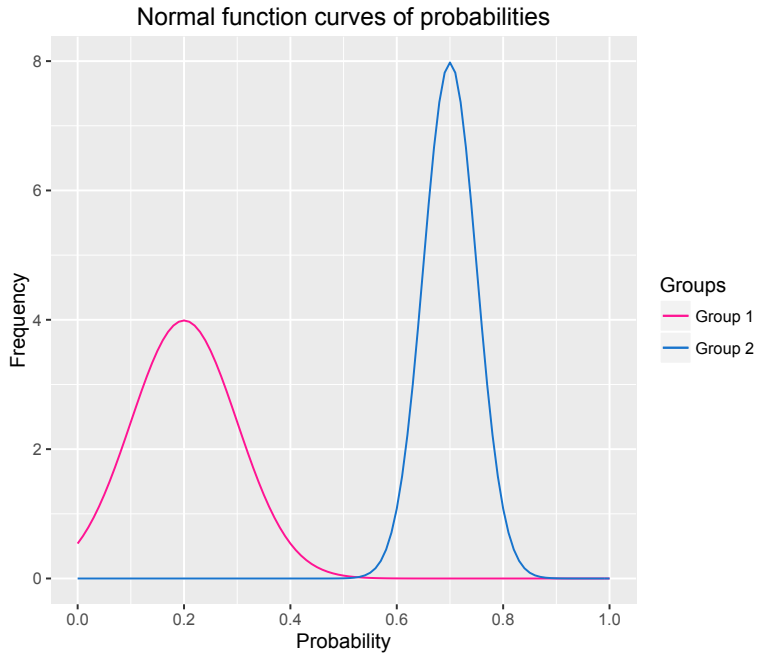
```
      breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
    scale_y_continuous(name = "Frequency") +
    ggtitle("Normal function curves of probabilities")
p9
```



9.9. Adding a legend

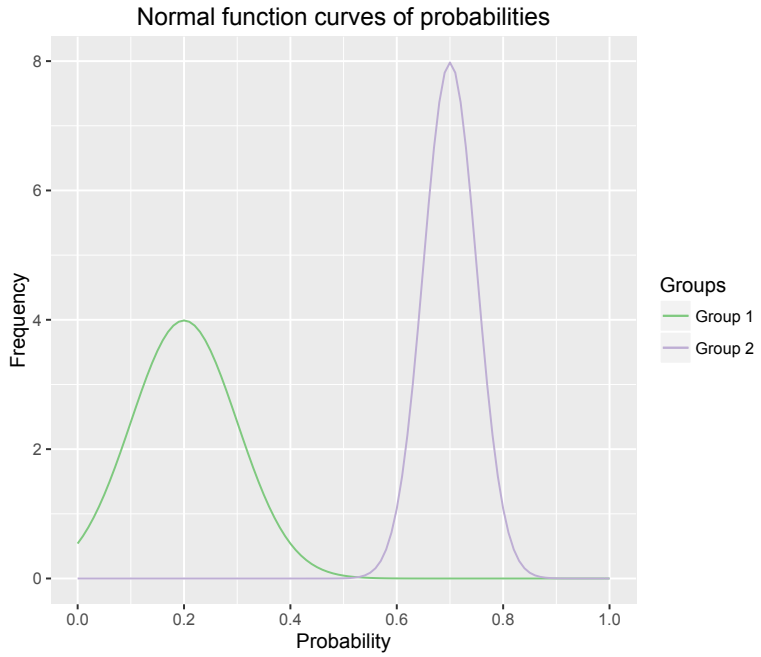
As we have added two separate commands to plot the two function curves, ggplot does not automatically recognise that it needs to create a legend. We can make a legend by swapping out the `colour` argument in each of the `stat_function` commands for `aes(colour = )`, and assigning it the name of the group. We also need to add the `scale_colour_manual` command to make the legend appear, and also assign colours and a title.

```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
    stat_function(fun = dnorm, args = list(0.2, 0.1),
      aes(colour = "Group 1 ")) +
    stat_function(fun = dnorm, args = list(0.7, 0.05),
      aes(colour = "Group 2 ")) +
    scale_x_continuous(name = "Probability",
      breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
    scale_y_continuous(name = "Frequency") +
    ggtitle("Normal function curves of probabilities") +
    scale_colour_manual("Groups ", values = c("deeppink", "dodgerblue3"))
p9
```



If you want to use one of the automatic brewer palettes, you can swap `scale_colour_manual` for `scale_colour_brewer`, and call your favouite brewer colour scheme. You can see all of the brewer palettes using `display.brewer.all(5)` As this command doesn't allow you to assign a title to the legend, you can assign a title using `labs(colour = "Groups ")`.

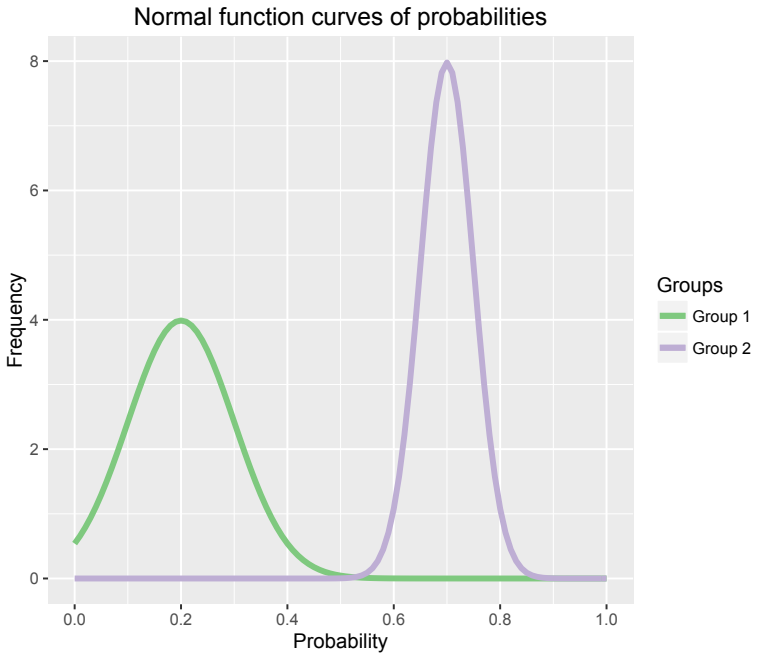
```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
    stat_function(fun = dnorm, args = list(0.2, 0.1),
      aes(colour = "Group 1 ")) +
    stat_function(fun = dnorm, args = list(0.7, 0.05),
      aes(colour = "Group 2 ")) +
    scale_x_continuous(name = "Probability",
      breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
    scale_y_continuous(name = "Frequency") +
    ggtitle("Normal function curves of probabilities") +
    scale_colour_brewer(palette="Accent") +
    labs(colour = "Groups ")
p9
```



9.10. Changing the size of the lines

As you can see, the lines are a little difficult to see. You can make them thicker (or thinner) using the argument `size` argument within `stat_function`. Here we have changed the thickness of each line to size 2.

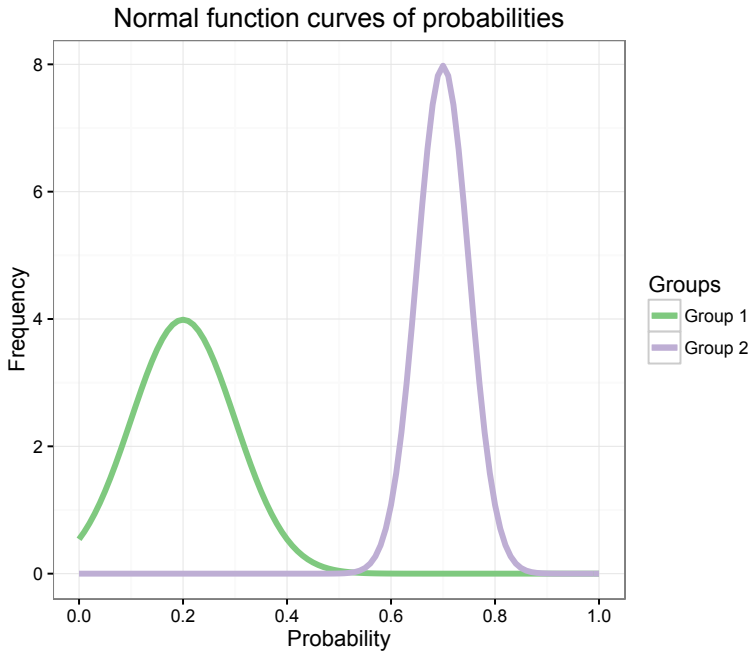
```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
    stat_function(fun = dnorm, args = list(0.2, 0.1),
      aes(colour = "Group 1 "), size = 1.5) +
    stat_function(fun = dnorm, args = list(0.7, 0.05),
      aes(colour = "Group 2 "), size = 1.5) +
    scale_x_continuous(name = "Probability",
      breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
    scale_y_continuous(name = "Frequency") +
    ggtitle("Normal function curves of probabilities") +
    scale_colour_brewer(palette="Accent") +
    labs(colour = "Groups ")
p9
```



9.11. Using the white theme

We can also change the overall look of the plot using themes. We'll start using a simple theme customisation by adding `theme_bw()` after `ggplot()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p9 <- p9 + theme_bw()
p9
```

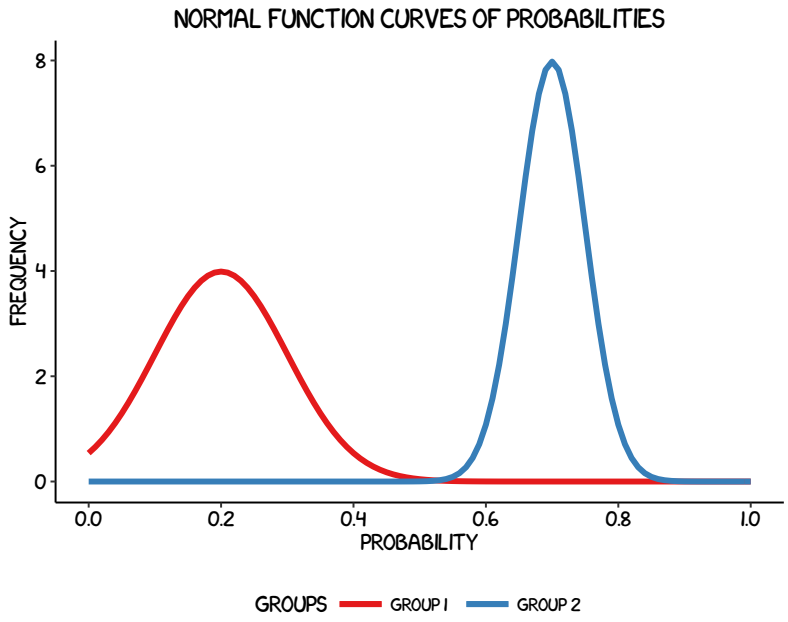


9.12. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(0.2, 0.1),
    aes(colour = "Group 1 "), size = 1.5) +
  stat_function(fun = dnorm, args = list(0.7, 0.05),
    aes(colour = "Group 2 "), size = 1.5) +
  scale_x_continuous(name = "Probability",
    breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
  scale_y_continuous(name = "Frequency") +
  ggtitle("Normal function curves of probabilities") +
  scale_colour_brewer(palette="Set1") +
  labs(colour = "Groups ") +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key = element_blank(),
```

```
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title=element_text(family="xkcd-Regular"),
    text=element_text(family="xkcd-Regular"))
p9
```

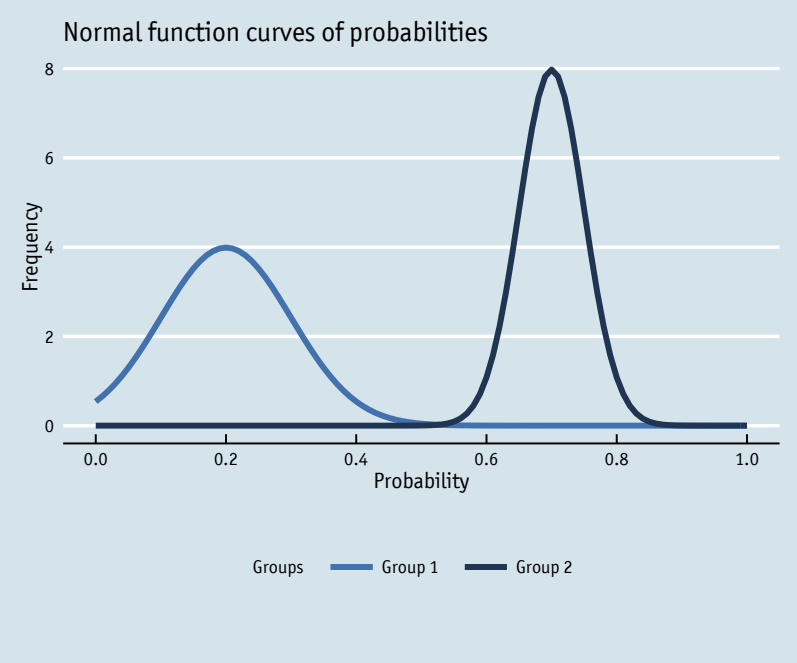


9.13. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(0.2, 0.1),
    aes(colour = "Group 1 "), size = 1.5) +
  stat_function(fun = dnorm, args = list(0.7, 0.05),
    aes(colour = "Group 2 "), size = 1.5) +
  scale_x_continuous(name = "Probability",
    breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
  scale_y_continuous(name = "Frequency") +
  ggtitle("Normal function curves of probabilities") +
```

```
  scale_colour_manual("Groups ", values = c("#4271AE", "#1F3552")) +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.title = element_text(size = 12),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.text = element_text(size = 10),
    text = element_text(family = "OfficinaSanITC-Book"),
    plot.title = element_text(family="OfficinaSanITC-Book"))
p9
```

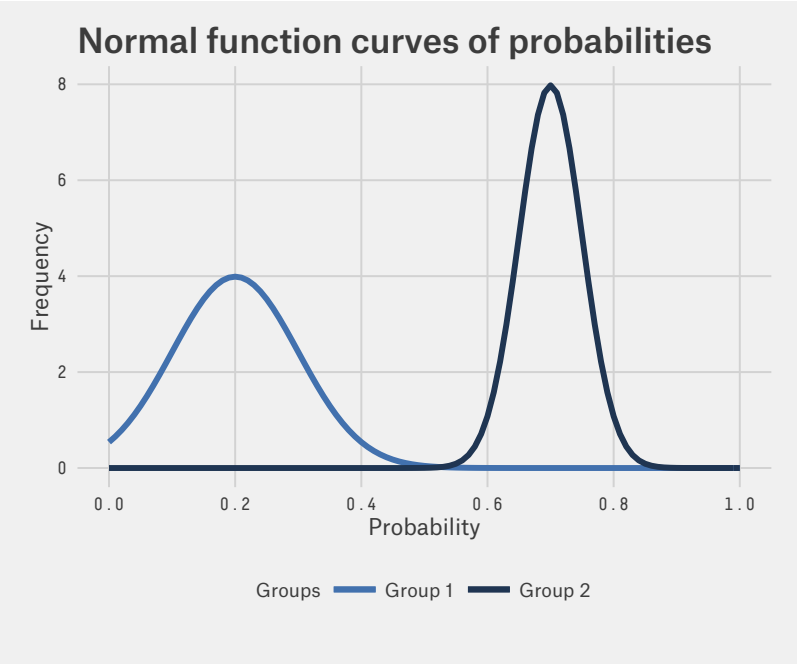


9.14. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(0.2, 0.1),
    aes(colour = "Group 1 "), size = 1.5) +
  stat_function(fun = dnorm, args = list(0.7, 0.05),
    aes(colour = "Group 2 "), size = 1.5) +
  scale_x_continuous(name = "Probability",
```

```
    breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
  scale_y_continuous(name = "Frequency") +
  ggtitle("Normal function curves of probabilities") +
  scale_colour_manual("Groups ", values = c("#4271AE", "#1F3552")) +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.title=element_text(family="Atlas Grotesk Regular", size = 10),
    legend.text=element_text(family="Atlas Grotesk Regular", size = 10),
    plot.title=element_text(family="Atlas Grotesk Medium"),
    text=element_text(family="DecimaMonoPro"))
p9
```

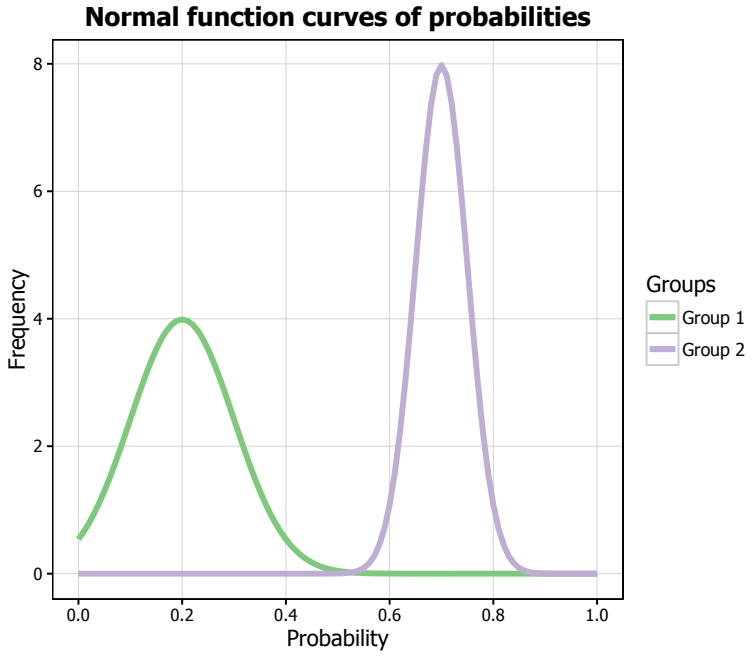


9.15. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here is a custom plot where we have modified the axes, background and font.

```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(0.2, 0.1),
    aes(colour = "Group 1 "), size = 1.5) +
  stat_function(fun = dnorm, args = list(0.7, 0.05),
    aes(colour = "Group 2 "), size = 1.5) +
```

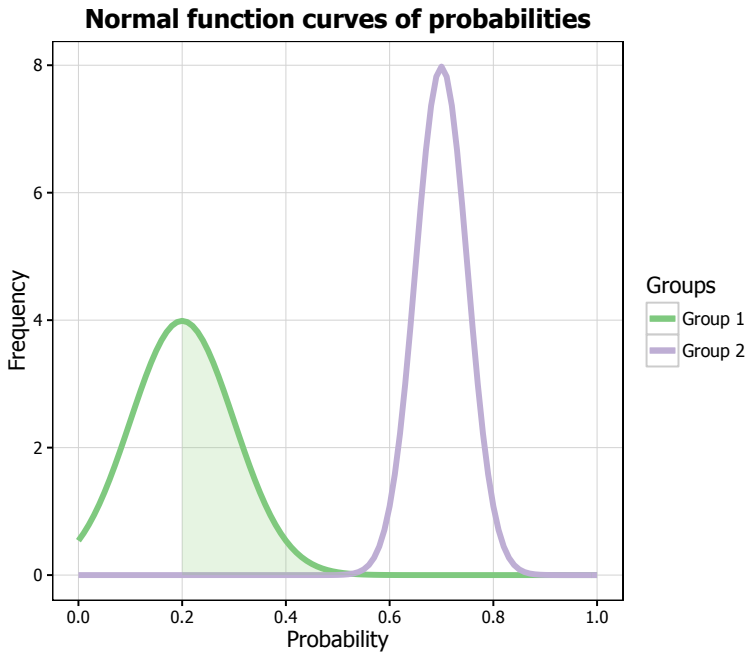
```
  scale_x_continuous(name = "Probability",
    breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
  scale_y_continuous(name = "Frequency") +
  ggtitle("Normal function curves of probabilities") +
  scale_colour_brewer(palette="Accent") +
  labs(colour = "Groups ") +
  theme_bw() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p9
```



9.16. Adding areas under the curve

If we want to shade an area under the curve, we can do so by creating a function that generates a range of normal values with a given mean and standard deviation, and then only retains those values that lie within the desired range (by assigning NAs to everything outside of the range). In this case, we have created a shaded area under the group 1 curve which covers between the mean and 4 standard deviations above the mean (as given by `0.2 + 4 * 0.1`). We then add another `stat_function` command to the graph which plots the area specified by this function, indicates it should be an `area` plot, and makes it semi-transparent using the `alpha` argument.

```
funcShaded <- function(x) {
    y <- dnorm(x, mean = 0.2, sd = 0.1)
    y[x < 0.2 | x > (0.2 + 4 * 0.1)] <- NA
    return(y)
}

p9 <- p9 + stat_function(fun=funcShaded, geom="area", fill="#84CA72", alpha=0.2)
p9
```

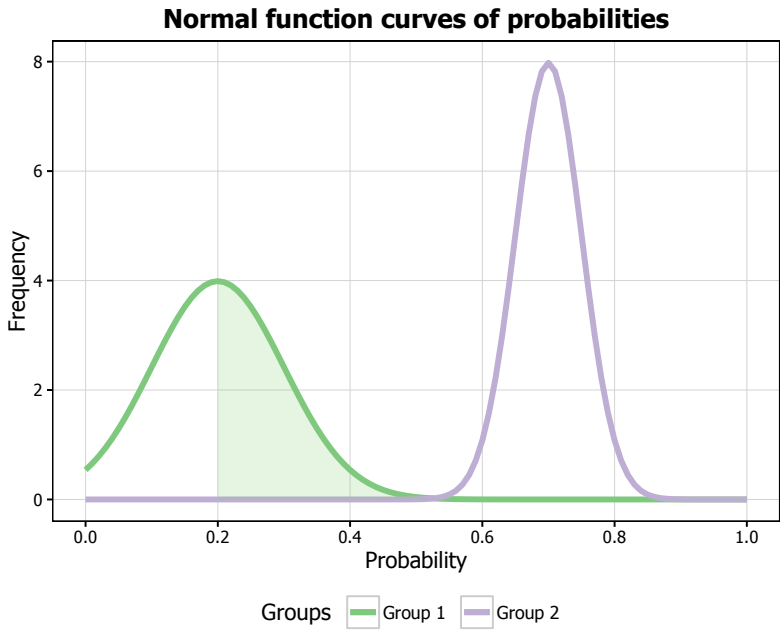


9.17. Formatting the legend

Finally, we can format the legend by changing the position. We simply add the `legend.position = "bottom"` argument to the `theme` option, which moves the legend under the plot. We now have the graph we showed at the beginning of this chapter.

```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(0.2, 0.1),
    aes(colour = "Group 1 "), size = 1.5) +
  stat_function(fun = dnorm, args = list(0.7, 0.05),
    aes(colour = "Group 2 "), size = 1.5) +
  stat_function(fun=funcShaded, geom="area", fill="#84CA72", alpha=0.2) +
  scale_x_continuous(name = "Probability",
    breaks = seq(0, 1, 0.2), limits=c(0, 1)) +
  scale_y_continuous(name = "Frequency") +
  ggtitle("Normal function curves of probabilities") +
```

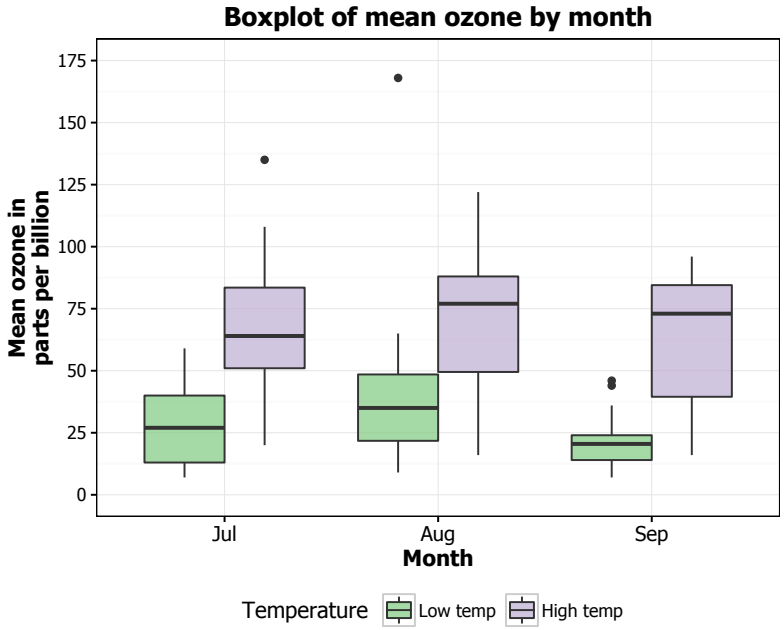
```
  scale_colour_brewer(palette="Accent") +
  labs(colour = "Groups ") +
  theme_bw() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    axis.text.x=element_text(colour="black", size = 9),
    axis.text.y=element_text(colour="black", size = 9),
    legend.position = "bottom", legend.position = "horizontal",
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"))
p9
```



CHAPTER 10

Boxplots

In this chapter, we will work towards creating the boxplot below. We will take you from a basic boxplot and explain all the customisations we add to the code step-by-step.



In this chapter, we will be using R's airquality dataset, which is contained in the `datasets` package. The first thing to do is load in the data and the libraries, as below. We'll convert `Month` into a labelled factor in order to use it as our grouping variable.

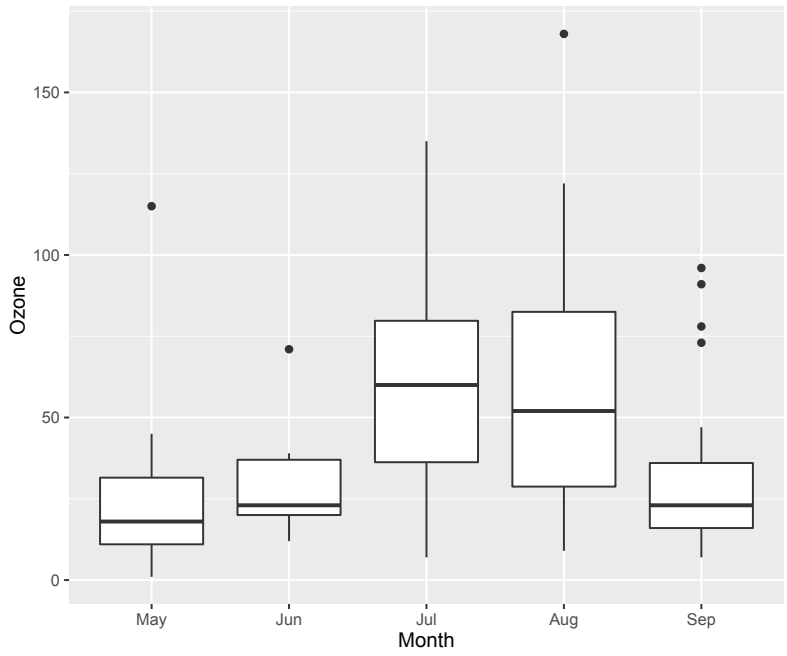
```
library(datasets)
library(ggplot2)
library(ggthemes)
library(grid)
library(RColorBrewer)
```

```
data(airquality)
airquality$Month <- factor(airquality$Month,
    labels = c("May", "Jun", "Jul", "Aug", "Sep"))
```

10.1. Basic boxplot

In order to initialise a plot we tell ggplot that `airquality` is our data, and specify that our x-axis plots the `Month` variable and our y-axis plots the `Ozone` variable. We then instruct ggplot to render this as a boxplot by adding the `geom_boxplot()` option.

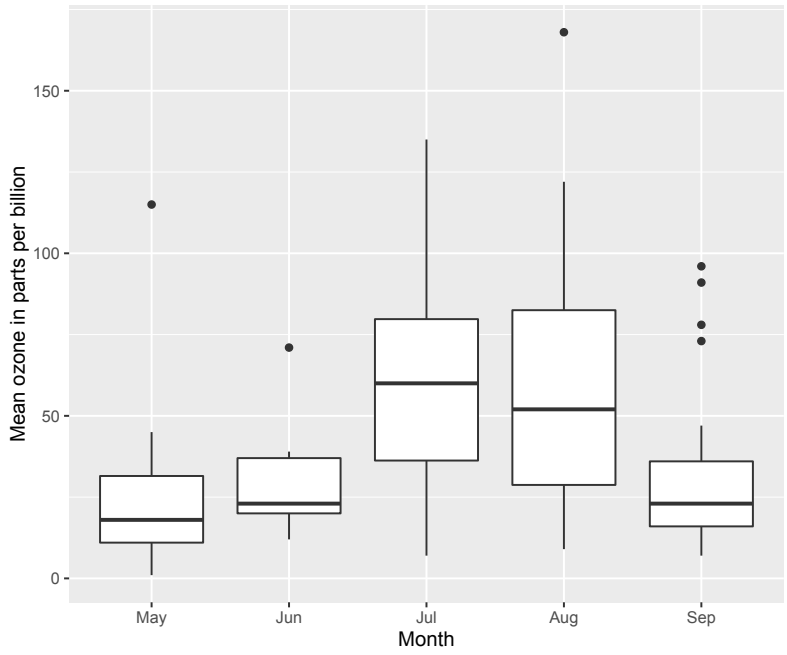
```
p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot()
p10
```



10.2. Customising axis labels

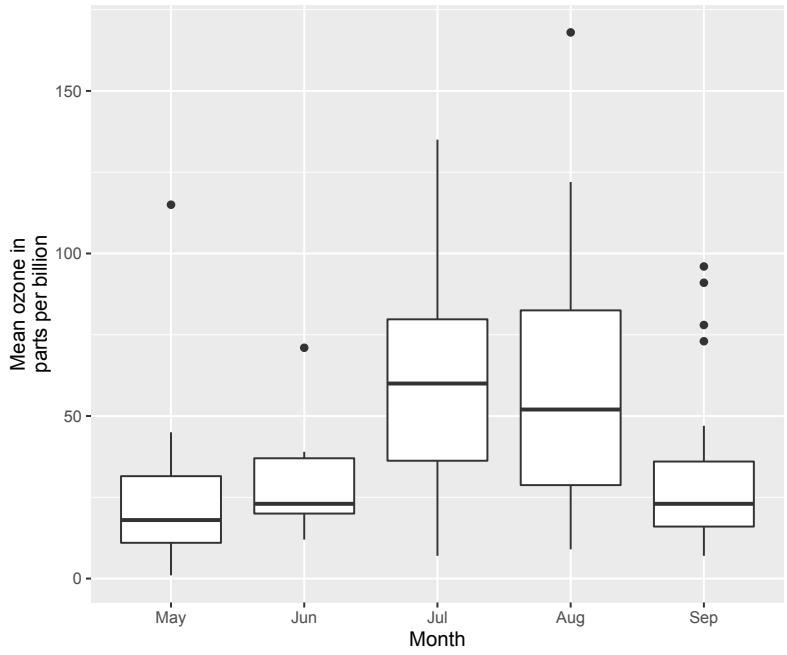
In order to change the axis labels, we have a couple of options. In this case, we have used the `scale_x_discrete` and `scale_y_continuous` options, as these have further customisation options for the axes we will use below. In each, we add the desired name to the `name` argument as a string.

```
p10 <- p10 + scale_x_discrete(name = "Month") +
  scale_y_continuous(name = "Mean ozone in parts per billion")
p10
```



ggplot also allows for the use of multiline names (in both axes and titles). Here, we've changed the y-axis label so that it goes over two lines using the `\n` character to break the line.

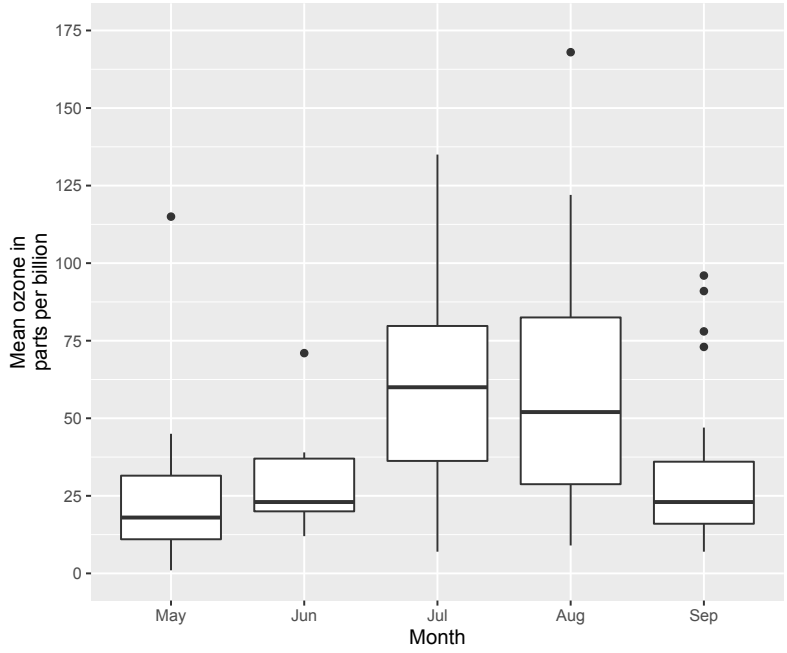
```
p10 <- p10 + scale_y_continuous(name = "Mean ozone in\nparts per billion")
p10
```



10.3. Changing axis ticks

The next thing we will change is the axis ticks. Let's make the y-axis ticks appear at every 25 units rather than 50 using the `breaks = seq(0, 175, 25)` argument in `scale_y_continuous`. (The `seq` function is a base R function that indicates the start and endpoints and the units to increment by respectively. See `help(seq)` for more information.) We ensure that the y-axis begins and ends where we want by also adding the argument `limits = c(0, 175)` to `scale_y_continuous`.

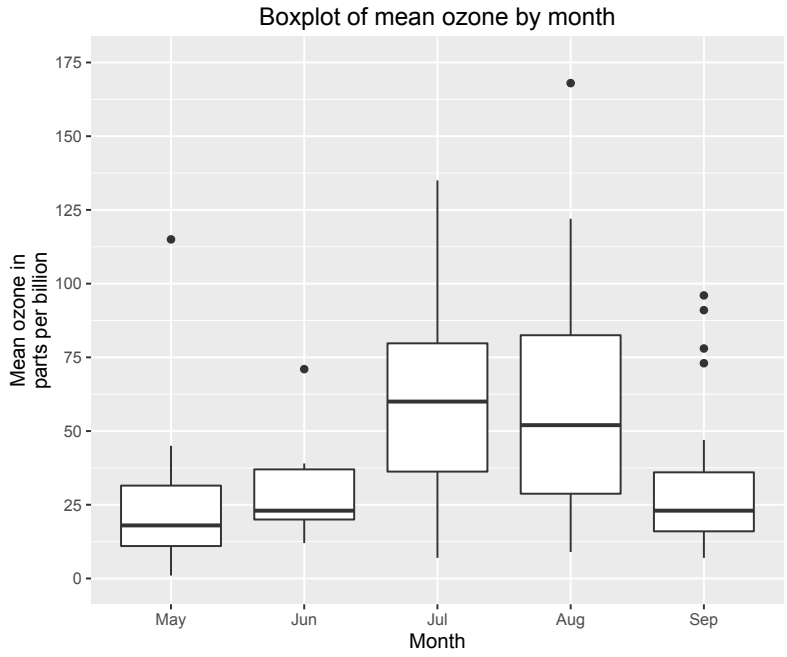
```
p10 <- p10 + scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175))
p10
```



10.4. Adding a title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument.

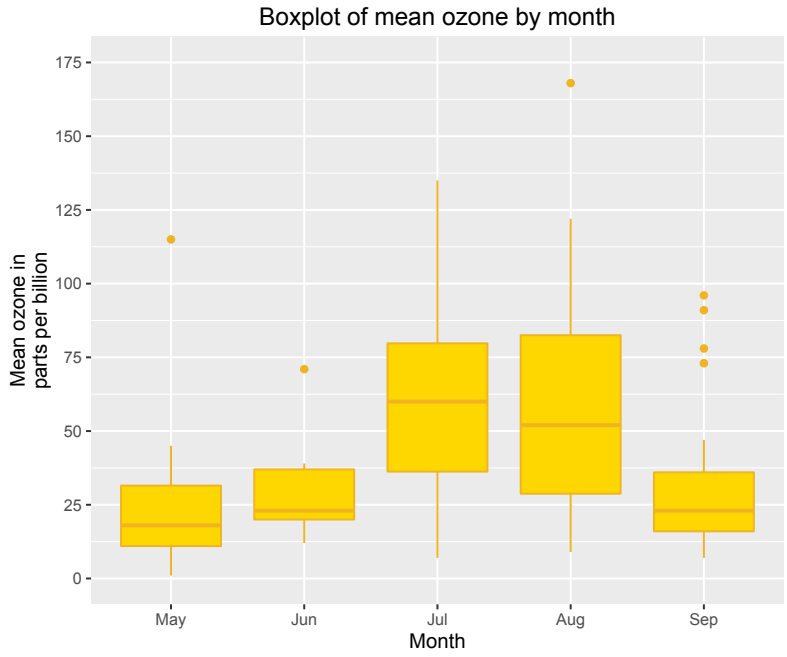
```
p10 <- p10 + ggtitle("Boxplot of mean ozone by month")
p10
```



10.5. Changing the colour of the boxes

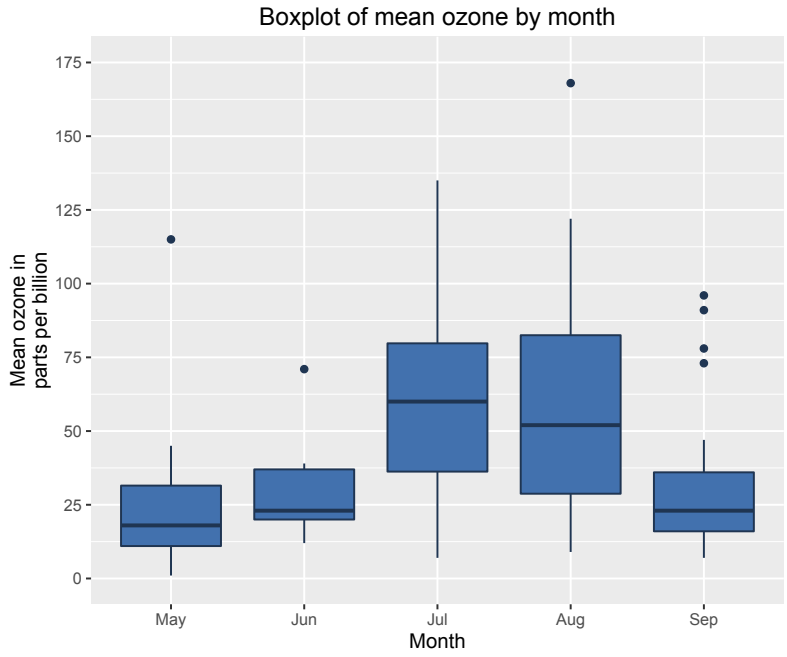
To change the line and fill colours of the box plot, we add a valid colour to the `colour` and `fill` arguments in `geom_boxplot()` (note that we assigned these colours to variables outside of the plot to make it easier to change them). A list of valid colours is here.

```
fill <- "gold1"; line <- "goldenrod2"

p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot(fill = fill, colour = line) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month")
p10
```



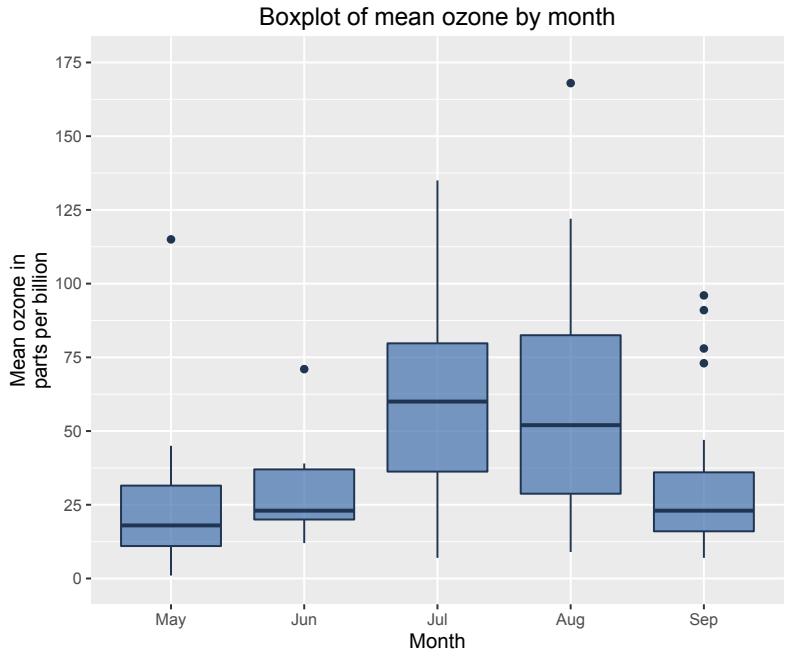
If you want to go beyond the options in the list above, you can also specify exact HEX colours by including them as a string preceded by a hash, e.g., "#FFFFFF". Below, we have called two shades of blue for the fill and lines using their HEX codes.

```
fill <- "#4271AE"; line <- "#1F3552"

p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot(fill = fill, colour = line) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month")
p10
```



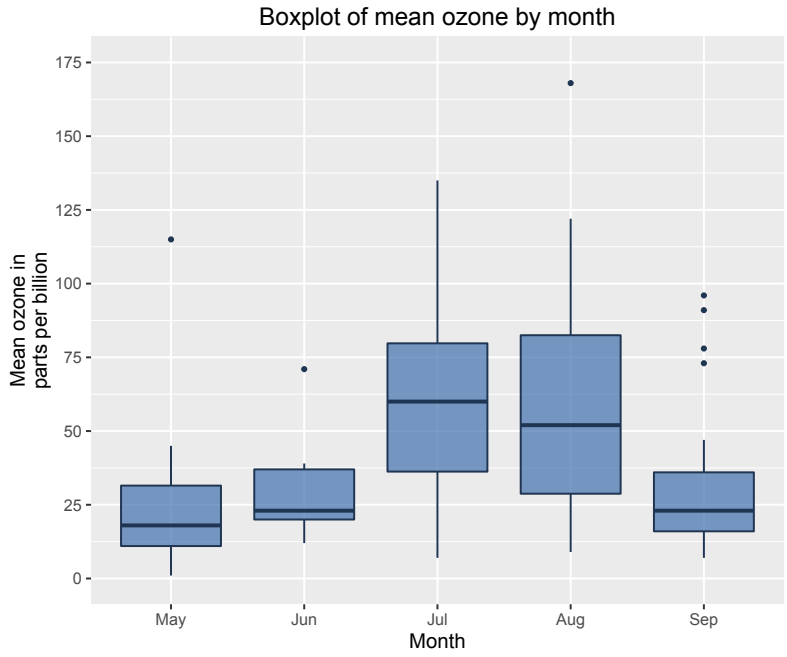
You can also specify the degree of transparency in the box fill area using the argument `alpha` in `geom_boxplot`. This ranges from 0 to 1.

```
p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot(fill = fill, colour = line, alpha = 0.7) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month")
p10
```



Finally, you can change the appearance of the outliers as well, using the arguments `outlier.colour` and `outlier.shape` in `geom_boxplot` to change the colour and shape respectively. An explanation of the allowed arguments for shape are described in this article, although be aware that because there is no "fill" argument for outlier, you cannot create circles with separate outline and fill colours. Here we will make the outliers small solid circles (using `outlier.shape = 20`) and make them the same colour as the box lines (using `outlier.colour = "#1F3552"`).

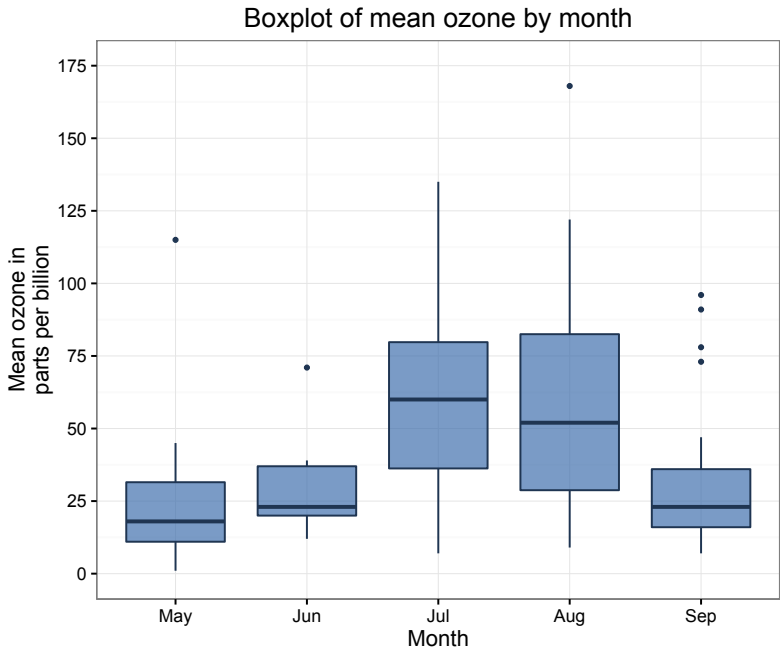
```
p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot(fill = fill, colour = line, alpha = 0.7,
    outlier.colour = "#1F3552", outlier.shape = 20) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month")
p10
```



10.6. Using the white theme

We can also change the overall look of the plot using themes. We'll start using a simple theme customisation by adding `theme_bw()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p10 <- p10 + theme_bw()
p10
```

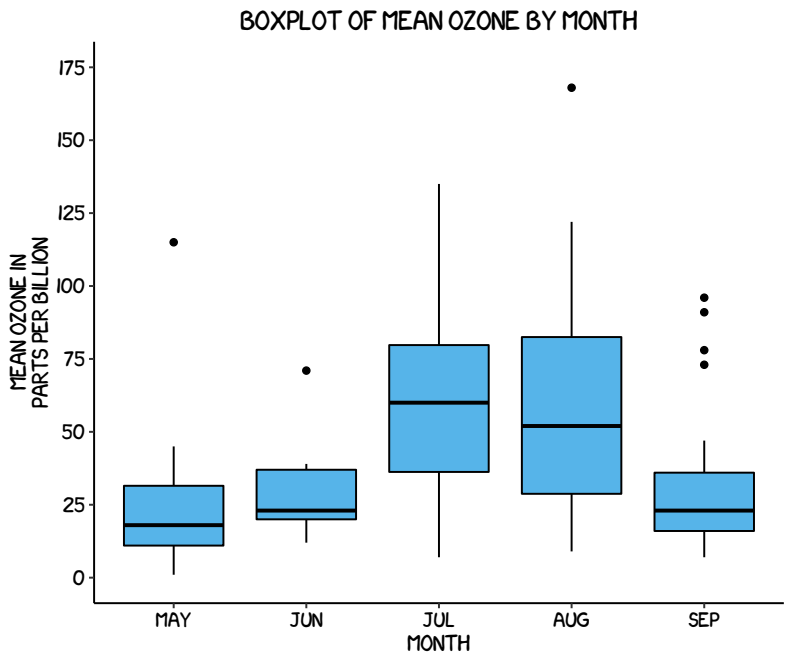


10.7. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot(colour = "black", fill = "#56B4E9") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month") +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.line.y = element_line(size=.5, colour = "black"),
    axis.text.x=element_text(colour="black", size = 10),
    axis.text.y=element_text(colour="black", size = 10),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.key = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    plot.title=element_text(family="xkcd-Regular"),
```

```
    text=element_text(family="xkcd-Regular"))
p10
```

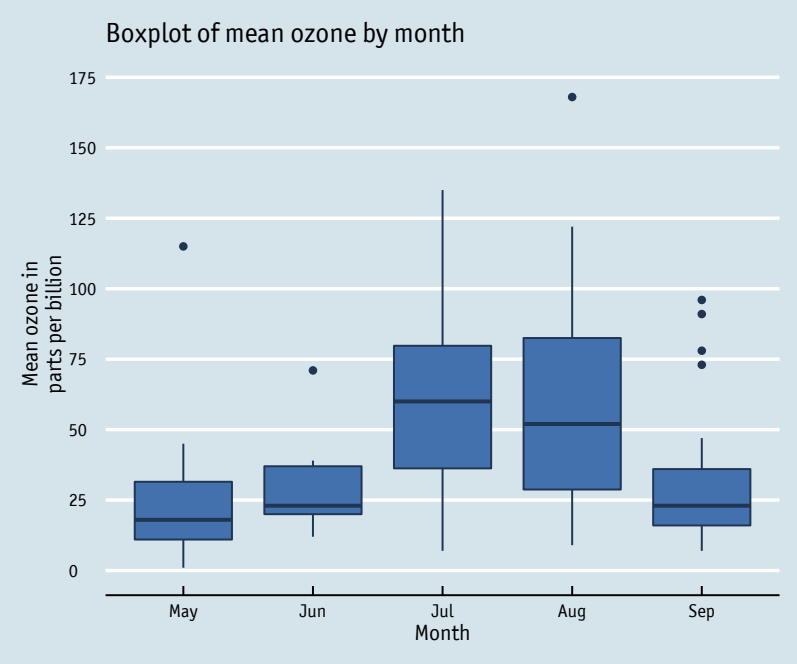


10.8. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

```
p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot(fill = fill, colour = line) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month") +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
    axis.title = element_text(size = 12),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.text = element_text(size = 10),
    text = element_text(family = "OfficinaSanITC-Book"),
```

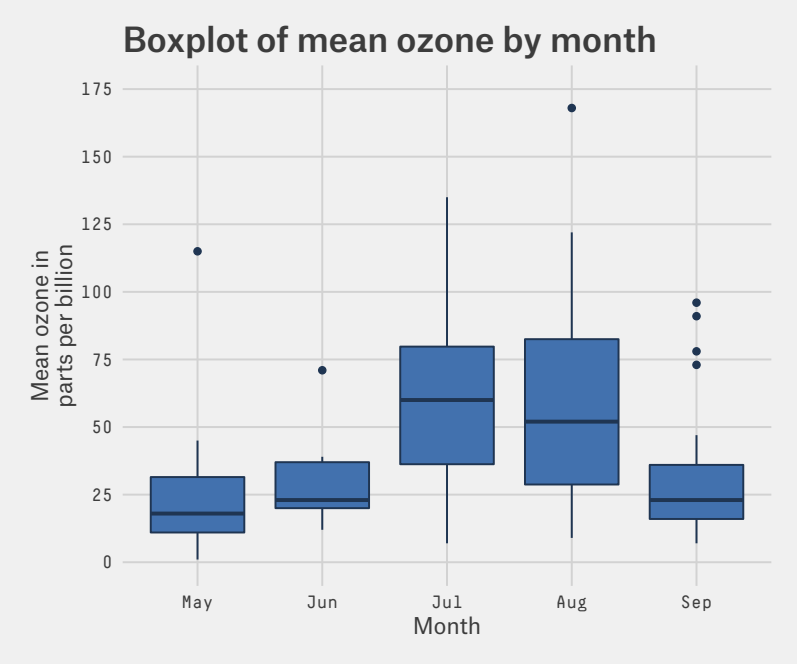
```
    plot.title = element_text(family="OfficinaSanITC-Book"))
p10
```



10.9. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

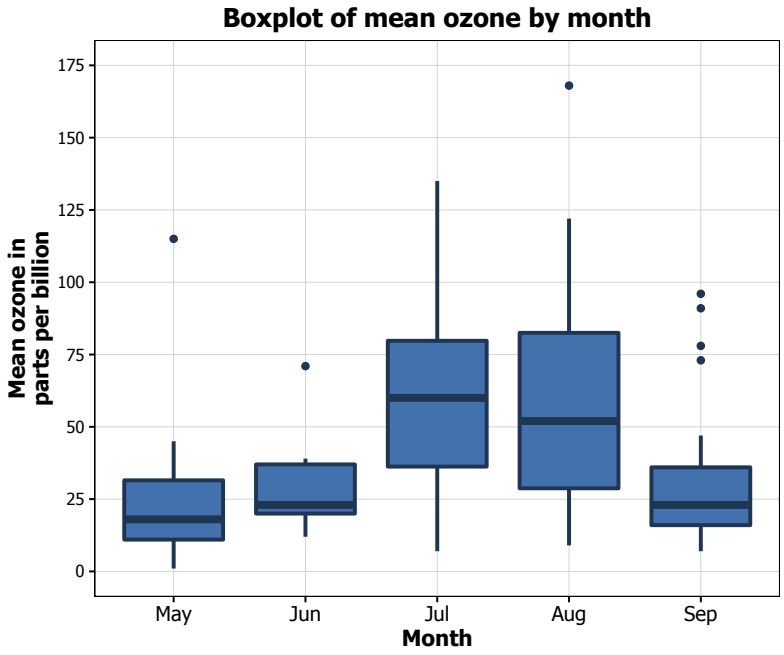
```
p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot(fill = fill, colour = line) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month") +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
    legend.position="bottom",
    legend.direction="horizontal",
    legend.box = "horizontal",
    legend.title=element_text(family="Atlas Grotesk Regular", size = 10),
    legend.text=element_text(family="Atlas Grotesk Regular", size = 10),
    plot.title=element_text(family="Atlas Grotesk Medium"),
    text=element_text(family="DecimaMonoPro"))
p10
```



10.10. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here is a custom plot where we have modified the axes, background and font.

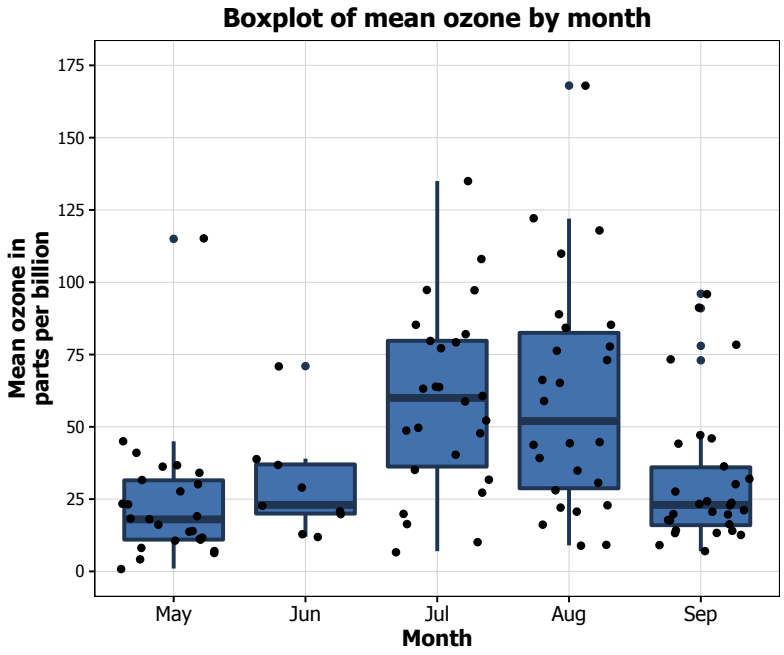
```
fill <- "#4271AE"; lines <- "#1F3552"

p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot(colour = lines, fill = fill, size = 1) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month") +
  theme_bw() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family = "Tahoma"),
    axis.title = element_text(face="bold"),
    axis.text.x = element_text(colour="black", size = 11),
    axis.text.y = element_text(colour="black", size = 9))
p10
```



10.11. Boxplot extras

An extra feature you can add to boxplots is to overlay all of the points for that group on each boxplot in order to get an idea of the sample size of the group. This can be achieved using by adding the `geom_jitter` option.

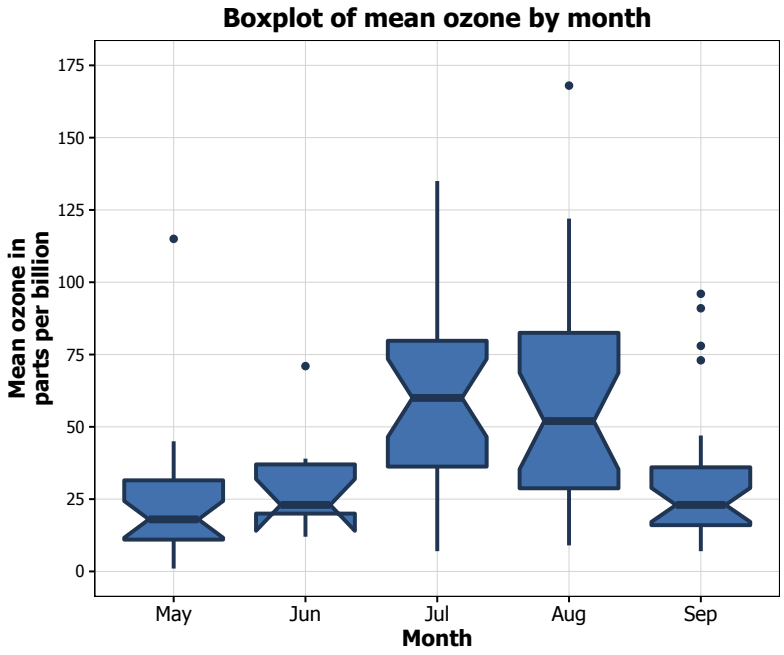
```
p10 <- p10 + geom_jitter()
p10
```



We can see that June has a pretty small sample, indicating that information based on this group may not be very reliable.

Another thing you can do with your boxplot is add a notch to the box where the median sits to give a clearer visual indication of how the data are distributed within the IQR. You achieve this by adding the argument `notch = TRUE` to the `geom_boxplot` option. You can see on our graph that the box for June looks a bit weird due to the very small gap between the 25th percentile and the median.

```
p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
  geom_boxplot(colour = lines, fill = fill, size = 1, notch = TRUE) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month") +
  theme_bw() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
    panel.grid.major = element_line(colour = "#d3d3d3"),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(), panel.background = element_blank(),
    plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    text=element_text(family="Tahoma"),
    axis.title = element_text(face="bold"),
    axis.text.x=element_text(colour="black", size = 11),
    axis.text.y=element_text(colour="black", size = 9))
p10
```



10.12. Grouping by another variable

You can also easily group box plots by the levels of another variable. There are two options, in separate (panel) plots, or in the same plot.

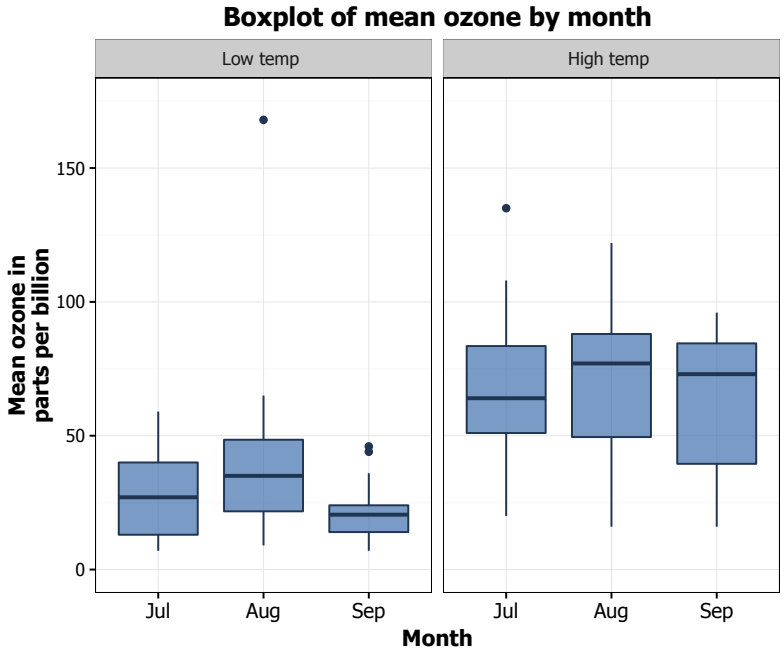
10.12.1. In panel plots

We first need to do a little data wrangling. In order to make the graphs a bit clearer, we've kept only months "July", "Aug" and "Sep" in a new dataset `airquality_trimmed`. We've also mean-split `Temp` so that this is also categorical, and made it into a new labelled factor variable called `Temp.f`.

In order to produce a panel plot by Temperature , we add the `facet_grid(. ~ Temp.f)` option to the plot.

```
airquality_trimmed <- airquality[which(airquality$Month == "Jul" |
    airquality$Month == "Aug" | airquality$Month == "Sep"), ]
airquality_trimmed$Temp.f <- factor(ifelse(airquality_trimmed$Temp >
    mean(airquality_trimmed$Temp), 1, 0), labels = c("Low temp ", "High temp "))

p10 <- ggplot(airquality_trimmed, aes(x = Month, y = Ozone)) +
  geom_boxplot(fill = fill, colour = line,
    alpha = 0.7) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 50), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month") +
```

```
  theme_bw() +
  theme(plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    panel.border = element_rect(colour = "black", fill=NA, size=.5),
    text = element_text(size = 12, family = "Tahoma"),
    axis.title = element_text(face="bold"),
    axis.text.x=element_text(size = 11)) +
  facet_grid(. ~ Temp.f)
p10
```

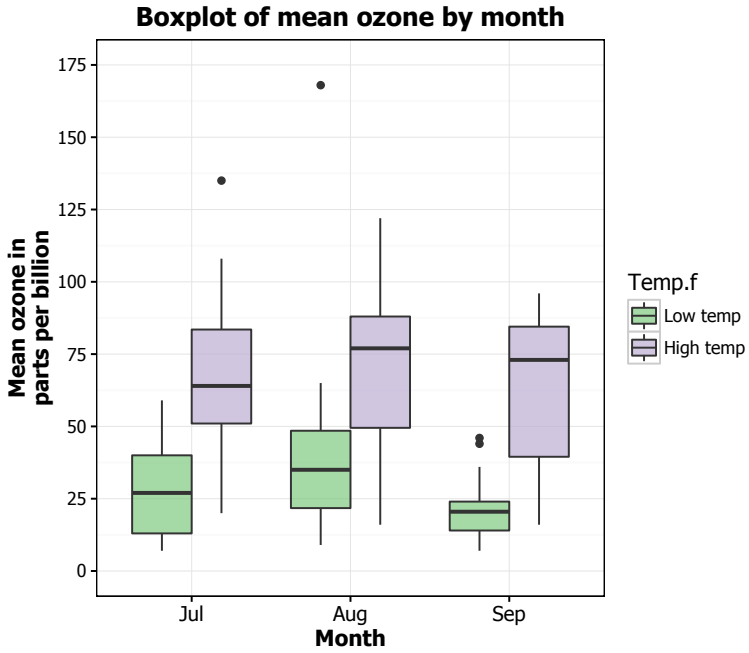


10.12.2. In the same plot

In order to plot the two Temperature levels in the same plot, we need to add a couple of things. Firstly, in the `ggplot` function, we add a `fill = Temp.f` argument to `aes`. Secondly, we customise the colours of the boxes by adding the `scale_fill_brewer` to the plot from the `RColorBrewer` package. This blog post describes the available packages.

```
p10 <- ggplot(airquality_trimmed, aes(x = Month, y = Ozone, fill = Temp.f)) +
  geom_boxplot(alpha=0.7) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month") +
  theme_bw() +
  theme(plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    panel.border = element_rect(colour = "black", fill=NA, size=.5),
    text = element_text(size = 12, family = "Tahoma"),
```

```
    axis.title = element_text(face="bold"),
    axis.text.x=element_text(size = 11)) +
  scale_fill_brewer(palette = "Accent")
p10
```

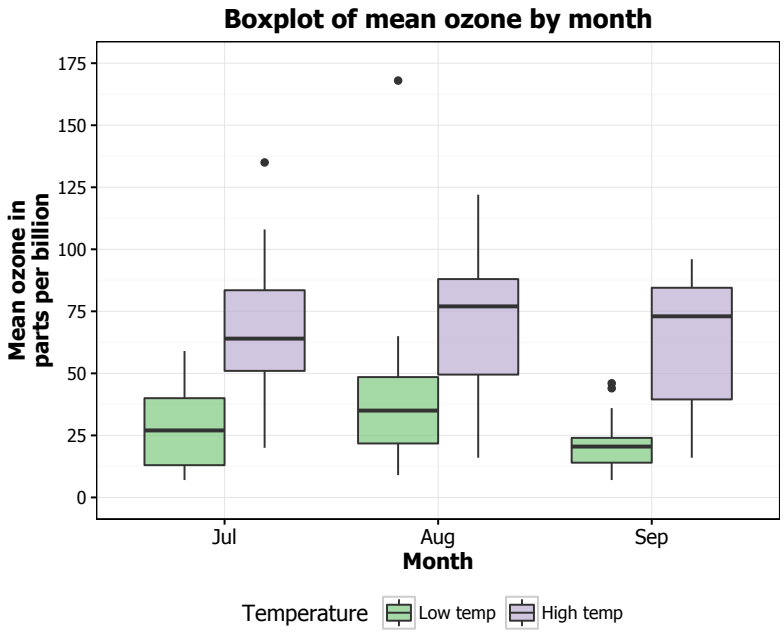


10.13. Formatting the legend

Finally, we can format the legend. Firstly, we can change the position by adding the `legend.position = "bottom"` argument to the `theme` option, which moves the legend under the plot. Secondly, we can fix the title by adding the `labs(fill = "Temperature ")` option to the plot. We now have our final graph that we showed at the beginning of this chapter.

```
p10 <- ggplot(airquality_trimmed, aes(x = Month, y = Ozone, fill = Temp.f)) +
  geom_boxplot(alpha=0.7) +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
    breaks = seq(0, 175, 25), limits=c(0, 175)) +
  scale_x_discrete(name = "Month") +
  ggtitle("Boxplot of mean ozone by month") +
  theme_bw() +
  theme(plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
    panel.border = element_rect(colour = "black", fill=NA, size=.5),
    text = element_text(size = 12, family = "Tahoma"),
    axis.title = element_text(face="bold"),
    axis.text.x=element_text(size = 11),
    legend.position = "bottom") +
```

```
  scale_fill_brewer(palette = "Accent") +
  labs(fill = "Temperature ")
p10
```



CHAPTER 11

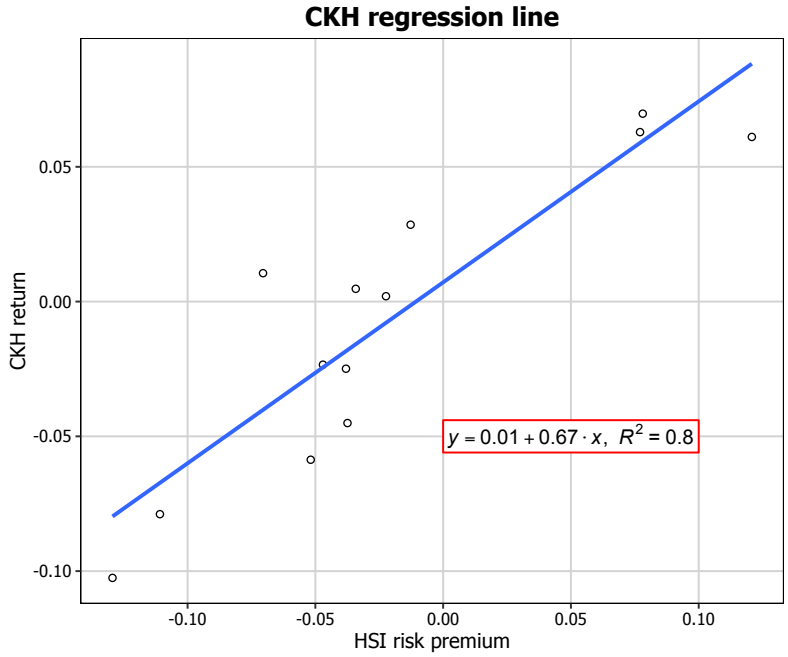
Linear regression plots

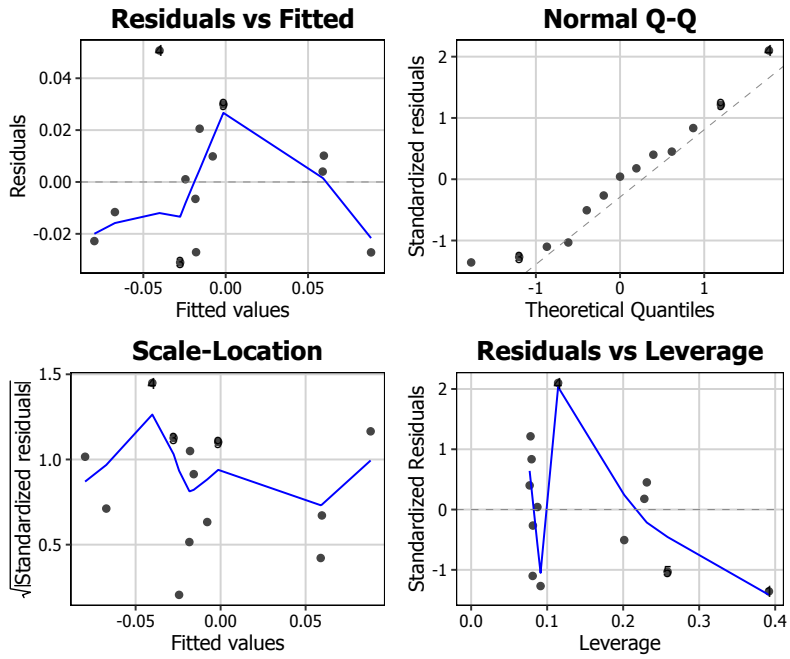
This chapter will be much more than showing you how to create regression plots. Here we are extracting, cleaning and processing financial data from Quandl.

Before going ahead, we strongly suggest that you create a Quandl account in order to obtain an API key that allows you to download their data without restrictions. It is even possible to log into Quandl using Github or Linkedin accounts. Quandl's website has complete instructions and they have an API that is 100% R compatible.

The goal is to estimate the CAPM model $R_i = R_f + \beta_i[R_m - R_f] + e_i$ where R_i is the return of an asset, R_f is the risk-free return (e.g. US Treasury Bonds), R_m is the return of the market portfolio (e.g. NYSE) and β_i is a measure of risk relative to the market (e.g. $\beta_i = 1$ means that the asset is exactly as risky as the market portfolio). More on the CAPM model can be read here, but in this chapter we will focus on plotting, so don't worry if you don't understand all of the details!

In this chapter, we will work towards creating the trend line and diagnostics plots below. We will take you from a basic regression plot and explain all the customisations we add to the code step-by-step.





The first thing to do is download and load in the libraries and the data of the monthly price of Hang Seng Index and Cheung Kong Holdings Hong Kong from 2015-03-01 to 2016-04-01.

```
library(ggplot2)
library(ggthemes)
library(grid)
library(ggfortify)
library(Quandl)
```

```
Quandl.api_key("XXX")

hsi.df <- Quandl("YAHOO/INDEX_HSI", start_date="2015-03-01", end_date="2016-04-01",
     collapse="monthly", type = "raw")

ckh.df <- Quandl("YAHOO/HK_0001", start_date="2015-03-01",
     end_date="2016-04-01", collapse="monthly", type = "raw")

saveRDS(hsi.df, "hsi.rds"); saveRDS(ckh.df,"ckh.rds")
```

Before calculating return as $R_i = \frac{P_t - P_{t-1}}{P_t}$ we need to order HSI and CKH data by dates and in decreasing order.

```
hsi.df <- readRDS("hsi.rds")
colnames(hsi.df)[7] <- "Adjusted.Close"
hsi.df <- hsi.df[order(as.Date(hsi.df$Date)),]
```

With ordered dates it is possible to obtain the correct return for each month.

```
hsi.Adjusted.Close <- hsi.df$Adjusted.Close
hsi.Return <- diff(hsi.Adjusted.Close)/
  hsi.Adjusted.Close[-length(hsi.Adjusted.Close)]
hsi.Return <- c(NA,hsi.Return)
hsi.df$Return <- hsi.Return
hsi.df <- na.omit(hsi.df)
hsi.Return <- hsi.df[,c("Date","Return")]

ckh.df <- readRDS("ckh.rds")
colnames(ckh.df)[7] <- "Adjusted.Close"
ckh.df <- ckh.df[order(as.Date(ckh.df$Date)),]
ckh.Adjusted.Close <- ckh.df$Adjusted.Close
ckh.Return <- diff(ckh.Adjusted.Close)/
  ckh.Adjusted.Close[-length(ckh.Adjusted.Close)]
ckh.Return <- c(NA,ckh.Return)
ckh.df <- na.omit(ckh.df)
ckh.df$Return <- ckh.Return
ckh.Return <- ckh.df[,c("Date","Return")]
```

The returns can be arranged in one data frame before doing plots and regression.

```
hsi.ckh.returns <- merge(hsi.Return, ckh.Return, by='Date')
hsi.ckh.returns <- na.omit(hsi.ckh.returns)
colnames(hsi.ckh.returns) <- c("Date","hsi.Return","ckh.Return")
```

Using Damodaran and Bloomberg data we can work with an estimate of HSI risk premium over risk-free rate.

```
usa.risk.free <- 0.3/100
hsi.risk.premium <- 0.6/100
```

11.1. Trend line plot

11.1.1. Basic trend line plot

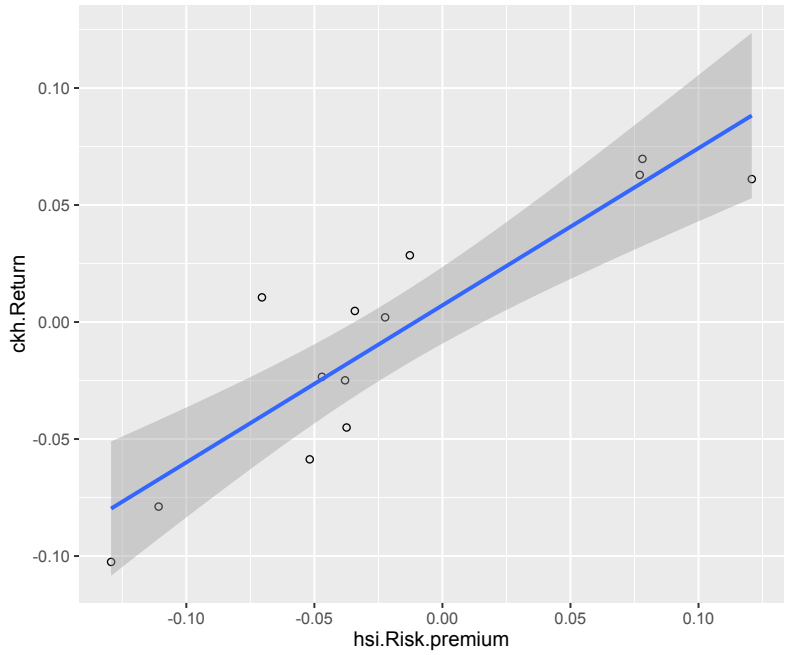
Now we can fit a linear regression. One interesting thing is that in CAPM context the regression line slope can be calculated as $\beta_i = \frac{\sigma_{i,m}}{\sigma_m^2}$.

```
hsi.ckh.returns$hsi.Risk.free <- usa.risk.free + hsi.risk.premium
hsi.ckh.returns$hsi.Risk.premium <- hsi.ckh.returns$hsi.Return -
  hsi.ckh.returns$hsi.Risk.free
hsi.Return.vector <- as.vector(hsi.ckh.returns$hsi.Return)
ckh.Return.vector <- as.vector(hsi.ckh.returns$ckh.Return)
cov.hsi.ckh <- cov(ckh.Return.vector, hsi.Return.vector)
var.hk <- var(hsi.Return.vector)
capm_beta = cov.hsi.ckh/var.hk

fit <- lm(ckh.Return ~ hsi.Risk.premium, data = hsi.ckh.returns)
summary(fit)
```

```
Call:
lm(formula = ckh.Return ~ hsi.Risk.premium, data = hsi.ckh.returns)

Residuals:
       Min          1Q      Median          3Q         Max
-0.031033 -0.022800  0.001032  0.010137  0.050709

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)         0.007142   0.007437   0.960    0.357
hsi.Risk.premium 0.671372   0.101209   6.634 3.69e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.02565 on 11 degrees of freedom
Multiple R-squared:     0.8, Adjusted R-squared:  0.7818
F-statistic:     44 on 1 and 11 DF,  p-value: 3.692e-05
```

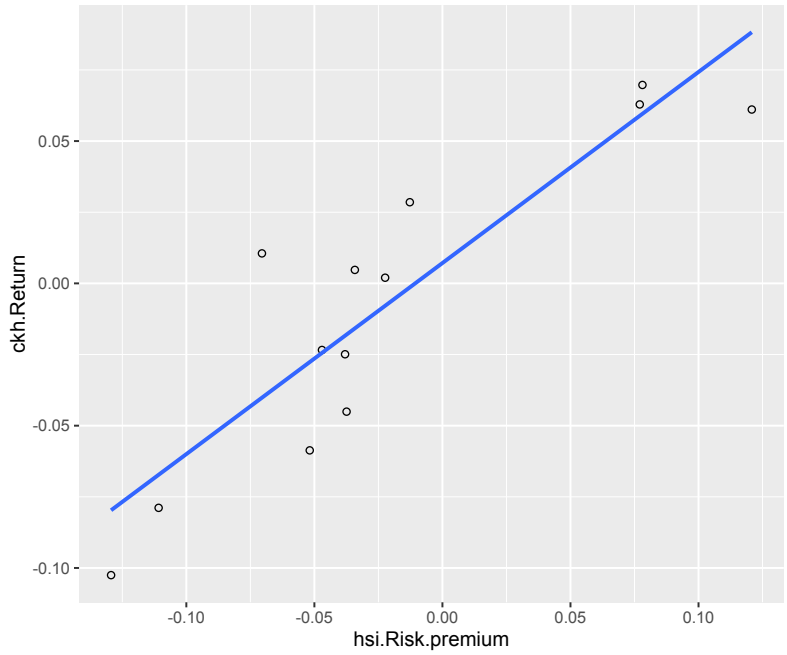
Up to this point we have all we need to plot regressions. We will start with a basic regression plot.

```
p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) +
  geom_point(shape=1) + geom_smooth(method=lm)
p11
```



`geom_smooth` can be customized, for example, not to include the confidence interval.

```
p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) +
  geom_point(shape=1) + geom_smooth(method=lm, se=FALSE)
p11
```

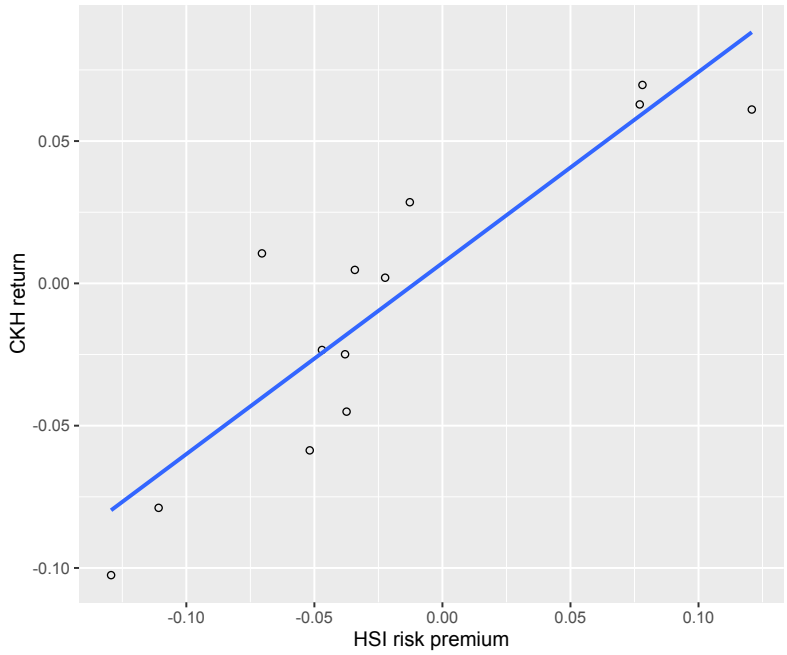


Before continuing it is a good idea to fix the axis labels and add a title.

11.1.2. Customising axis labels

We can change the text of the axis labels using the `scale_x_continuous` and `scale_y_continuous` options, with the names passed as a string to the `name` arguments in each.

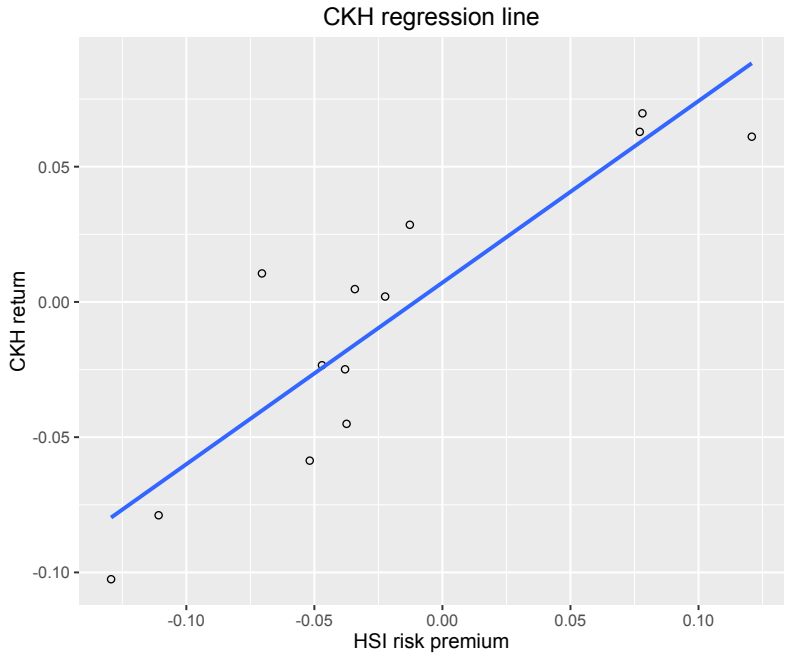
```
p11 <- p11 + scale_x_continuous(name = "HSI risk premium") +
  scale_y_continuous(name = "CKH return")
p11
```



11.1.3. Adding a title

Similarly, we can add a title using the `ggtitle` option.

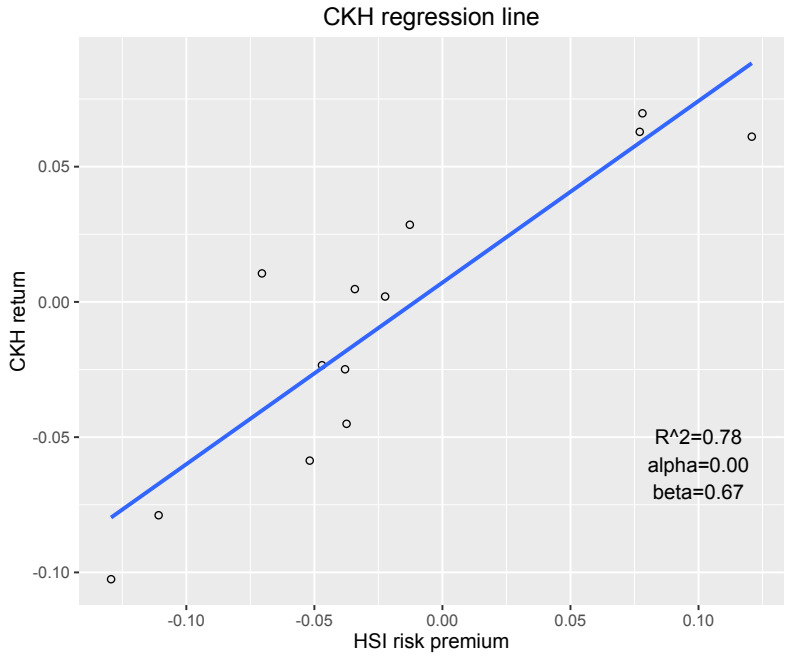
```
p11 <- p11 + ggtitle("CKH regression line")
p11
```



11.1.4. Including regression coefficients

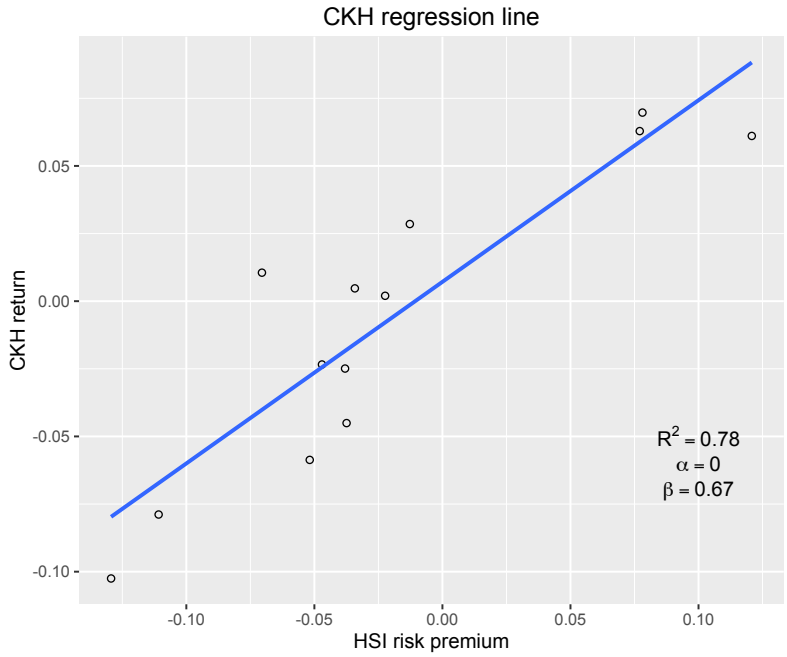
We can also include more information about the regression line itself. It would be interesting to show the R^2 and regression coefficients within the plot. We do this by adding these values in separate `annotate` options.

```
p11 <- p11 + annotate("text", x=0.1, y=-0.05, label = "R^2=0.78") +
  annotate("text", x=0.1, y=-0.06, label = "alpha=0.00") +
  annotate("text", x=0.1, y=-0.07, label = "beta=0.67")
p11
```



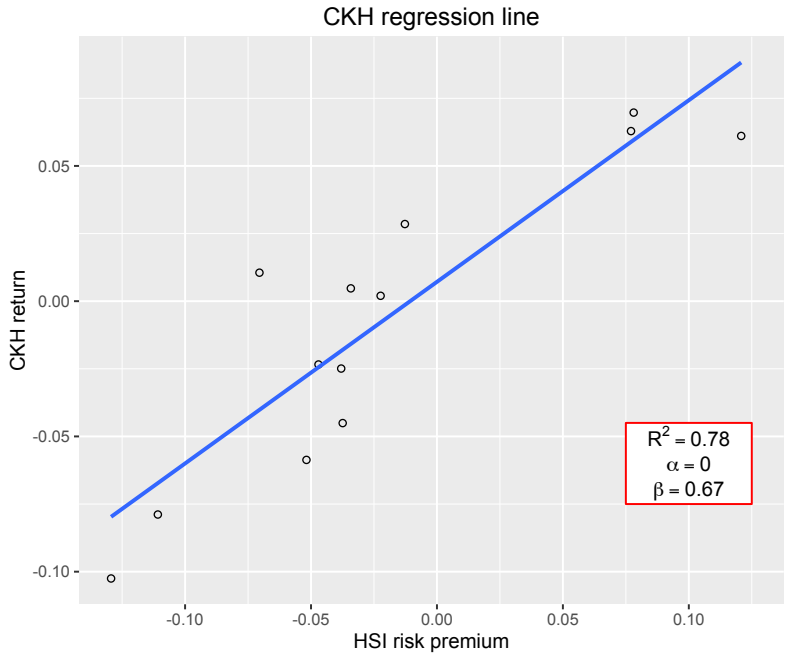
Another option would be to add greek letters and exponents. We can do this by calling the argument `parse=T` within the `annotate` option.

```
p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) +
  geom_point(shape=1) +
  geom_smooth(method=lm, se=FALSE) + ggtitle("CKH regression line") +
  scale_x_continuous(name = "HSI risk premium") +
  scale_y_continuous(name = "CKH return") +
  annotate("text", x=0.1, y=-0.05, label = "R^2 == 0.78", parse=T) +
  annotate("text", x=0.1, y=-0.06, label = "alpha == 0.00", parse=T) +
  annotate("text", x=0.1, y=-0.07, label = "beta == 0.67", parse=T)
p11
```



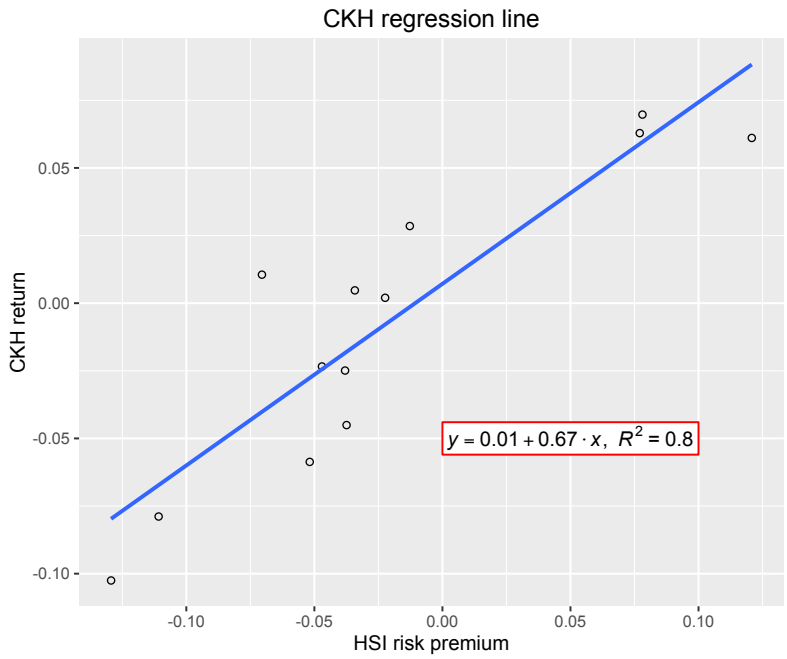
It's a little hard to see these values. To make the the coefficients more clear we will add some elements to increase visibility. You can see we've added a box with a red outline and white background using another `annotate` option, this time calling co-ordinate arguments and colour arguments.

```
p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) +
  geom_point(shape=1) + geom_smooth(method=lm, se=FALSE) +
  ggtitle("CKH regression line") +
  scale_x_continuous(name = "HSI risk premium") +
  scale_y_continuous(name = "CKH return") +
  annotate("rect", xmin = 0.075, xmax = 0.125, ymin = -0.075, ymax = -0.045,
     fill="white", colour="red") +
  annotate("text", x=0.1, y=-0.05, label = "R^2 == 0.78", parse=T) +
  annotate("text", x=0.1, y=-0.06,
     label = "alpha == 0.00", parse=T) +
  annotate("text", x=0.1, y=-0.07, label = "beta == 0.67", parse=T)
p11
```



Another customization could be to show the trend line using rounded digits (or even significant digits) from regression coefficients. This requires us to write a function and is not as easy to obtain as the last plot.

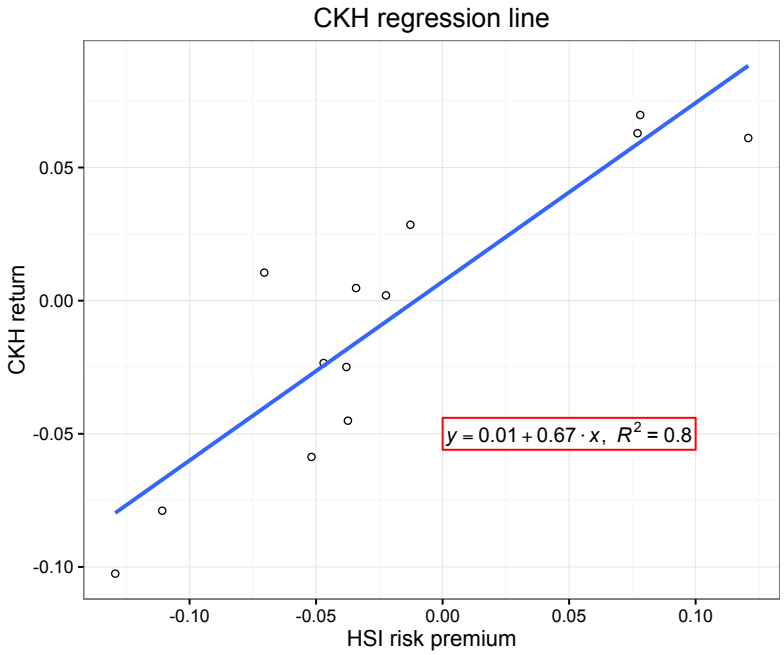
```
equation = function(x) {
  lm_coef <- list(a = round(coef(x)[1], digits = 2),
     b = round(coef(x)[2], digits = 2),
     r2 = round(summary(x)$r.squared, digits = 2));
  lm_eq <- substitute(italic(y) == a + b %.% italic(x)*","~~italic(R)^2~"="~r2,lm_coef)
  as.character(as.expression(lm_eq));
}

p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) +
  geom_point(shape=1) + geom_smooth(method=lm, se=FALSE) +
  ggtitle("CKH regression line") +
  scale_x_continuous(name = "HSI risk premium") +
  scale_y_continuous(name = "CKH return") +
  annotate("rect", xmin = 0.00, xmax = 0.1, ymin = -0.056, ymax = -0.044,
     fill="white", colour="red") +
  annotate("text", x = 0.05, y = -0.05, label = equation(fit), parse = TRUE)
p11
```



11.1.5. Using the white theme

We can also change the overall look of the plot using themes. We'll start using a simple theme customisation by adding `theme_bw()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
p11 <- p11 + theme_bw()
p11
```

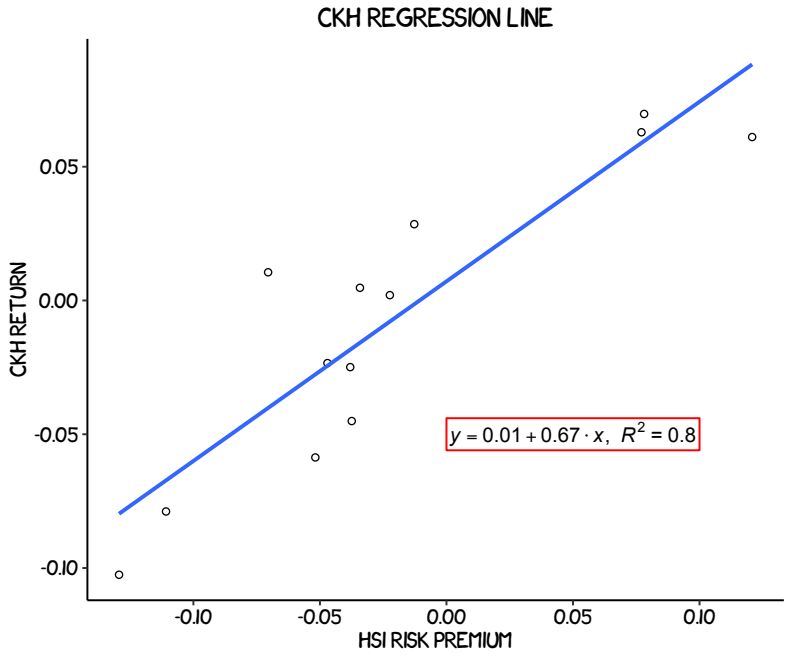


11.1.6. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) +
  geom_point(shape=1) + geom_smooth(method=lm, se=FALSE) +
  ggtitle("CKH regression line") +
  scale_x_continuous(name = "HSI risk premium") +
  scale_y_continuous(name = "CKH return") +
  annotate("rect", xmin = 0.00, xmax = 0.1, ymin = -0.056, ymax = -0.044,
     fill="white", colour="red") +
  annotate("text", x = 0.05, y = -0.05, label = equation(fit), parse = TRUE) +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
     axis.line.y = element_line(size=.5, colour = "black"),
     axis.text.x=element_text(colour="black", size = 10),
     axis.text.y=element_text(colour="black", size = 10),
     legend.position="bottom",
     legend.direction="horizontal",
     legend.box = "horizontal",
     legend.key = element_blank(),
     panel.grid.major = element_blank(),
     panel.grid.minor = element_blank(),
     panel.border = element_blank(),
     panel.background = element_blank(),
```

```
    plot.title=element_text(family="xkcd-Regular"),
    text=element_text(family="xkcd-Regular"))
p11
```

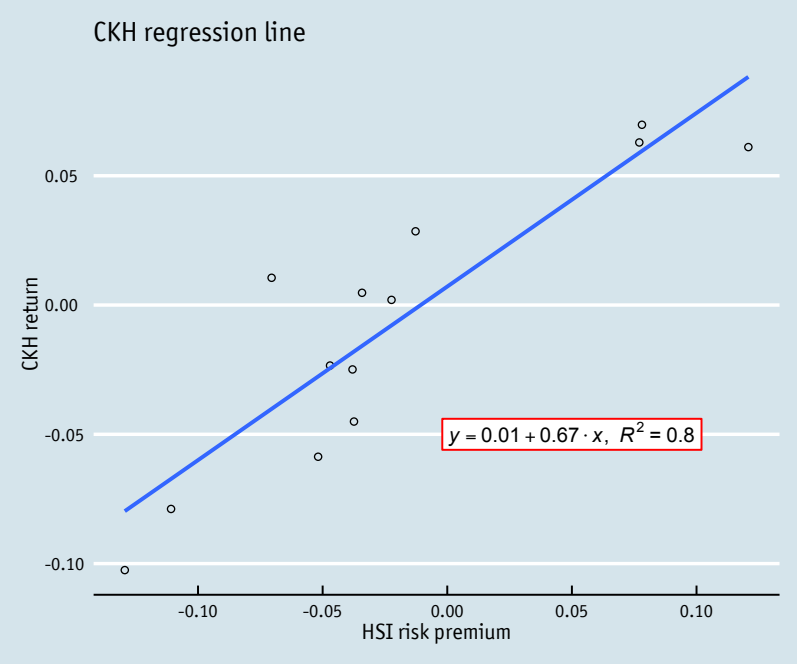


11.1.7. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

```
p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) +
  geom_point(shape=1) + geom_smooth(method=lm, se=FALSE) +
  ggtitle("CKH regression line") +
  scale_x_continuous(name = "HSI risk premium") +
  scale_y_continuous(name = "CKH return") +
  annotate("rect", xmin = -0.002, xmax = 0.102, ymin = -0.056, ymax = -0.044,
     fill="white", colour="red") +
  annotate("text", x = 0.05, y = -0.05, label = equation(fit), parse = TRUE) +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
     axis.title = element_text(size = 12),
     legend.position="bottom",
     legend.direction="horizontal",
     legend.box = "horizontal",
```

```
      legend.text = element_text(size = 10),
      text = element_text(family = "OfficinaSanITC-Book"),
      plot.title = element_text(family="OfficinaSanITC-Book"))
p11
```

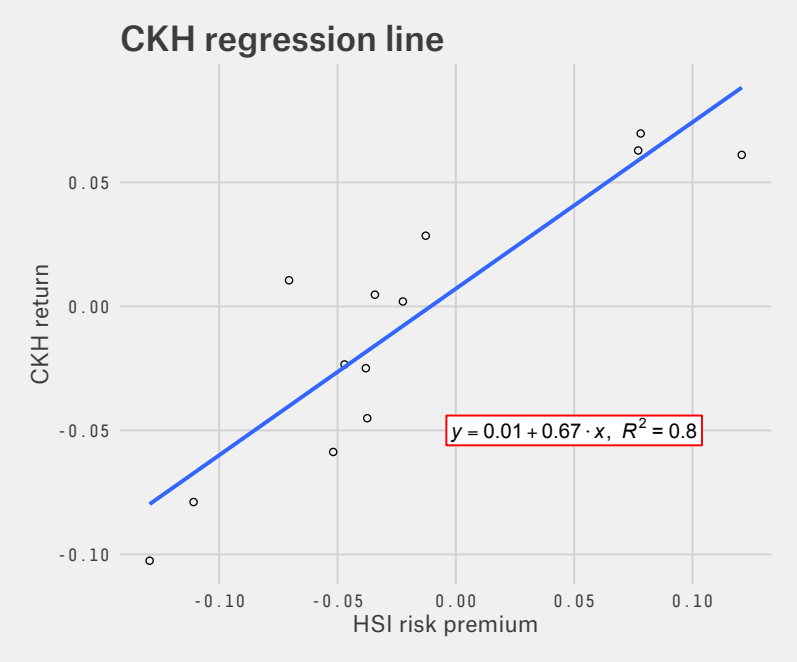


11.1.8. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

```
p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) +
  geom_point(shape=1) + geom_smooth(method=lm, se=FALSE) +
  ggtitle("CKH regression line") +
  scale_x_continuous(name = "HSI risk premium") +
  scale_y_continuous(name = "CKH return") +
  annotate("rect", xmin = -0.004, xmax = 0.104, ymin = -0.056, ymax = -0.044,
     fill="white", colour="red") +
  annotate("text", x = 0.05, y = -0.05, label = equation(fit), parse = TRUE) +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
     legend.position="bottom",
     legend.direction="horizontal",
     legend.box = "horizontal",
     legend.title=element_text(family="Atlas Grotesk Regular", size = 10),
```

```
    legend.text=element_text(family="Atlas Grotesk Regular", size = 10),
    plot.title=element_text(family="Atlas Grotesk Medium"),
    text=element_text(family="DecimaMonoPro"))
p11
```

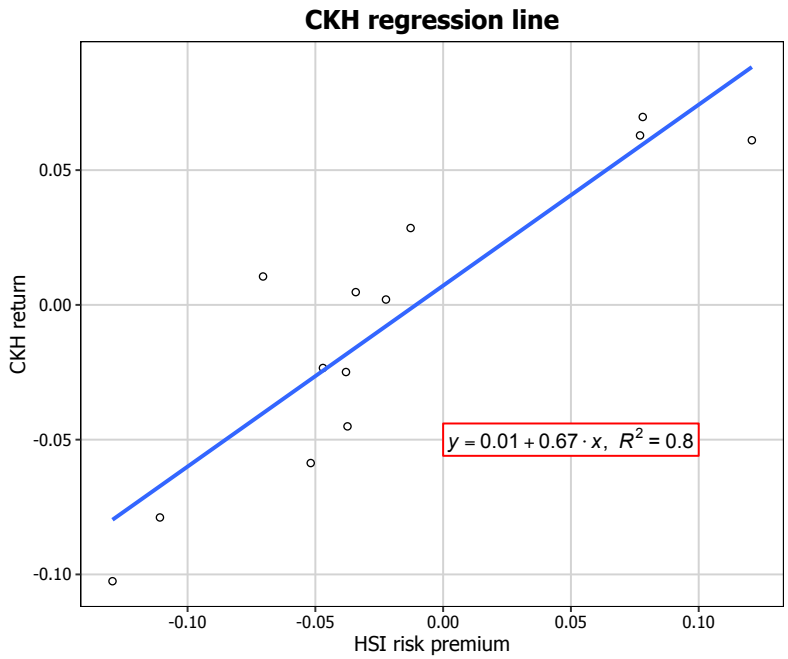


11.1.9. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here is a custom plot where we have modified the axes, background and font.

```
p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) +
  geom_point(shape=1) + geom_smooth(method=lm, se=FALSE) +
  ggtitle("CKH regression line") +
  scale_x_continuous(name = "HSI risk premium") +
  scale_y_continuous(name = "CKH return") +
  annotate("rect", xmin = 0.00, xmax = 0.1, ymin = -0.056, ymax = -0.044,
     fill="white", colour="red") +
  annotate("text", x = 0.05, y = -0.05, label = equation(fit), parse = TRUE) +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
     axis.text.x=element_text(colour="black", size = 9),
     axis.text.y=element_text(colour="black", size = 9),
     legend.position = "bottom", legend.position = "horizontal",
     panel.grid.major = element_line(colour = "#d3d3d3"),
     panel.grid.minor = element_blank(),
     panel.border = element_blank(), panel.background = element_blank(),
     plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
```

```
      text=element_text(family="Tahoma"))
p11
```

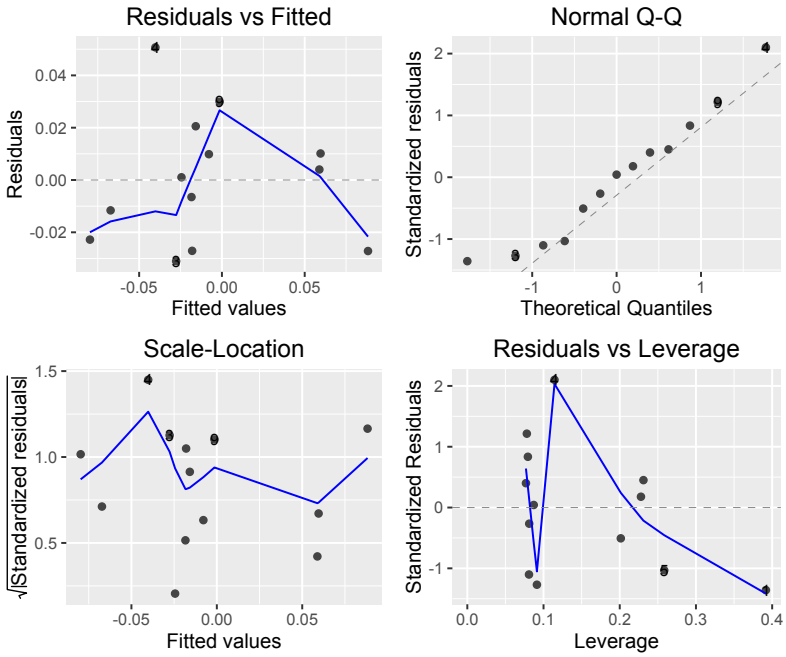


11.2. Regression diagnostics plots

11.2.1. Basic diagnostics plots

An important part of creating regression models is evaluating how well they fit the data. We can use the package `ggfortify` to let `ggplot2` interpret `lm` objects and create diagnostic plots.

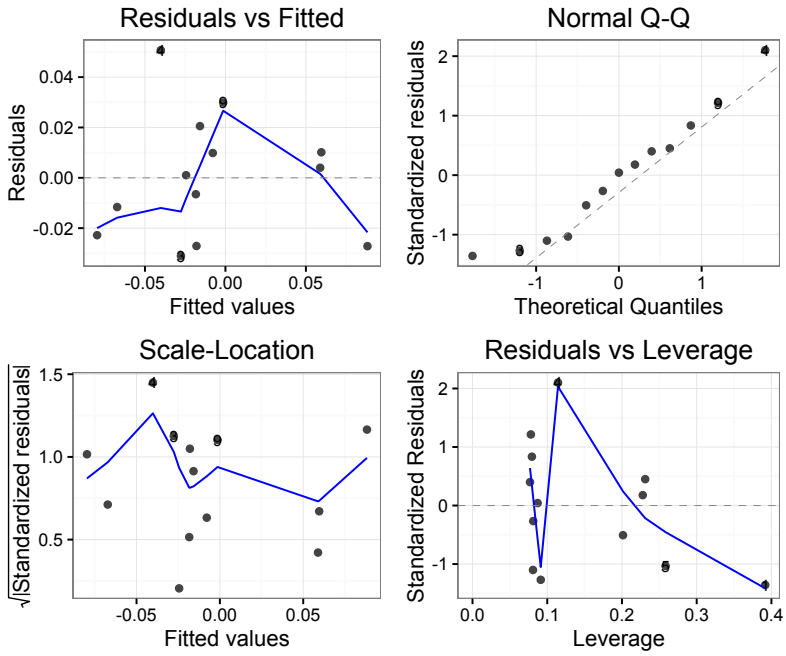
```
autoplot(fit, label.size = 3)
```



11.2.2. Using the white theme

We can also customise the appearance of our diagnostic plots. Let's first use the white theme by again adding `theme_bw()`.

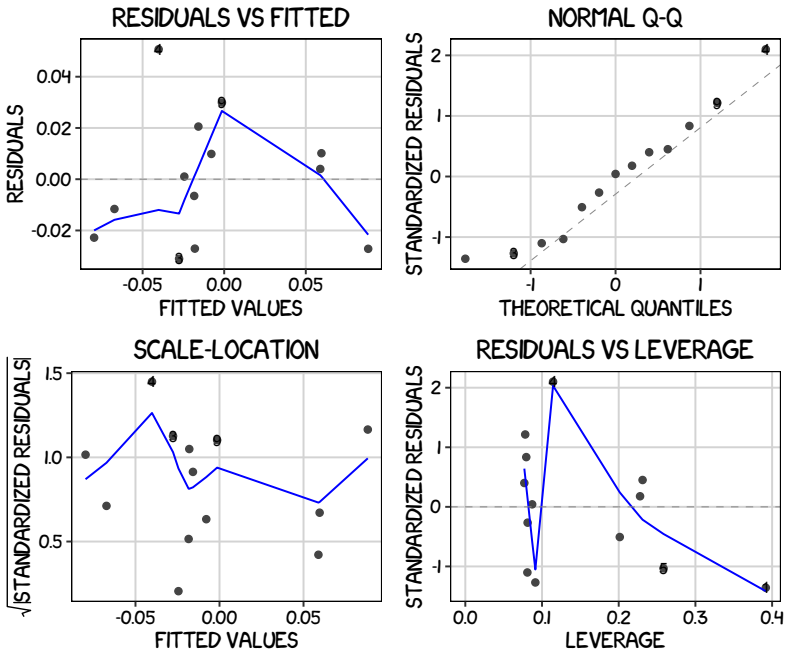
```
autoplot(fit, label.size = 3) + theme_bw()
```



11.2.3. Creating an XKCD style chart

We can of course apply our other themes as well. Let's try the XKCD theme.

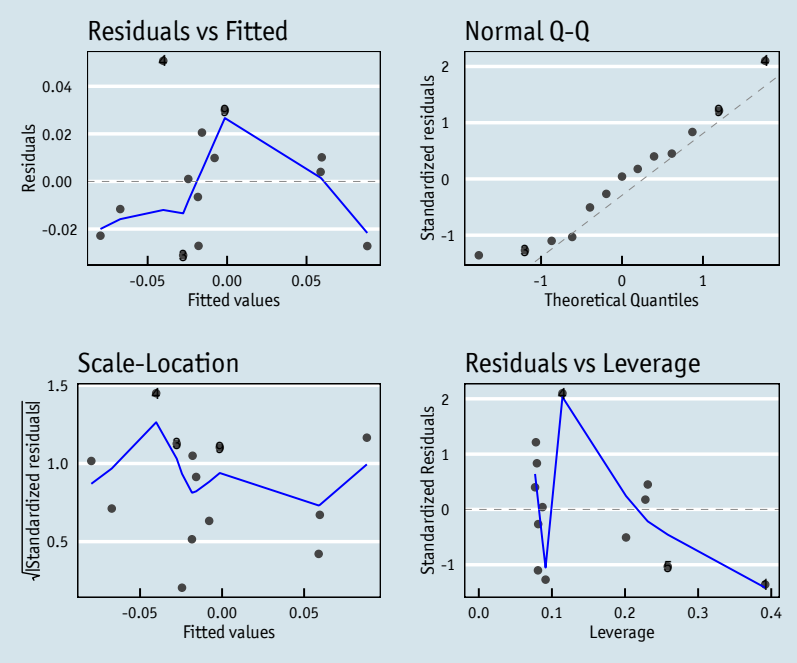
```
autoplot(fit, label.size = 3) + theme(panel.border = element_rect(colour = "black",
     fill=NA, size=.5),
     axis.text.x=element_text(colour="black", size = 9),
     axis.text.y=element_text(colour="black", size = 9),
     panel.grid.major = element_line(colour = "#d3d3d3"),
     panel.grid.minor = element_blank(),
     panel.border = element_blank(), panel.background = element_blank(),
     plot.title = element_text(family = "xkcd-Regular"),
     text=element_text(family="xkcd-Regular"))
```



11.2.4. Using 'The Economist' theme

And now the Economist theme.

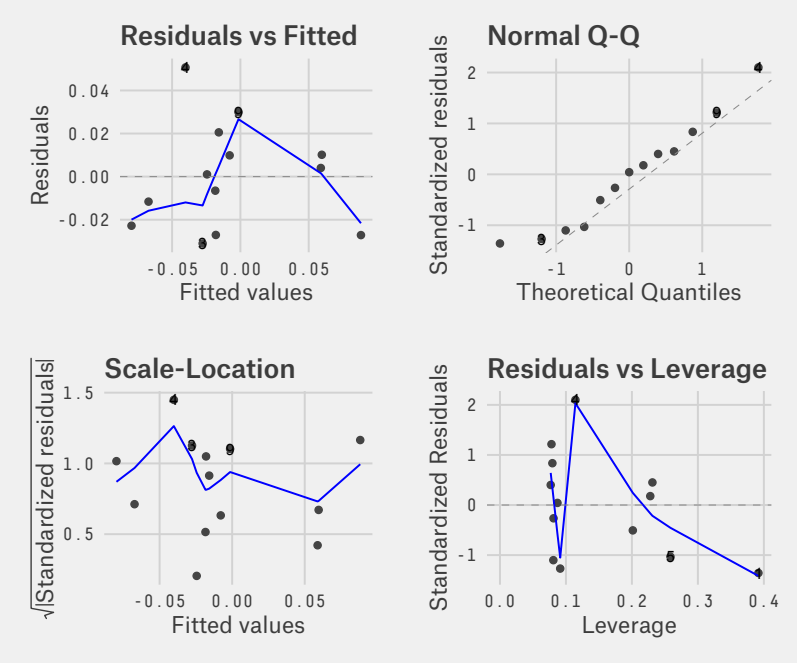
```
autoplot(fit, label.size = 3) + theme_economist() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
     axis.text.x=element_text(colour="black", size = 9),
     axis.text.y=element_text(colour="black", size = 9),
     panel.border = element_blank(), panel.background = element_blank(),
     plot.title = element_text(family = "OfficinaSanITC-Book"),
     text=element_text(family="OfficinaSanITC-Book"))
```



11.2.5. Using 'Five Thirty Eight' theme

Finally, let's see how the Five Thirty Eight theme looks.

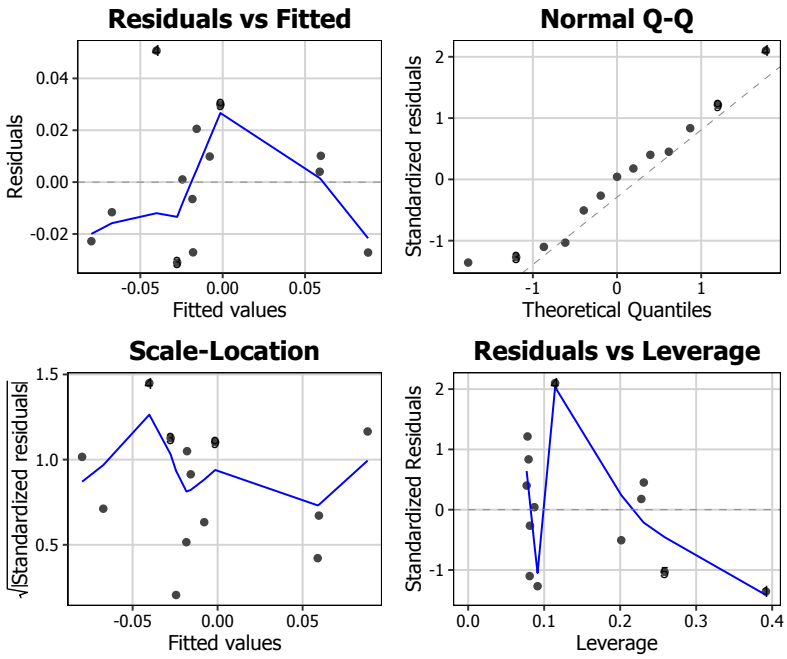
```
autoplot(fit, label.size = 3) + theme_fivethirtyeight() +
  theme(axis.title = element_text(family="Atlas Grotesk Regular"),
     legend.position="bottom",
     legend.direction="horizontal",
     legend.box = "horizontal",
     plot.title=element_text(family="Atlas Grotesk Medium", size = 14),
     text=element_text(family="DecimaMonoPro"))
```



11.2.6. Creating your own theme

As with our regression plot, we can also fully customise the diagnostic plots to match our regression plot simply by applying all of the same theme options. We now have our final two graphs that we showed at the beginning of this chapter.

```
autoplot(fit, label.size = 3) + theme(panel.border = element_rect(colour = "black",
     fill=NA, size=.5),
     axis.text.x=element_text(colour="black", size = 9),
     axis.text.y=element_text(colour="black", size = 9),
     legend.position = "bottom", legend.position = "horizontal",
     panel.grid.major = element_line(colour = "#d3d3d3"),
     panel.grid.minor = element_blank(),
     panel.border = element_blank(), panel.background = element_blank(),
     plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
     text=element_text(family="Tahoma"))
```

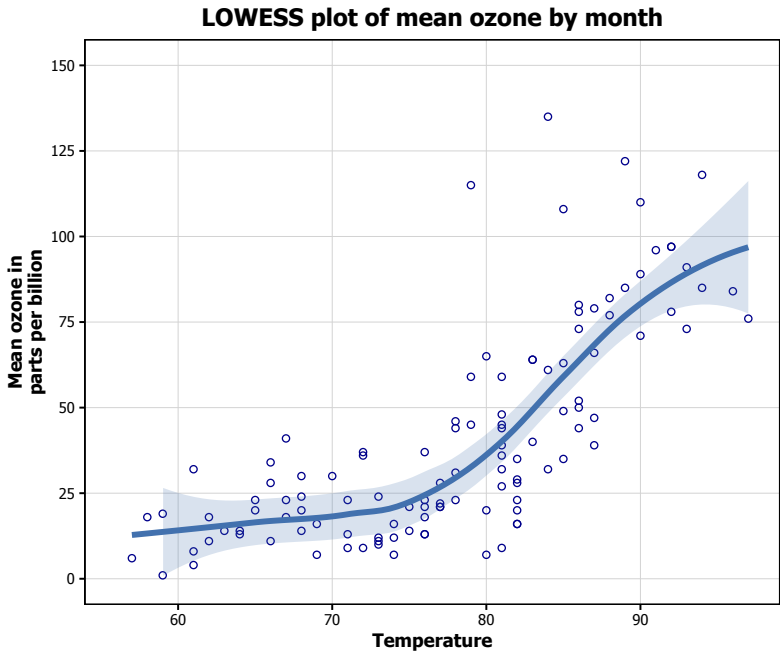

Residuals vs Fitted
Residuals
Fitted values
Normal Q-Q
Standardized residuals
Theoretical Quantiles
Scale-Location
√|Standardized residuals|
Fitted values
Residuals vs Leverage
Standardized Residuals
Leverage

CHAPTER 12

LOWESS plots

This is the twelfth and final chapter. In this chapter we will demonstrate some of the many options the `ggplot2` package has for creating and customising LOWESS plots. LOWESS, or *LO*cally *WE*ighted *S*catterplot *S*moothing, is a form of regression that creates different models for different subsets of the data. LOWESS plots represent this graphically by breaking down a continuous x-variable into a number of small bins and plotting the relationship with the y-variable individually for each subsection. Even if you don't intend to fit a LOWESS model, LOWESS plots are really useful for getting an initial feel for the relationship between your outcome and predictor, especially when you think that relationship isn't linear.

In this chapter, we will work towards creating the LOWESS plot below using R's airquality dataset in the `datasets` package. We will take you from a basic LOWESS plot and explain all the customisations we add to the code step-by-step.



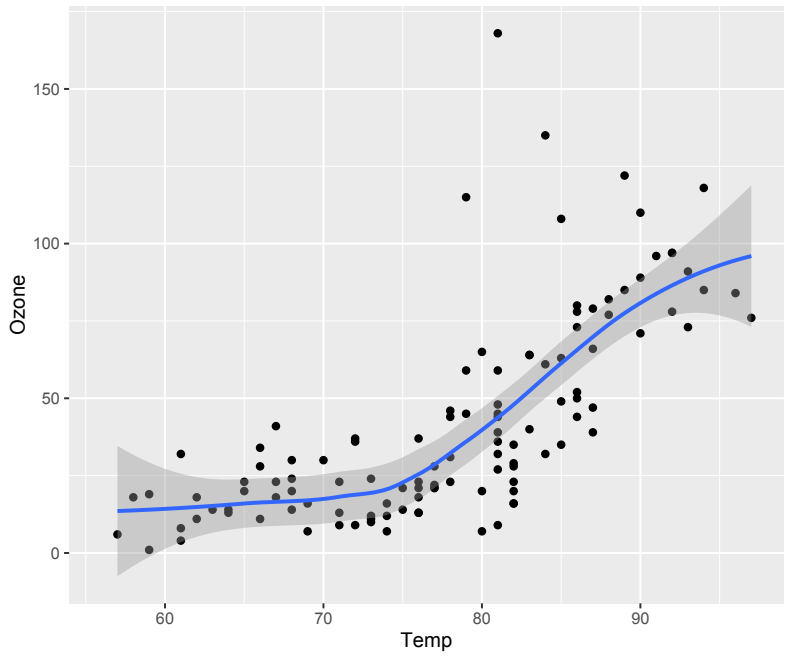
The first thing to do is load in the data and the libraries, as below.

```
library(datasets)
library(ggplot2)
library(ggthemes)
library(grid)
library(RColorBrewer)

data(airquality)
```

12.1. Creating a basic LOWESS plot, and what it can tell us about our data

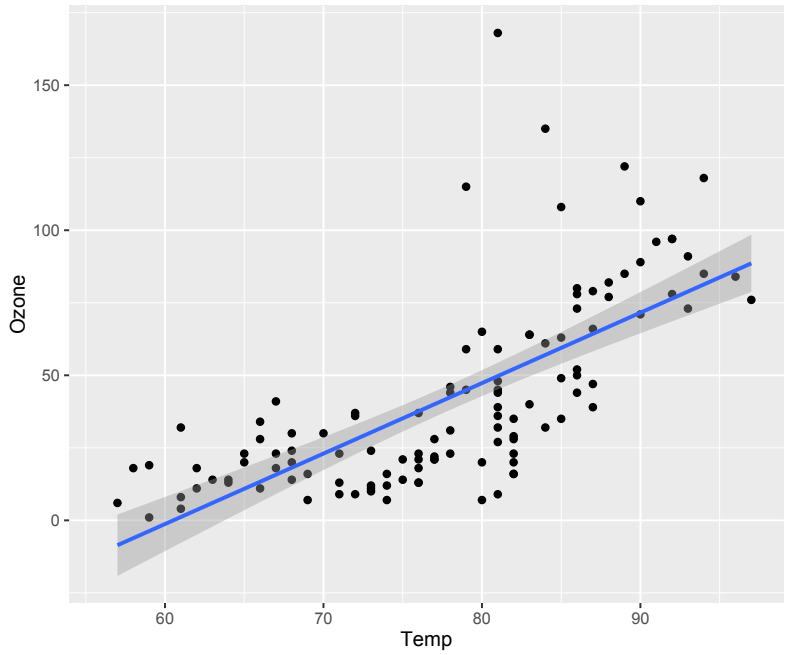
In order to initialise a plot we tell ggplot that `airquality` is our data, and specify that our x-axis plots the `Temp` variable and our y-axis plots the `Ozone` variable. We then instruct ggplot to render this as a LOWESS curve by adding the `stat_smooth(method = "loess")` option. Note that the default for `stat_smooth` is to include the confidence interval.

```
p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  stat_smooth(method = "loess")
p12
```



We can see that while the relationship between `Temp` and `Ozone` is fairly linear, the LOWESS plot is demonstrating there may be a threshold effect where ozone only starts increasing as temperatures pass around 75 degrees Fahrenheit. To assess whether this is the case, let's see how a standard linear fit between these variables looks.

```
p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  geom_smooth(method=lm)
p12
```



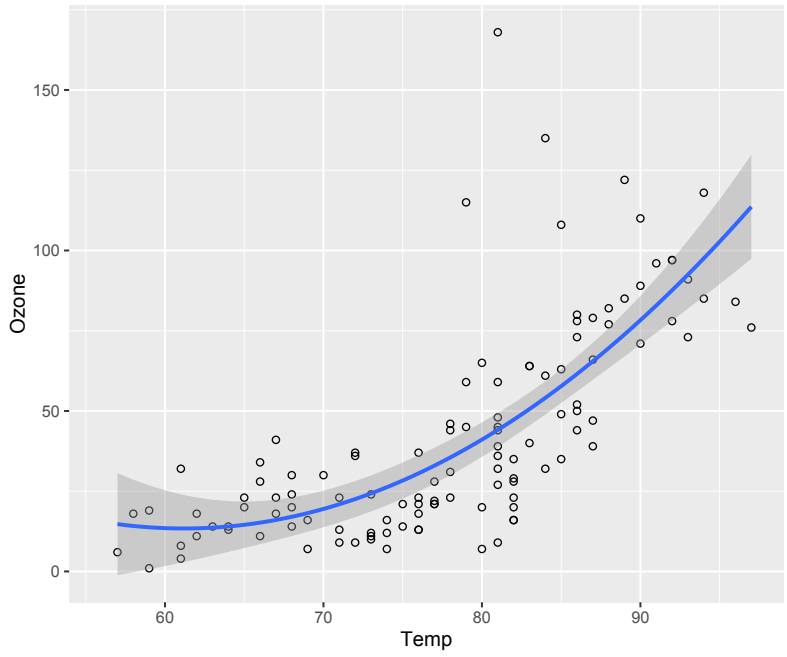
Let's now have a look at the amount of variance it explains in ozone levels by extracting the adjusted R^2 from the linear regression model between these two variables.

```
m1 <- summary(lm(Ozone ~ Temp, data = airquality))
m1$adj.r.squared
```

```
[1] 0.4832134
```

You can see that the line comes away from the data at several points, which will have increased the error in the regression model and brought down the overall R^2. Let's see whether we can get a better result by fitting a quadratic model.

```
p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(shape=1) +
  stat_smooth(method = "lm", formula = y ~ x + I(x^2))
p12
```



You can see this fits the data *much* better. Let's see if the regression model confirms this:

```
m2 <- summary(lm(Ozone ~ Temp + I(Temp^2), data = airquality))
m2$adj.r.squared
```

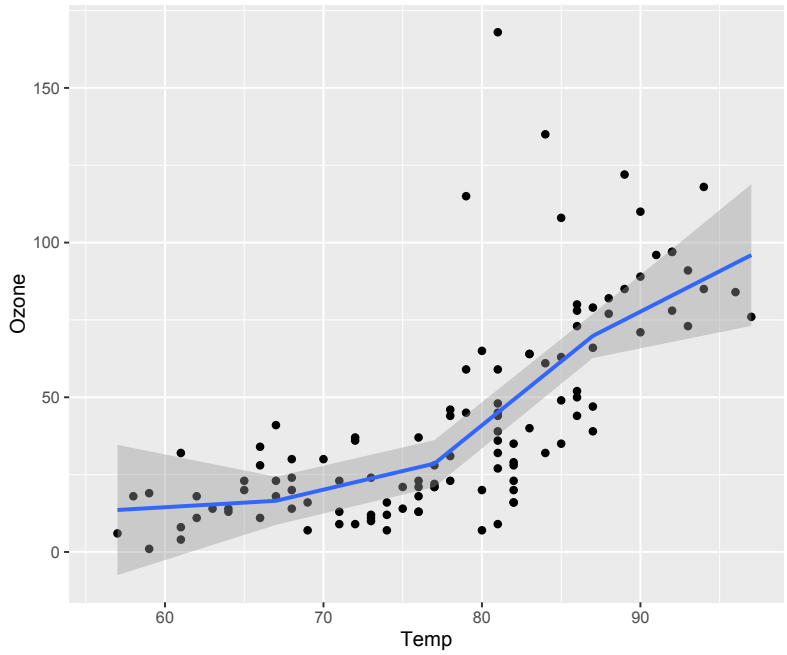
```
[1] 0.5361501
```

You can see that we've managed to explain an additional 5% of variance in ozone levels by fitting a quadratic model rather than defaulting to a linear model. Using LOWESS plots to explore the relationships between your variables can therefore guide you in choosing the the right regression model in a fairly pain-free way.

12.2. Changing the width of the bins

An important part of fitting LOWESS curves is that you can change the number of bins that the x-axis is divided into by using the argument `n`. More bins smooth out the line more, while less make it closer to linear. The default number is 80, and here we will change it to 5 so you can see the difference.

```
p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  geom_smooth(method = "loess", n = 5)
p12
```

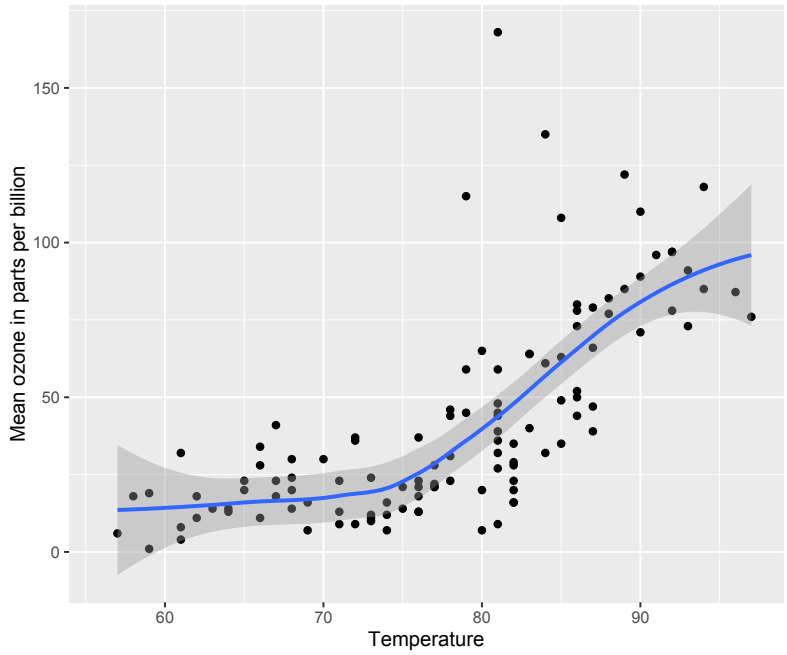


12.3. Customising axis labels

Now that we've established the rationale for using them, let's get down to customising our basic LOWESS plot.

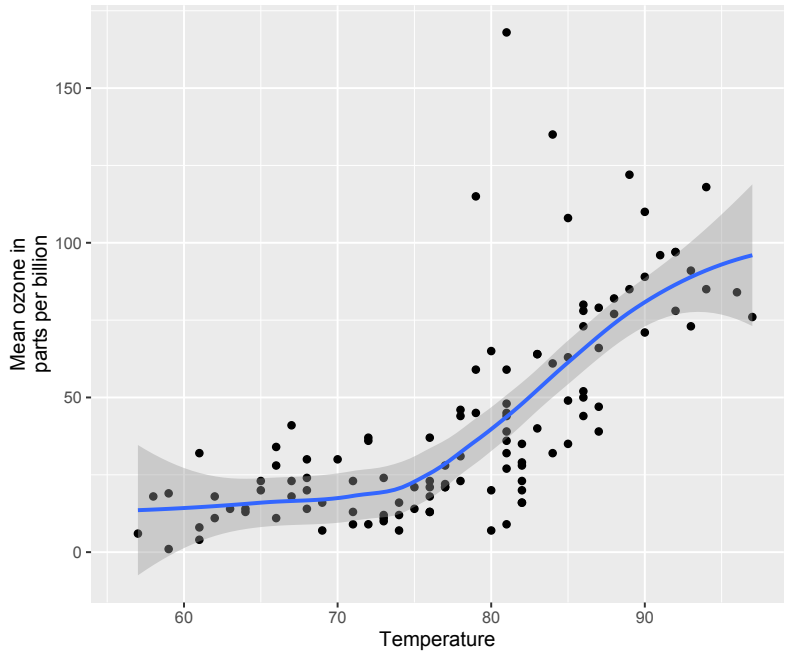
In order to change the axis labels, we have a couple of options. In this case, we have used the `scale_x_continuous` and `scale_y_continuous` options, as these have further customisation options for the axes we will use below. In each, we add the desired name to the `name` argument as a string.

```
p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  stat_smooth(method = "loess") +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in parts per billion")
p12
```



ggplot also allows for the use of multiline names (in both axes and titles). Here, we've changed the y-axis label so that it goes over two lines using the `\n` character to break the line.

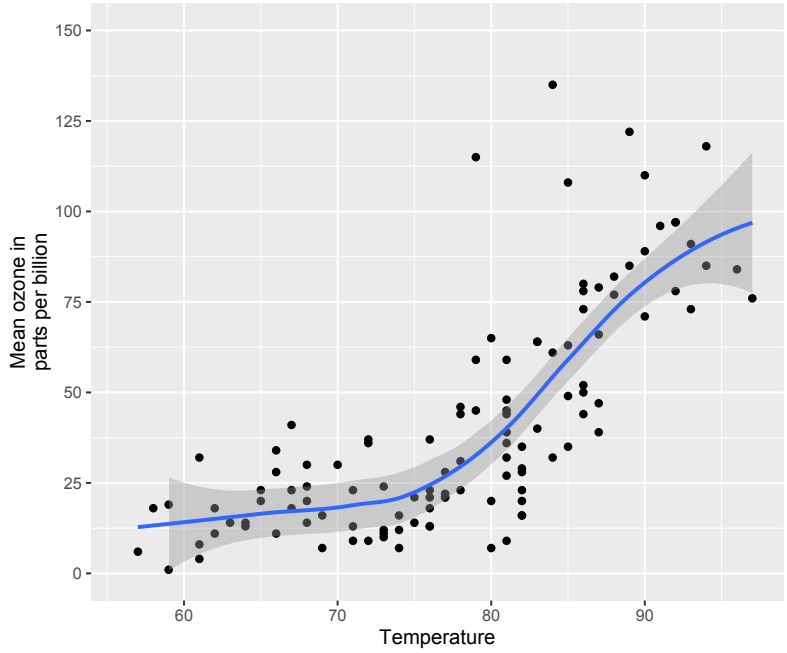
```
p12 <- p12 + scale_y_continuous(name = "Mean ozone in\nparts per billion")
p12
```



12.4. Changing axis ticks

The next thing we will change is the axis ticks. Let's make the y-axis ticks appear at every 25 units rather than 50 using the `breaks = seq(0, 150, 25)` argument in `scale_y_continuous`. (The `seq` function is a base R function that indicates the start and endpoints and the units to increment by respectively. See `help(seq)` for more information.) We ensure that the y-axis begins and ends where we want by also adding the argument `limits = c(0, 150)` to `scale_y_continuous`.

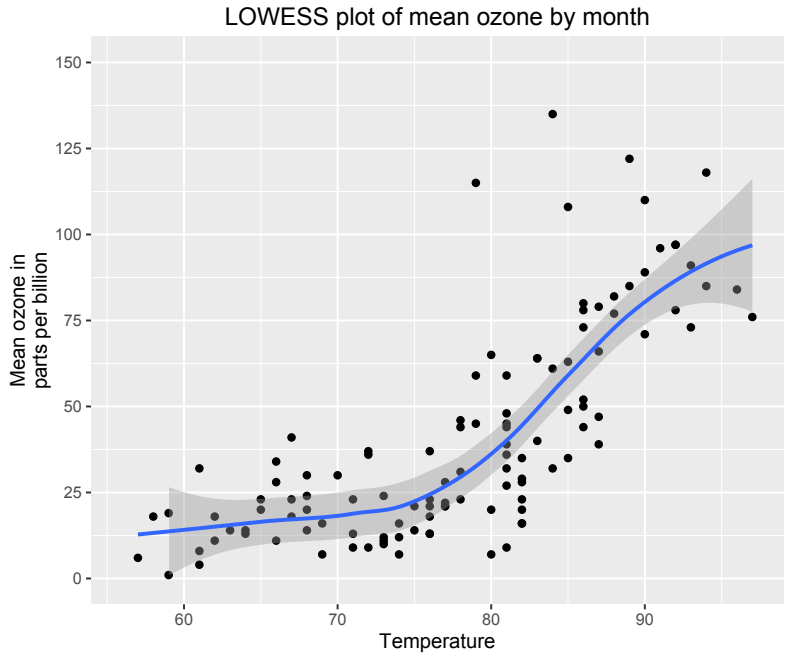
```
p12 <- p12 + scale_y_continuous(name = "Mean ozone in\nparts per billion",
  breaks = seq(0, 150, 25), limits=c(0, 150))
p12
```



12.5. Adding a title

To add a title, we include the option `ggtitle` and include the name of the graph as a string argument.

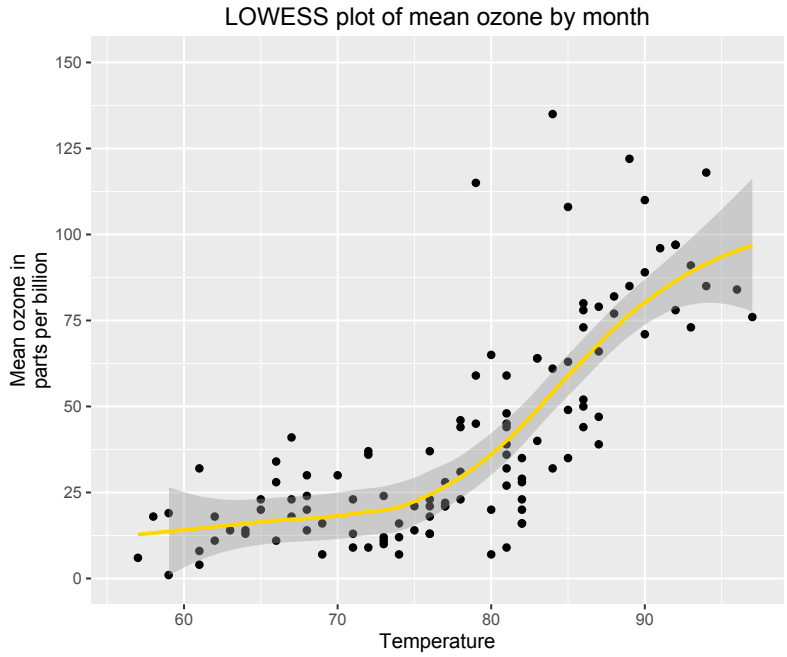
```
p12 <- p12 + ggtitle("LOWESS plot of mean ozone by month")
p12
```



12.6. Changing the colour and size of the LOWESS curve

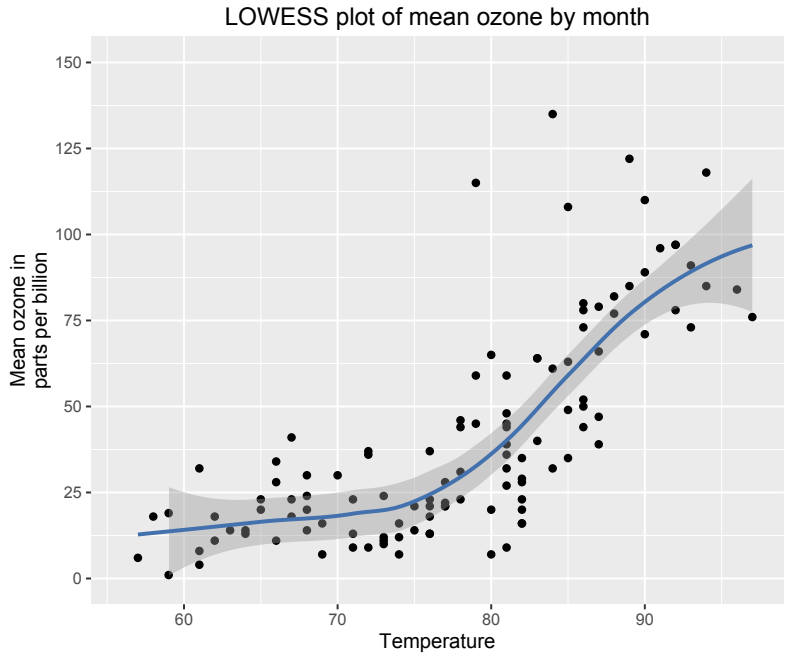
To change the colour of the LOWESS curve, we add a valid colour to the `colour` argument in `geom_smooth()` (note that we assigned this colour to a variable outside of the plot to make it easier to change it). A list of valid colours is here.

```
fill <- "gold1"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  geom_smooth(method = "loess", colour = fill) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month")
p12
```



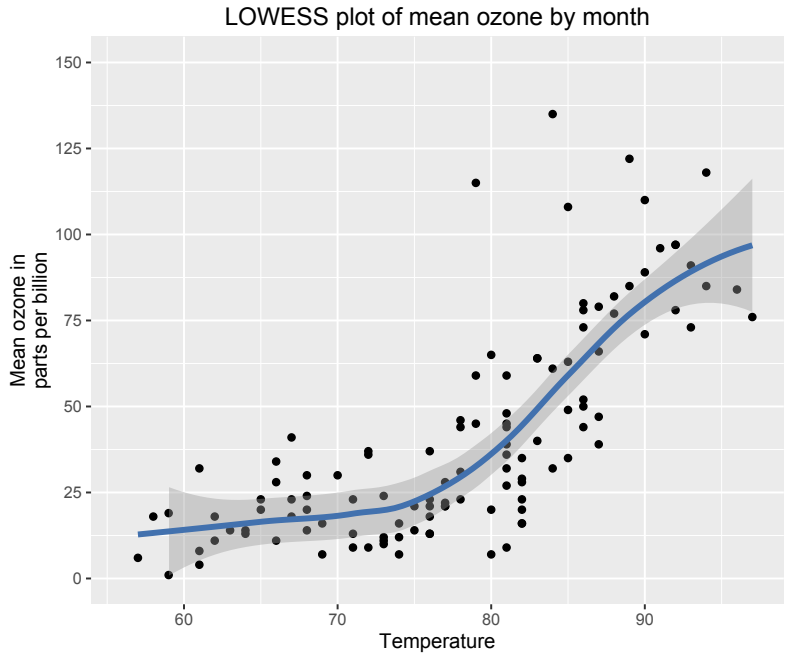
If you want to go beyond the options in the list above, you can also specify exact HEX colours by including them as a string preceded by a hash, e.g., "#FFFFFF". Below, we have called a shade of blue for the line using its HEX code.

```
fill <- "#4271AE"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  geom_smooth(method = "loess", colour = fill) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month")
p12
```



We can also increase the thickness of the line using the `size` option in `geom_smooth()`.

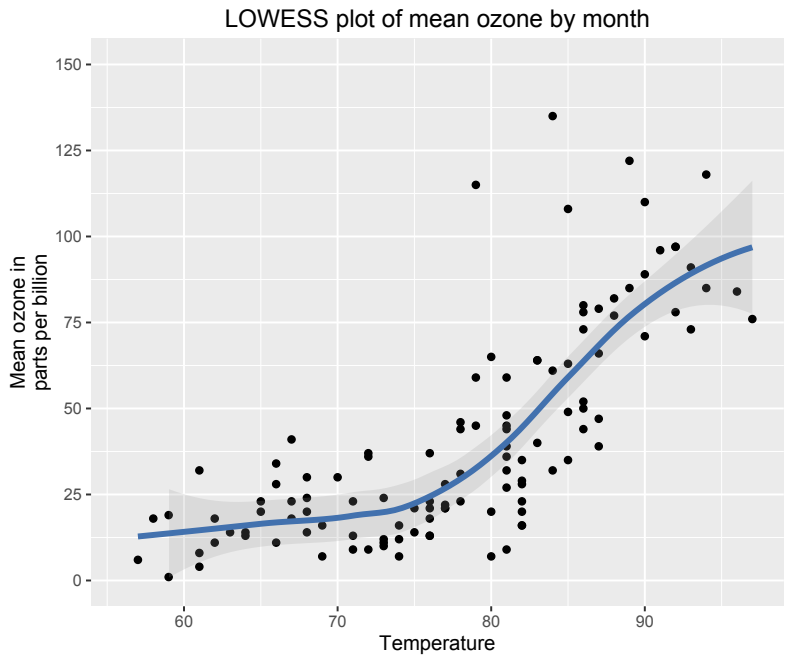
```
fill <- "#4271AE"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  geom_smooth(method = "loess", colour = fill, size = 1.5) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month")
p12
```



12.7. Changing the appearance of the confidence interval

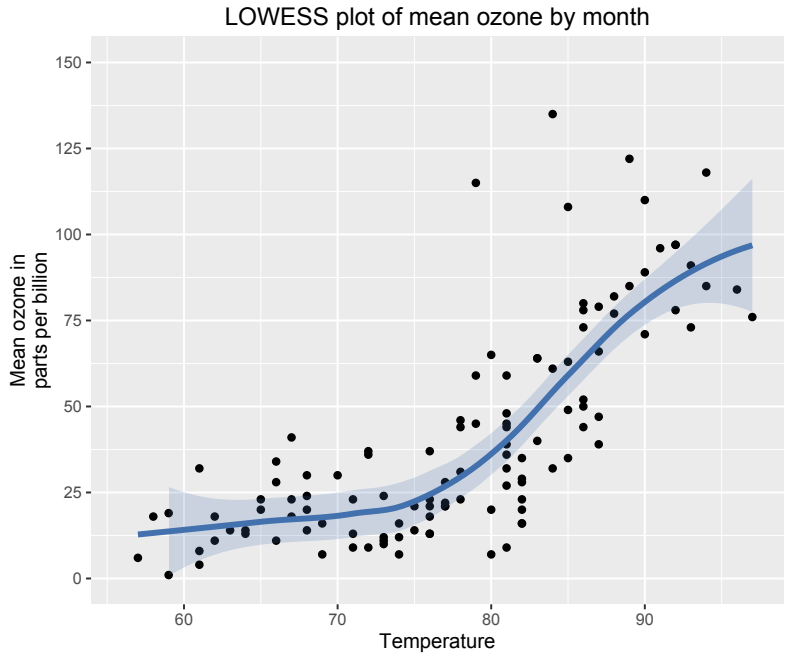
We can also alter how the confidence interval around the LOWESS curve looks. We can change the transparency using the argument `alpha` in `geom_smooth()`. This ranges from 0 to 1. Here we will increase the transparency of the confidence interval.

```
fill <- "#4271AE"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  geom_smooth(method = "loess", colour = fill, size = 1.5, alpha = 0.2) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month")
p12
```



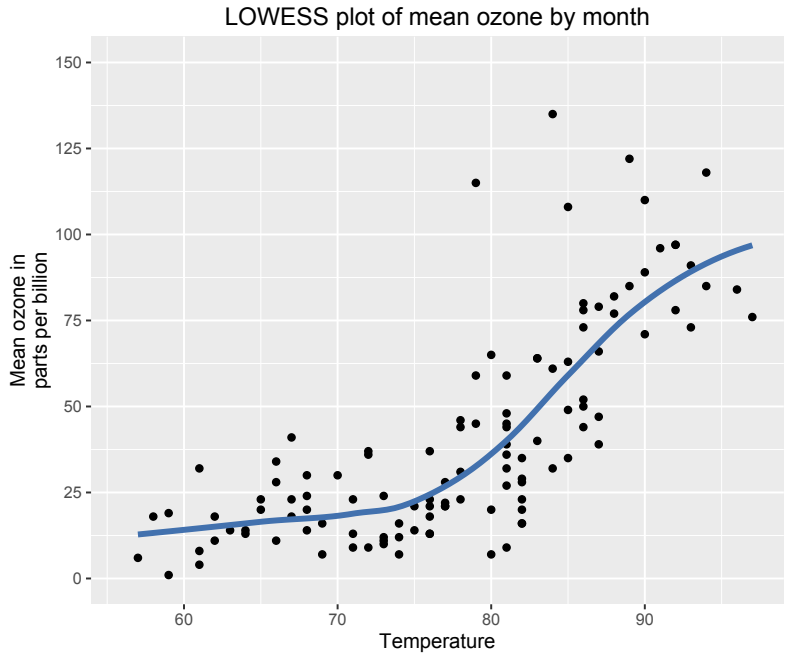
We can also change the colour of the confidence interval from the default grey using the argument `fill`, also within `geom_smooth()`. Let's change it to the same blue as our LOWESS curve.

```
fill <- "#4271AE"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  geom_smooth(method = "loess", colour = fill, size = 1.5,
   alpha = 0.2, fill = fill) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month")
p12
```



Finally, you can also turn off the confidence interval altogether by adding the argument `se = FALSE` to `geom_smooth()`.

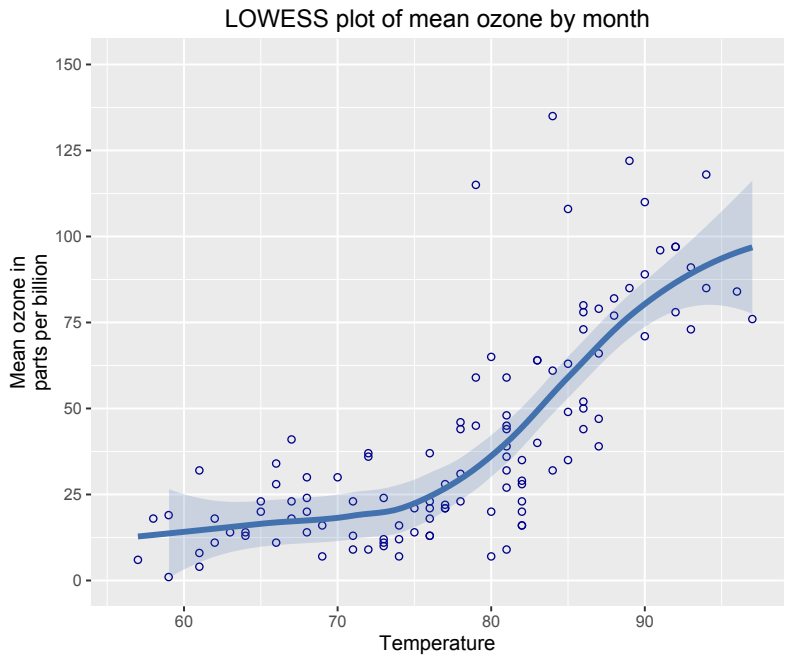
```
fill <- "#4271AE"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point() +
  geom_smooth(method = "loess", colour = fill, size = 1.5, se = FALSE) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month")
p12
```



12.8. Changing the appearance of the scatterplot

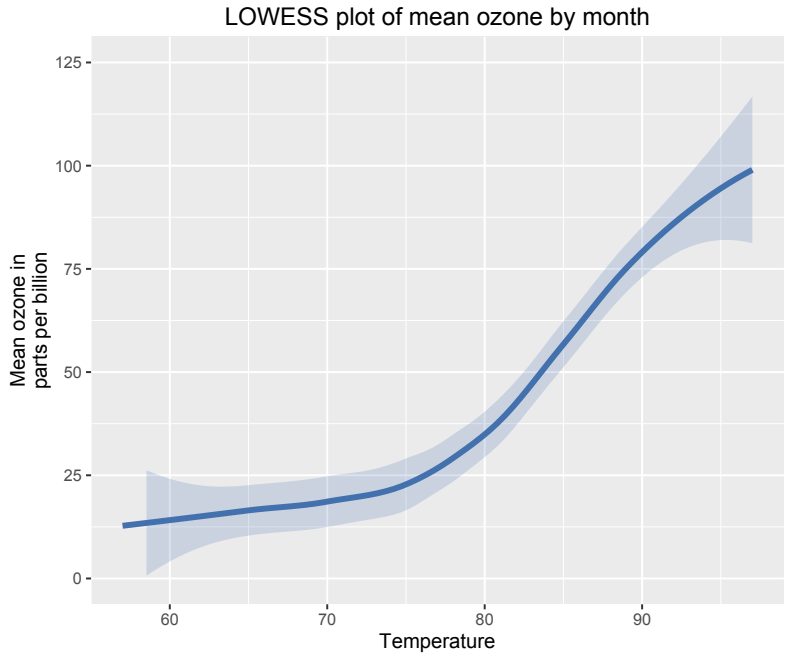
Of course, the LOWESS curve is not the only part of this plot. We can also customise the appearance of the scatterplot underlying the curve. Let's change the circles to shape 21, which is a circle that allows different colours for the outline and fill, and change the colour of the outline to dark blue. We can do this by adding the `shape` and `colour` arguments to `geom_point()` respectively.

```
fill <- "#4271AE"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(shape = 21, colour = "darkblue") +
  geom_smooth(method = "loess", colour = fill, size = 1.5,
   alpha = 0.2, fill = fill) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month")
p12
```



You can also get rid of the scatterplot points altogether by removing the `geom_point()` option. You can see we have also changed the range of the y-axis in `scale_y_continuous()` so the graph sits closer to the top of the LOWESS curve.

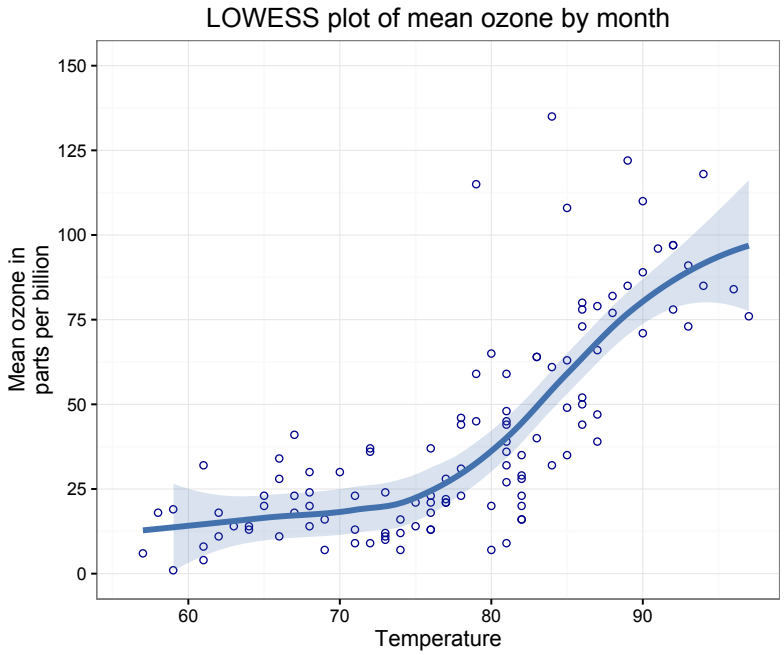
```
fill <- "#4271AE"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_smooth(method = "loess", colour = fill, size = 1.5,
   alpha = 0.2, fill = fill) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 125, 25), limits=c(0, 125)) +
  ggtitle("LOWESS plot of mean ozone by month")
p12
```



12.9. Using the white theme

We can also change the overall look of the plot using themes. We'll start using a simple theme customisation by adding `theme_bw()`. As you can see, we can further tweak the graph using the `theme` option, which we've used so far to change the legend.

```
fill <- "#4271AE"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(shape = 21, colour = "darkblue") +
  geom_smooth(method = "loess", colour = fill, size = 1.5,
   alpha = 0.2, fill = fill) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month") +
  theme_bw()
p12
```

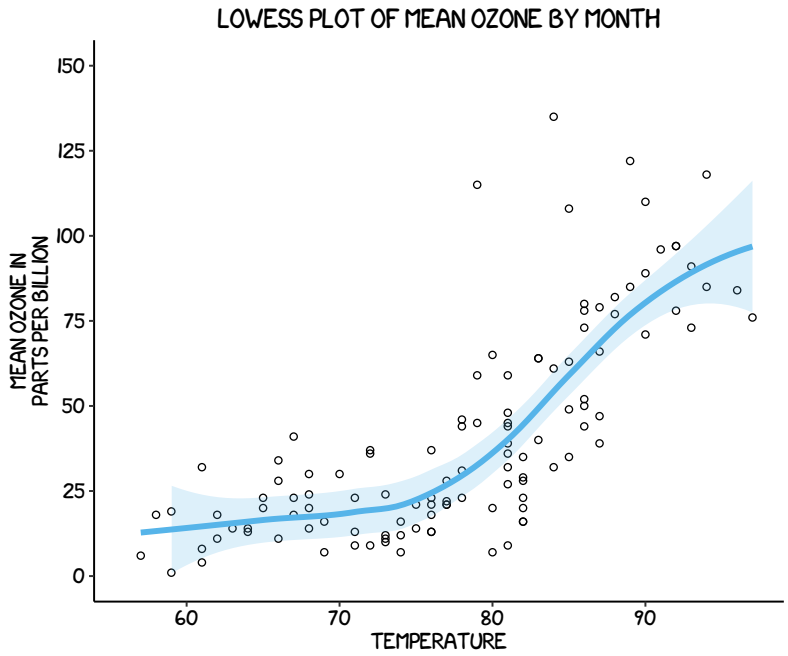


12.10. Creating an XKCD style chart

Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first need to import the XKCD font, and load it into R using the `extrafont` package.

```
p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(shape = 21, colour = "black") +
  geom_smooth(method = "loess", colour = "#56B4E9", size = 1.5,
   alpha = 0.2, fill = "#56B4E9") +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month") +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
   axis.line.y = element_line(size=.5, colour = "black"),
   axis.text.x=element_text(colour="black", size = 10),
   axis.text.y=element_text(colour="black", size = 10),
   legend.position="bottom",
   legend.direction="horizontal",
   legend.box = "horizontal",
   legend.key = element_blank(),
   panel.grid.major = element_blank(),
   panel.grid.minor = element_blank(),
   panel.border = element_blank(),
```

```
   panel.background = element_blank(),
   plot.title=element_text(family="xkcd-Regular"),
   text=element_text(family="xkcd-Regular"))
p12
```

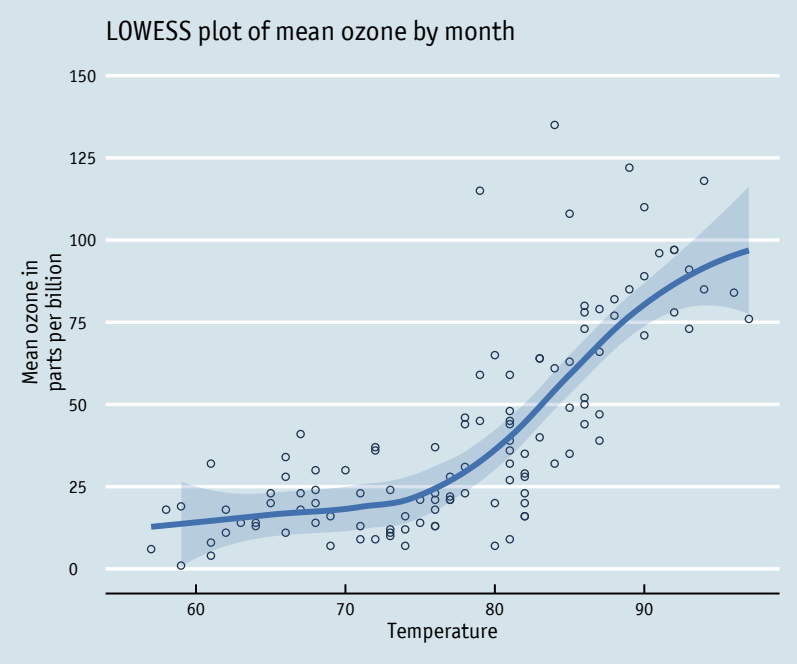


12.11. Using 'The Economist' theme

There are a wider range of pre-built themes available as part of the `ggthemes` package (more information on these here). Below we've applied `theme_economist()`, which approximates graphs in the Economist magazine. It is also important that the font change argument inside `theme` is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Officina Sans' which is a commercial font and is available here.

```
p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(shape = 21, colour = "#1F3552") +
  geom_smooth(method = "loess", colour = "#4271AE", size = 1.5,
   alpha = 0.2, fill = "#4271AE") +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month") +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
   axis.title = element_text(size = 12),
   legend.position="bottom",
```

```
   legend.direction="horizontal",
   legend.box = "horizontal",
   legend.text = element_text(size = 10),
   text = element_text(family = "OfficinaSanITC-Book"),
   plot.title = element_text(family="OfficinaSanITC-Book"))
p12
```

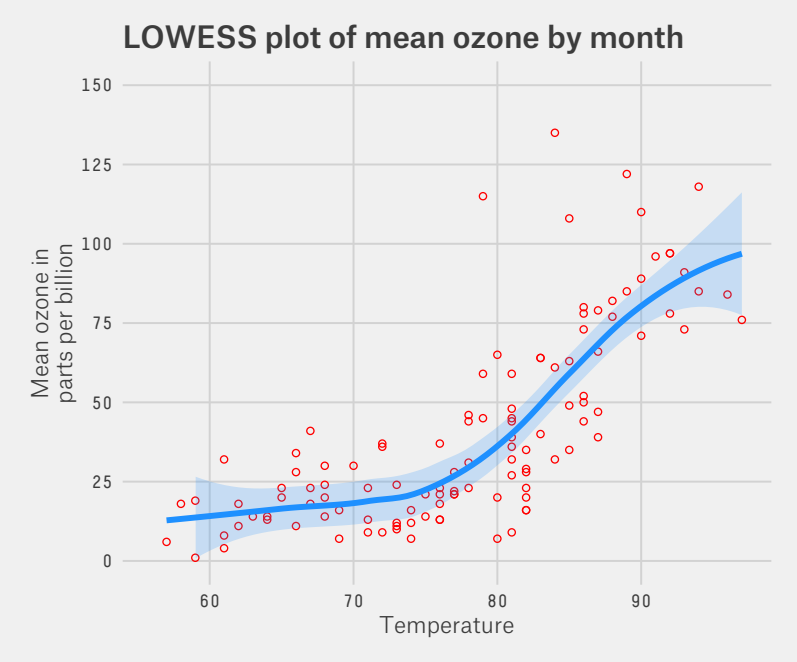


12.12. Using 'Five Thirty Eight' theme

Below we've applied `theme_fivethirtyeight()`, which approximates graphs in the nice FiveThirtyEight website. Again, it is also important that the font change is optional and it's only to obtain a more similar result compared to the original. For an exact result you need 'Atlas Grotesk' and 'Decima Mono Pro' which are commercial fonts and are available here and here.

```
p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(shape = 21, colour = "red") +
  geom_smooth(method = "loess", colour = "dodgerblue", size = 1.5,
   alpha = 0.2, fill = "dodgerblue") +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month") +
  theme_fivethirtyeight() + scale_fill_fivethirtyeight() +
  theme(axis.title = element_text(family="AtlasGrotesk-Light", size = 12),
   legend.position="bottom",
```

```
   legend.direction="horizontal",
   legend.box = "horizontal",
   legend.title=element_text(family="AtlasGrotesk-Light", size = 8),
   legend.text=element_text(family="AtlasGrotesk-Light", size = 8),
   plot.title=element_text(family="AtlasGrotesk-Medium", size = 16),
   text=element_text(family="DecimaMonoPro"))
p12
```

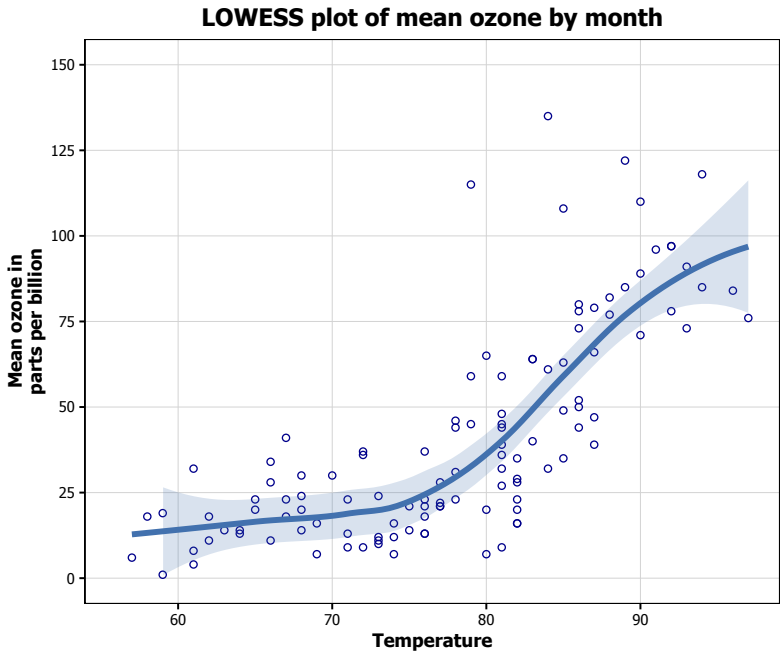


12.13. Creating your own theme

As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here we present our original result shown at the beginning of this chapter.

```
fill <- "#4271AE"

p12 <- ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(shape = 21, colour = "darkblue") +
  geom_smooth(method = "loess", colour = fill, size = 1.5,
   alpha = 0.2, fill = fill) +
  scale_x_continuous(name = "Temperature") +
  scale_y_continuous(name = "Mean ozone in\nparts per billion",
   breaks = seq(0, 150, 25), limits=c(0, 150)) +
  ggtitle("LOWESS plot of mean ozone by month") +
  theme_bw() +
  theme(panel.border = element_rect(colour = "black", fill=NA, size=.5),
```

```
   panel.grid.major = element_line(colour = "#d3d3d3"),
   panel.grid.minor = element_blank(),
   panel.border = element_blank(), panel.background = element_blank(),
   plot.title = element_text(size = 13, family = "Tahoma", face = "bold"),
   text=element_text(family = "Tahoma"),
   axis.title = element_text(face="bold", size = 10),
   axis.text.x = element_text(colour="black", size = 8),
   axis.text.y = element_text(colour="black", size = 8))
p12
```



Suggested material

(1) Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, 2009.

(2) Jenny Bryan. All the graph things.

(3) Jenny Bryan. Teaching materials for the R package ggplot2.

(4) R. Peng, J. Leek, B. Caffo. Exploratory Data Analysis.

(5) Winston Chang. *R Graphics Cookbook*. O'Reilly Media, 2012.