Creating plots in R using ggplot2 - Parts 1-10

Jodie Burchell
Mauricio Vargas Sepúlveda

2016-05-11

Contents

I recently teamed up with Mauricio Vargas Sepúlveda to create some graphing tutorials in R. In the coming weeks, we will be publishing a series of tutorials on how to use the `ggplot2` package to create beautiful and informative data visualisations. Each tutorial 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.

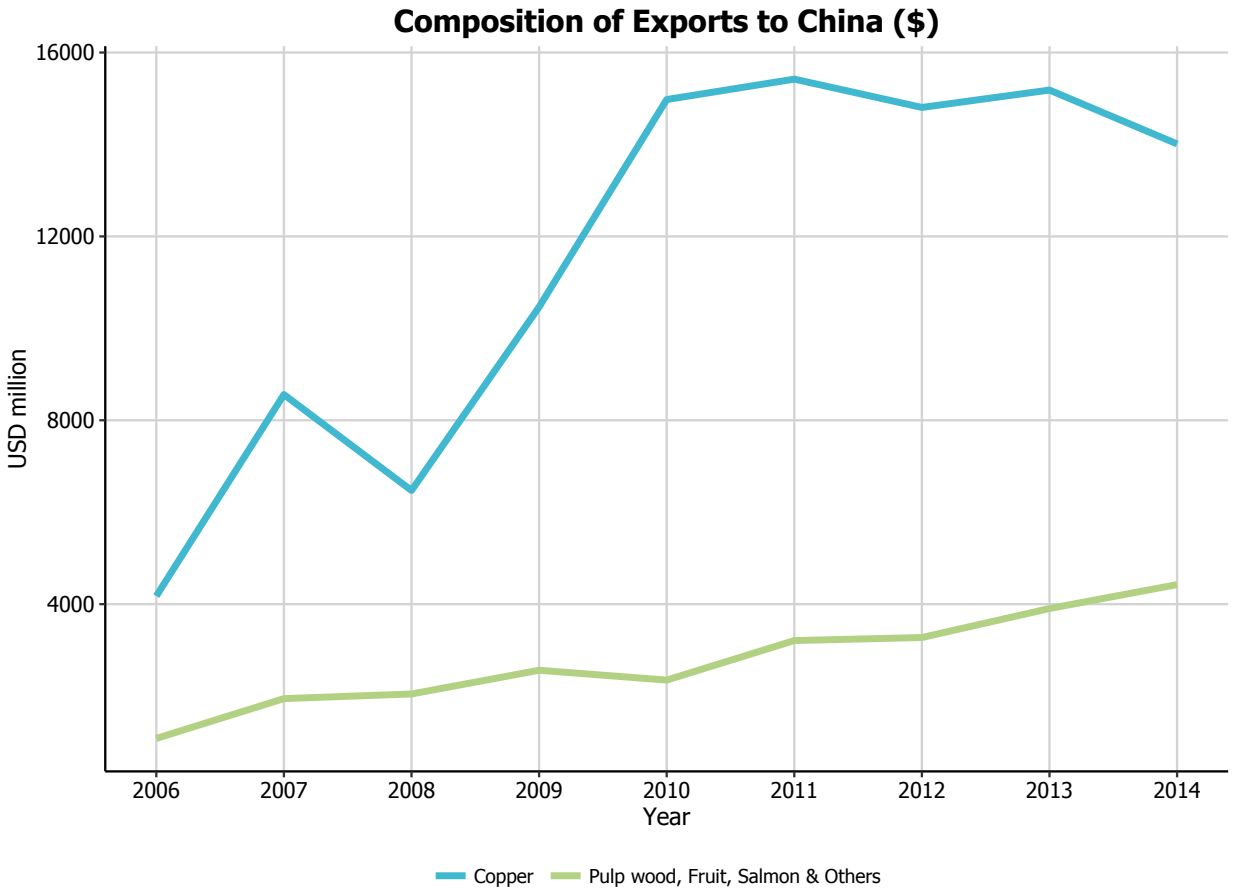
We will demonstrate some of the many options the `ggplot2` package has for creating and customising line plots. We will use an international trade dataset (parts 1-4) made by ourselves from different sources (Chile Customs, Central Bank of Chile and General Directorate of International Economic Relations).

Line Plots

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-tutorial.csv")
```

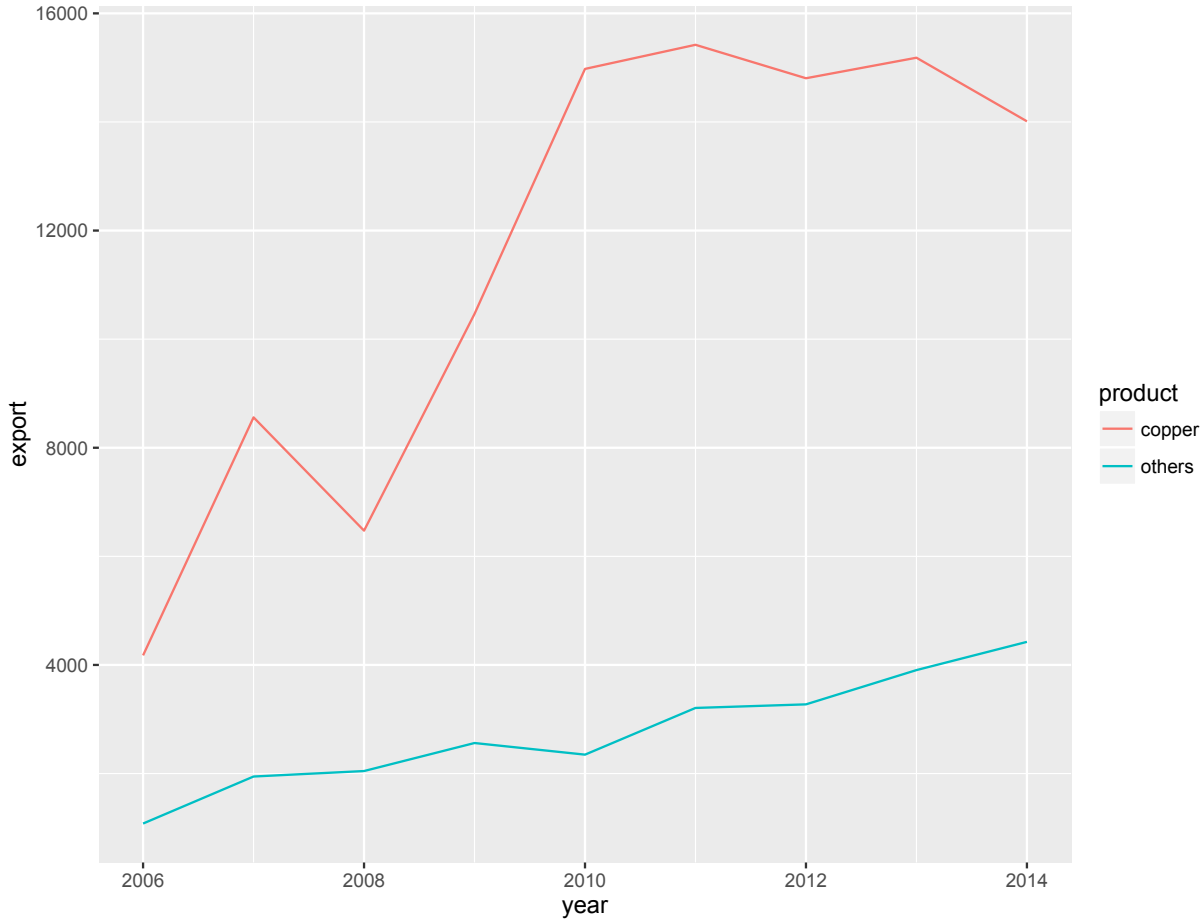
In this part, we will work towards creating the line plot below. We will take you from a basic line plot and explain all the customisations we add to the code step-by-step.



Basic graph

In order to initialise a plot we tell ggplot that `charts.data` is our data, and specify the variables on each axis. We then instruct ggplot to render this as a line plot by adding the `geom_line` command.

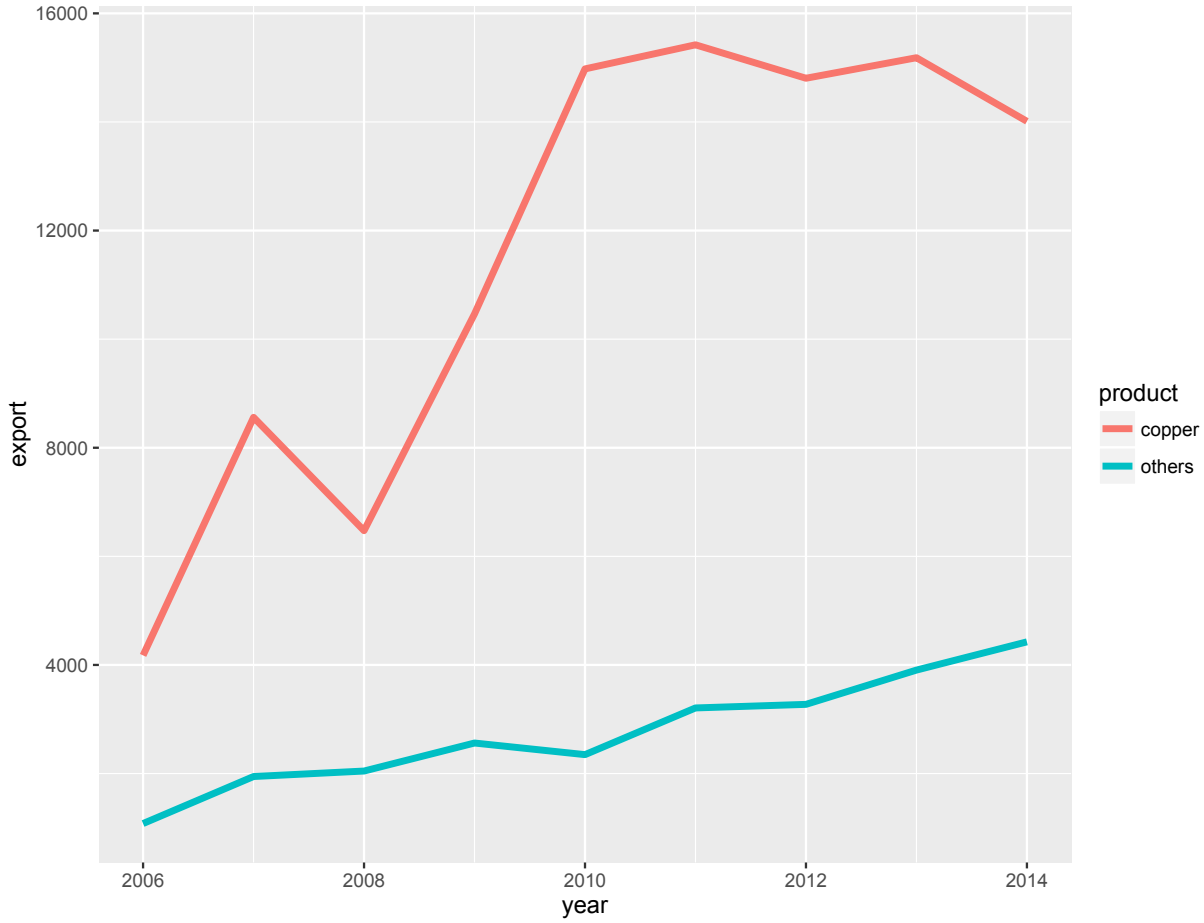
```
p1 <- ggplot() + geom_line(aes(y = export, x = year, colour = product),
                           data = charts.data, stat="identity")
p1
```



Adjusting line width

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

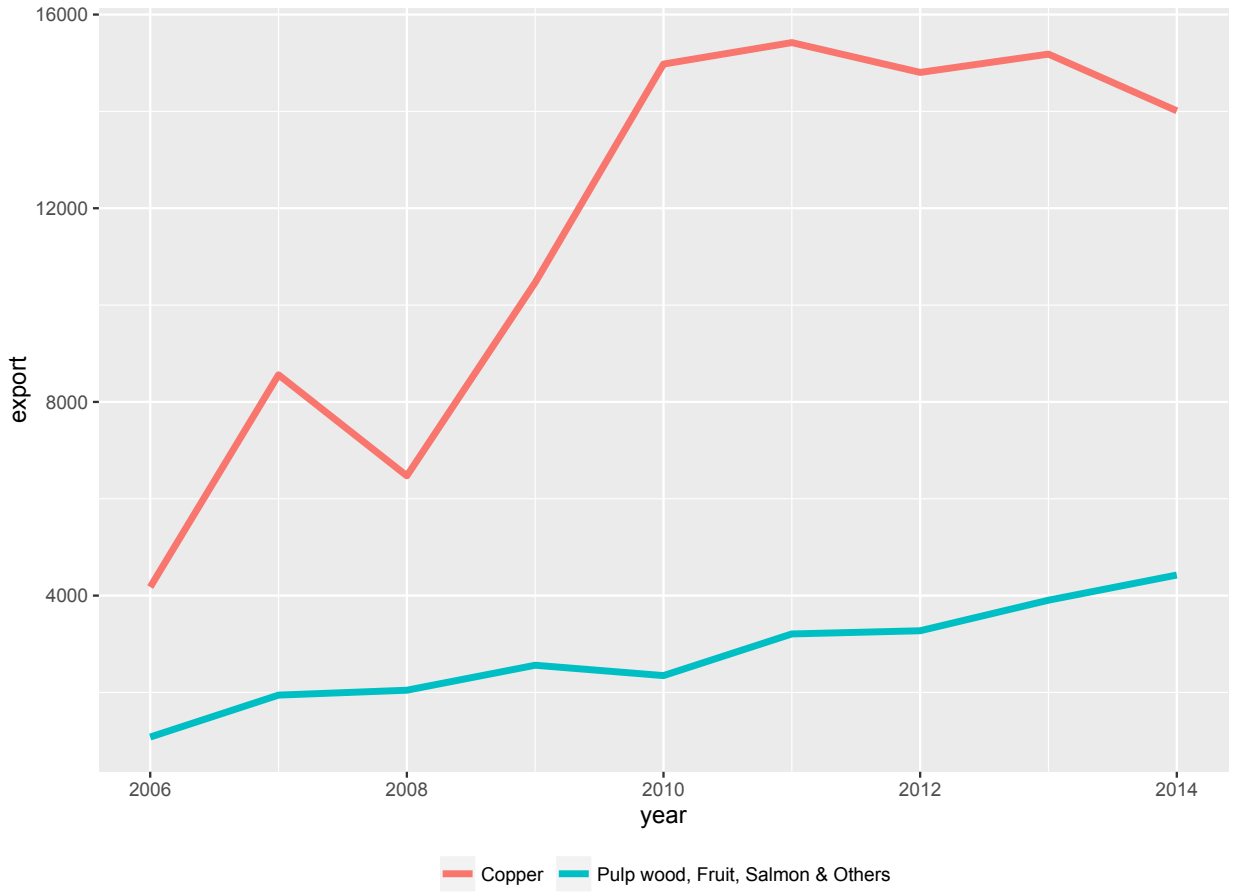
```
p1 <- ggplot() + geom_line(aes(y = export, x = year, colour = product), size=1.5,
                           data = charts.data, stat="identity")
p1
```



Changing variables display

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

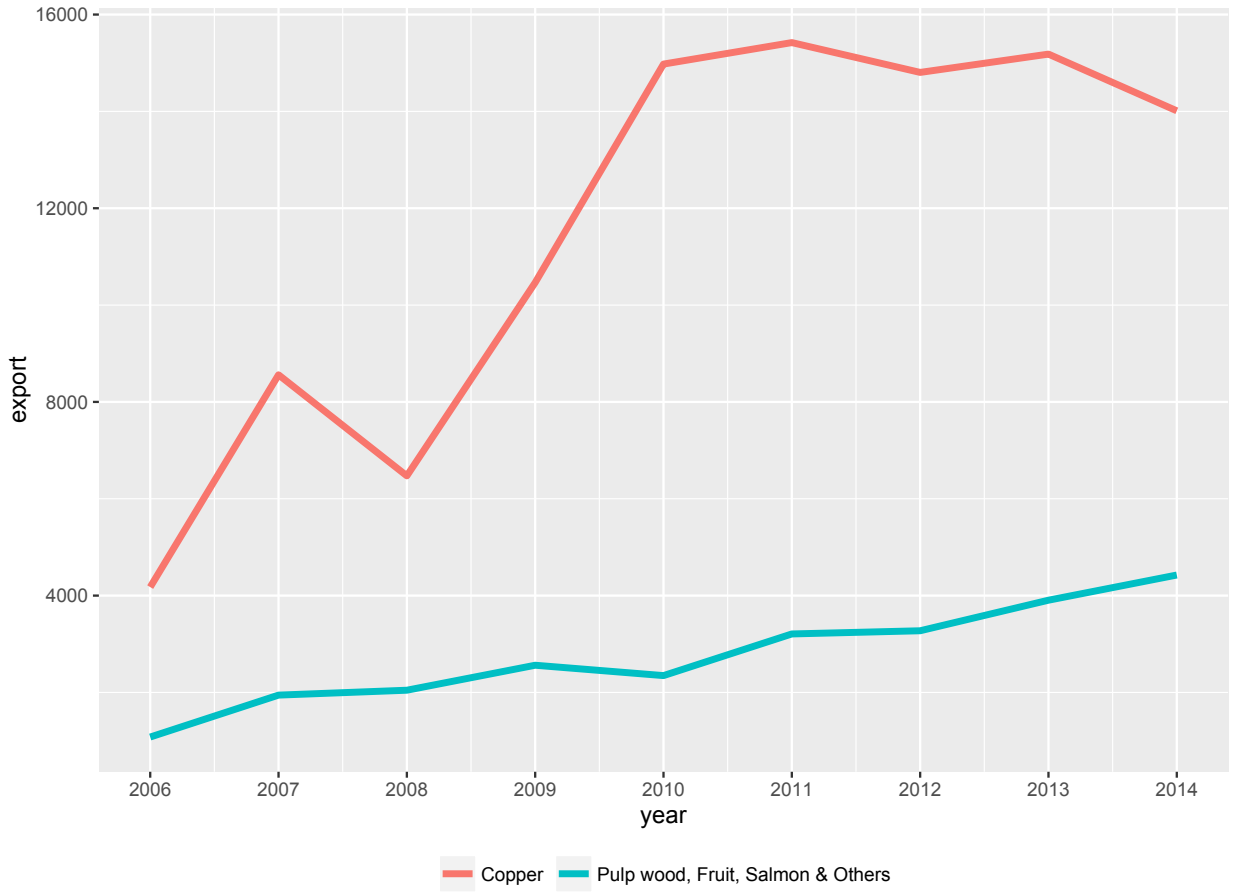
```
charts.data <- as.data.frame(charts.data)
charts.data$product <- factor(charts.data$product, levels = c("copper","others"),
                              labels = c("Copper ","Pulp wood, Fruit, Salmon & Others"))

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
```



Adjusting x-axis scale

To change the axis tick marks, we use the `scale_x_continuous` and/or `scale_y_continuous` commands.

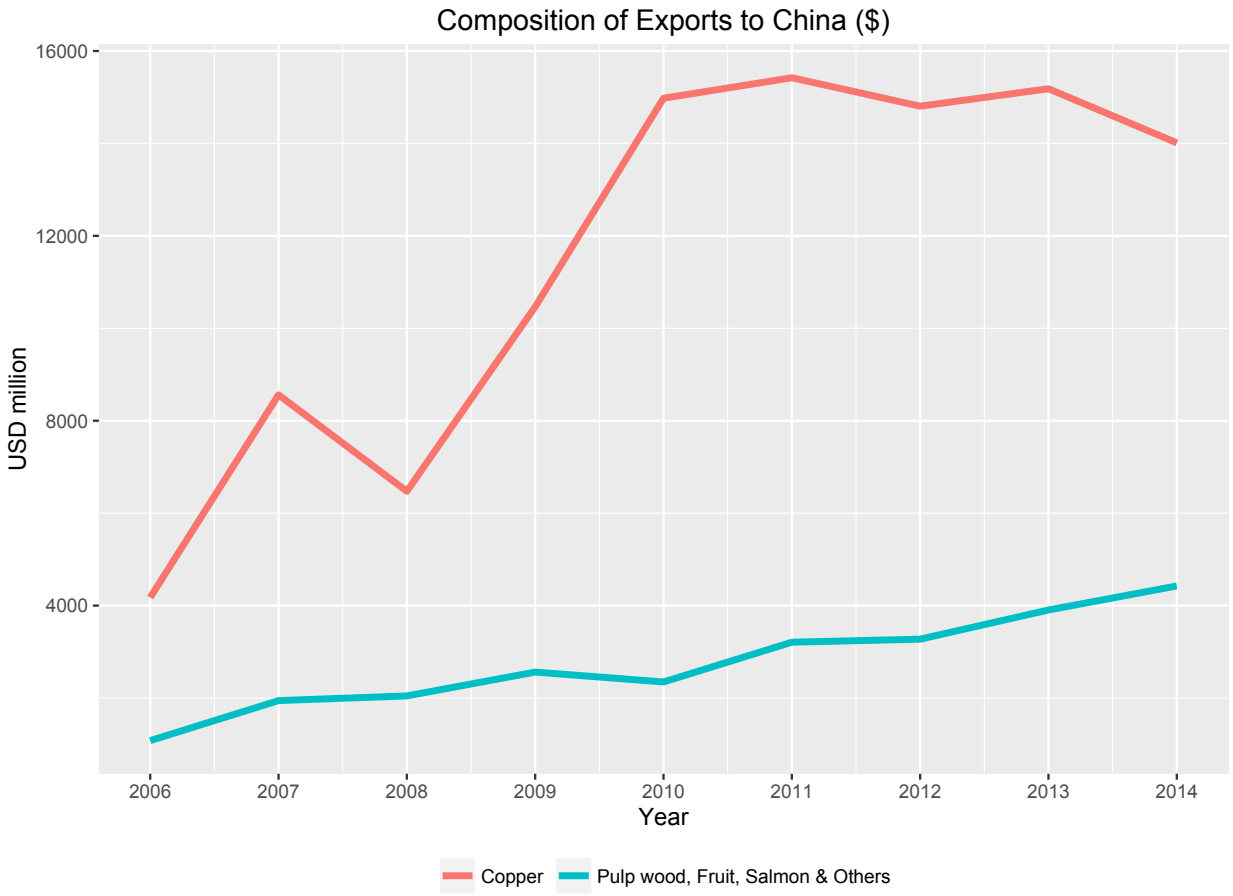
```
p1 <- p1 + scale_x_continuous(breaks=seq(2006,2014,1))
p1
```



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.

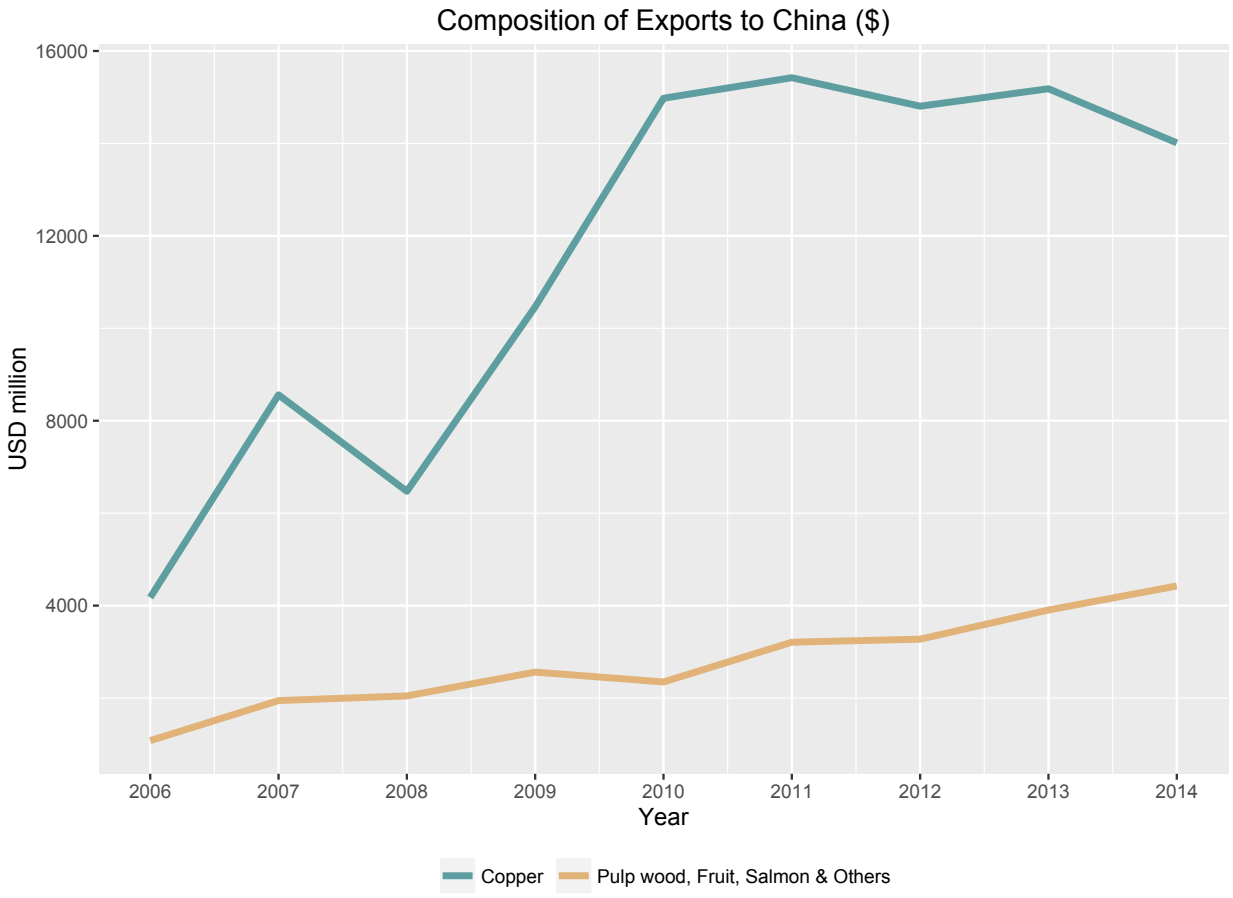
```
p1 <- p1 + ggtitle("Composition of Exports to China ($)") + labs(x="Year", y="USD million")
p1
```



Adjusting color palette

To change the colours, we use the `scale_colour_manual` command.

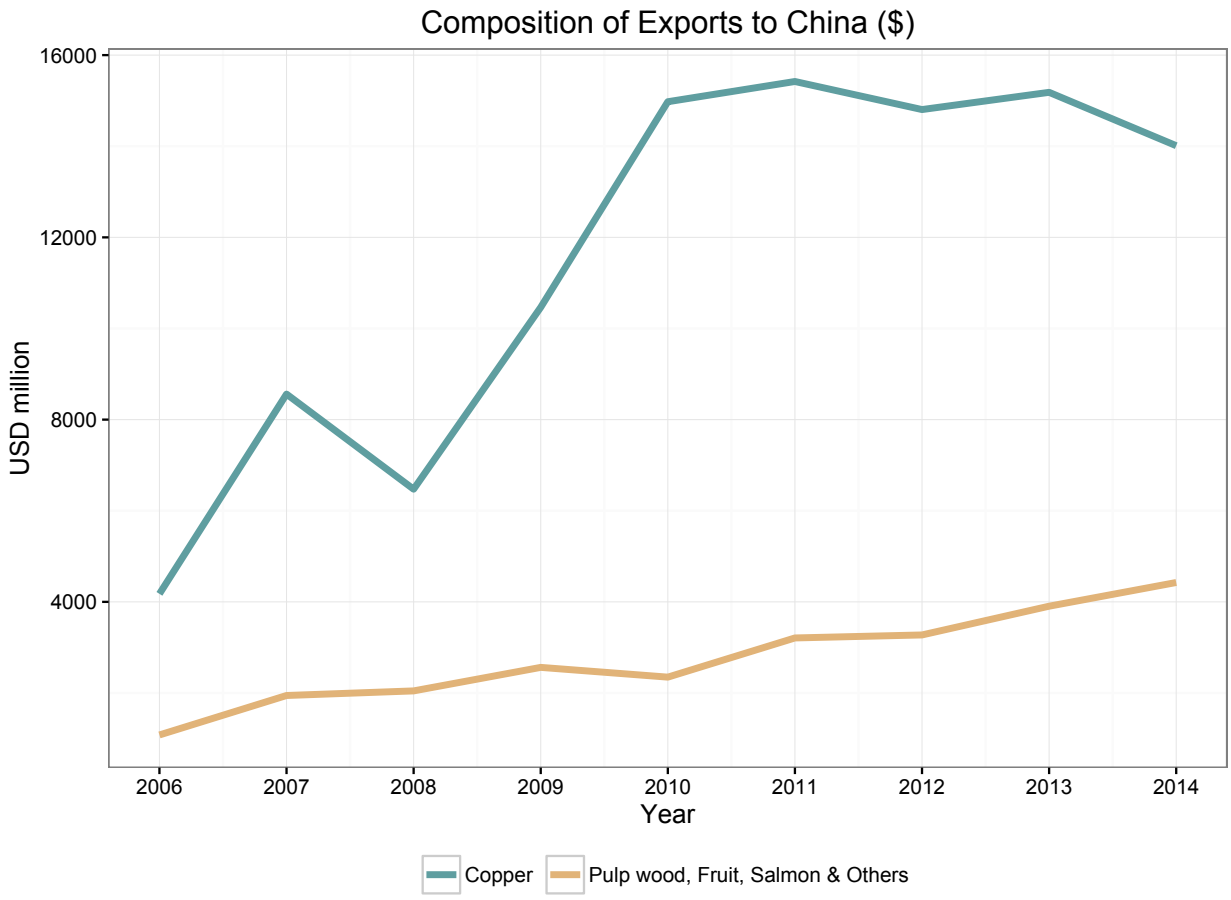
```
colour <- c("#5F9EA0", "#E1B378")
p1 <- p1 + scale_colour_manual(values=colour)
p1
```



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
```



Creating an XKCD style chart

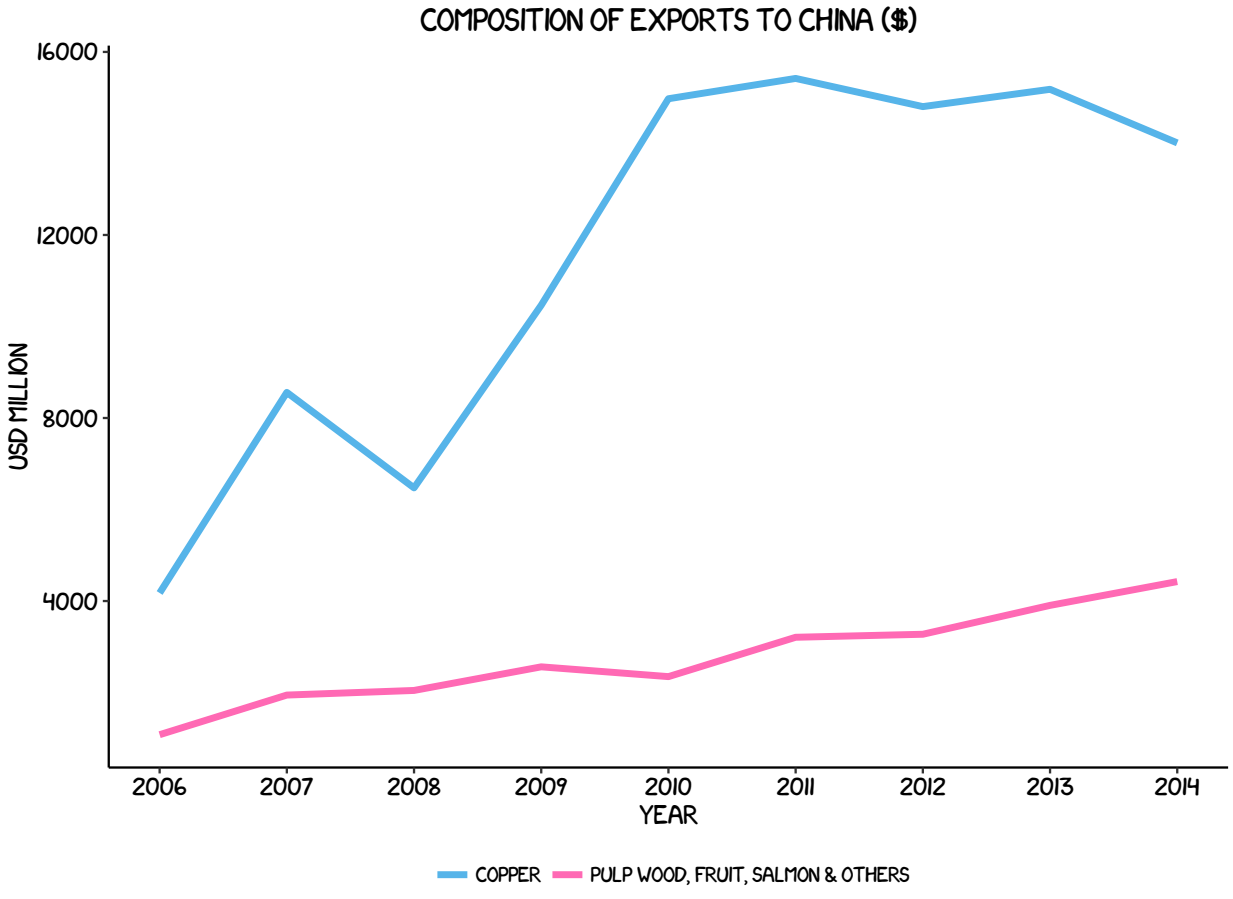
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
              dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

```
fill <- c("#56B4E9", "#ff69b4")
```

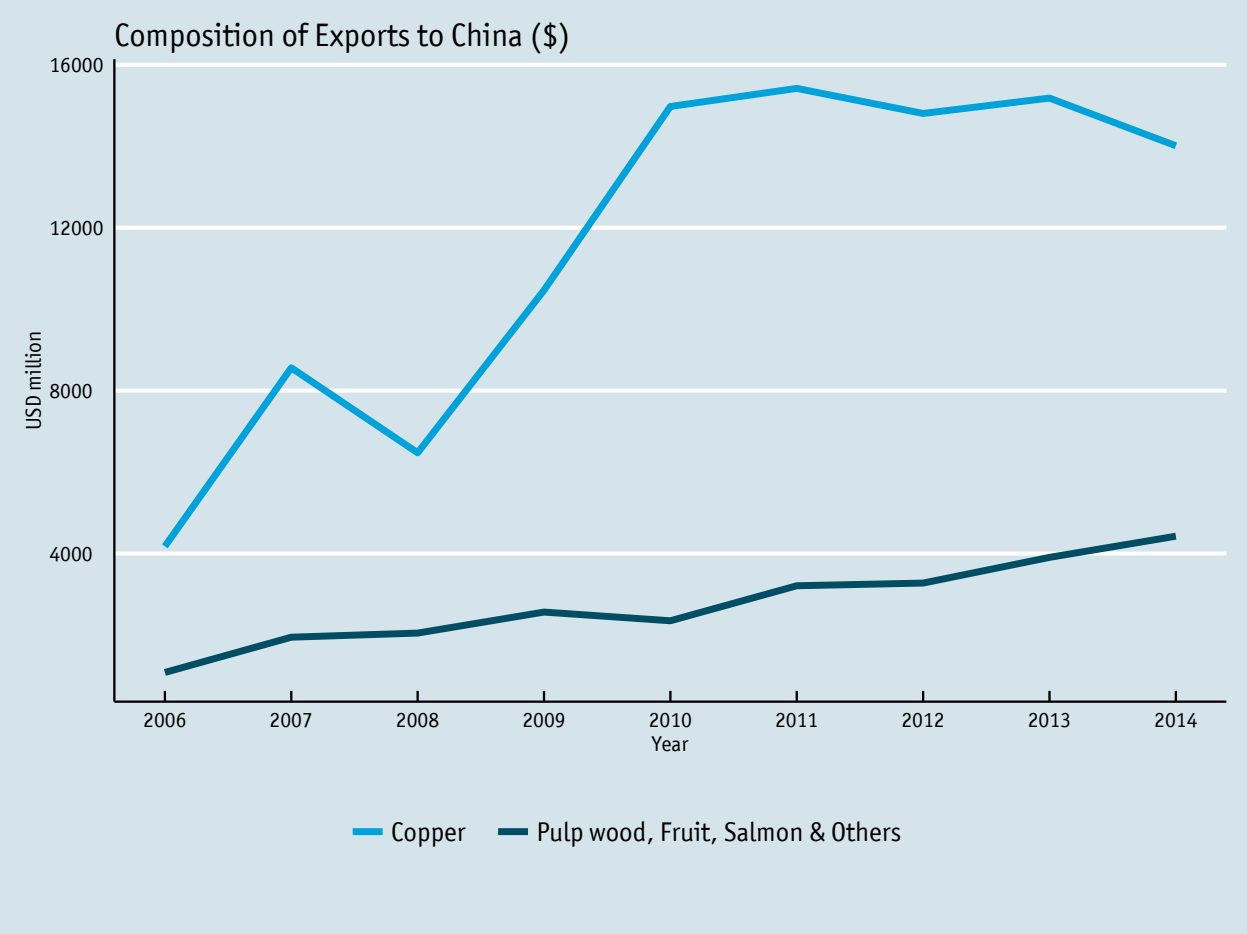
```
p1 <- ggplot() +
  geom_line(aes(y = export, x = year, colour = product), size=1.5, data = charts.data,
            stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_color_manual(values=fill) +
  theme(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"),
        axis.line.y = 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"))
p1
```



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.

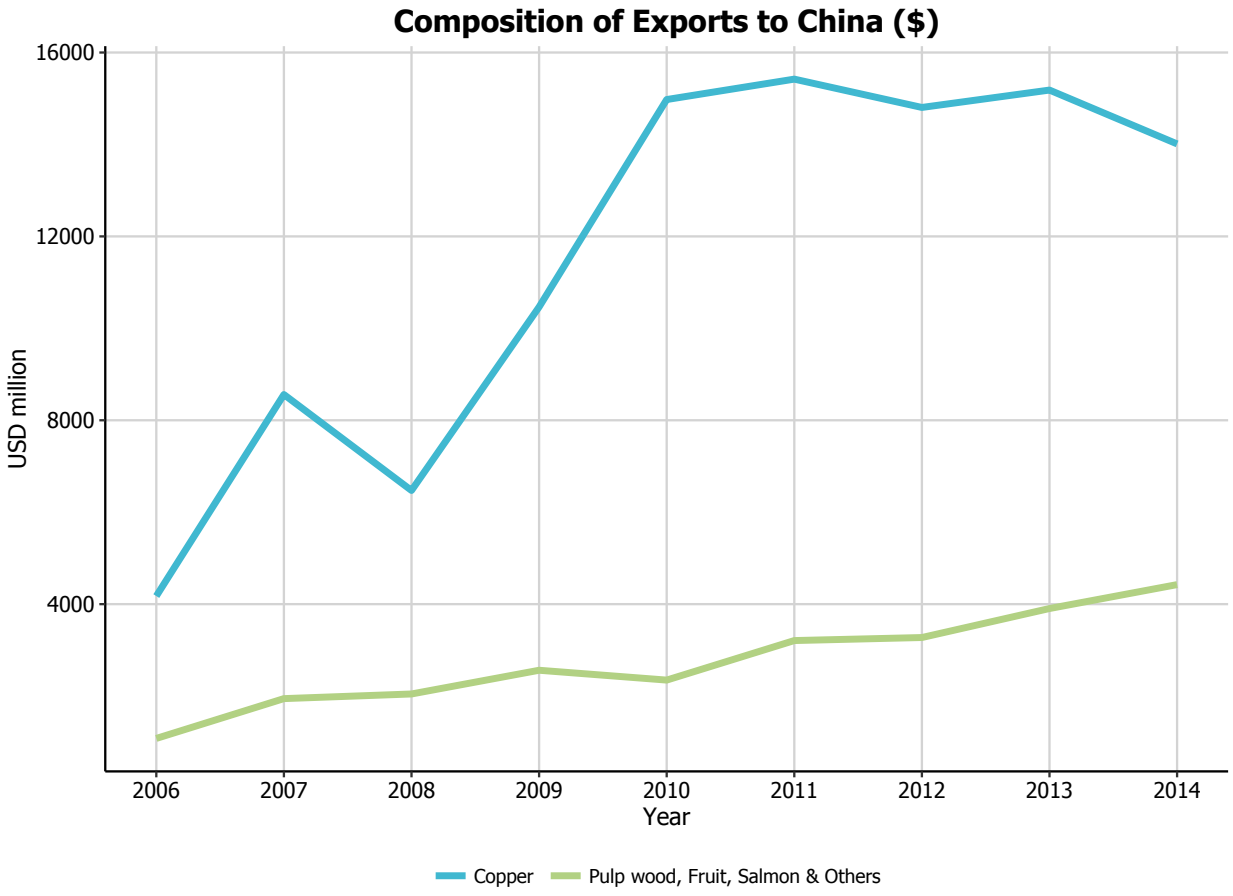
```
p1 <- ggplot() +
  geom_line(aes(y = export, x = year, colour = product), size=1.5, data = charts.data,
            stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  theme_economist() + scale_colour_economist()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



Creating your own theme

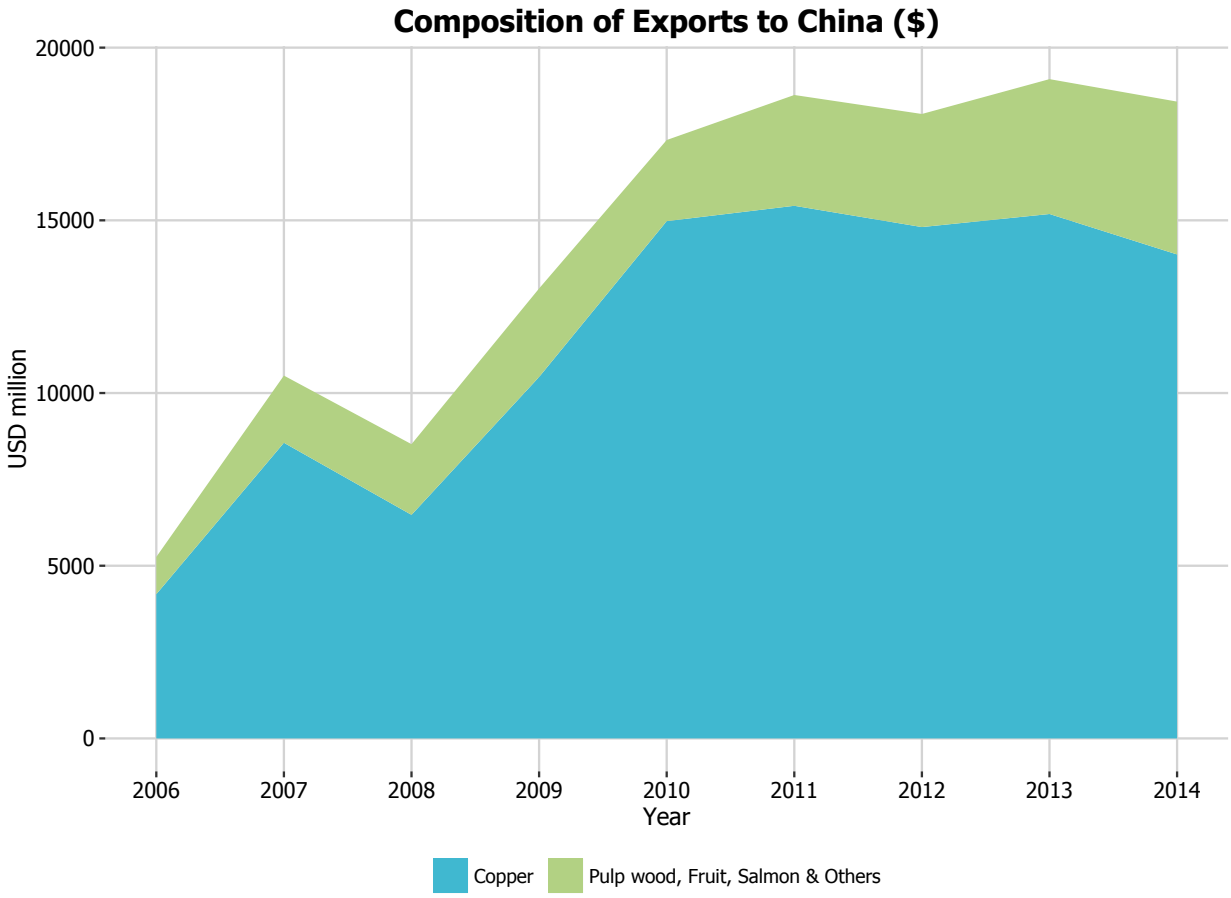
As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here we present our original result shown at the top of page.

```
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(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_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
```



Area Plots

In this part, 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.



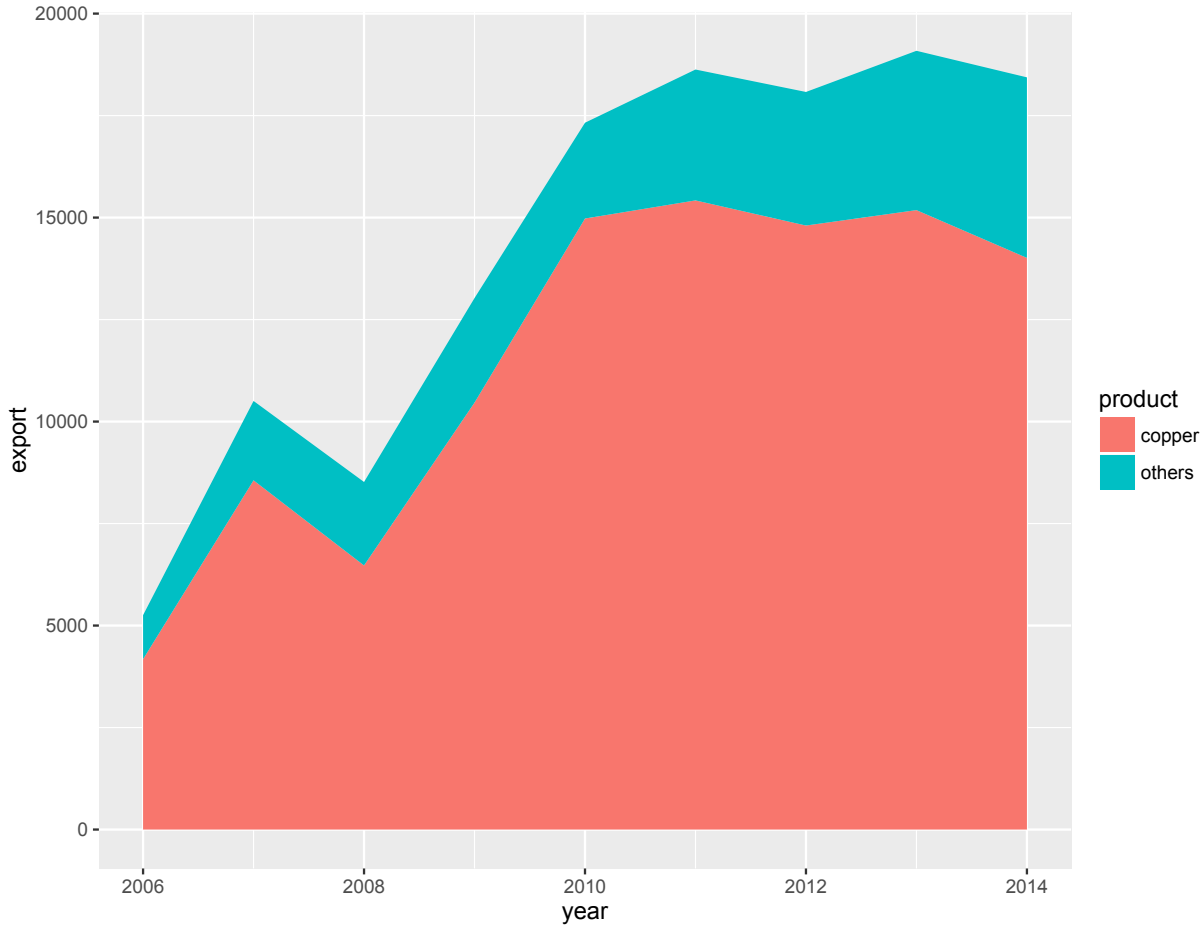
Basic graph

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-tutorial.csv")
```

In order to initialise a plot we tell ggplot that `charts.data` is our data, and specify the variables on each axis. We then instruct ggplot to render this as an area plot by adding the `geom_area` command.

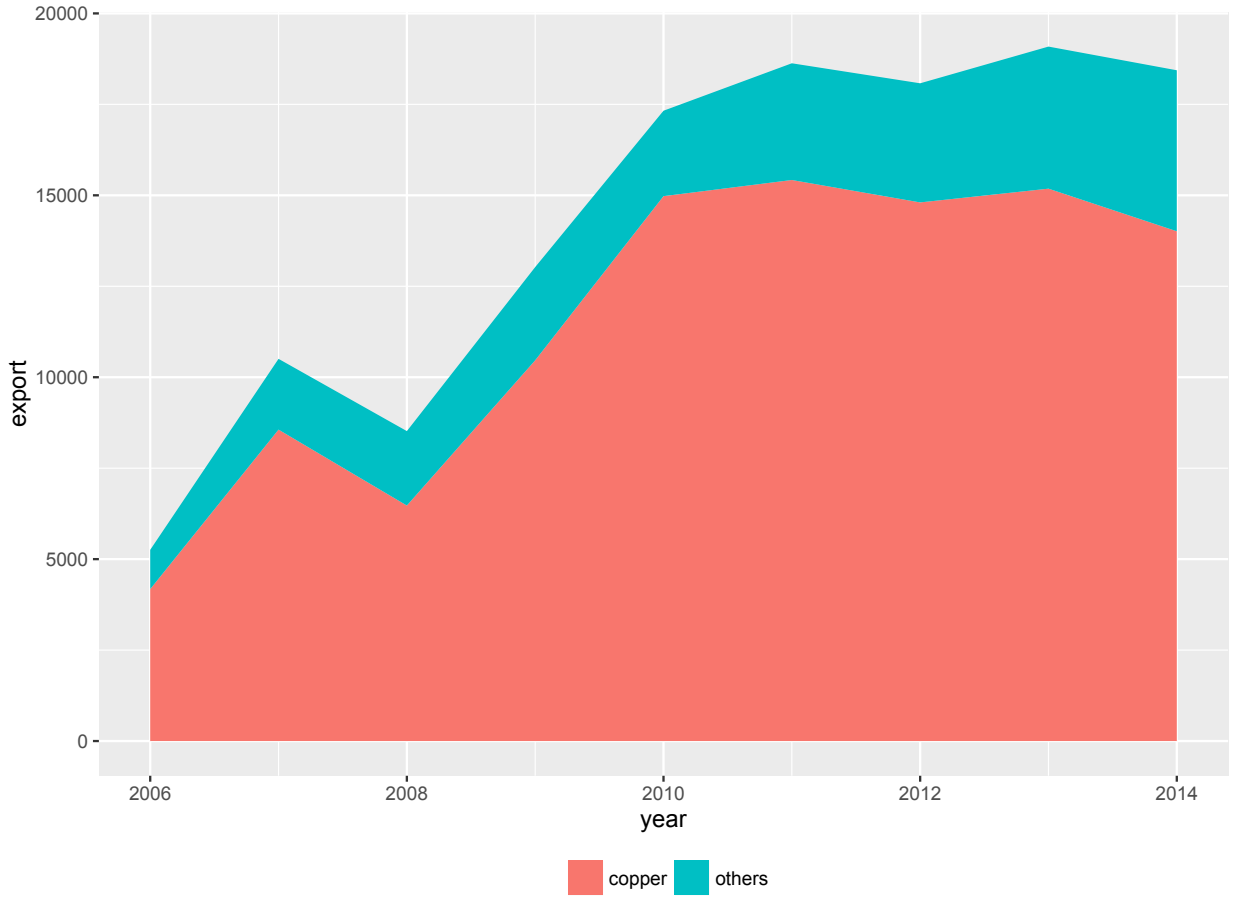
```
charts.data <- read.csv("copper-data-for-tutorial.csv")

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



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.

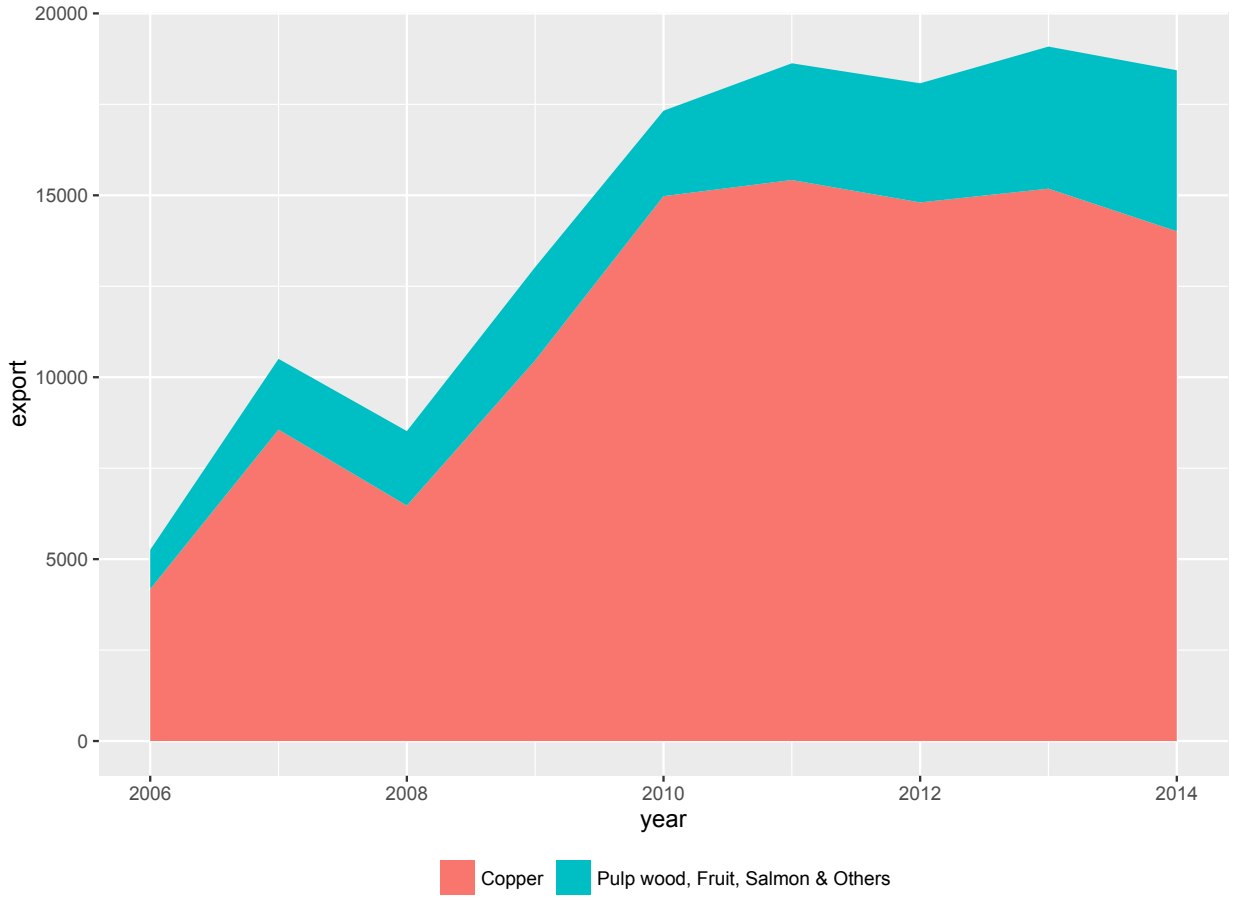
```
charts.data <- ddply(charts.data, .(year), transform, pos = cumsum(export) - (0.5 * export))

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



Changing variables display

To change the variables displayed name, we need to re-factor our data labels in `charts.data` data frame.

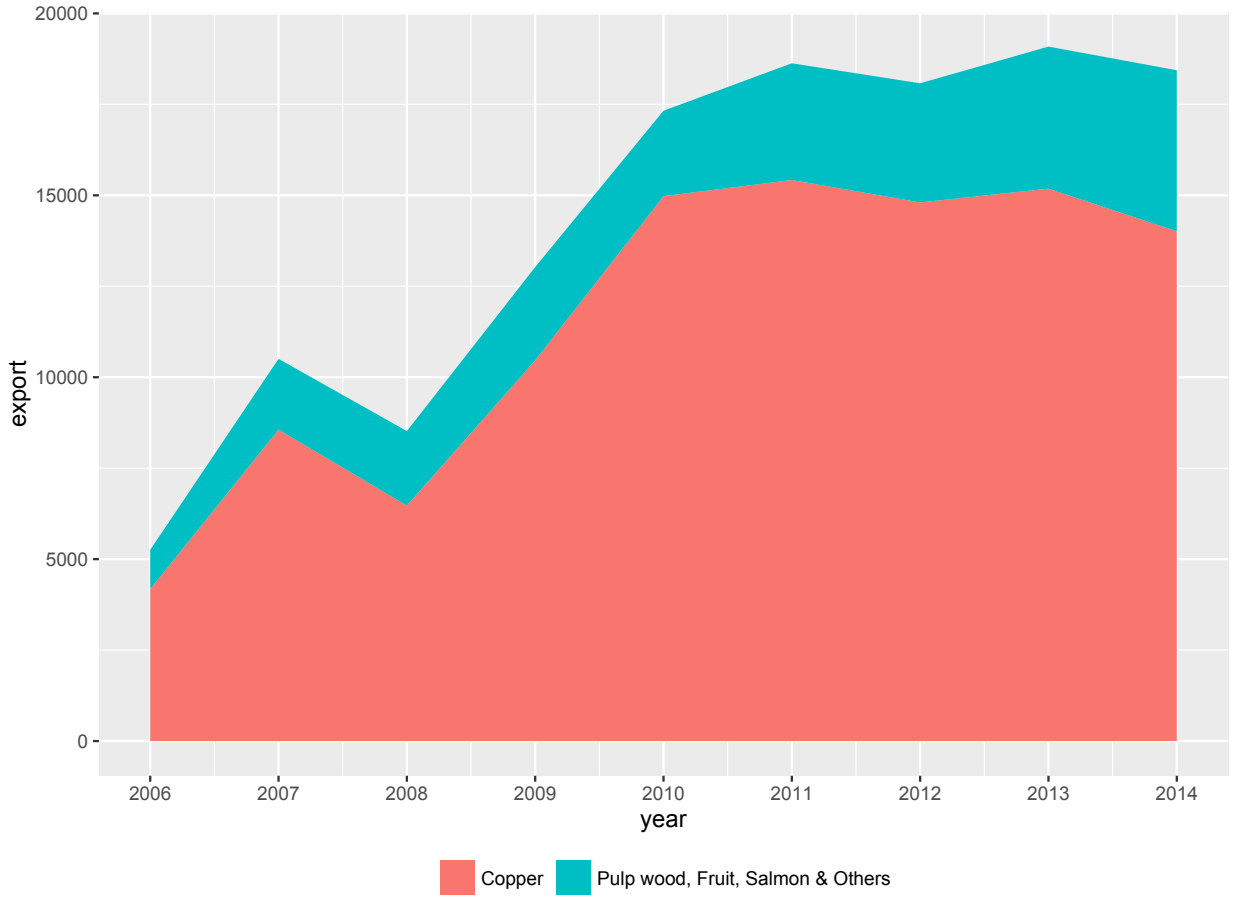
```
charts.data <- as.data.frame(charts.data)
charts.data$product <- factor(charts.data$product, levels = c("copper","others"),
                              labels = c("Copper ","Pulp wood, Fruit, Salmon & Others"))

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
```



Adjusting x-axis scale

To change the axis tick marks, we use the `scale_x_continuous` and/or `scale_y_continuous` commands.

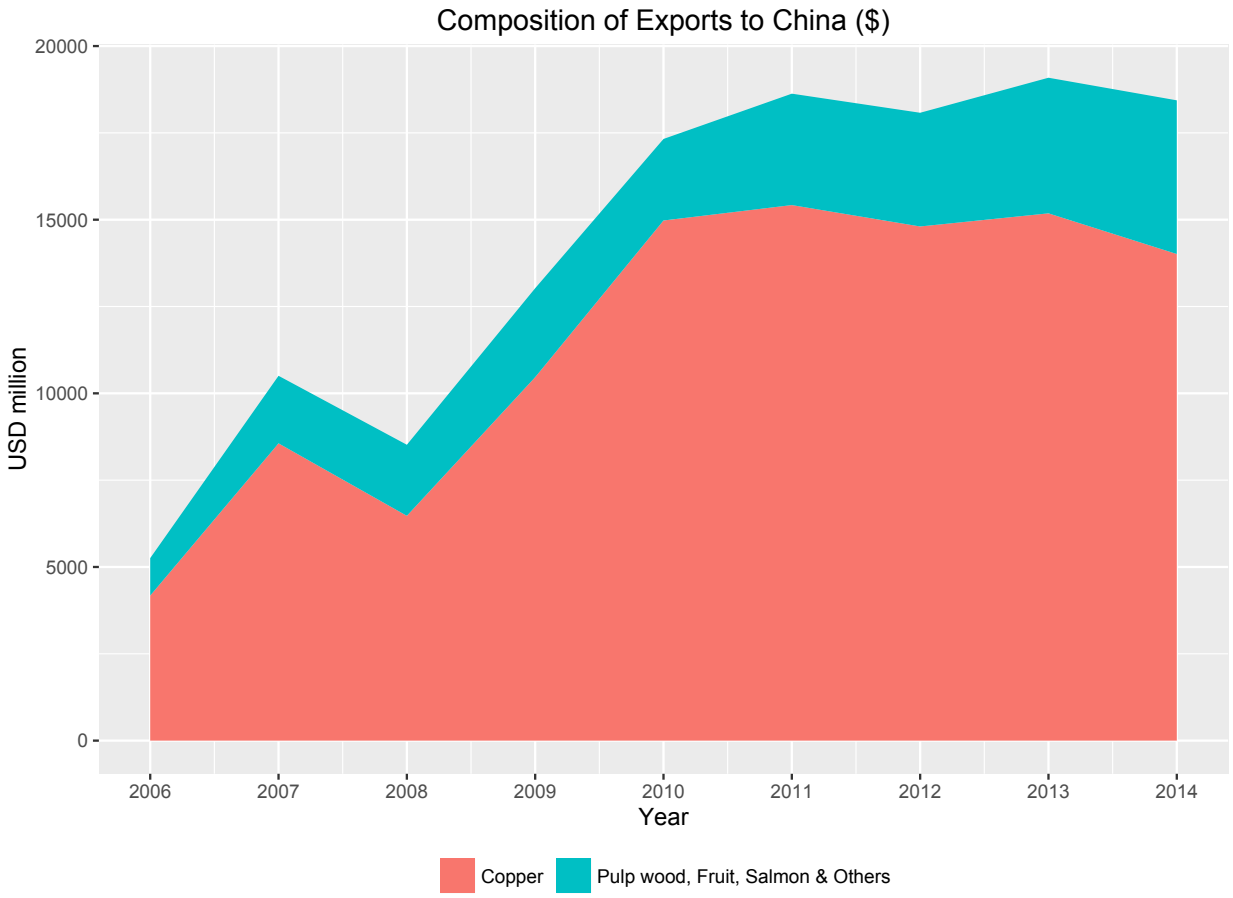
```
p2 <- p2 + scale_x_continuous(breaks=seq(2006,2014,1))
p2
```



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.

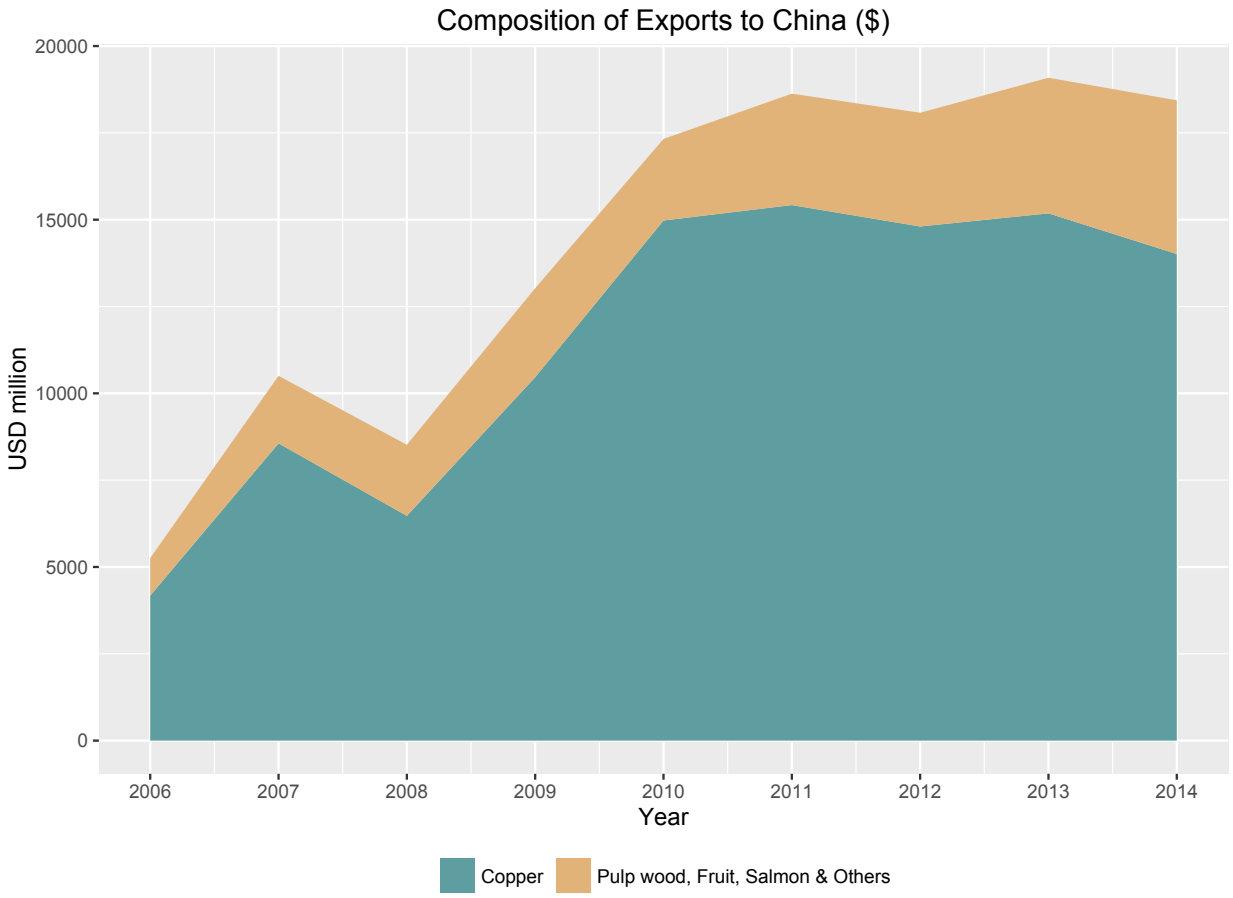
```
p2 <- p2 + ggtitle("Composition of Exports to China ($)") +
           labs(x="Year", y="USD million")
p2
```



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.

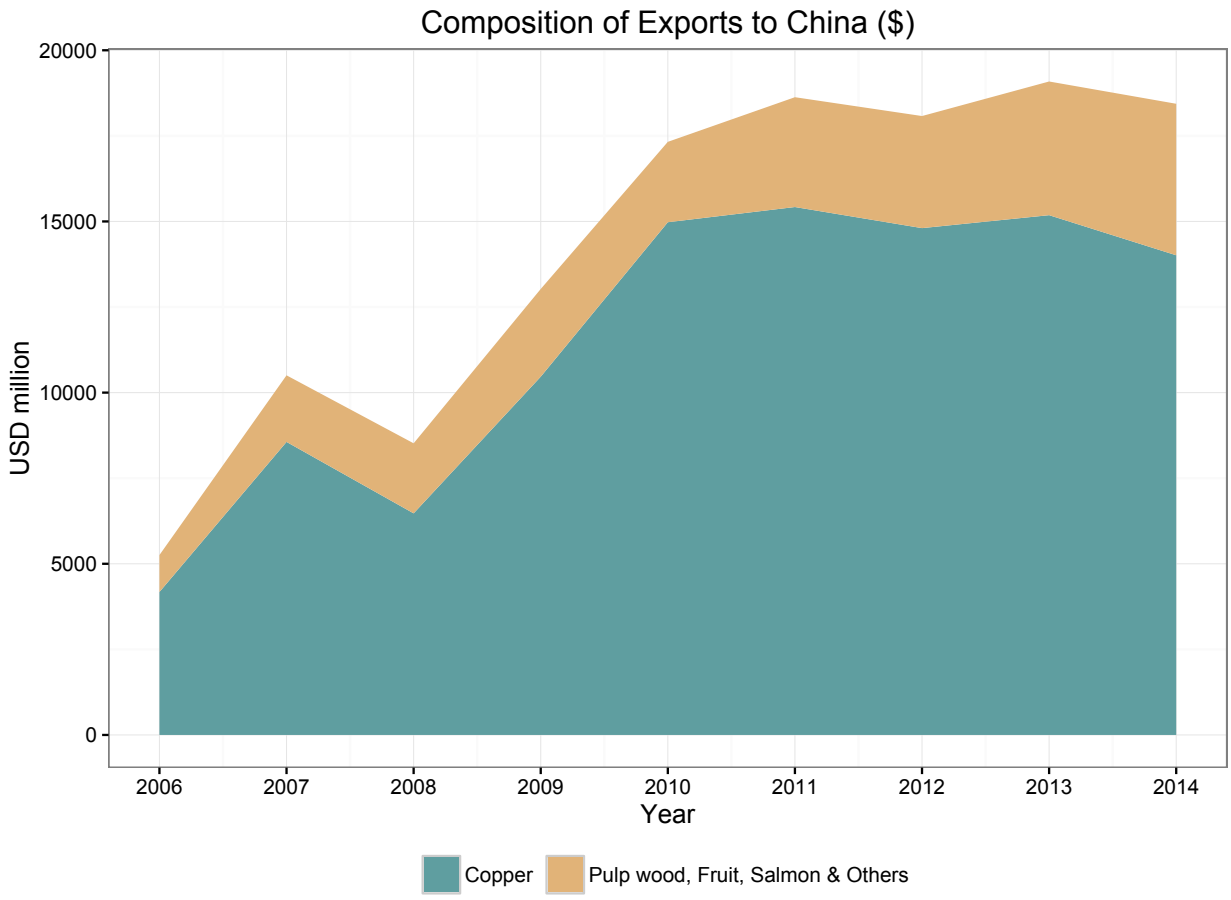
```
fill <- c("#5F9EA0", "#E1B378")
p2 <- p2 + scale_fill_manual(values=fill)
p2
```



Using the white theme

As explained in the previous post, we can also change the overall look of the site 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
```



Creating an XKCD style chart

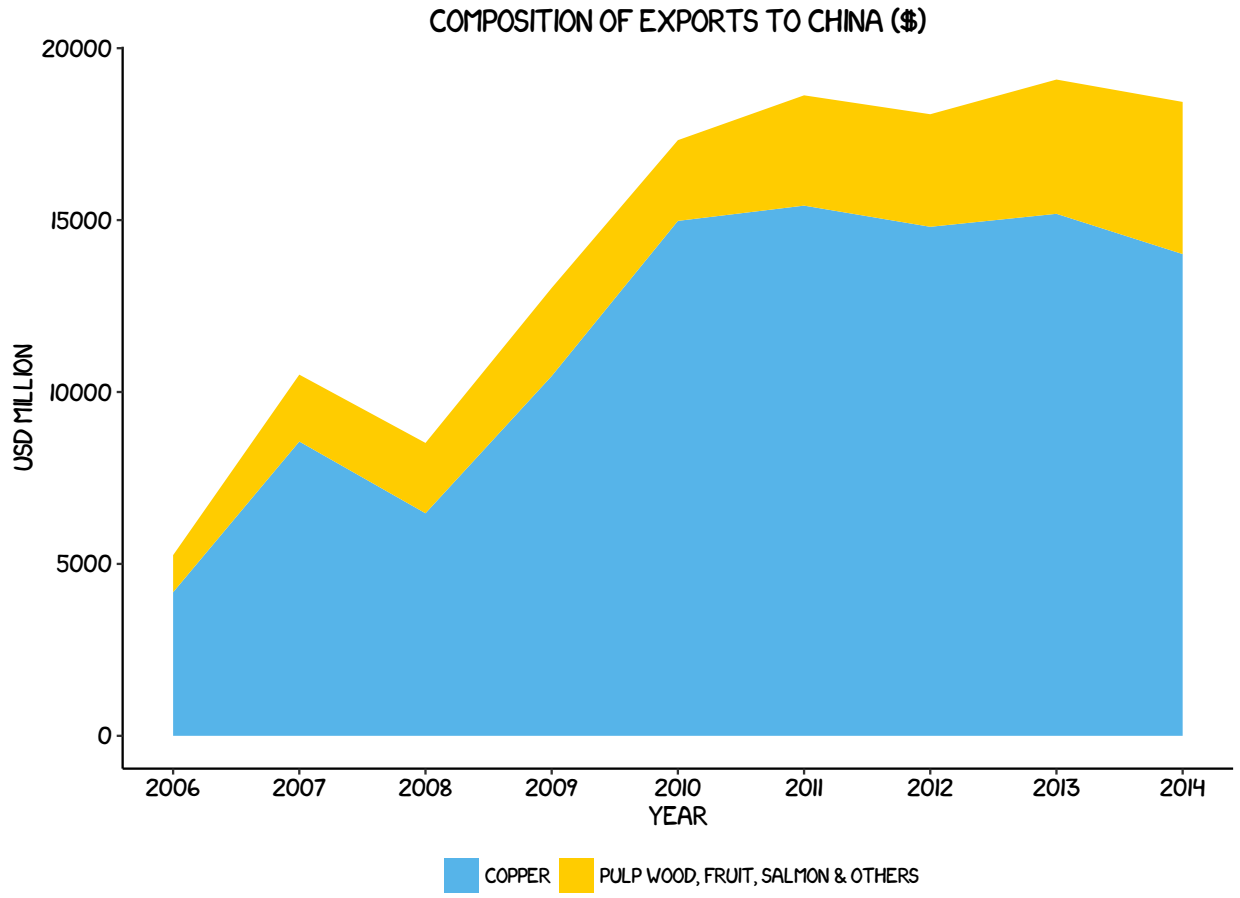
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
              dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

```
fill <- c("#56B4E9", "#ffcc00")
```

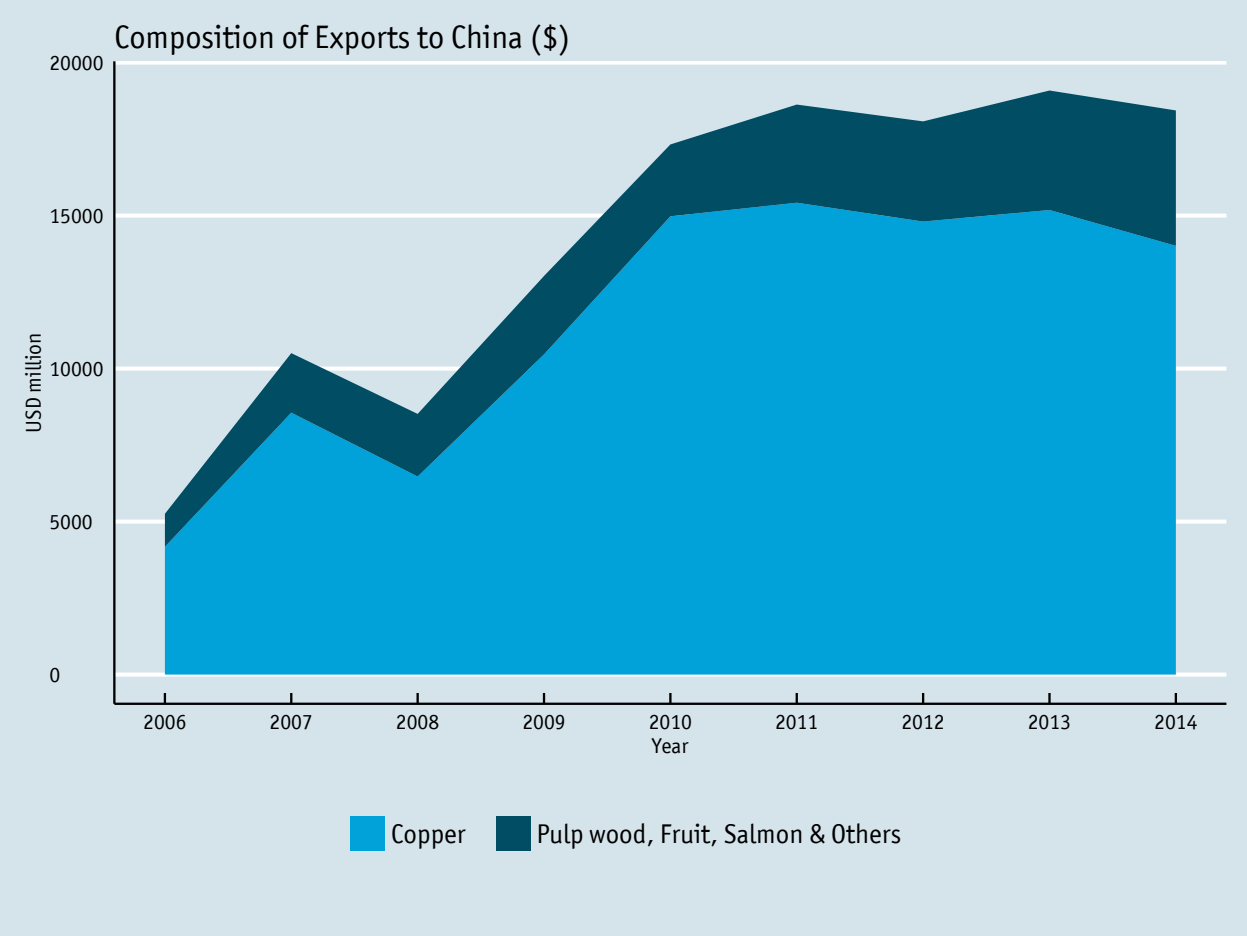
```
p2 <- ggplot() +
  geom_area(aes(y = export, x = year, fill = product), data = charts.data, stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  scale_fill_manual(values=fill) +
  theme(axis.text.x=element_text(colour="black", size = 10),
        axis.text.y=element_text(colour="black", size = 10),
        axis.line.x = element_line(size=.5, colour = "black"),
        axis.line.y = 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"))
p2
```



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.

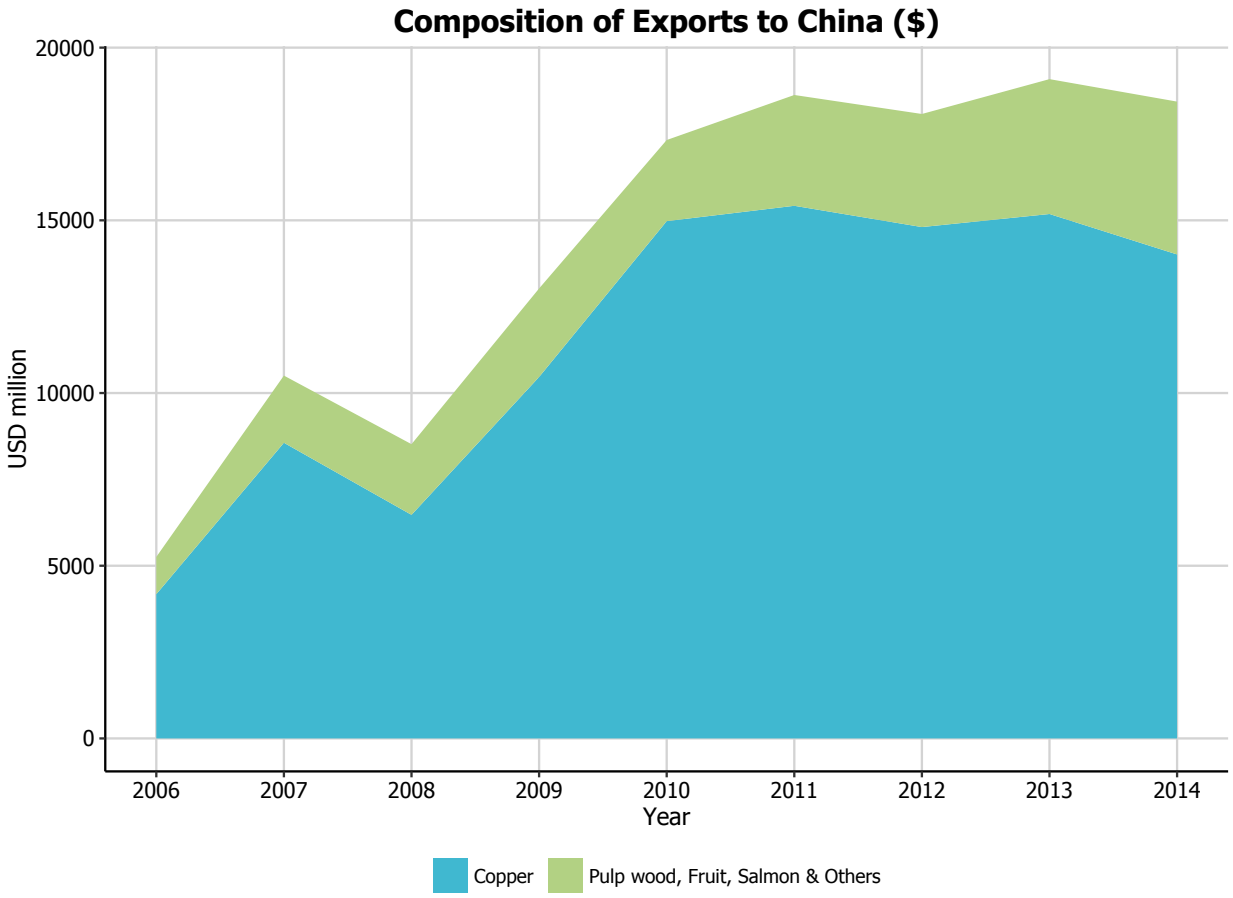
```
p2 <- ggplot() +
  geom_area(aes(y = export, x = year, fill = product), data = charts.data, stat="identity") +
  scale_x_continuous(breaks=seq(2006,2014,1)) +
  labs(x="Year", y="USD million") +
  ggtitle("Composition of Exports to China ($)") +
  theme_economist() + scale_fill_economist() +
  theme(axis.line.x = element_line(size=.5, colour = "black"),
        axis.line.y = 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
```



Creating your own theme

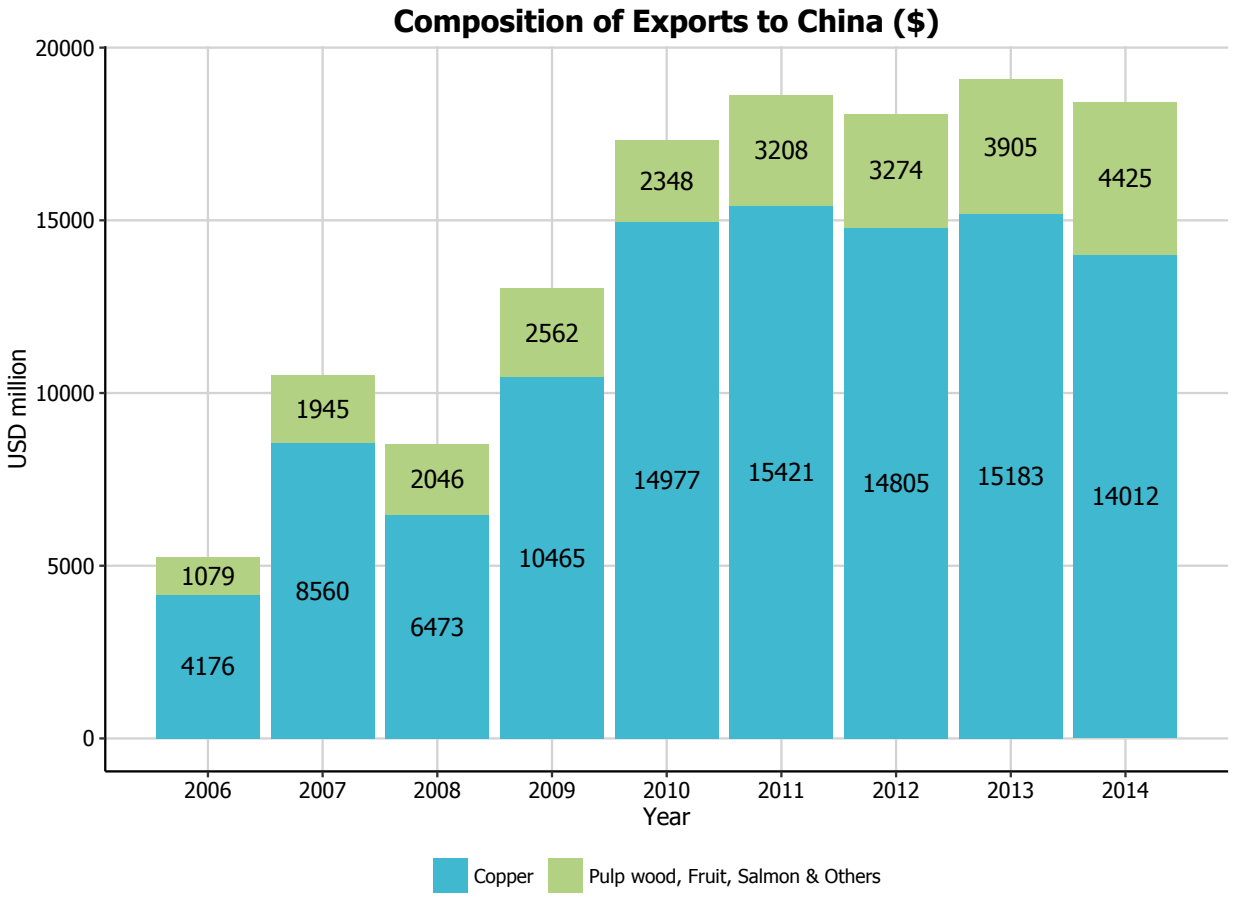
As before, you can modify your plots a lot as `ggplot2` allows many customisations. Here we present our original result shown at the top of page.

```
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_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
```



Bar Plots

In this part, 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.



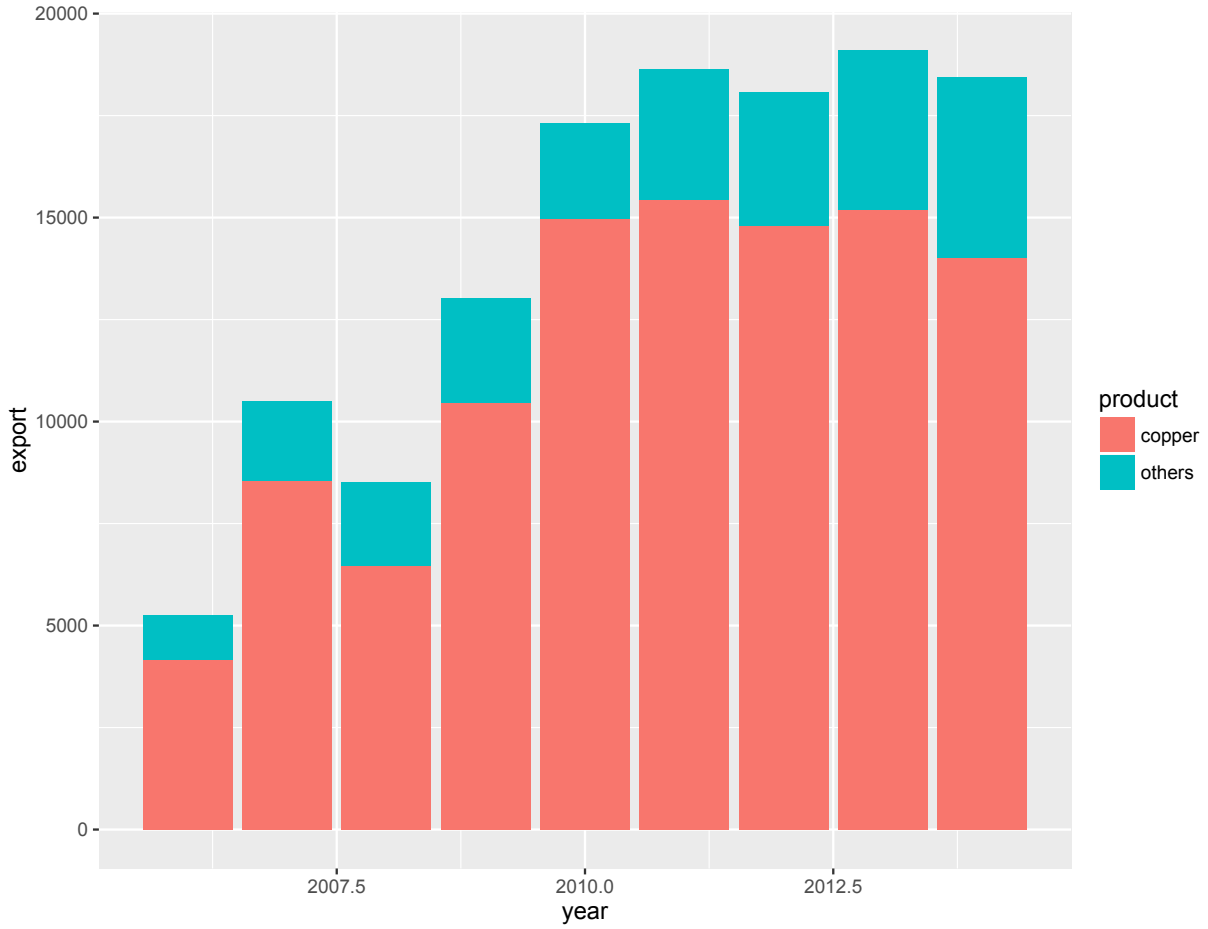
Basic graph

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-tutorial.csv")
```

In order to initialise a plot we tell ggplot that `charts.data` is our data, and specify the variables on each axis. We then instruct ggplot to render this as an bar plot by adding the `geom_area` command.

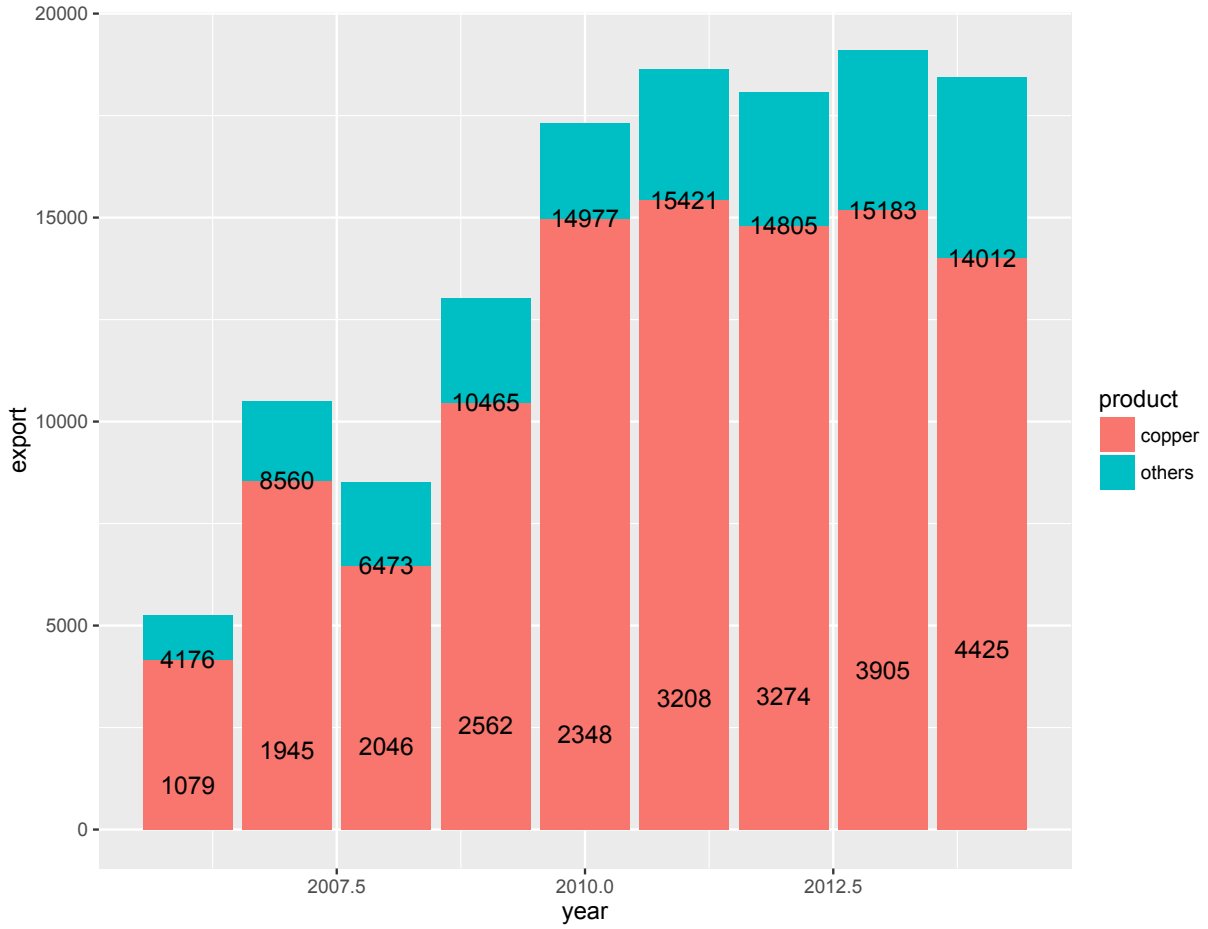
```
p3 <- ggplot() + geom_bar(aes(y = export, x = year, fill = product), data = charts.data,
                          stat="identity")
p3
```



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.

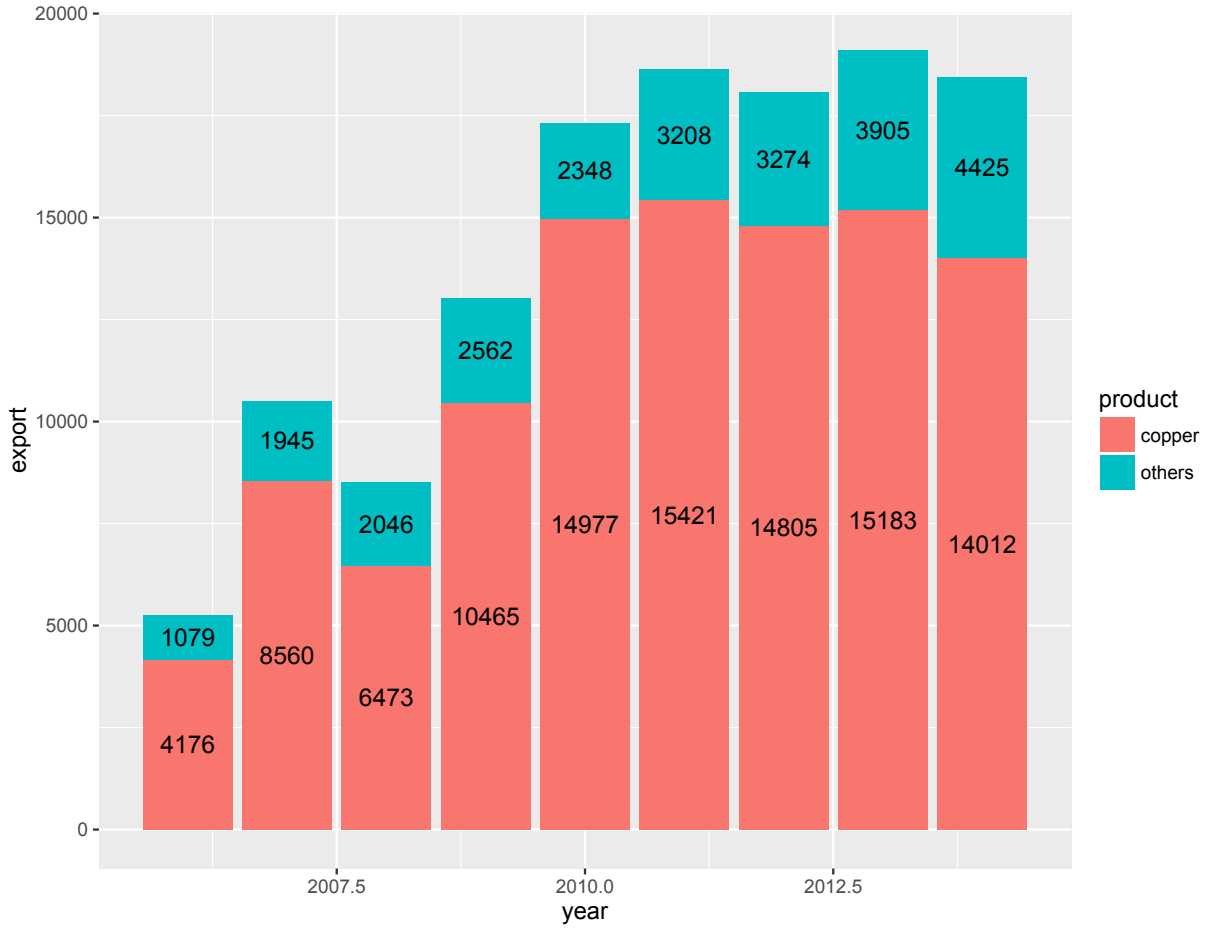
```
p3 <- p3 + geom_text(data=charts.data, aes(x = year, y = export, label = export), size=4)
p3
```



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 create a new variable called `pos`. 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())`.

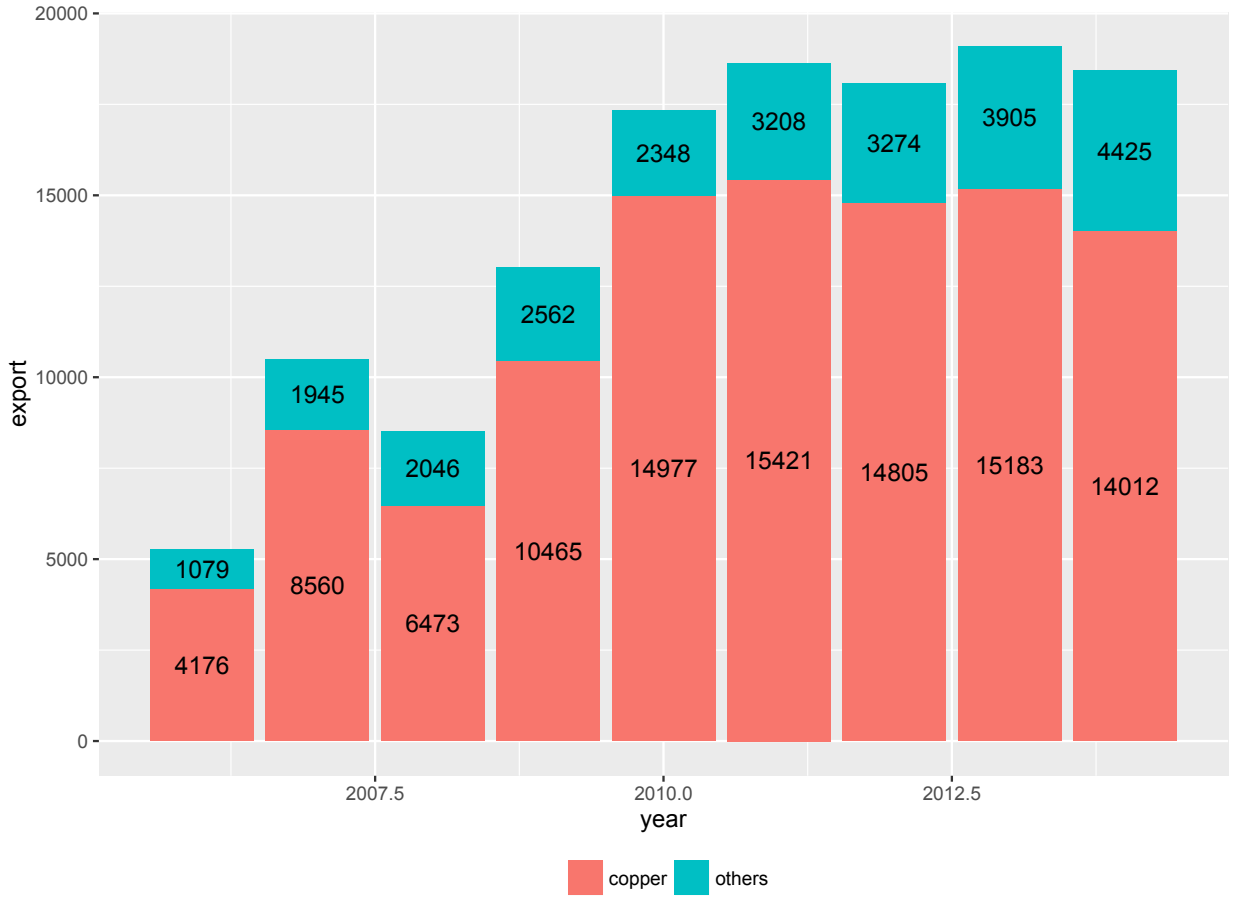
```
charts.data <- ddply(charts.data, .(year), transform, pos = cumsum(export) - (0.5 * export))

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
```



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.

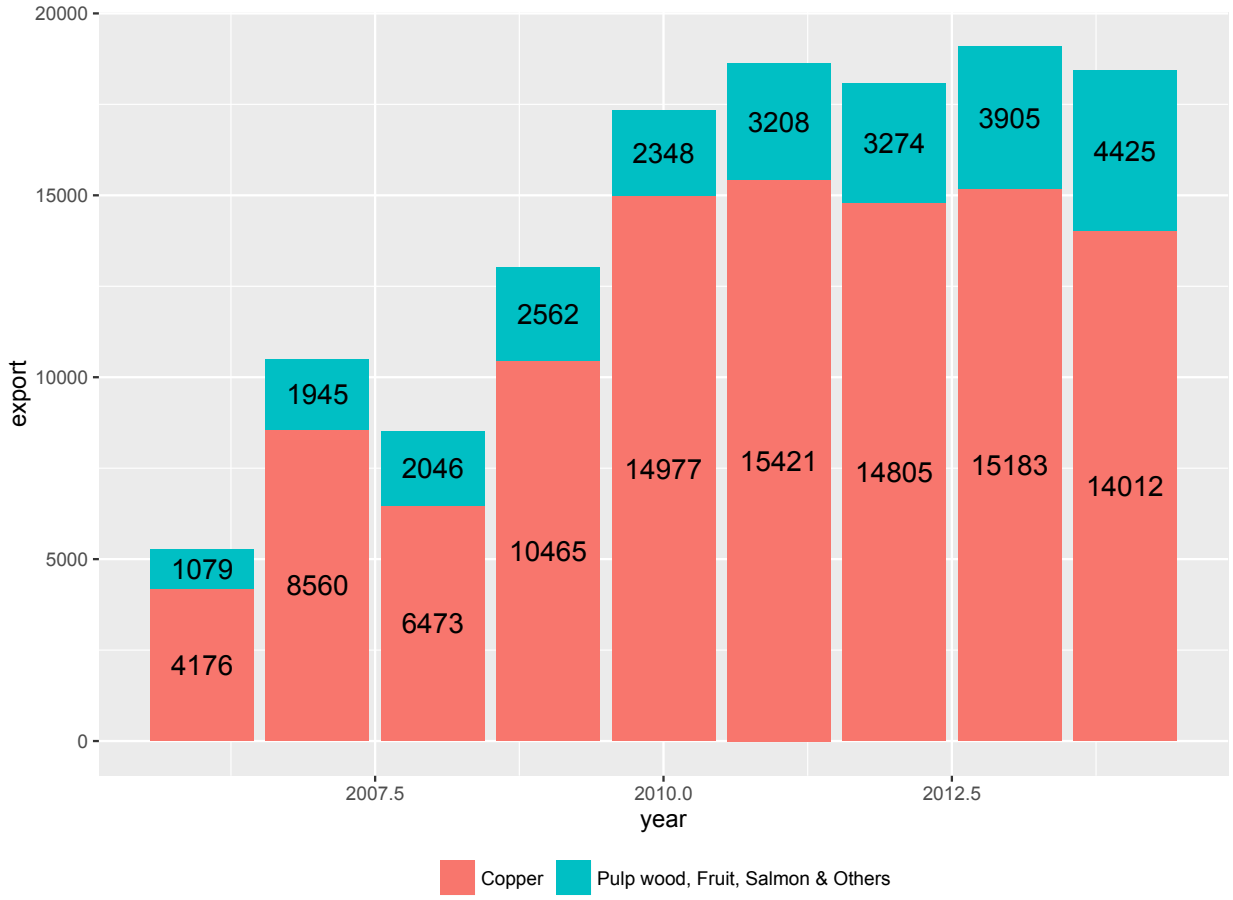
```
p3 <- p3 + theme(legend.position="bottom", legend.direction="horizontal",
                 legend.title = element_blank())
p3
```



Changing variables display

To change the variables' displayed name, 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
```



Adjusting x-axis scale

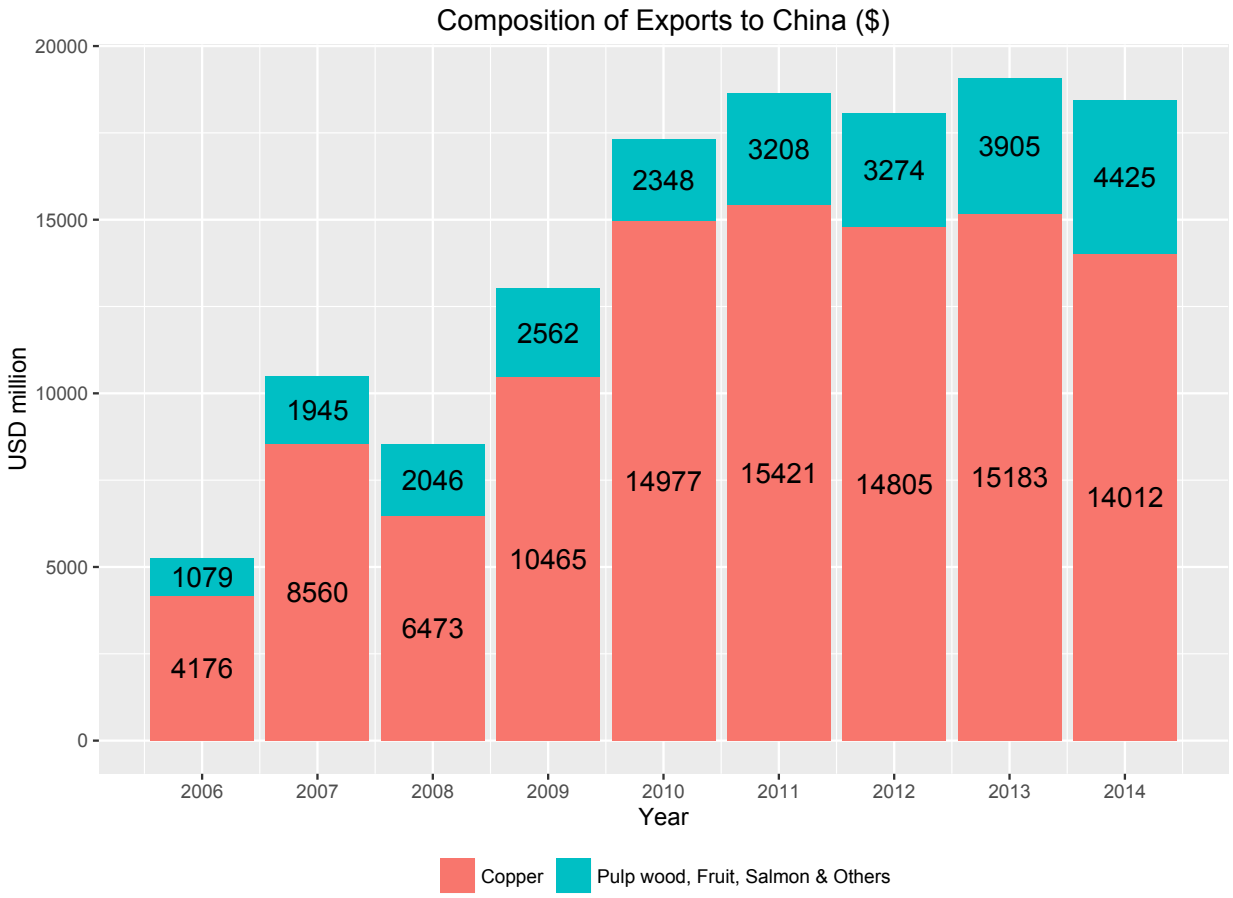
To change the axis tick marks, we use the `scale_x_continuous` and/or `scale_y_continuous` commands.

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

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.

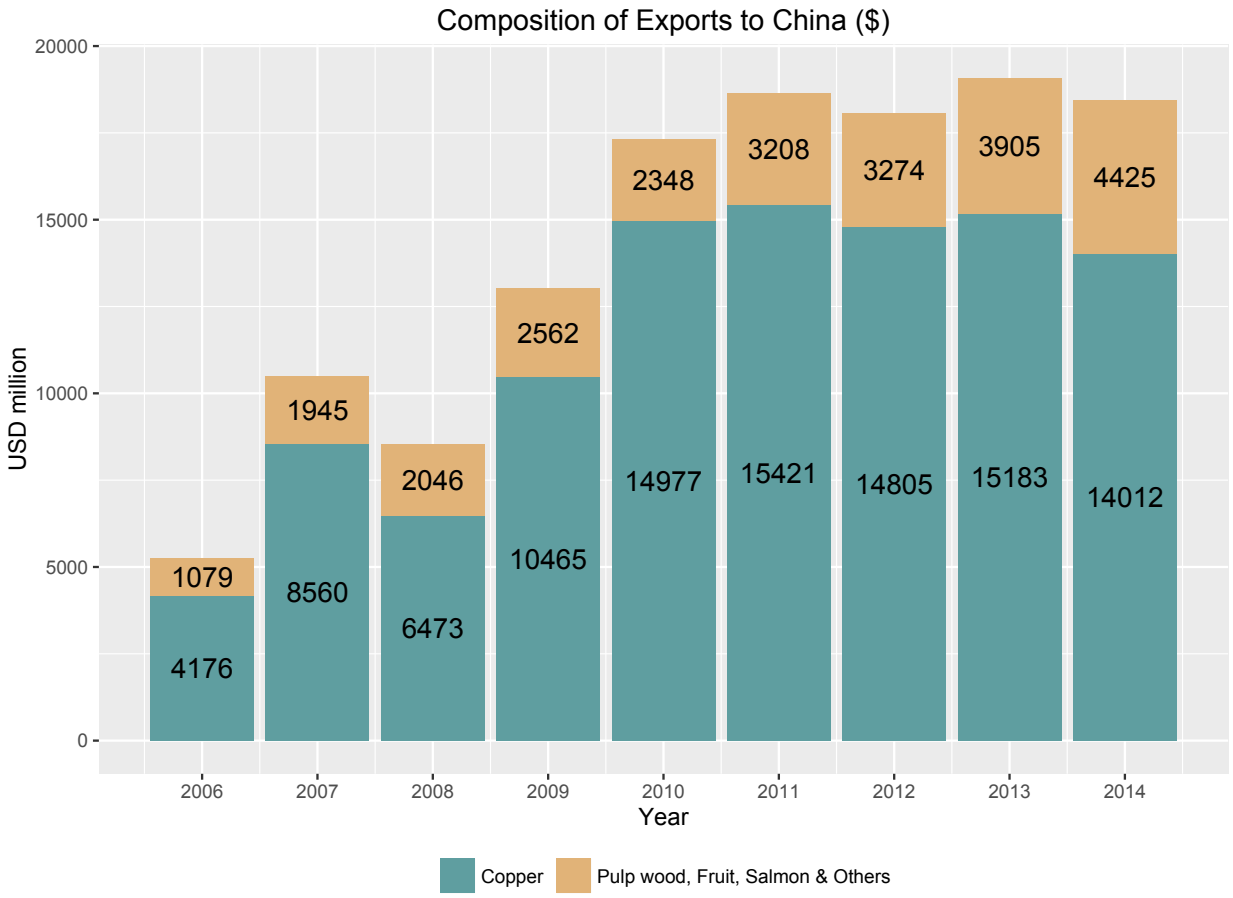
```
p3 <- p3 + ggtitle("Composition of Exports to China ($)") + labs(x="Year", y="USD million")
p3
```



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.

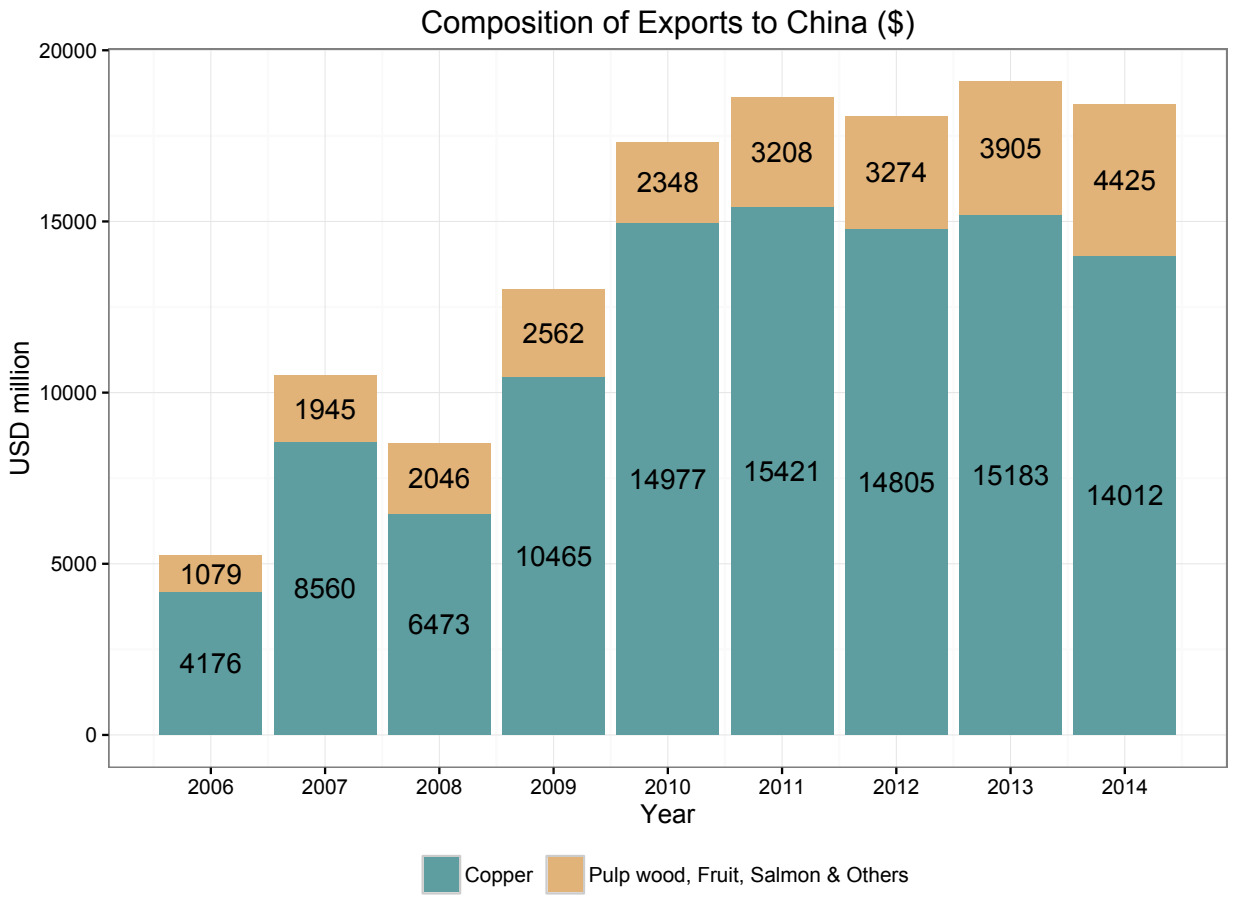
```
fill <- c("#5F9EA0", "#E1B378")
p3 <- p3 + scale_fill_manual(values=fill)
p3
```



Using the white theme

As explained in the previous posts, 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
```



Creating an XKCD style chart

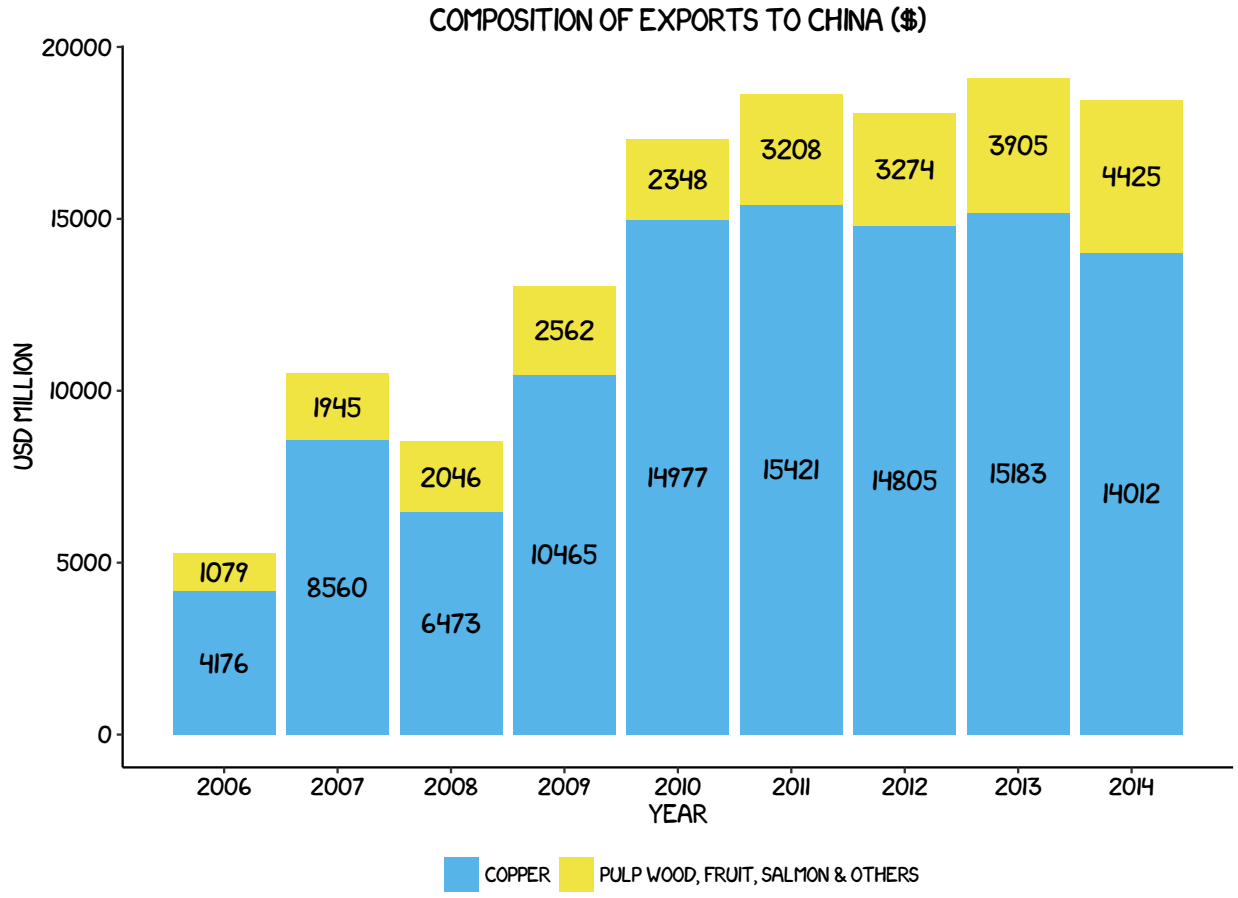
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
              dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

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

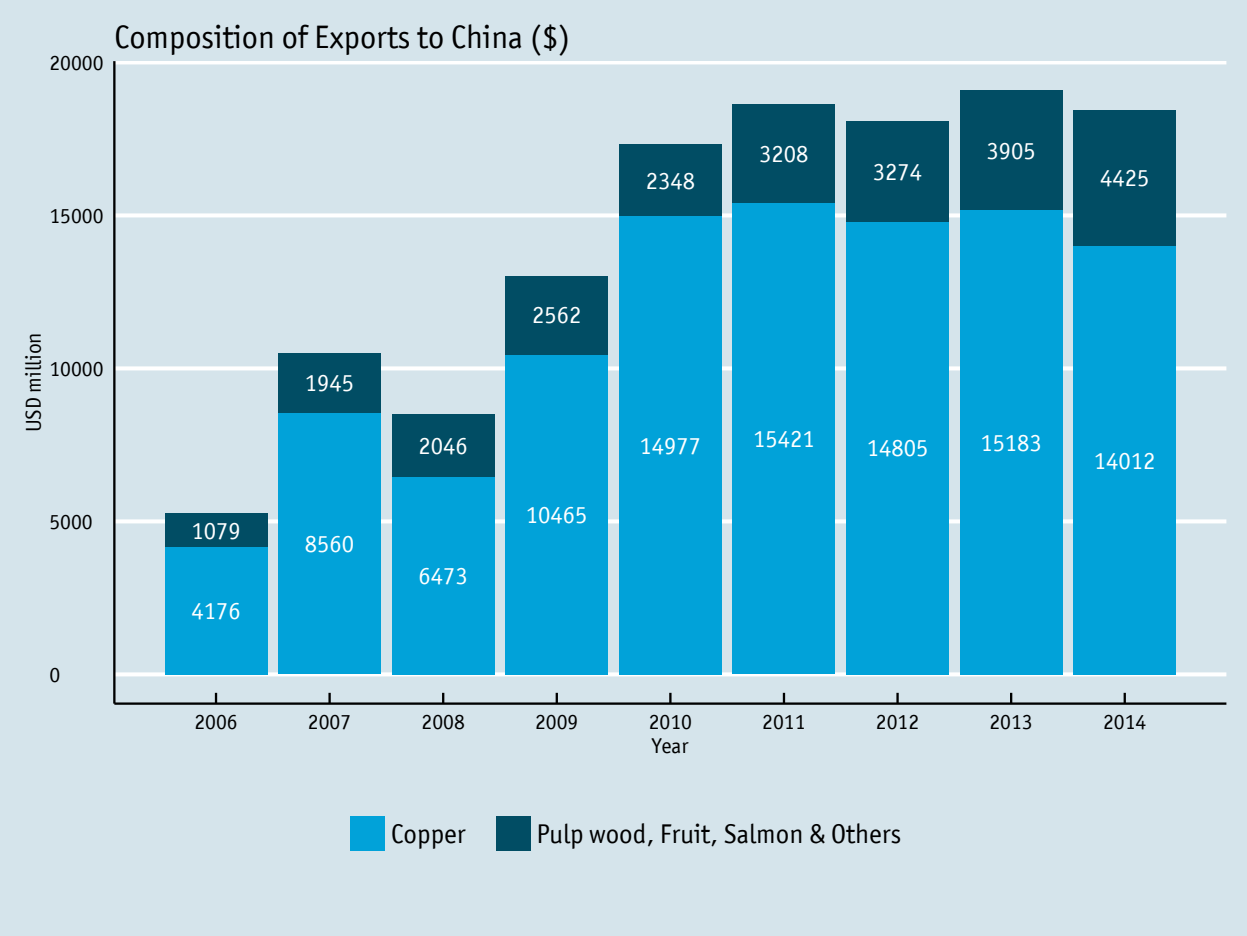
```
p3 <- ggplot() +
  geom_bar(aes(y = export, x = year, fill = product), data = charts.data, stat="identity") +
  geom_text(data=charts.data, aes(x = year, y = pos, label = export), colour="black",
            family="xkcd-Regular", size = 4, show.legend = F) +
  scale_x_continuous(breaks=seq(2006,2014,1))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



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"),
        axis.line.y = 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
```

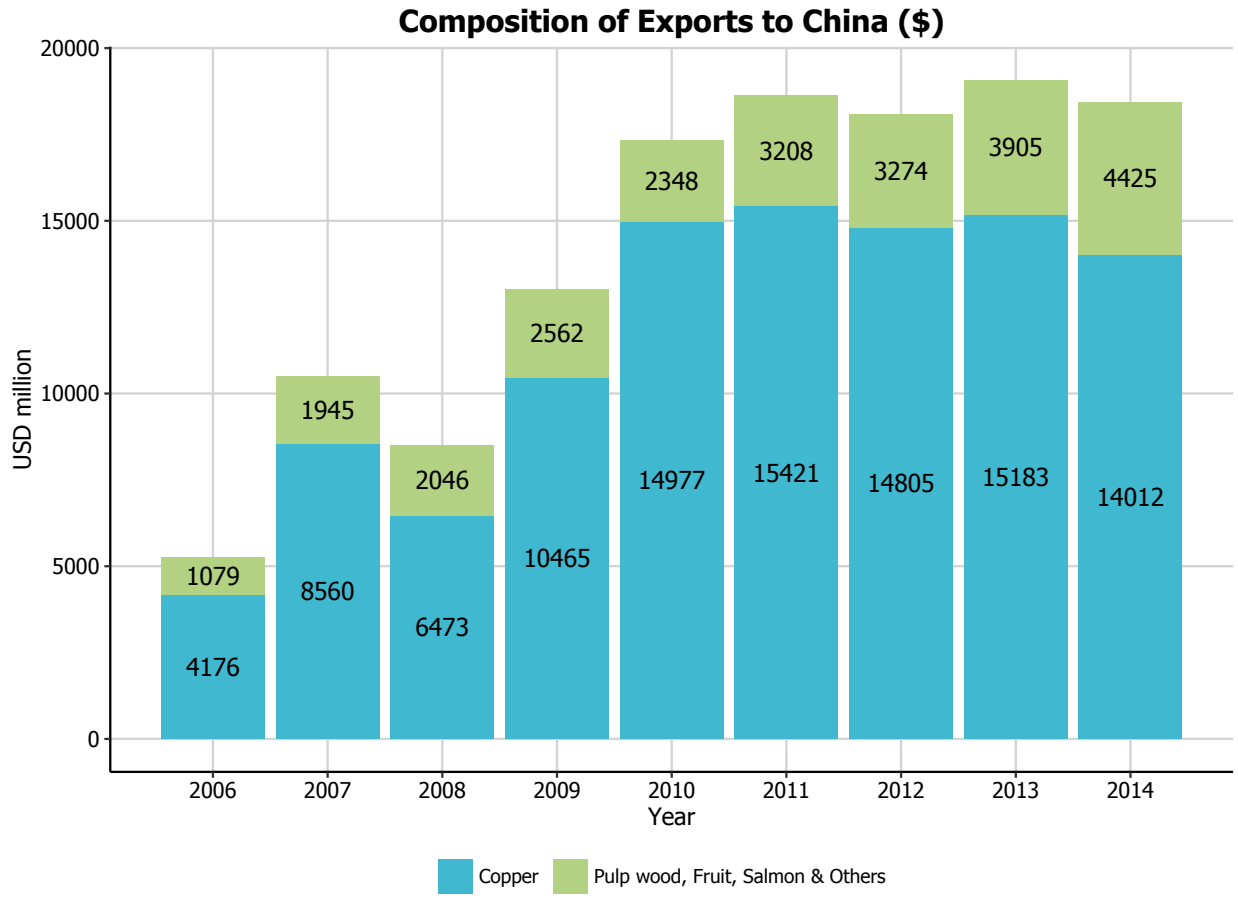


Creating your own theme

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

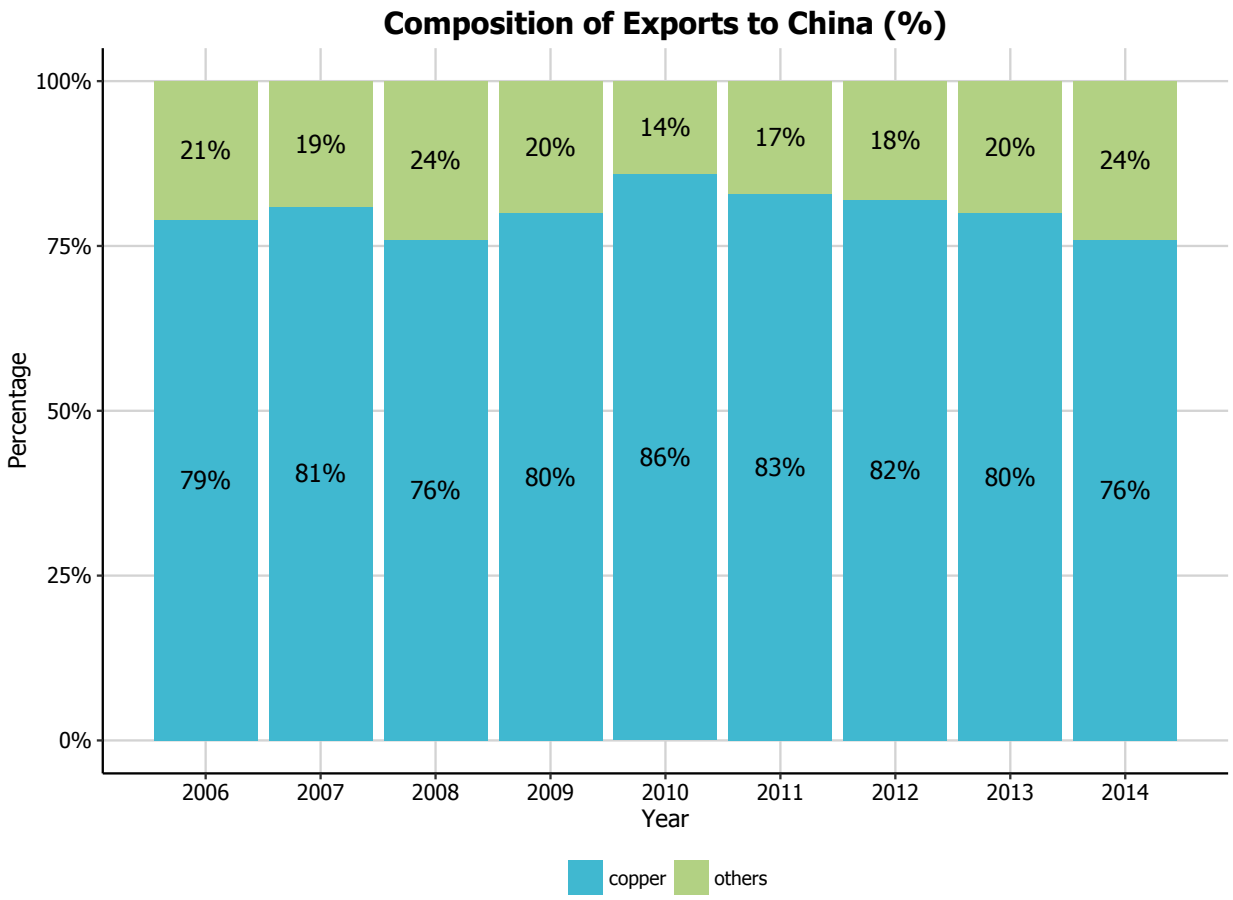
```
fill <- c("#40b8d0", "#b2d183")

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(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_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
```



Stacked Bar Plots

In this part, 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.



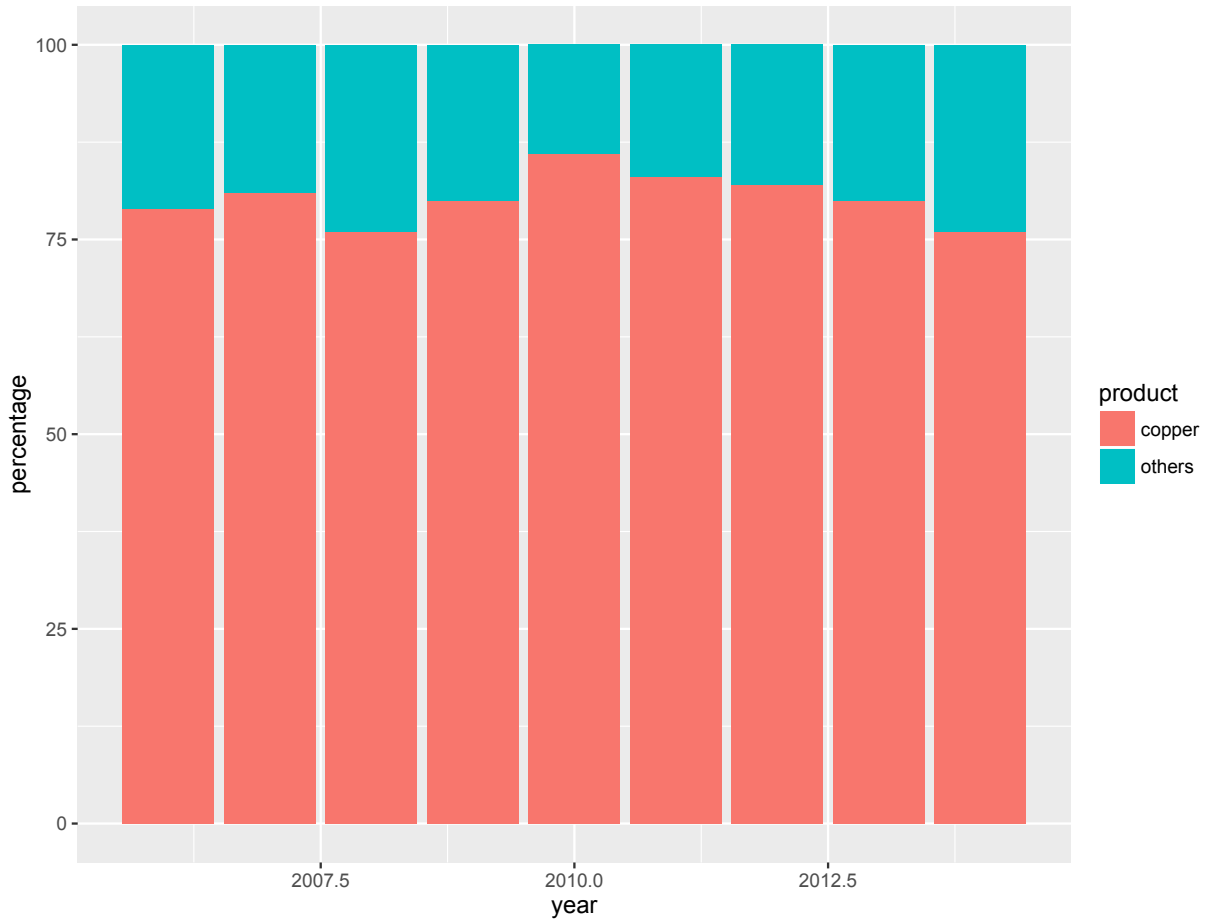
Basic graph

In order to initialise a plot we tell ggplot that `charts.data` is our data, and specify the variables on each axis. We then instruct ggplot to render this as a stacked bar plot by adding the `geom_bar` command.

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-tutorial.csv")
```

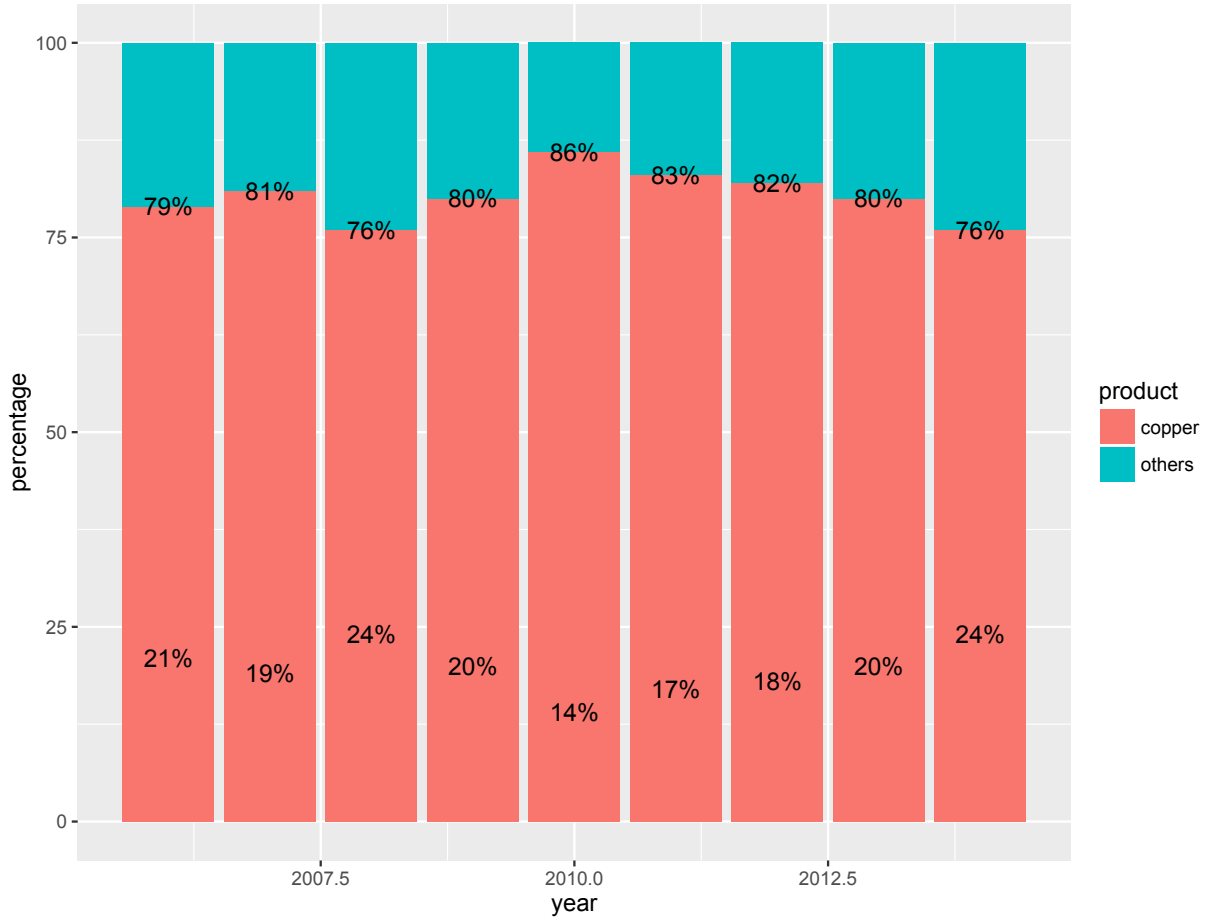
```
charts.data <- read.csv("copper-data-for-tutorial.csv")

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



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.

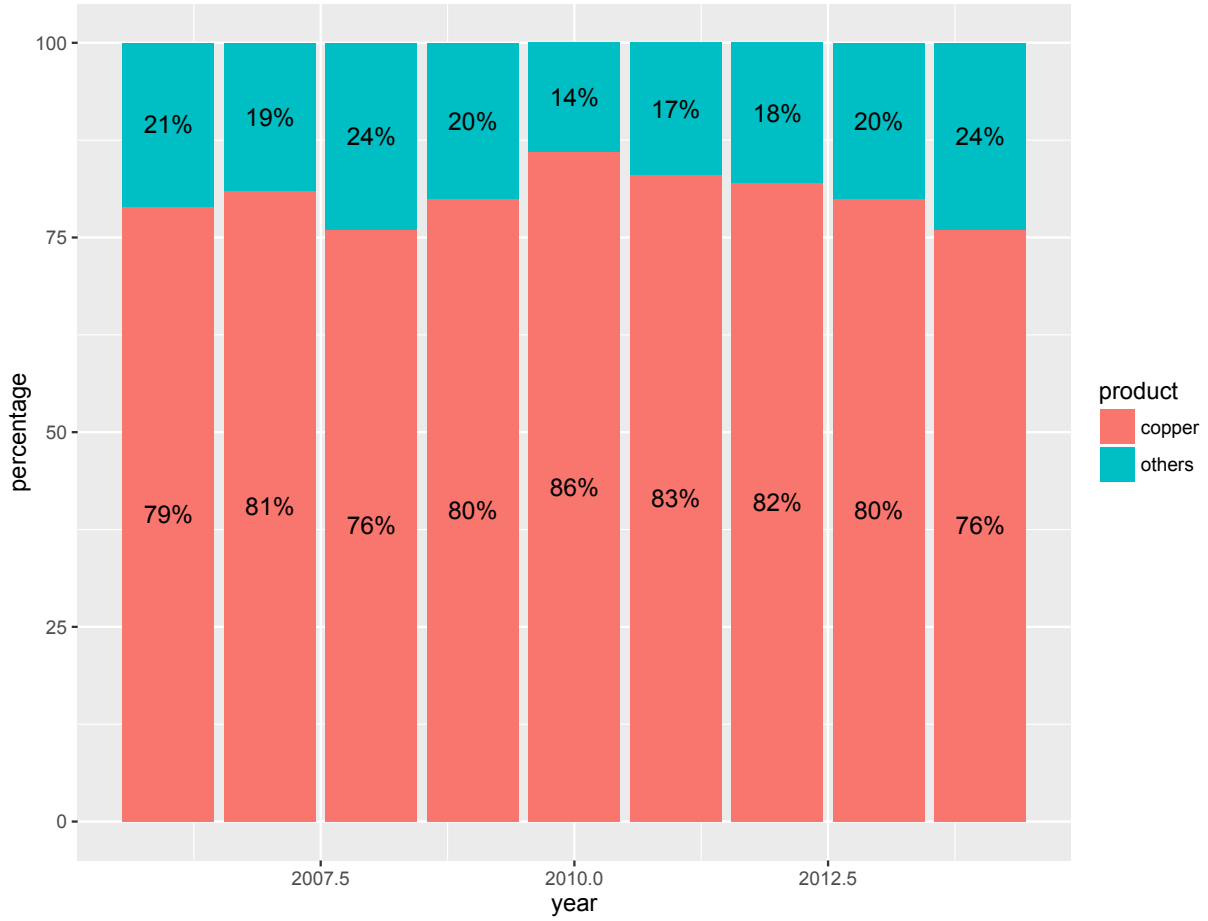
```
p4 <- p4 + geom_text(data=charts.data, aes(x = year, y = percentage,
                                           label = paste0(percentage,"%")), size=4)
p4
```



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 create a new variable called `pos`. 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())`.

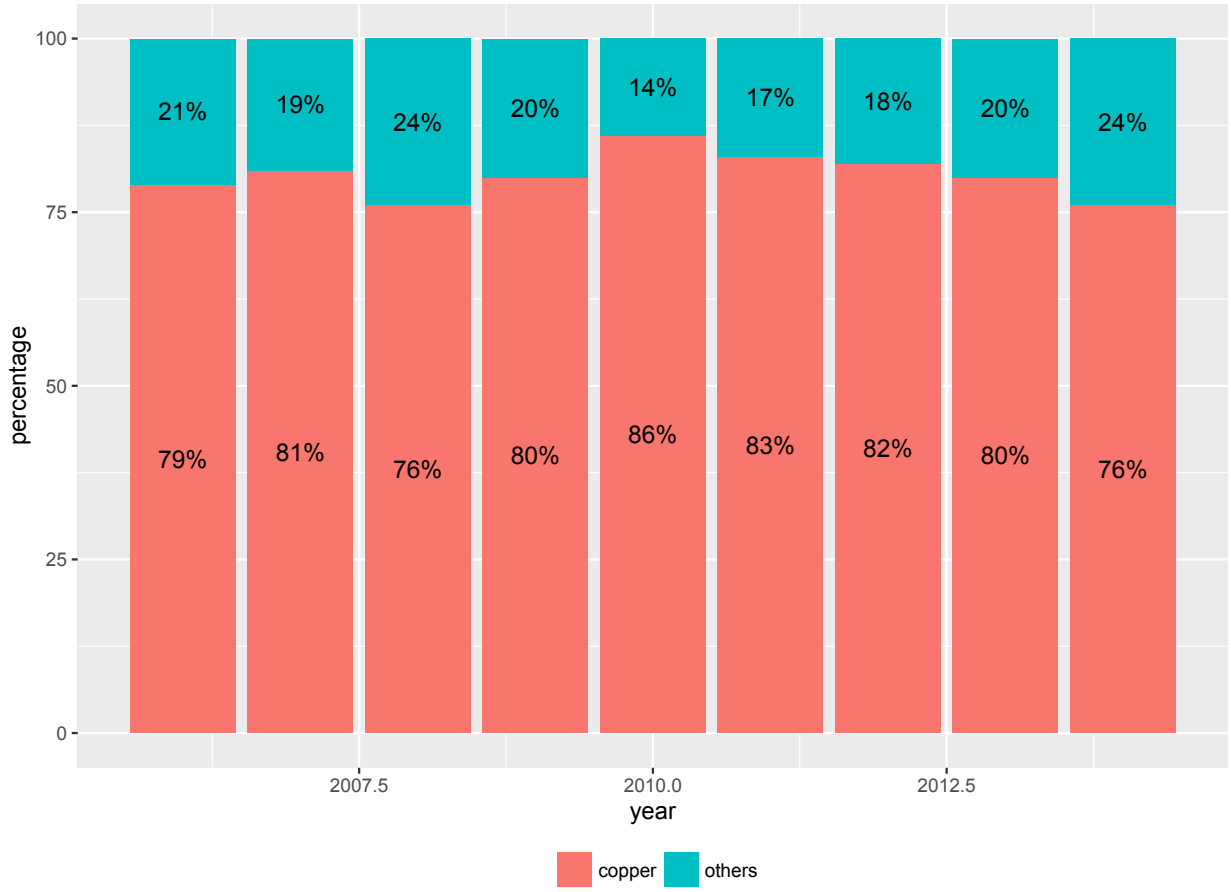
```
charts.data <- ddply(charts.data, .(year), transform, pos = cumsum(percentage) - (0.5 * percentage))

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
```



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.

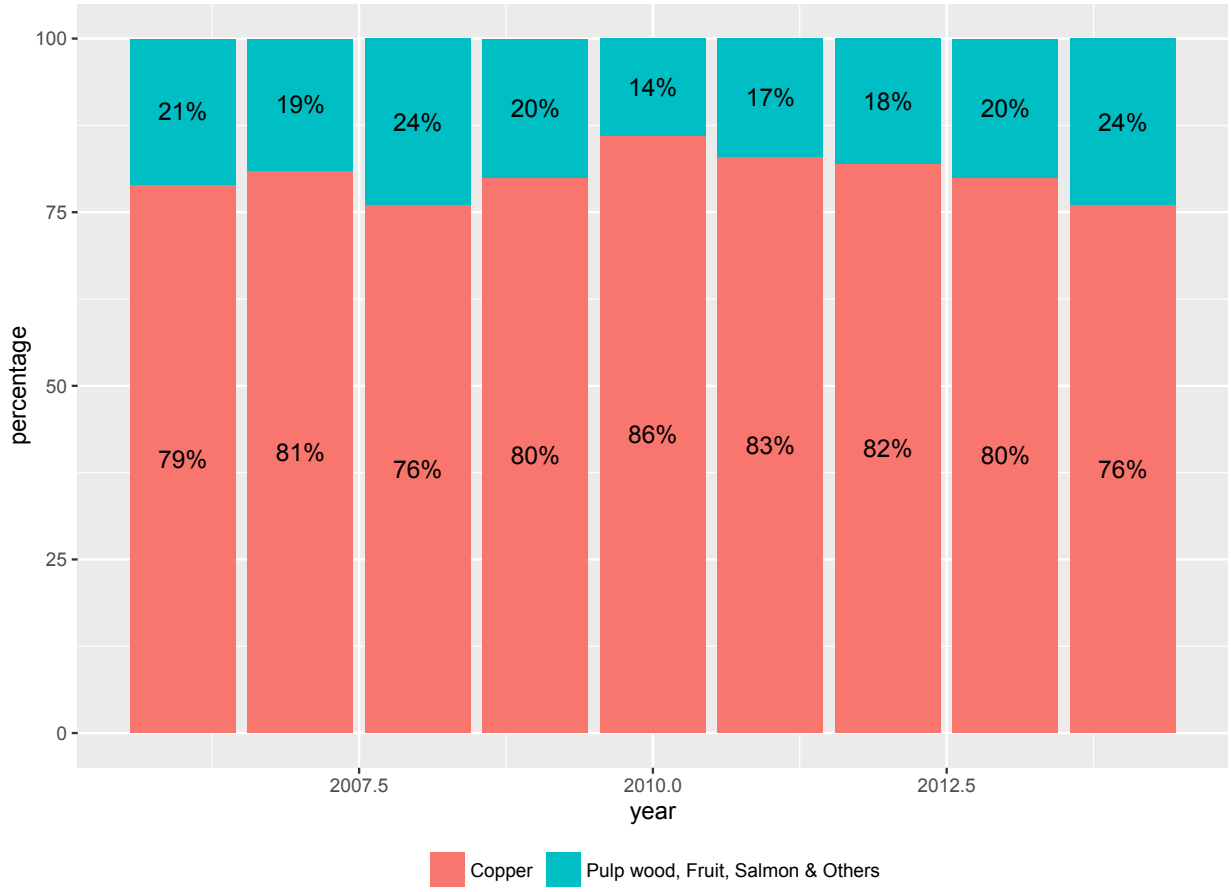
```
p4 <- p4 + theme(legend.position="bottom", legend.direction="horizontal",
                 legend.title = element_blank())
p4
```



Changing variables display

To change the variables' displayed name, we need to re-factor our data labels in `charts.data` data frame.

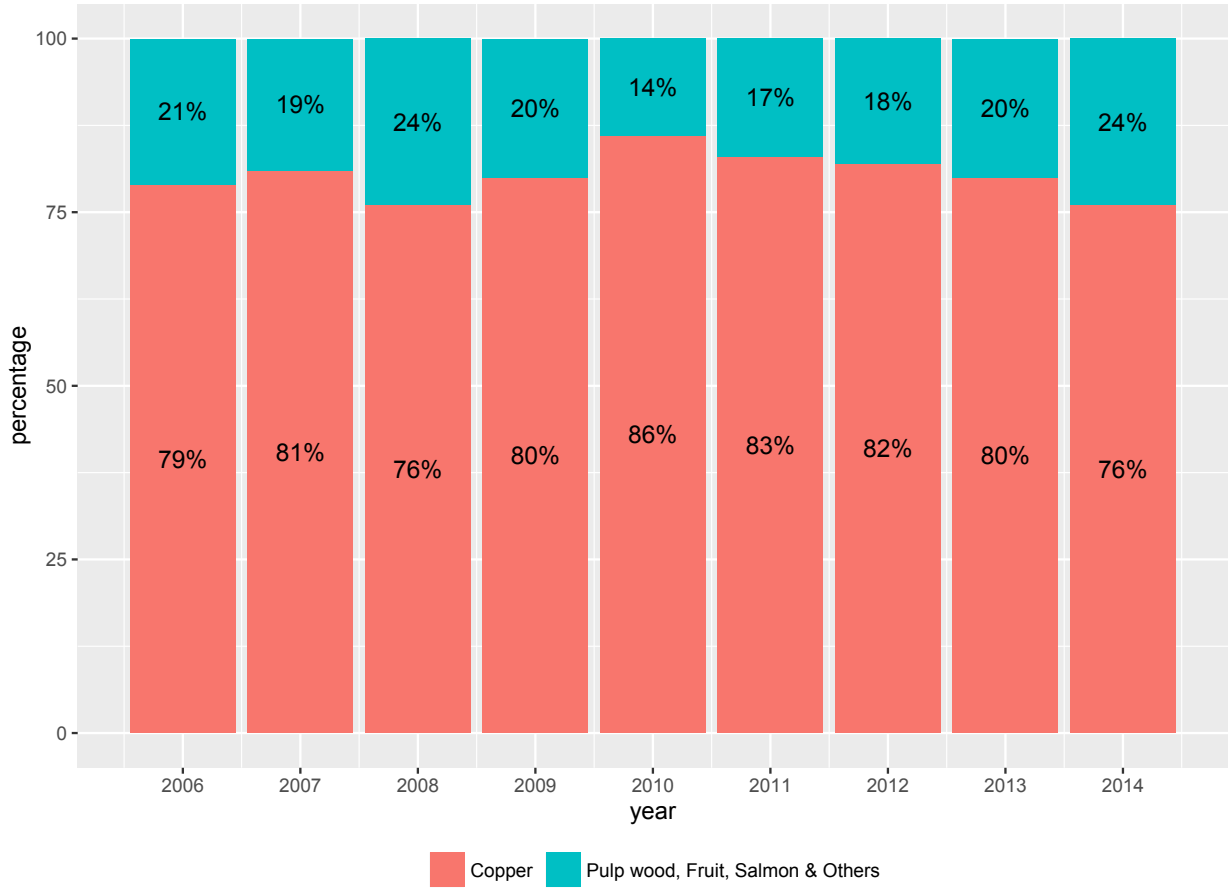
```
charts.data <- as.data.frame(charts.data)
charts.data$product <- factor(charts.data$product, levels = c("copper","others"),
                              labels = c("Copper ","Pulp wood, Fruit, Salmon & Others"))

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
```



Adjusting x-axis scale

To change the axis tick marks, we use the `scale_x_continuous` and/or `scale_y_continuous` commands.

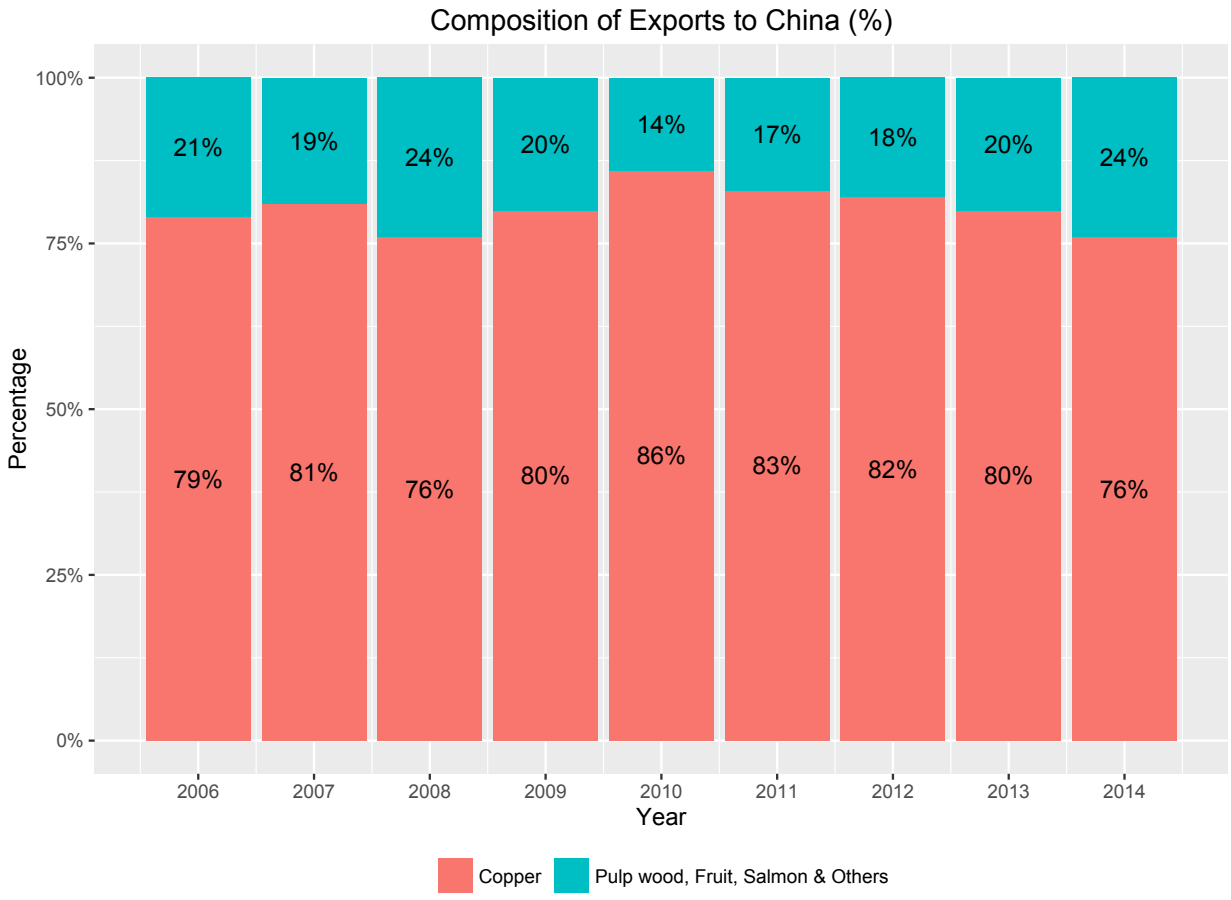
```
p4 <- p4 + scale_x_continuous(breaks=seq(2006,2014,1))
p4
```



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.

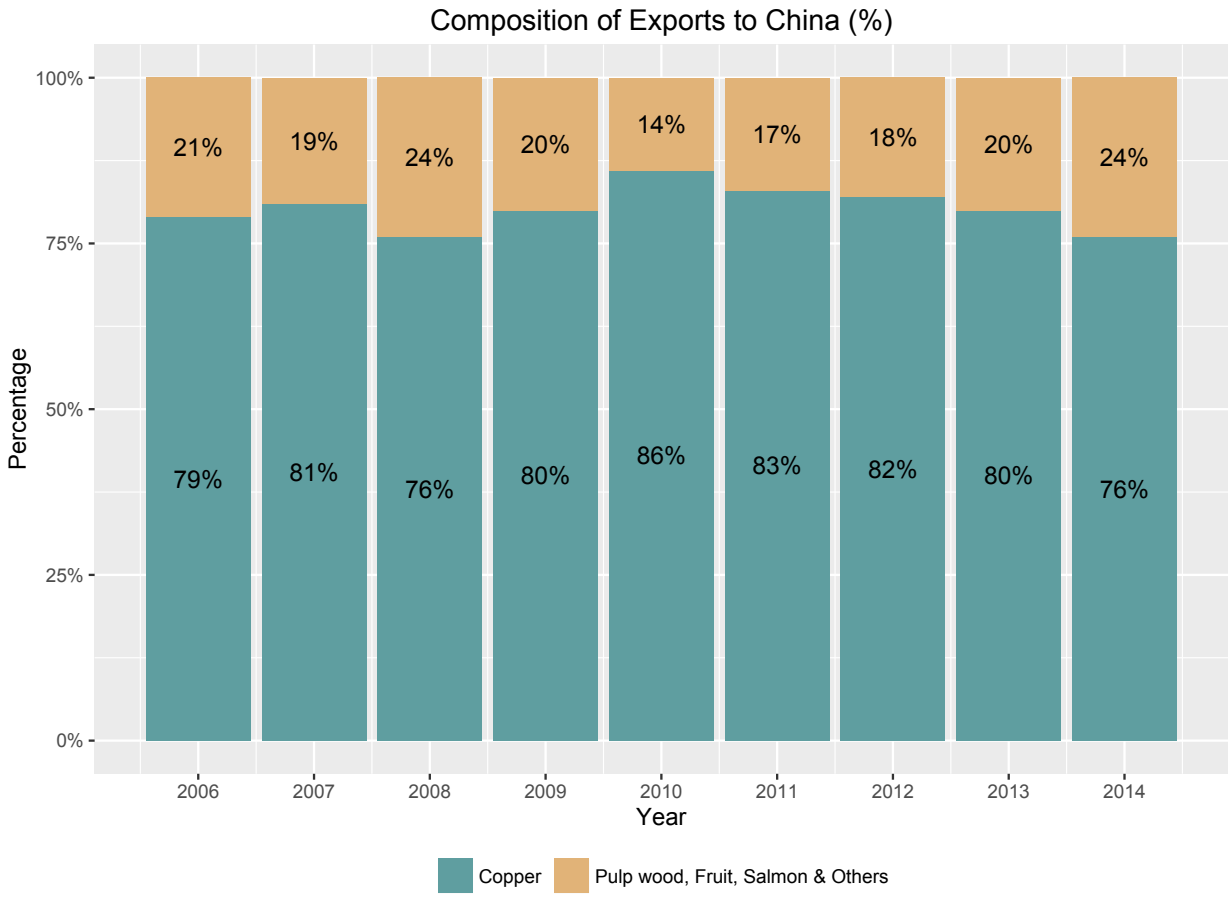
```
p4 <- p4 + labs(x="Year", y="Percentage") +
      scale_y_continuous(labels = dollar_format(suffix = "%", prefix = "")) +
      ggtitle("Composition of Exports to China (%)")
p4
```



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.

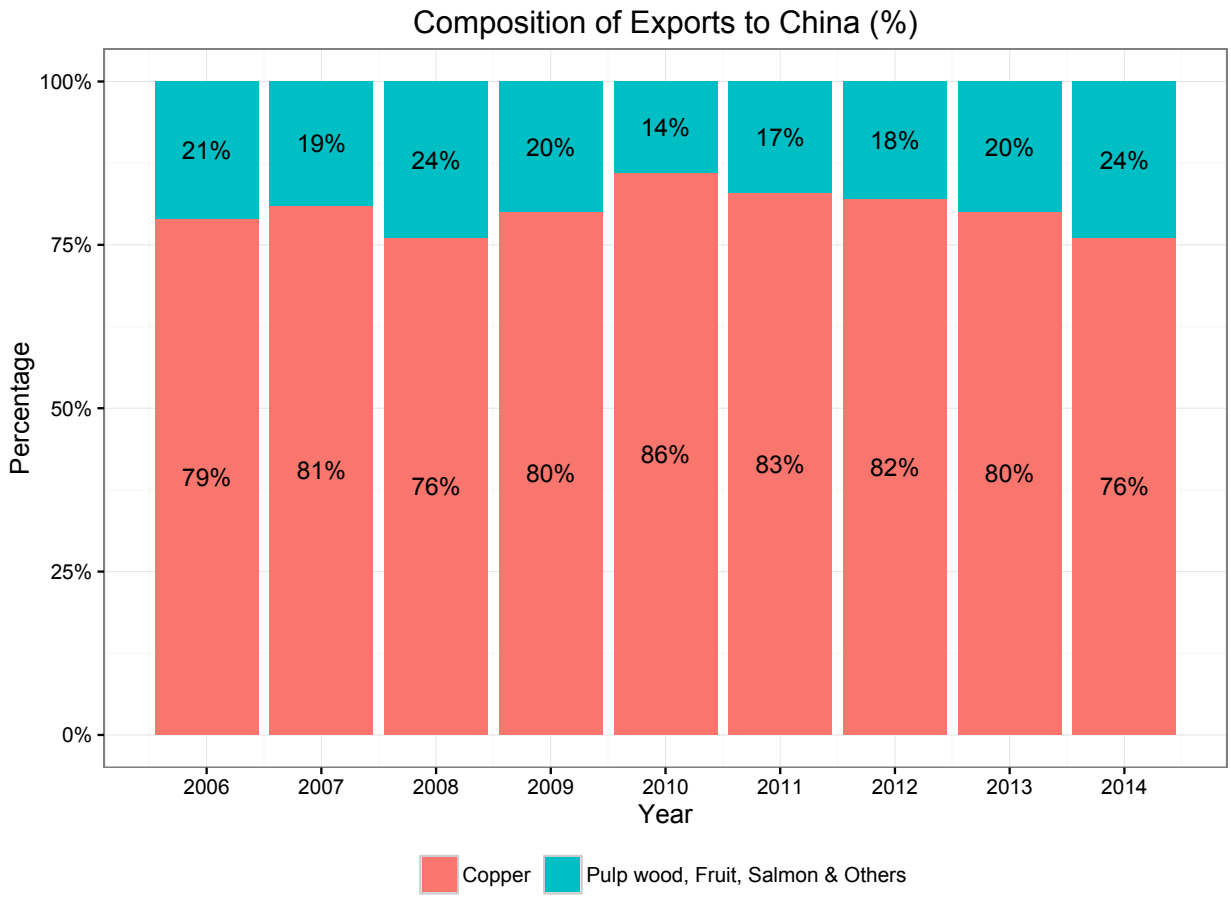
```
fill <- c("#5F9EA0", "#E1B378")
p4 <- p4 + scale_fill_manual(values=fill)
p4
```



Using the white theme

As explained in the previous posts, 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
```



Creating an XKCD style chart

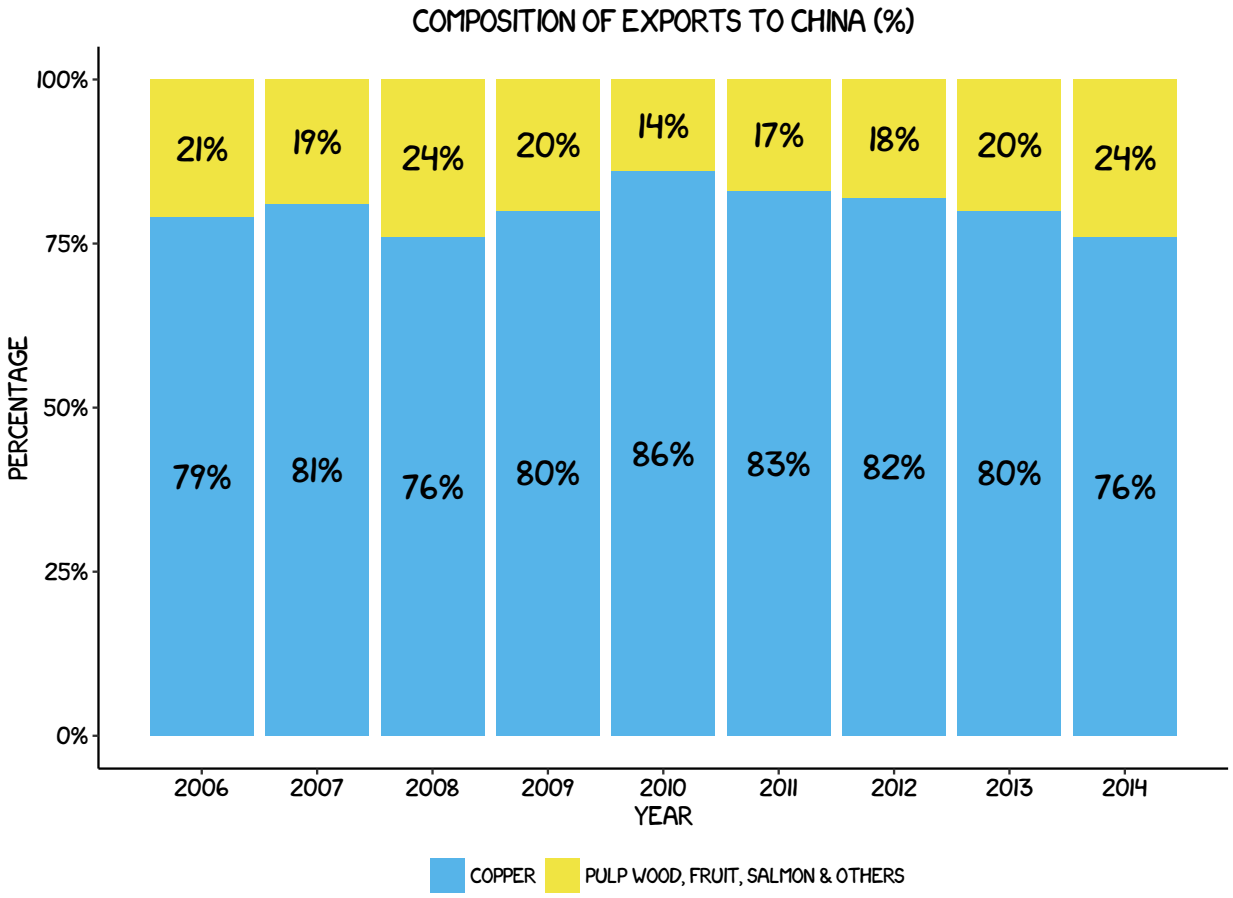
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
              dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

```
#font_import(pattern="[X/x]kcd")
#fonts()
```

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

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.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"),
            axis.line.y = 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
```

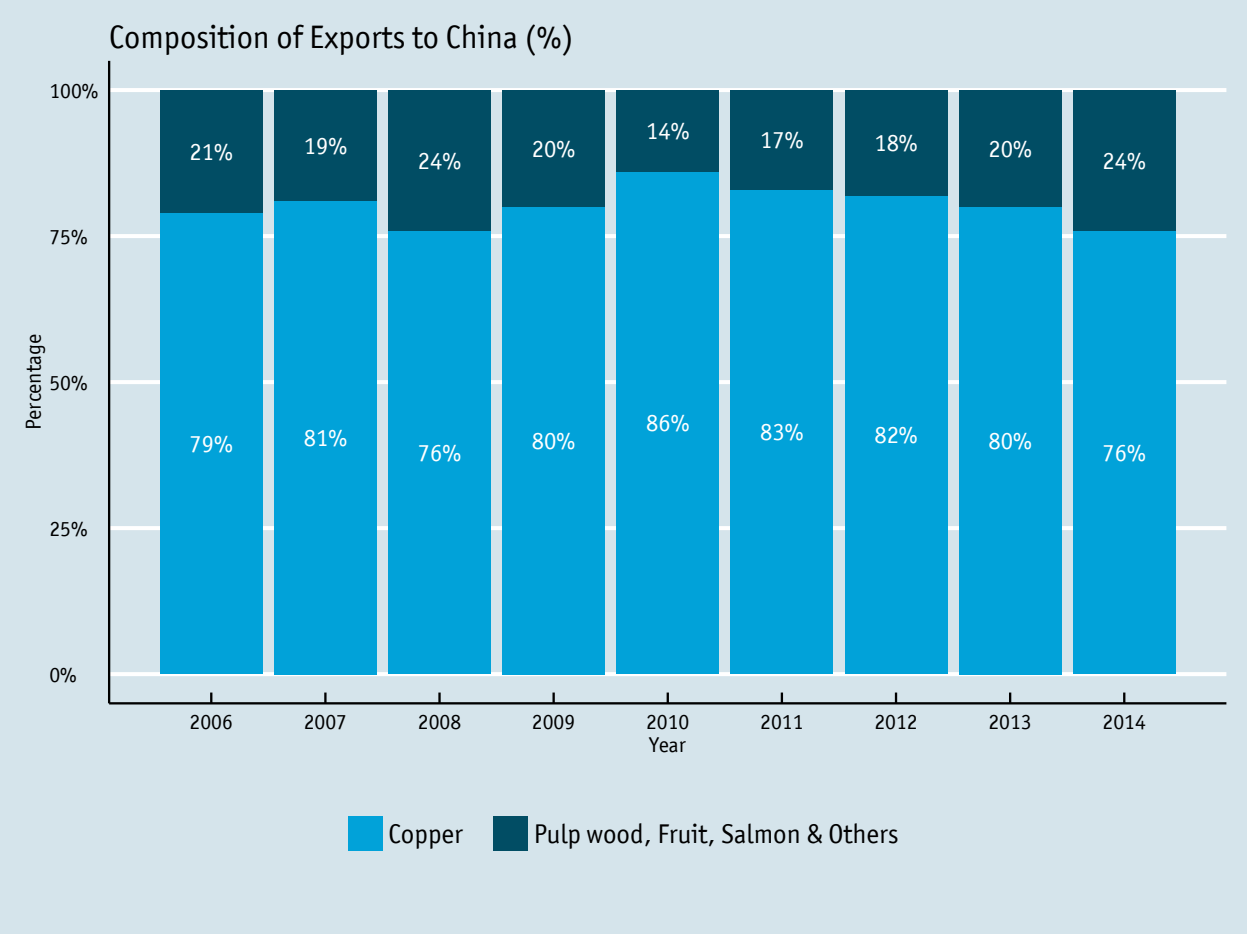


Using 'The Economist' theme

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.

```
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"),
            axis.line.y = 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
```

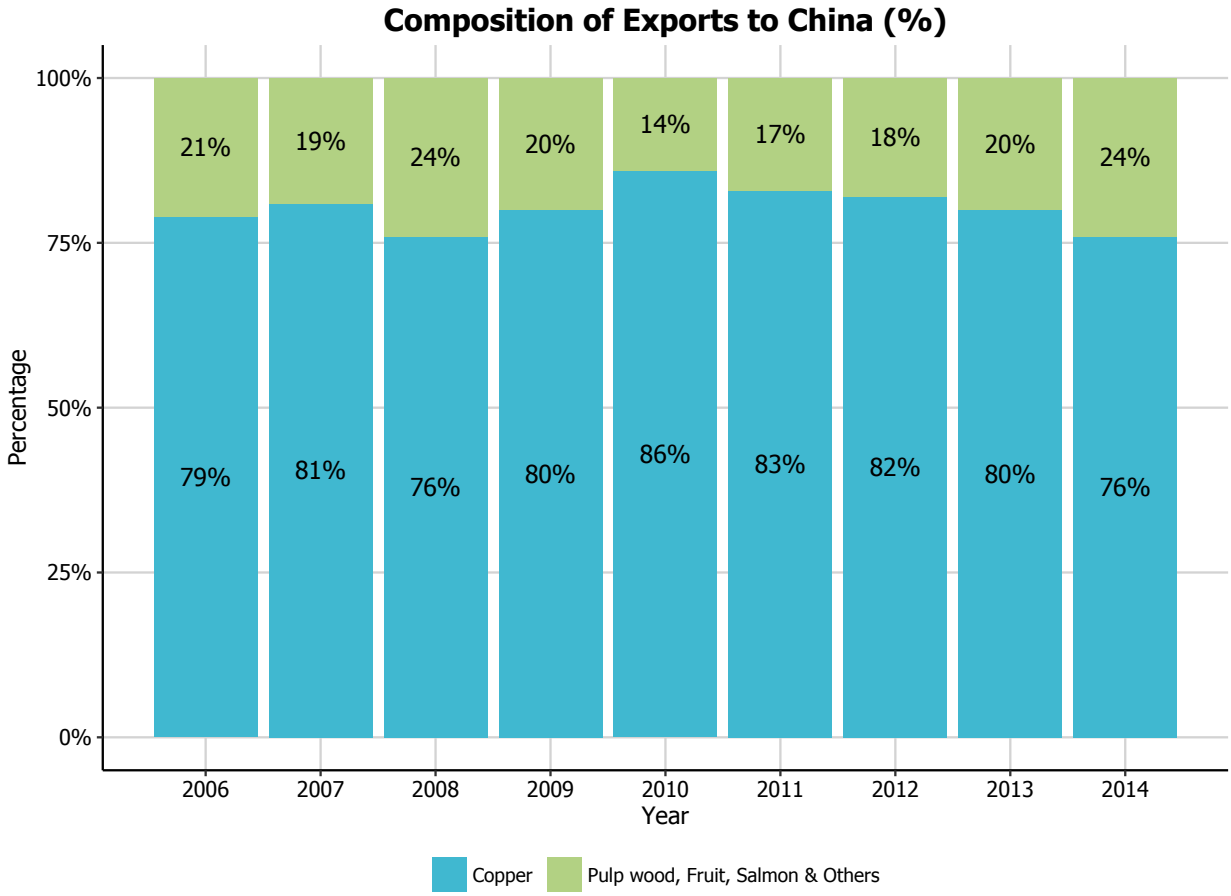


Creating your own theme

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

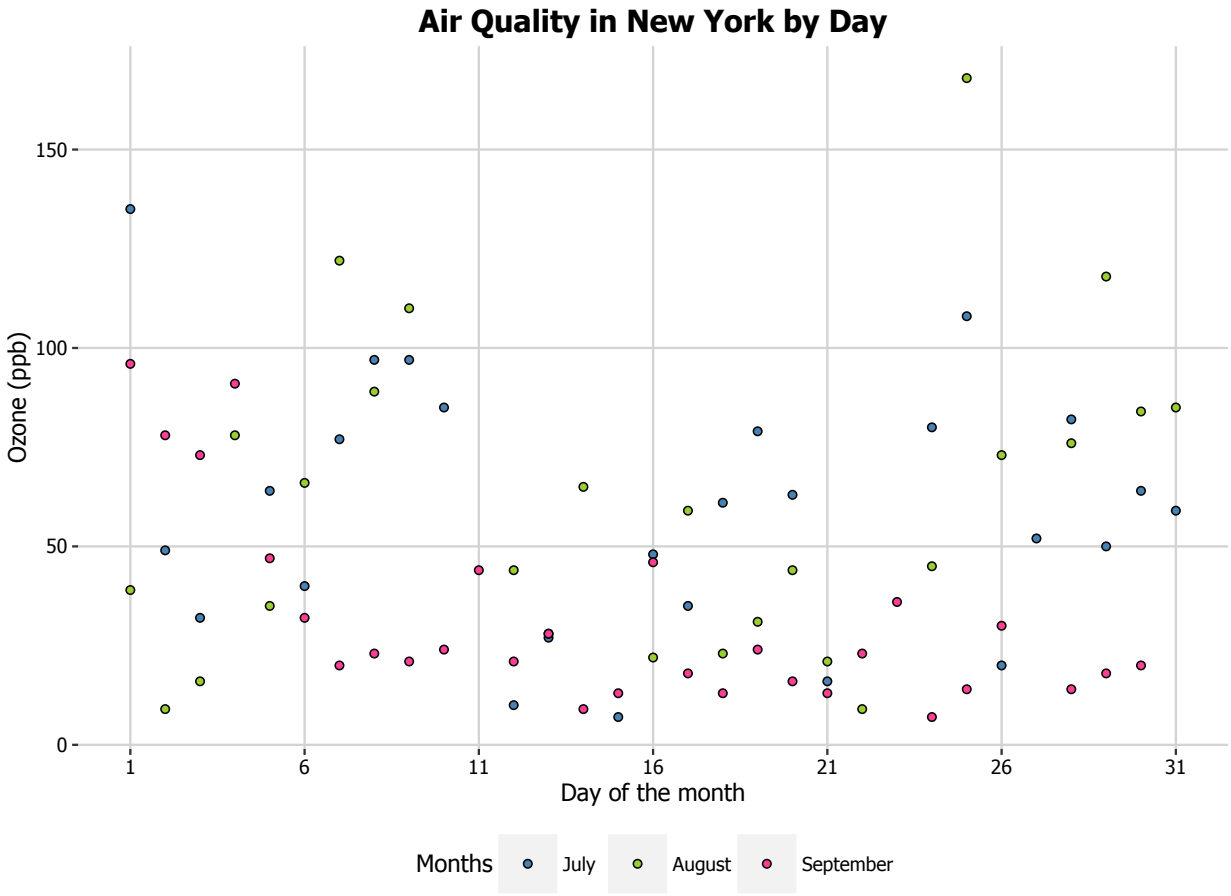
```
fill <- c("#40b8d0", "#b2d183")

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(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_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
```



Scatter Plots

In this part, 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.



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

```
library(ggplot2)
library(ggthemes)
library(extrafont)
library(datasets)
data(airquality)
```

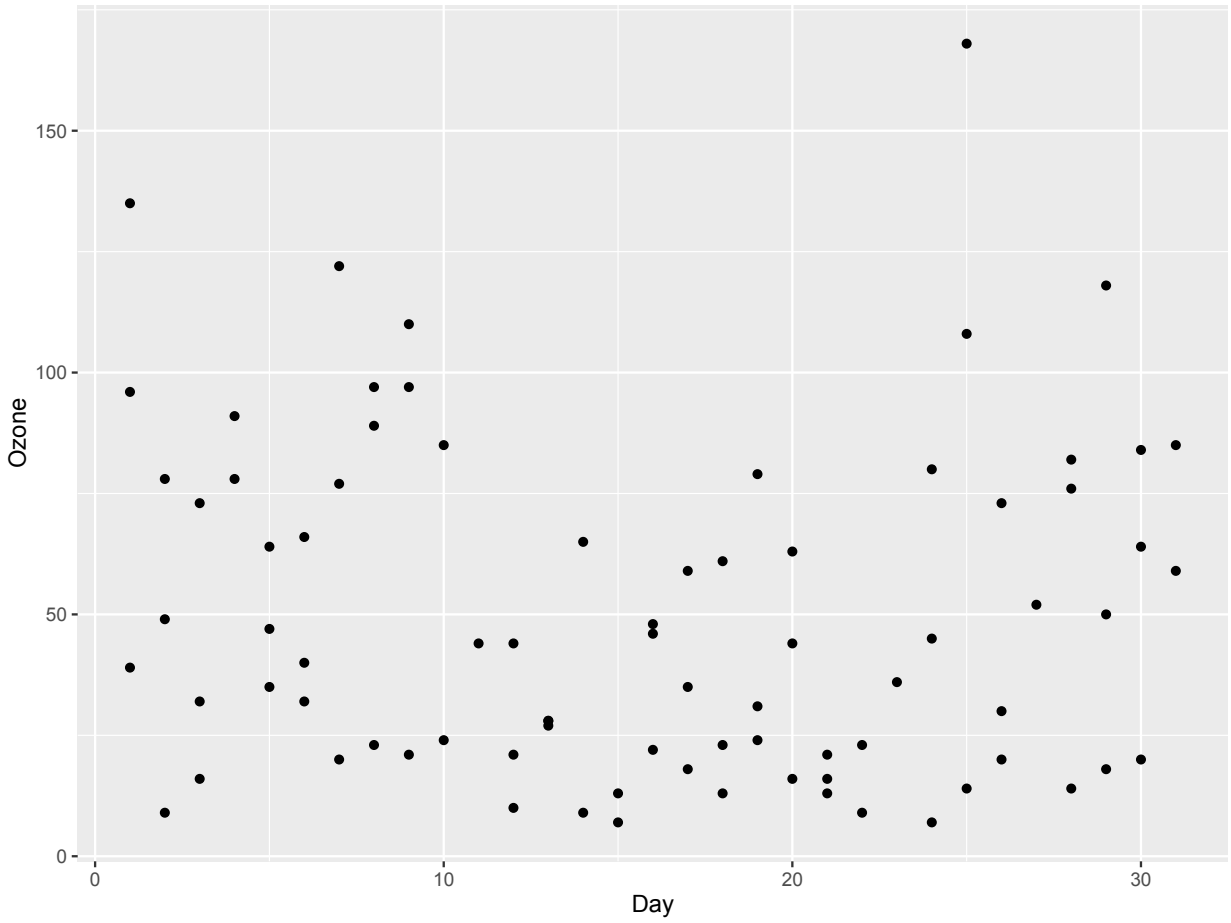
We will then trim the data down to the final three months and turn the `Month` variable into a labelled factor variable. We end up with a new dataset called `aq_trim`.

```
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"))
```

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.

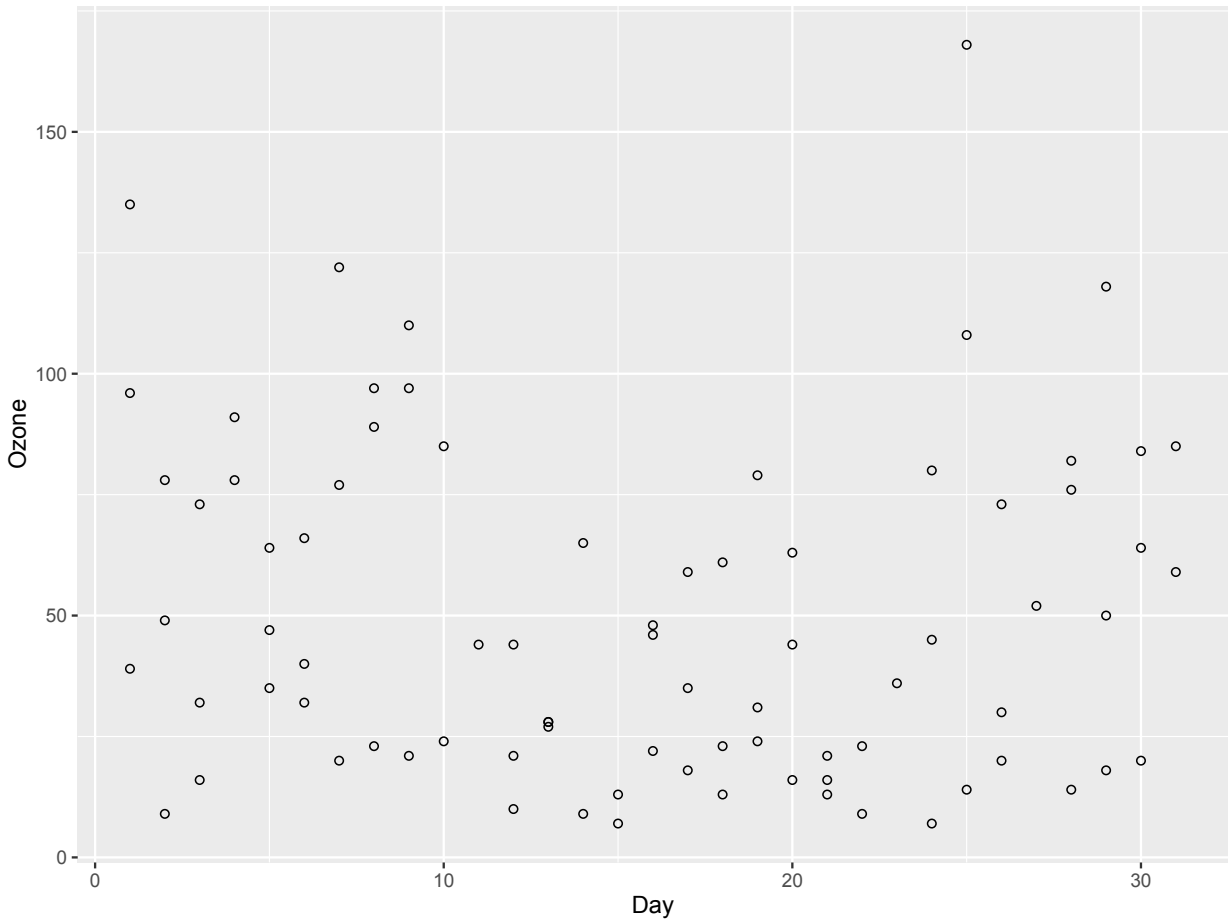
```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone)) +
      geom_point()
p5
```



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.

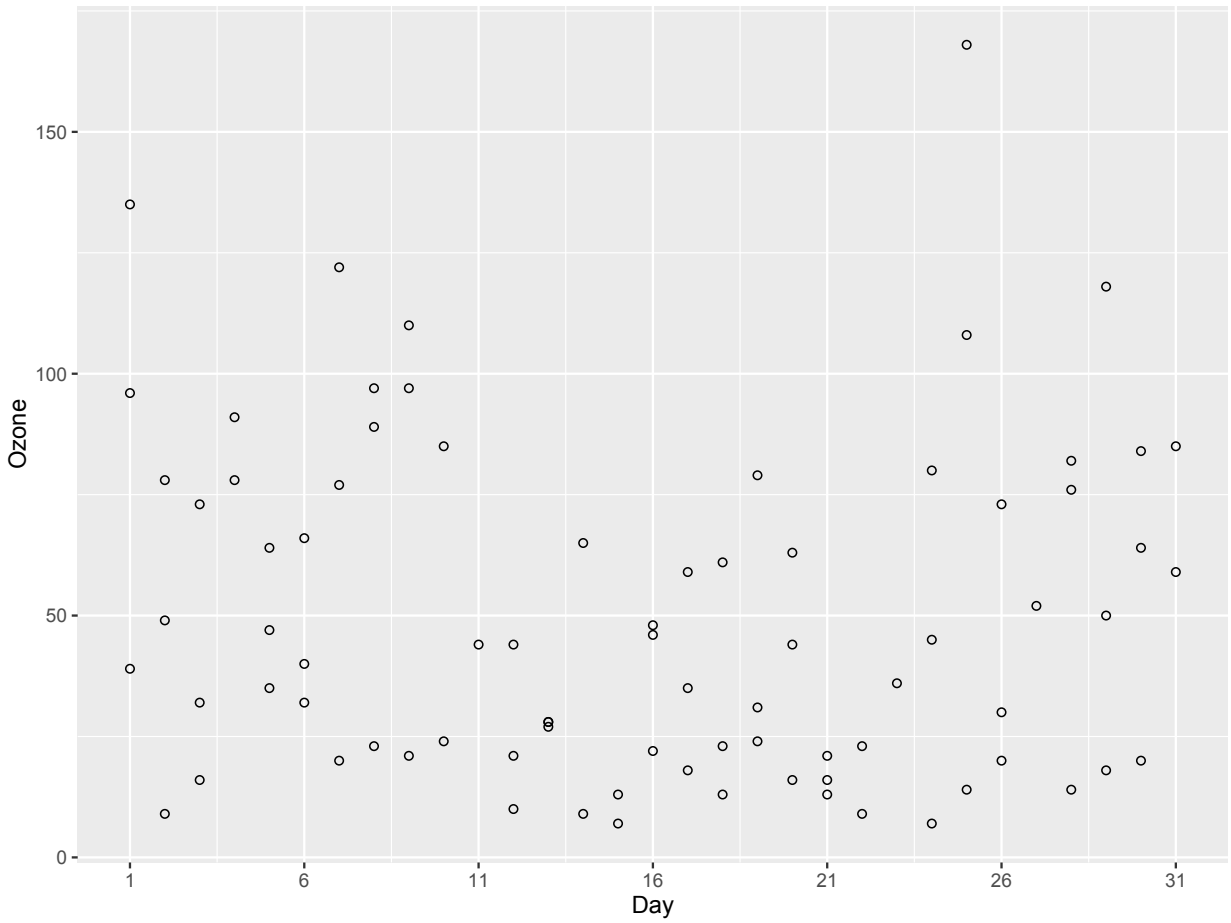
```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone)) + geom_point(shape = 21)
p5
```



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).

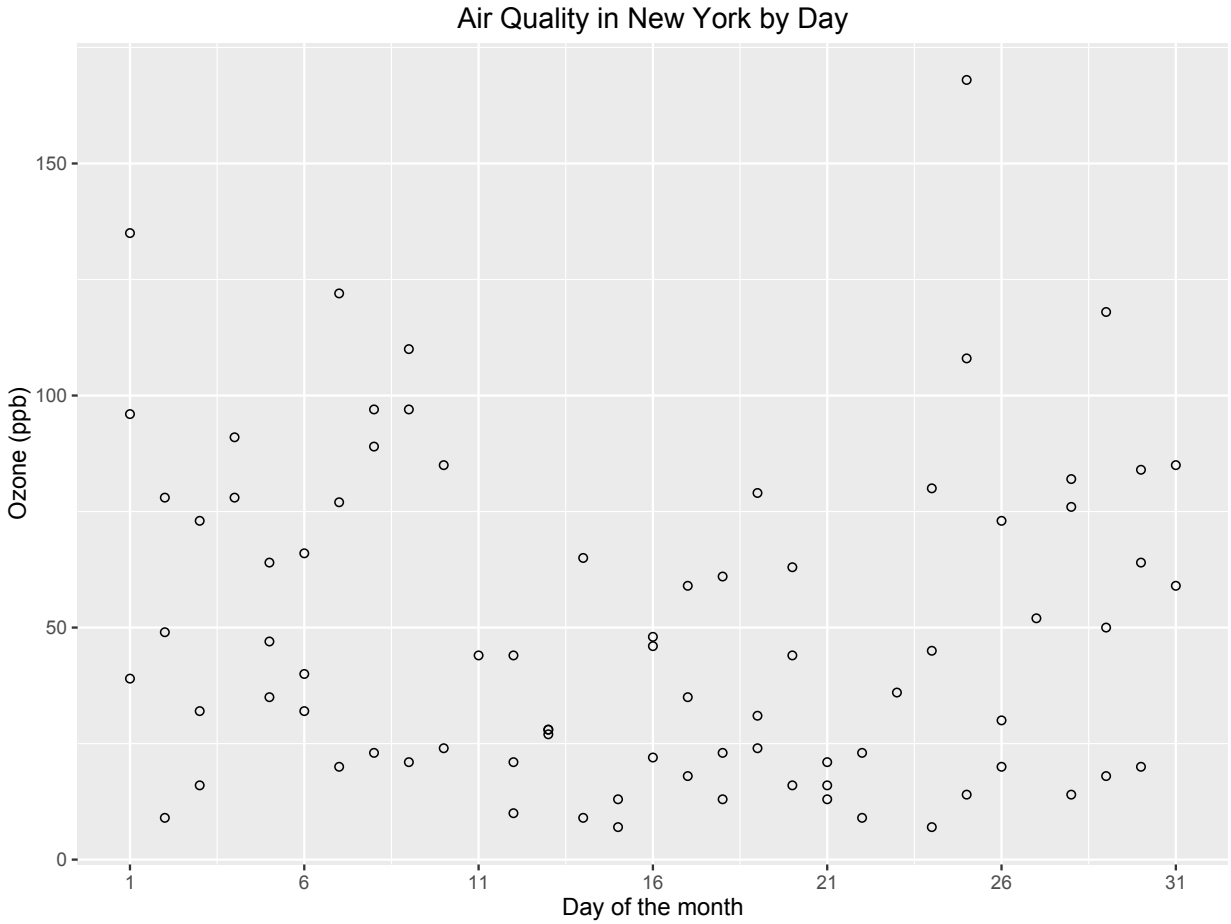
```
p5 <- p5 + scale_x_continuous(breaks = seq(1, 31, 5))
p5
```



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.

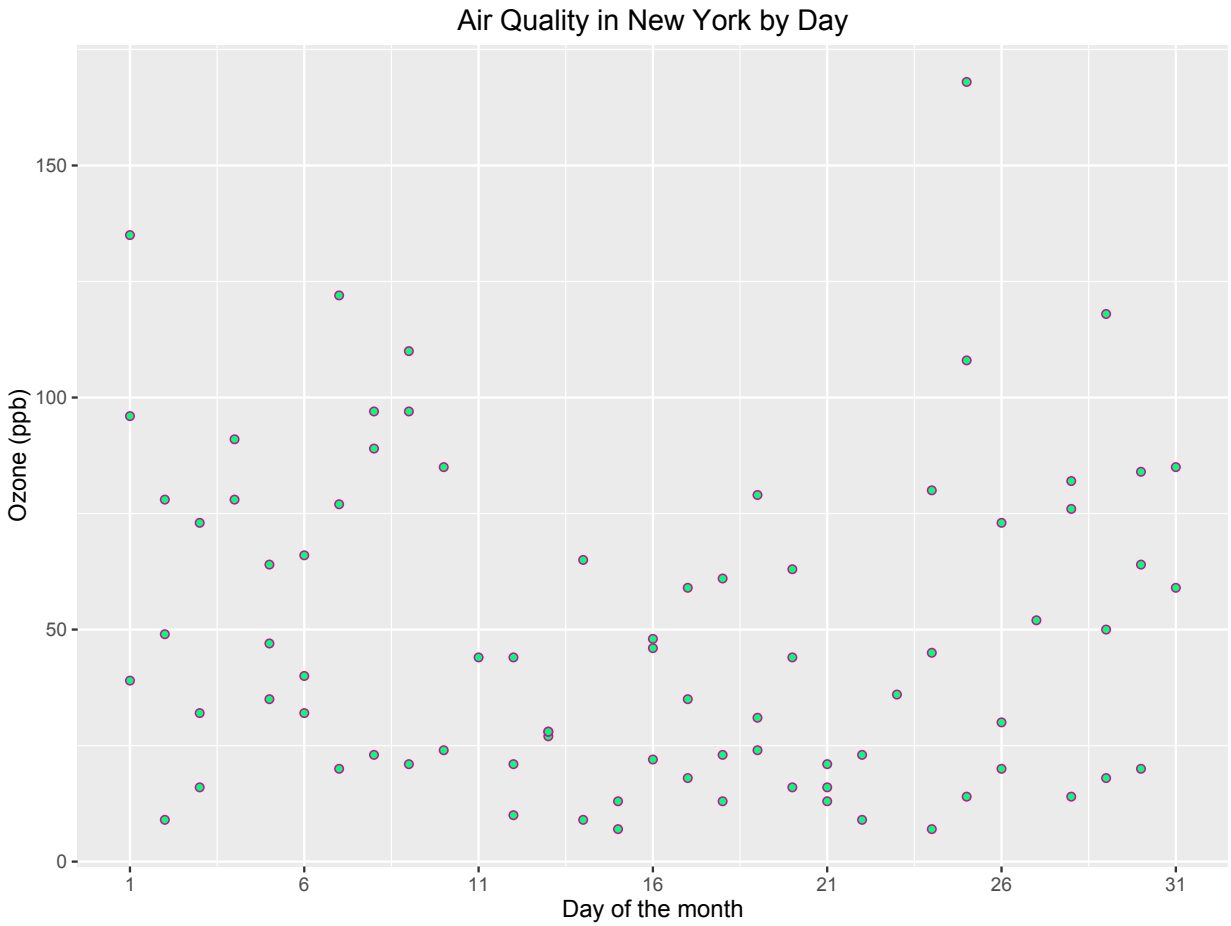
```
p5 <- p5 + ggtitle("Air Quality in New York by Day") +
      labs(x = "Day of the month", y = "Ozone (ppb)")
p5
```



Adjusting the colour palette

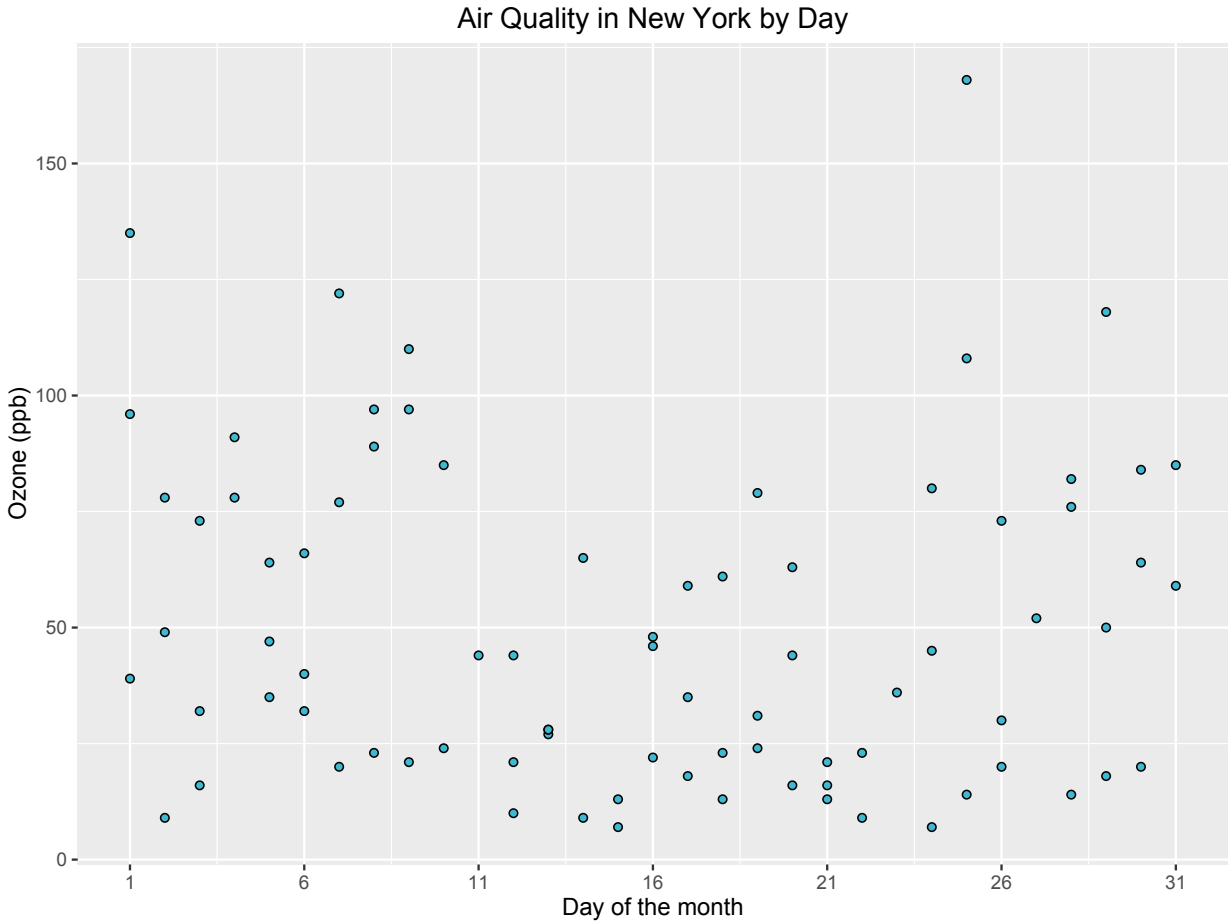
There are a few options for adjusting the colour. The most simple is to make every point one fixed colour. You can reference colours by name, with the full list of colours recognised by R here. Let's try making the outline `mediumvioletred` and the fill `springgreen`.

```
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
```



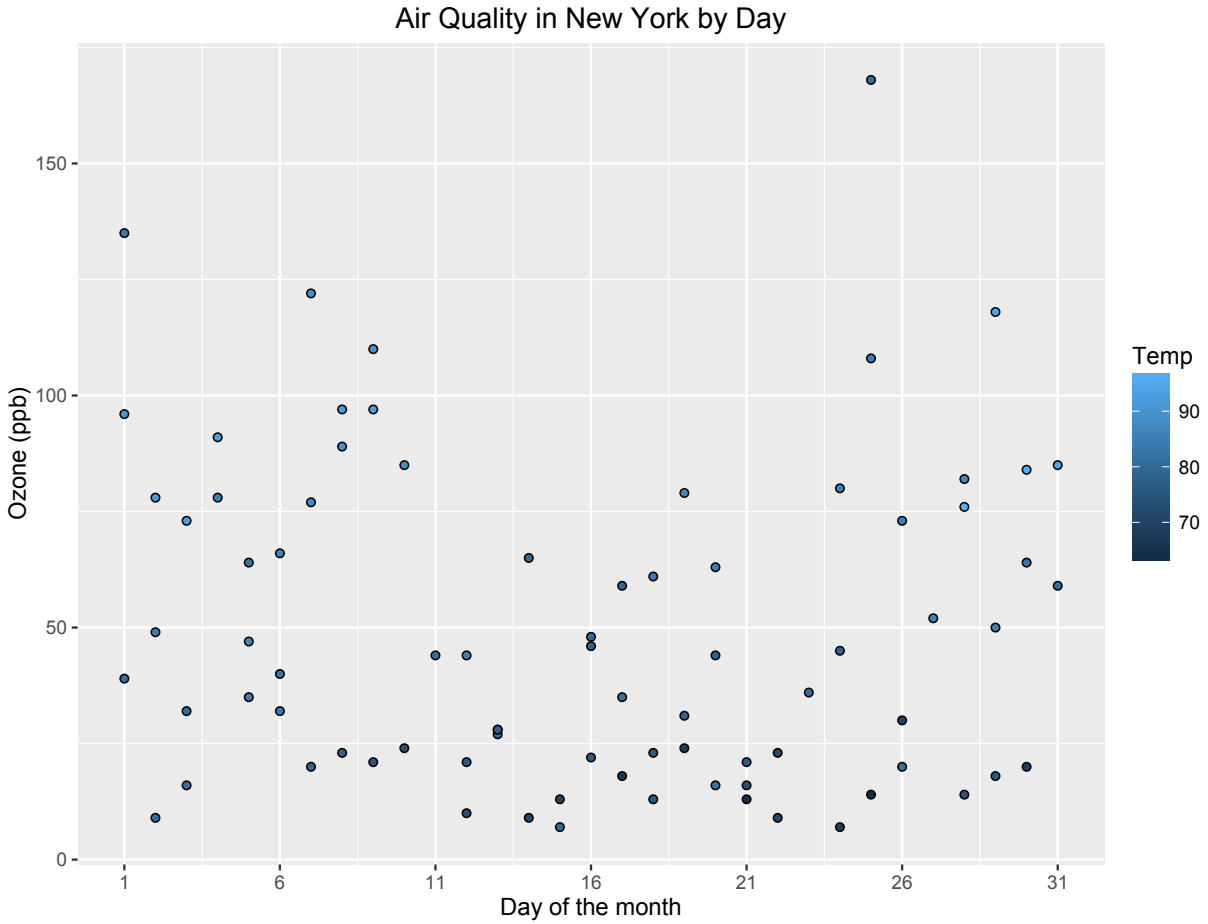
You can change the colours using specific HEX codes instead. Here we have made the outline #000000 (black) and the fill "#40b8d0 (vivid cyan).

```
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
```



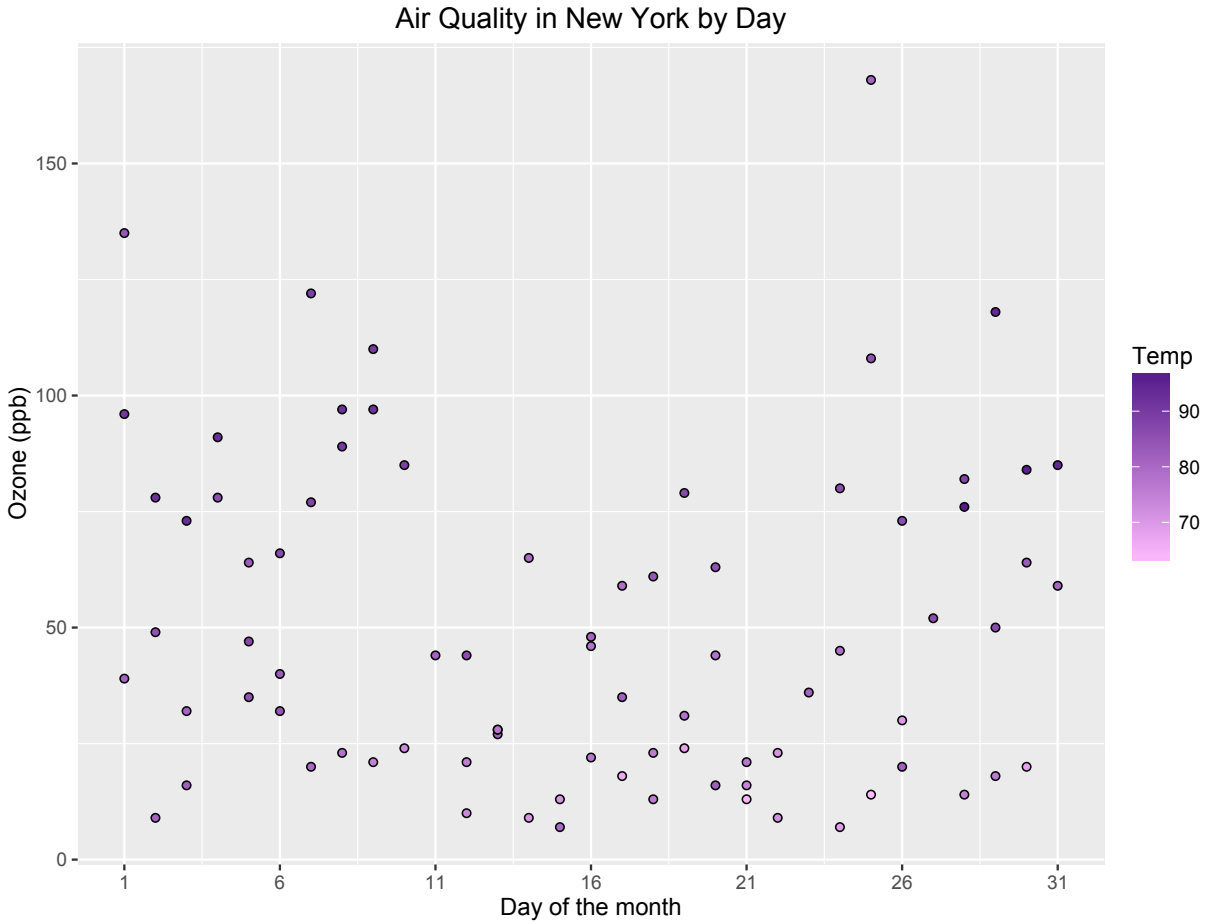
You can also change the colour of the data points according to the levels of another variable. This can be done either as a continuous gradient, or as a levels of a factor variable. Let's change the colour by the values of temperature:

```
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.

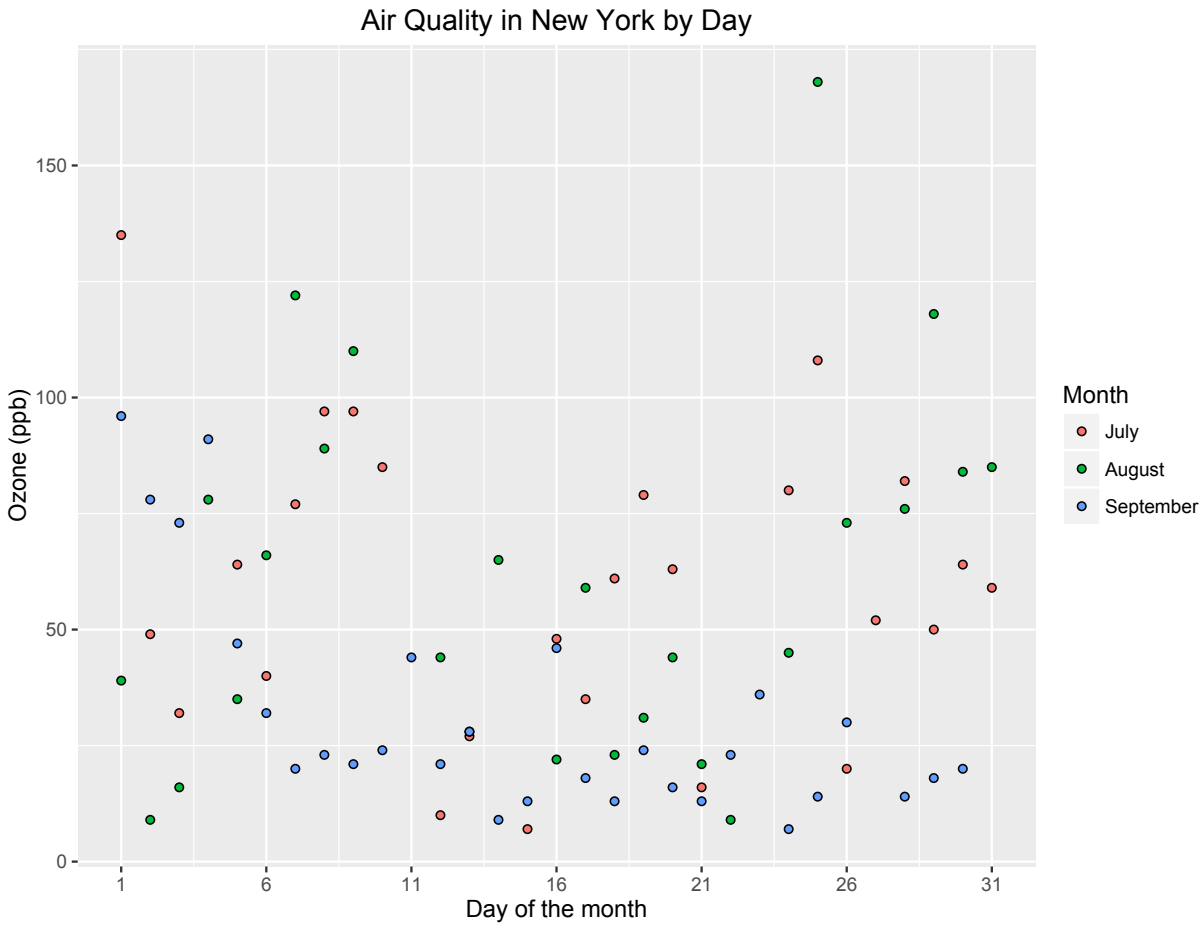
```
p5 <-  p5 + scale_fill_continuous(low = "plum1", high = "purple4")
p5
```



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

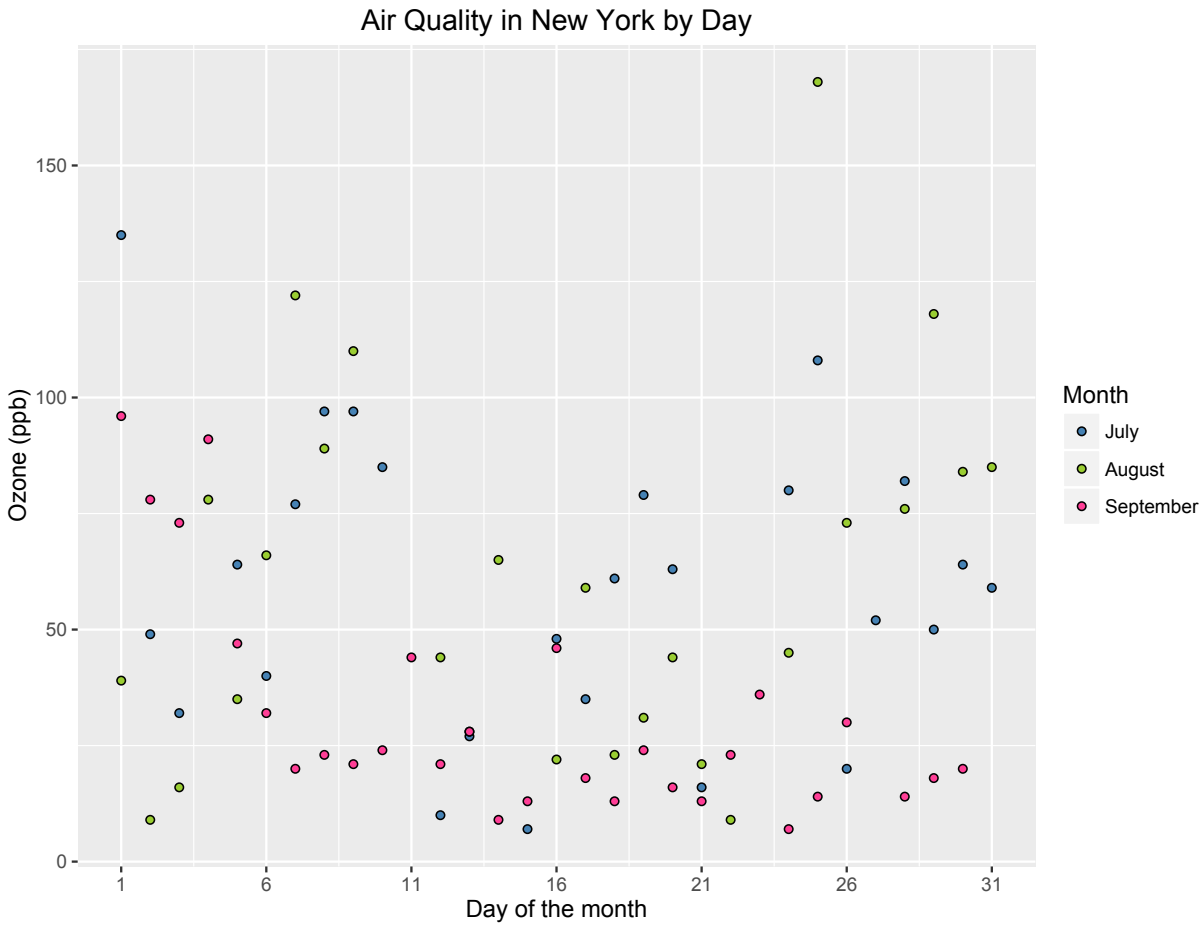
Let's now change the colours of the data points by a factor variable, `Month`.

```
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`.

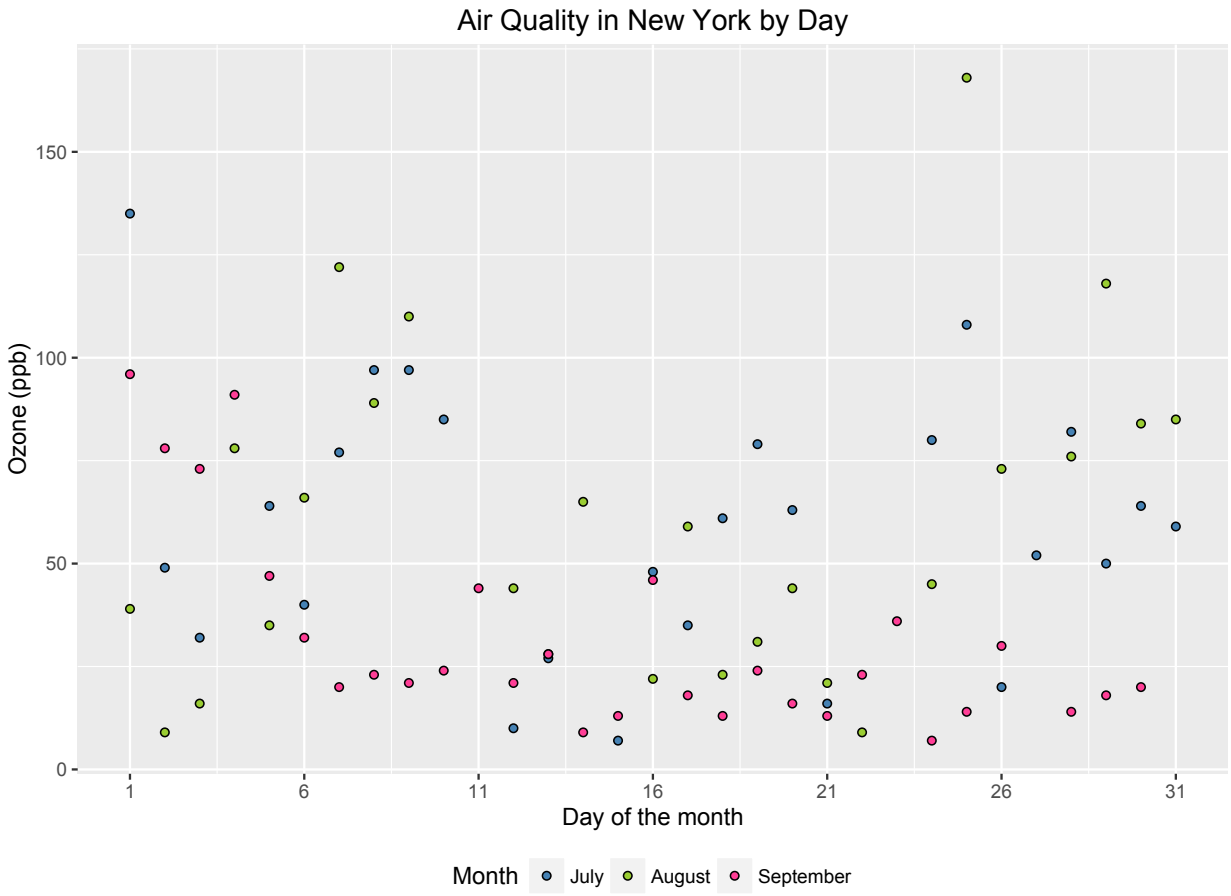
```
fill = c("steelblue", "yellowgreen", "violetred1")

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



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.

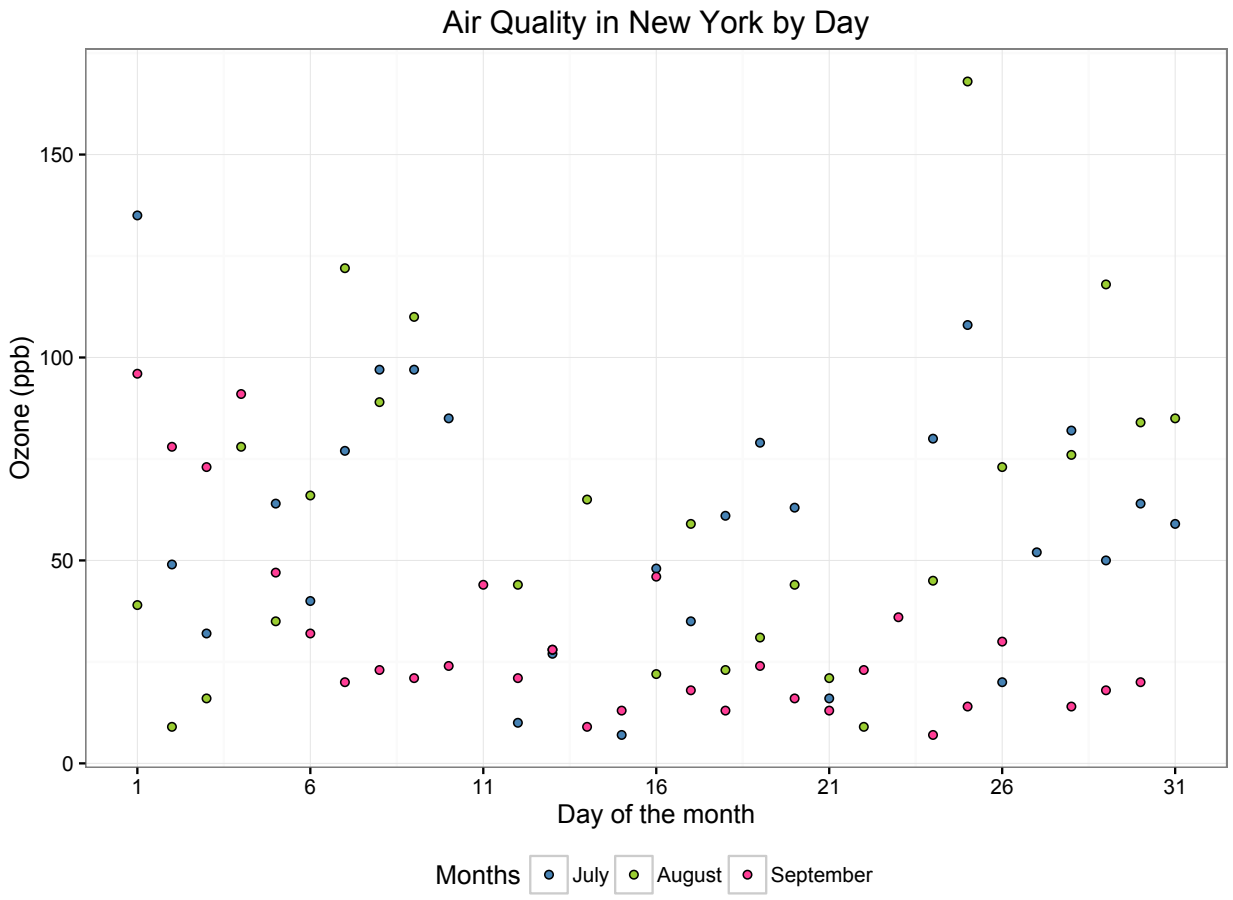
```
p5 <- p5 + theme(legend.position = "bottom", legend.direction = "horizontal")
p5
```



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)) + theme_bw() +
      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(legend.position="bottom", legend.direction="horizontal")
p5
```



Creating an XKCD style chart

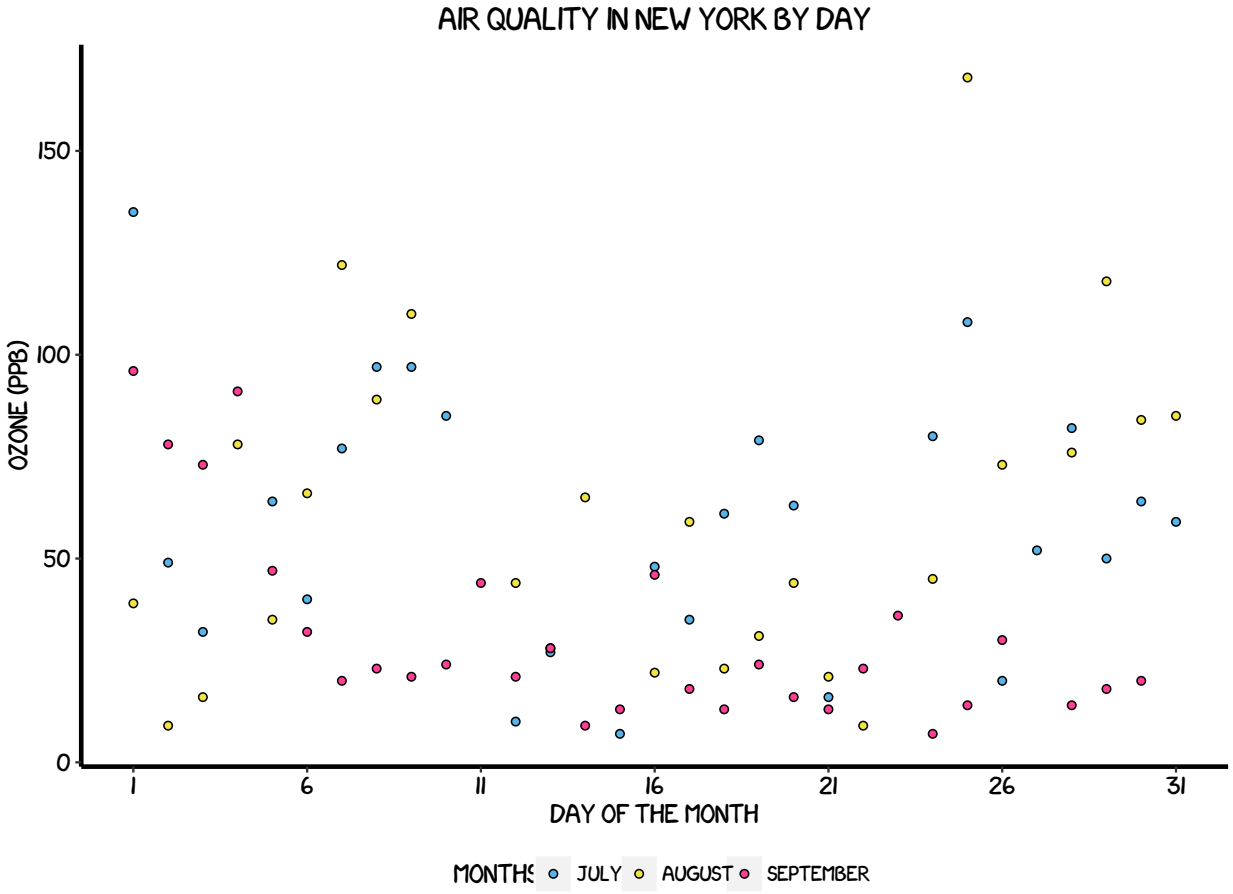
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf", dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

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

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

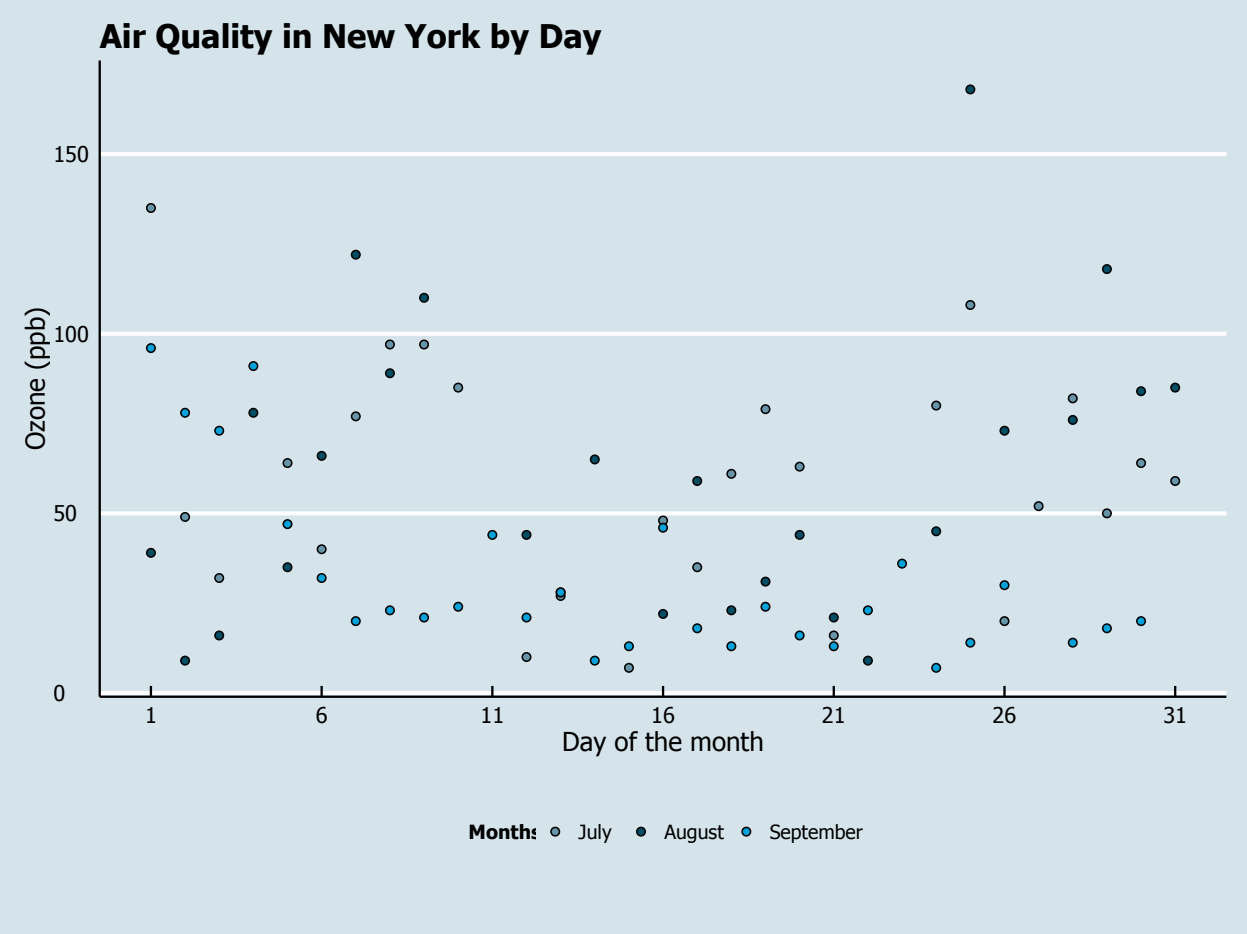
```
      geom_point(shape = 21) +
      ggtitle("Air Quality in New York by Day") +
      labs(x = "Day of the month", y = "Ozone (ppb)", fill = "Months") +
      scale_x_continuous(breaks = seq(1, 31, 5)) +
      scale_fill_manual(values = fill) +
      scale_size(range = c(1, 10)) +
      theme(axis.line.x = element_line(size=1, colour = "black"),
            axis.line.y = element_line(size=1, 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",
            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
```



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.

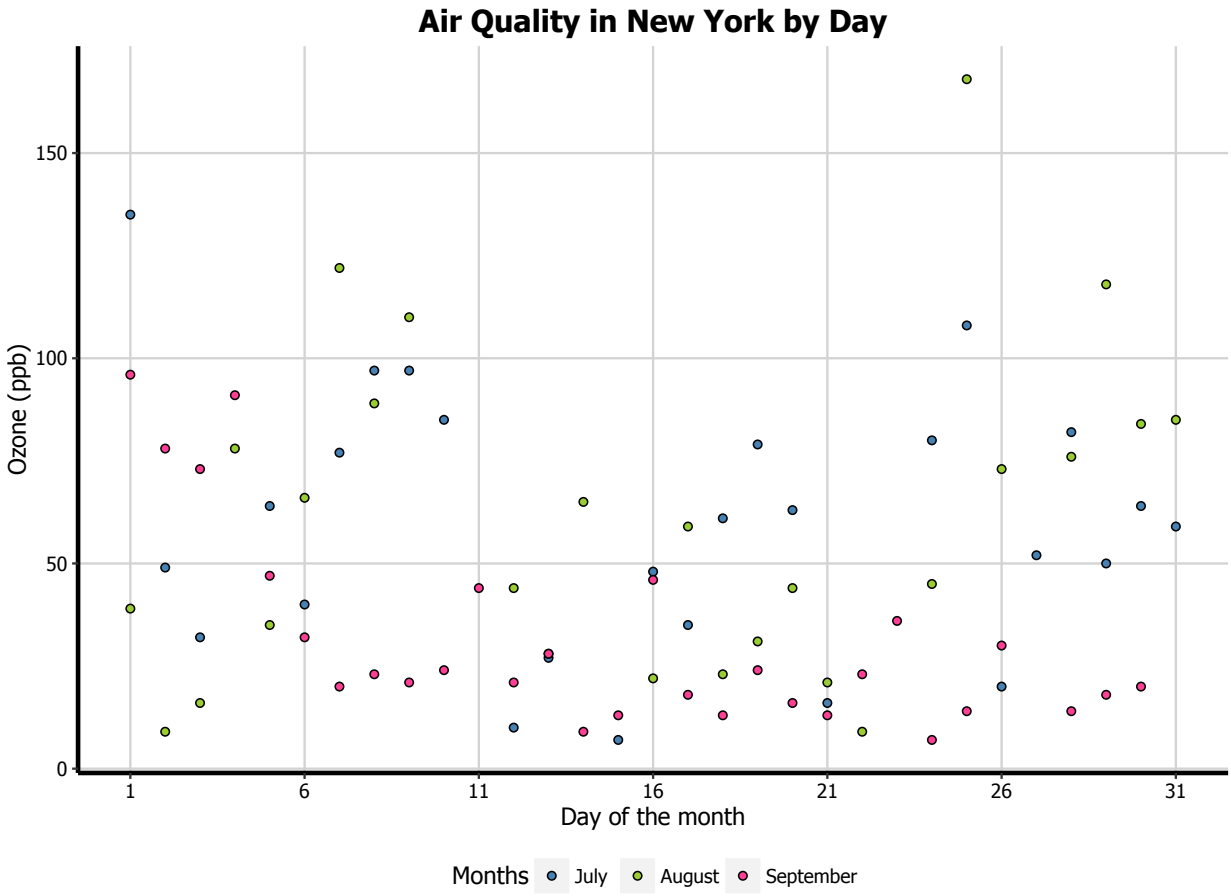
```
p5 <- ggplot(aq_trim, aes(x = Day, y = Ozone, fill = Month)) +
      scale_fill_economist() +
      geom_point(shape = 21) +
      ggtitle("Air Quality in New York by Day") +
      labs(x = "Day of the month", y = "Ozone (ppb)", fill = "Months") +
      scale_x_continuous(breaks = seq(1, 31, 5)) +
      scale_size(range = c(1, 10)) +
      theme_economist() +
      theme(axis.line.x = element_line(size=.5, colour = "black"),
            axis.line.y = element_line(size=.5, colour = "black"),
            axis.title = element_text(size = 12),
            legend.position = "bottom", legend.direction = "horizontal",
            legend.text = element_text(size = 9),
            legend.title=element_text(face = "bold", size = 9),
            text = element_text(family = "Tahoma"),
            plot.title = element_text(family="Tahoma"))
p5
```



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 top of page.

```
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(axis.line.x = element_line(size=1, colour = "black"),
            axis.line.y = element_line(size=1, colour = "black"),
            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
```



Weighted Scatter Plots

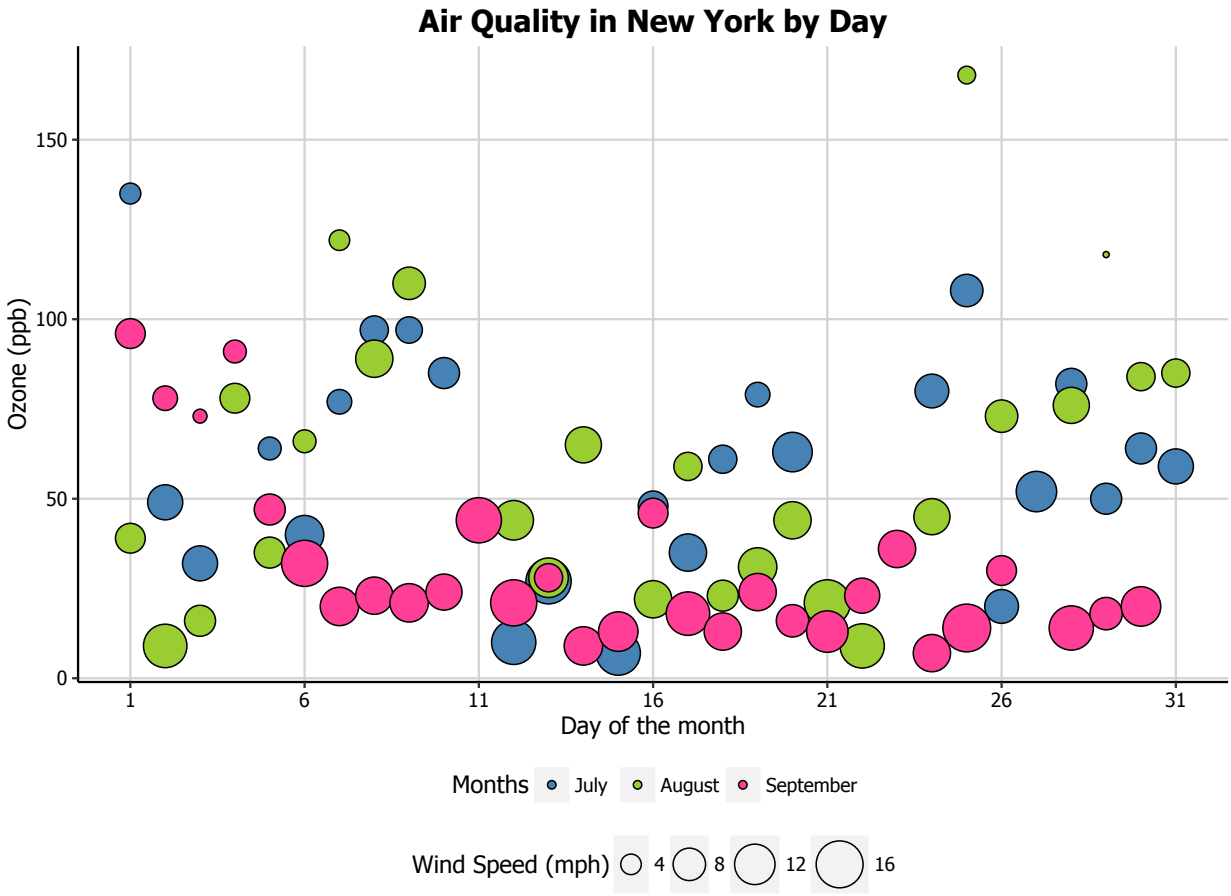
The first thing to do is load in the data, as below:

```
rm(list = ls())
library(datasets)
library(ggplot2)
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"))
```

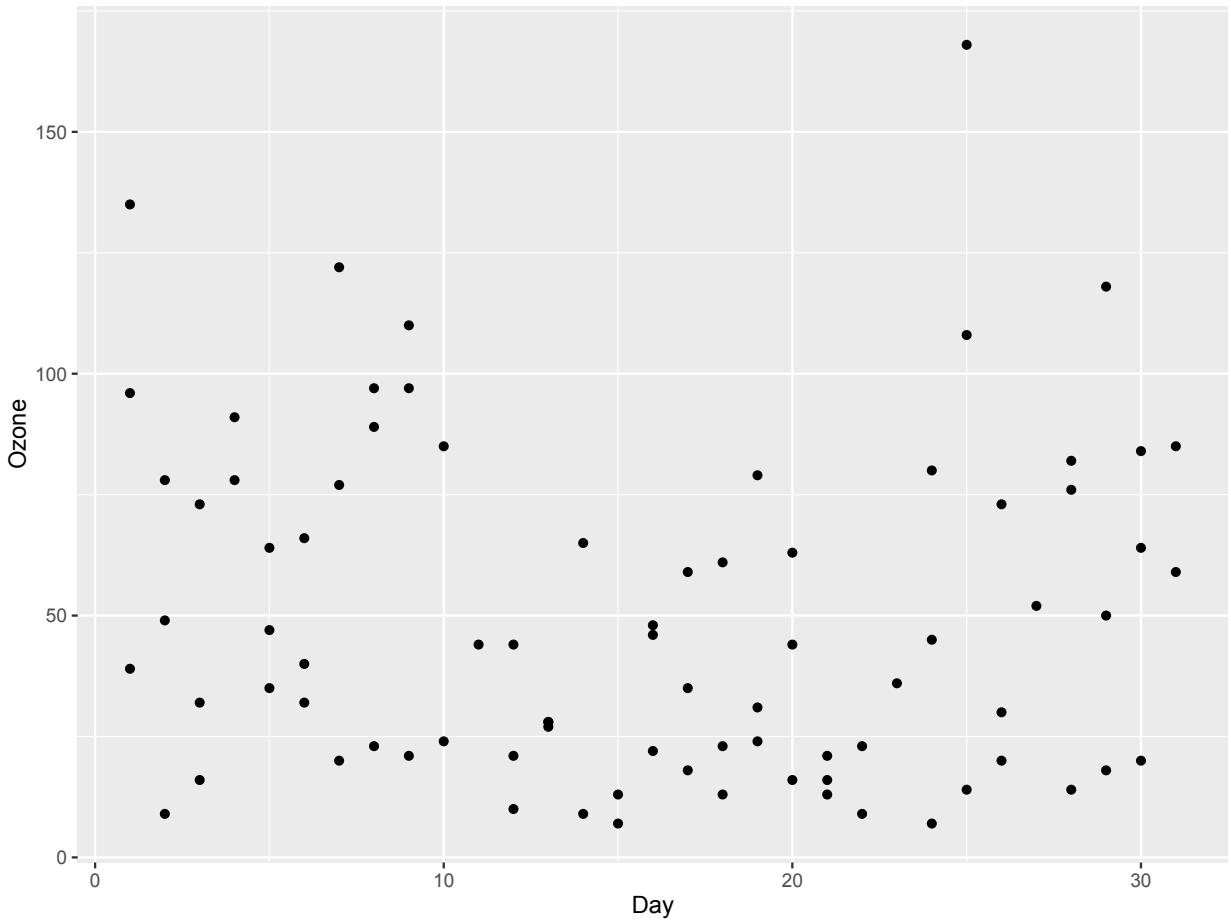
In this part, we will work towards creating the weighted scatterplot below. We will take you from a basic scatterplot and explain all the customisations we add to the code step-by-step.



Basic weighted scatterplot

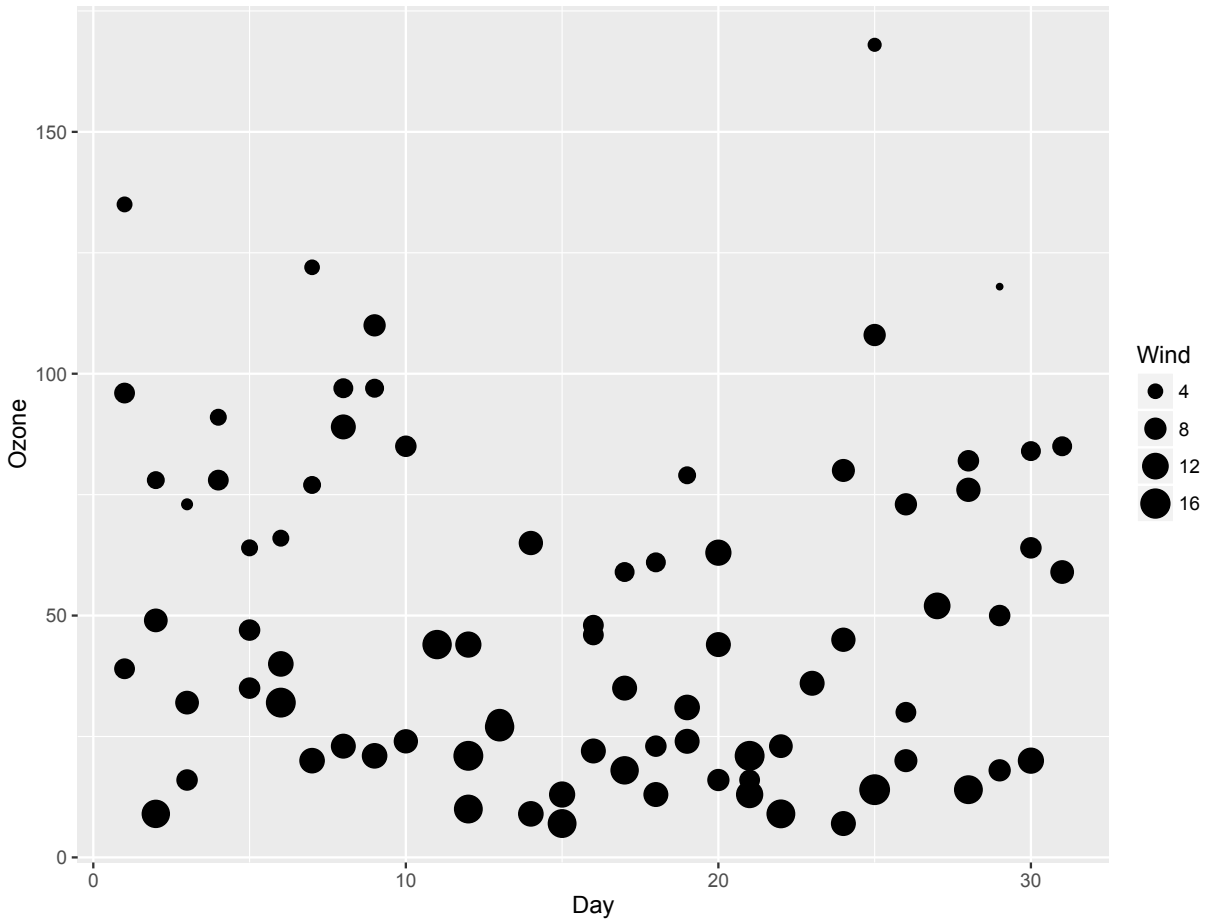
Let's start really slowly by revisiting how to create a basic scatterplot. In order to initialise this plot we tell ggplot that `aq_trim` is our data, and specify that our x-axis plots the `Day` variable and our y-axis plots the `Ozone` variable. We then instruct ggplot to render this as a scatterplot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

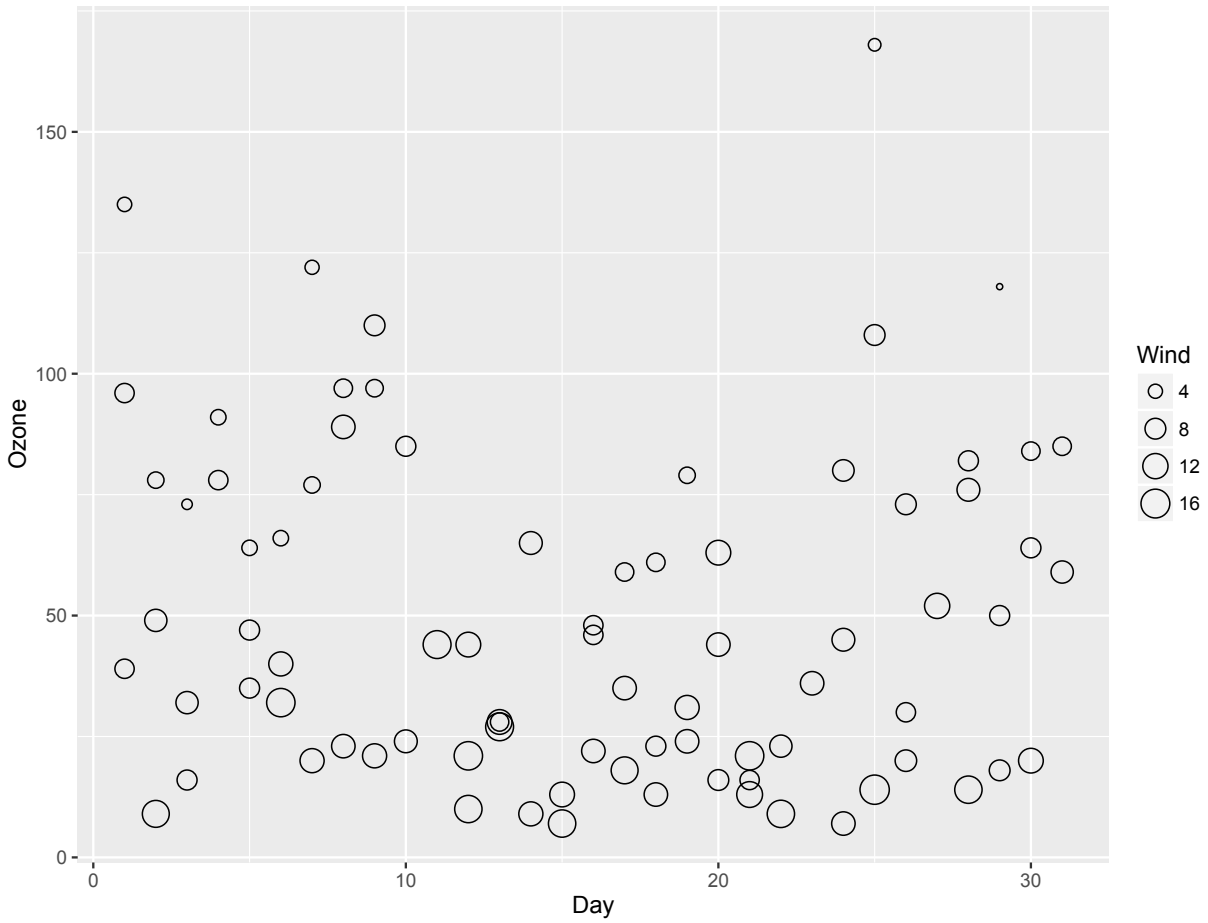


You can see we already have an interesting looking pattern, where days with higher wind speed tend to have lower ozone (or in other words, better air quality). Now let's make it beautiful!

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.

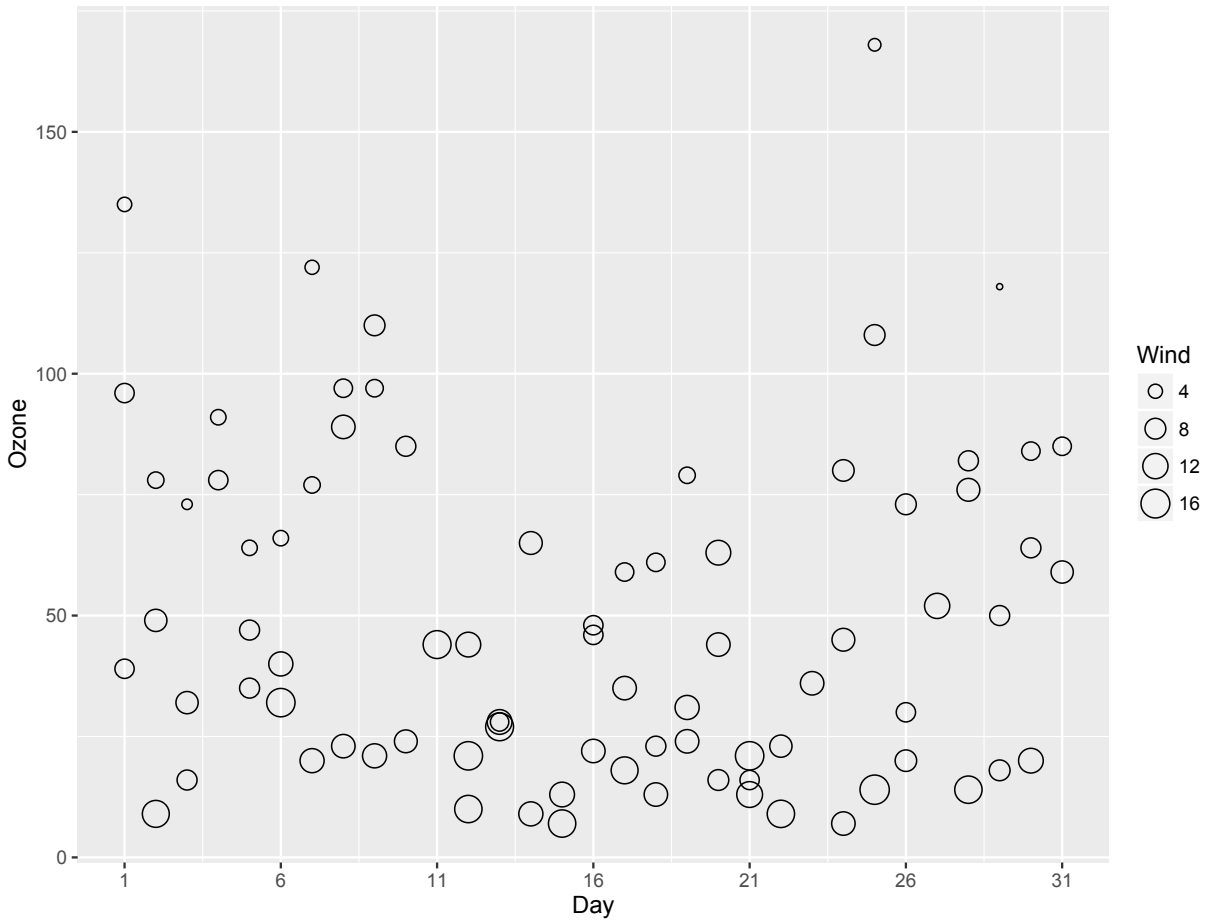
```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind)) +
      geom_point(shape = 21)
p6
```



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).

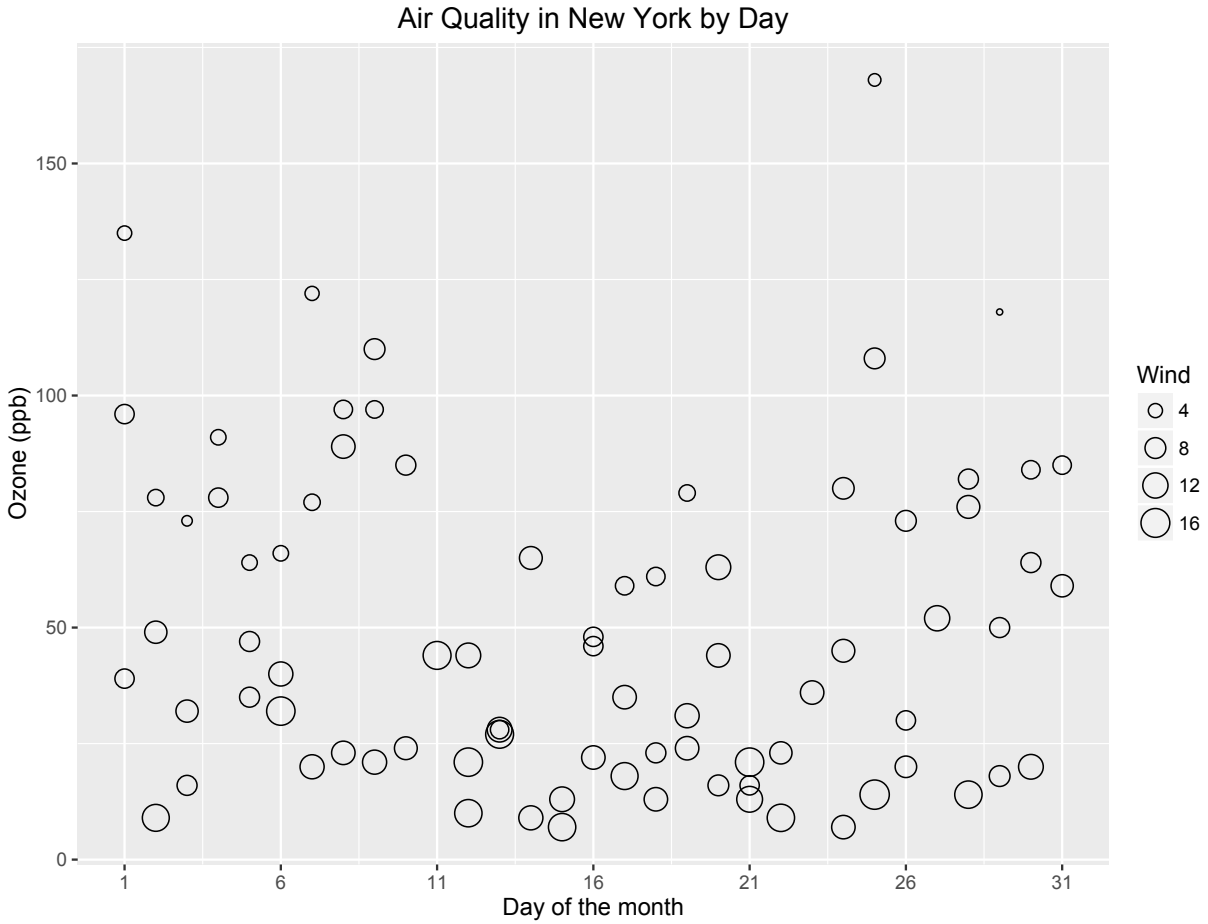
```
p6 <- p6 + scale_x_continuous(breaks = seq(1, 31, 5))
p6
```



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.

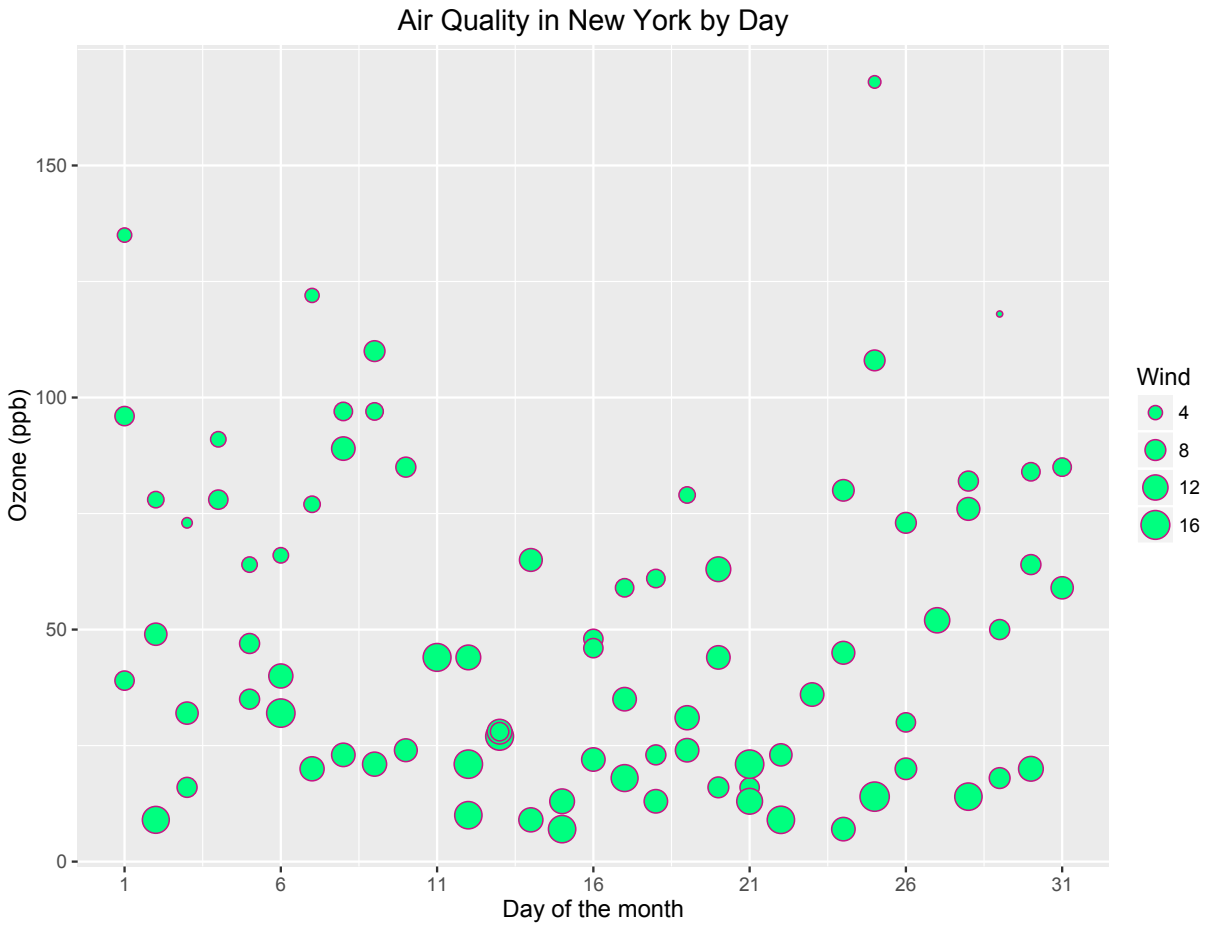
```
p6 <- p6 + ggtitle("Air Quality in New York by Day") +
      labs(x = "Day of the month", y = "Ozone (ppb)")
p6
```



Adjusting the colour palette

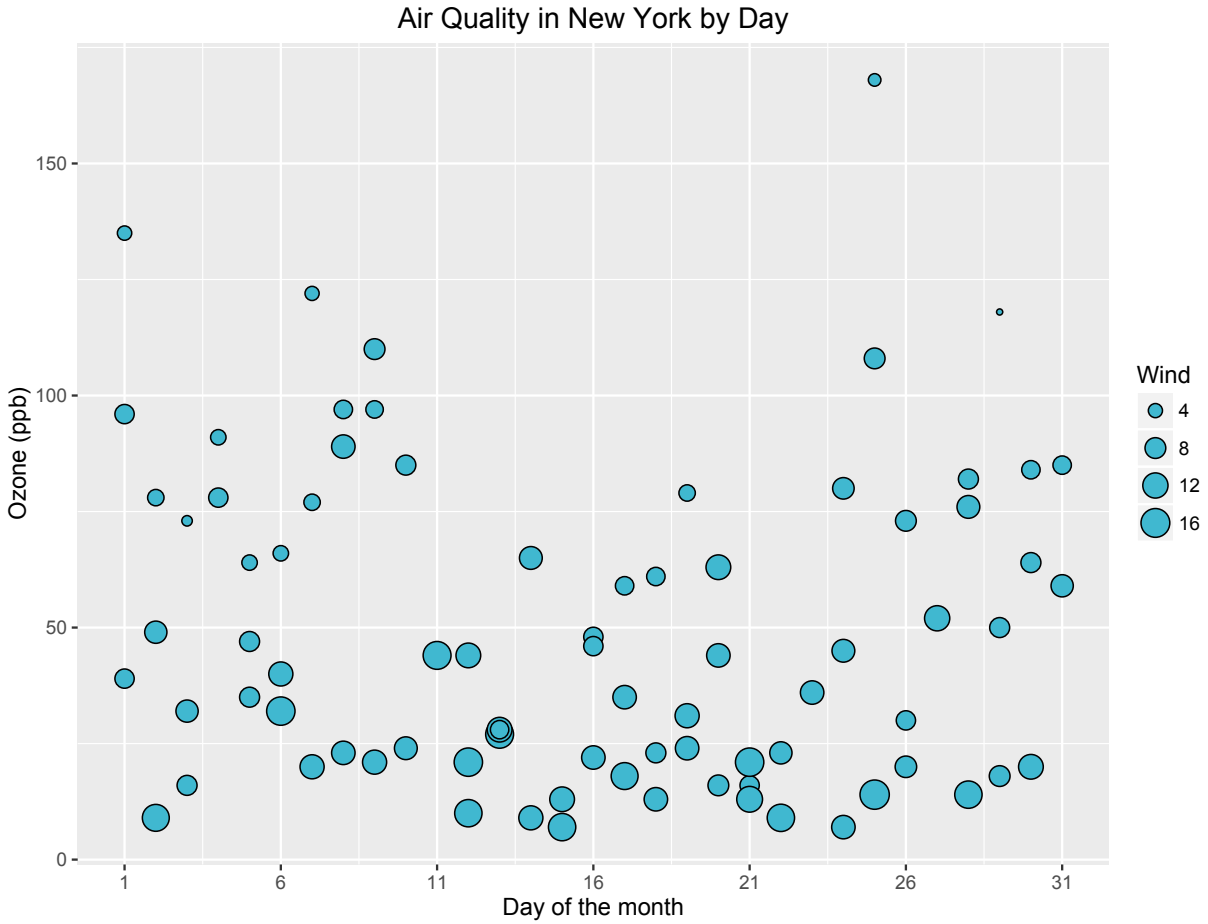
There are a few options for adjusting the colour. The most simple is to make every point one fixed colour. You can reference colours by name, with the full list of colours recognised by R here. Let's try making the outline `mediumvioletred` and the fill `springgreen`.

```
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
```



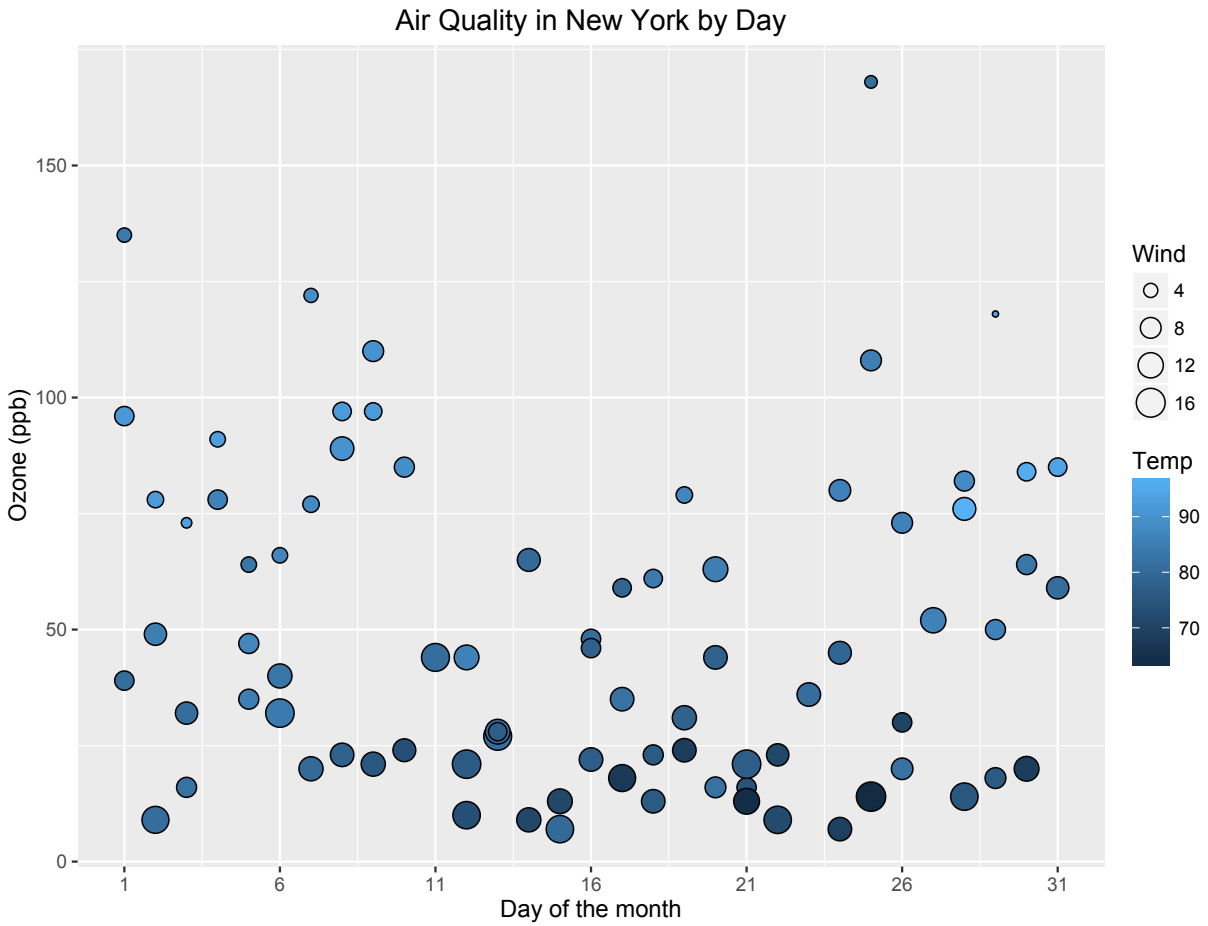
You can change the colours using specific HEX codes instead. Here we have made the outline #000000 (black) and the fill "#40b8d0 (vivid cyan).

```
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
```



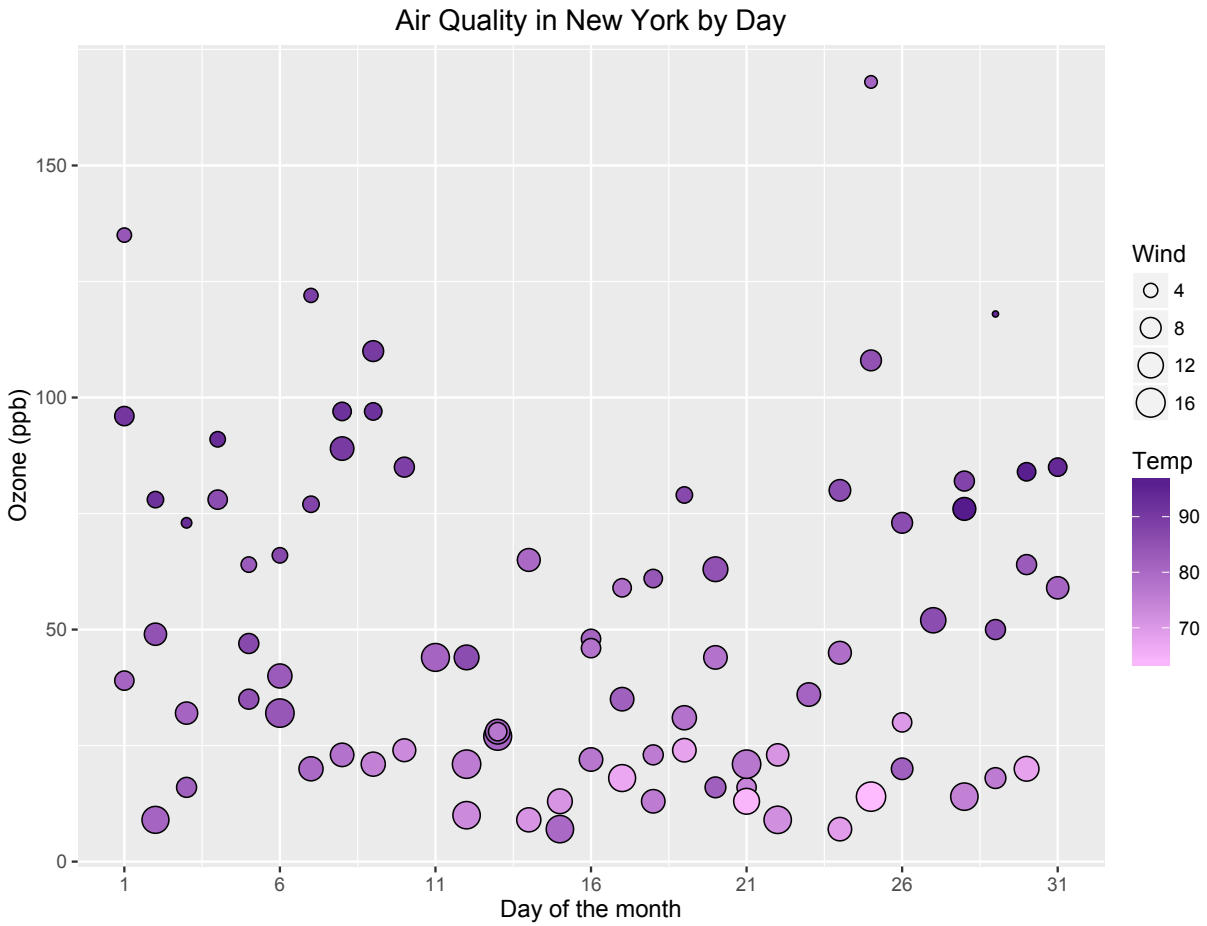
You can also change the colour of the data points according to the levels of another variable. This can be done either as a continuous gradient, or as a levels of a factor variable. Let's change the colour by the values of temperature:

```
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.

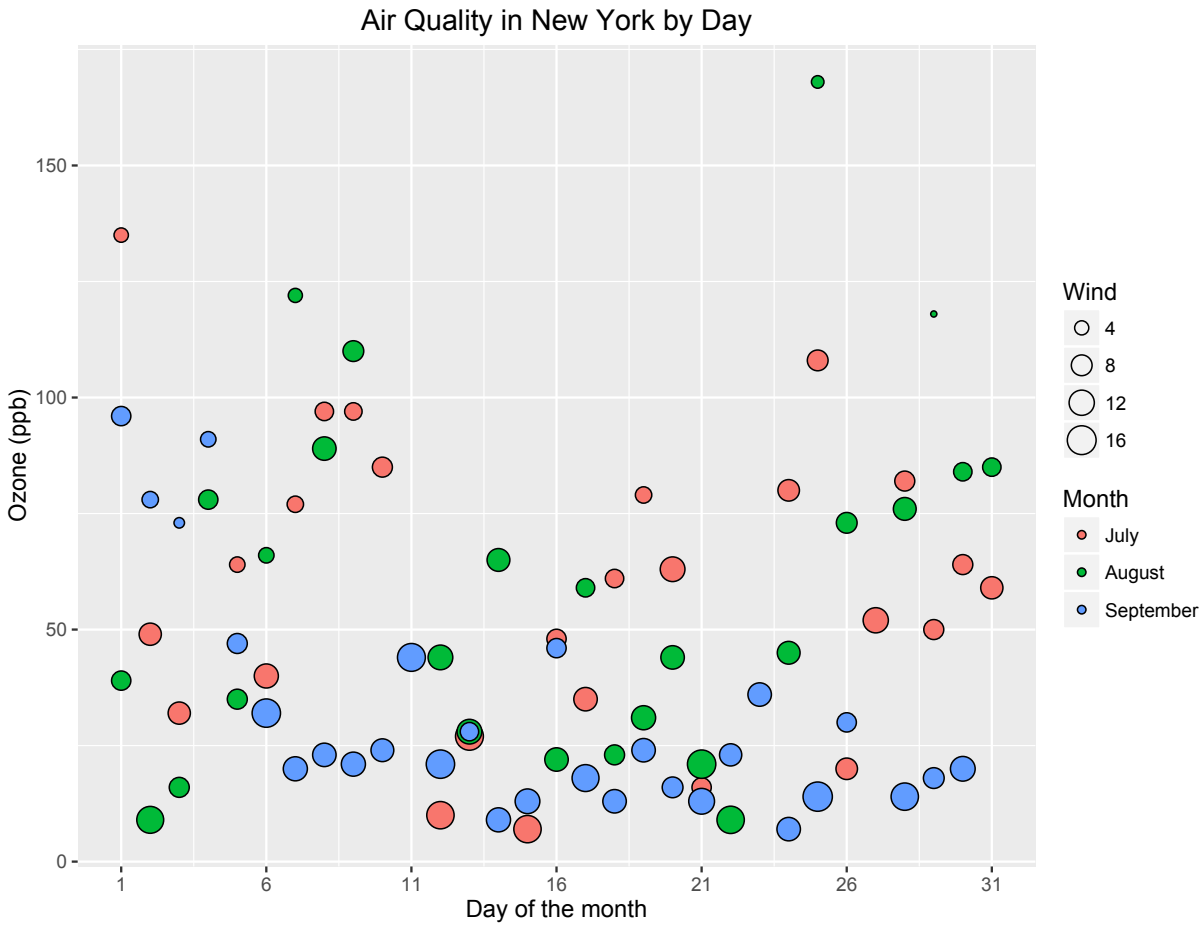
```
p6 <-p6 + scale_fill_continuous(low = "plum1", high = "purple4")
p6
```



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

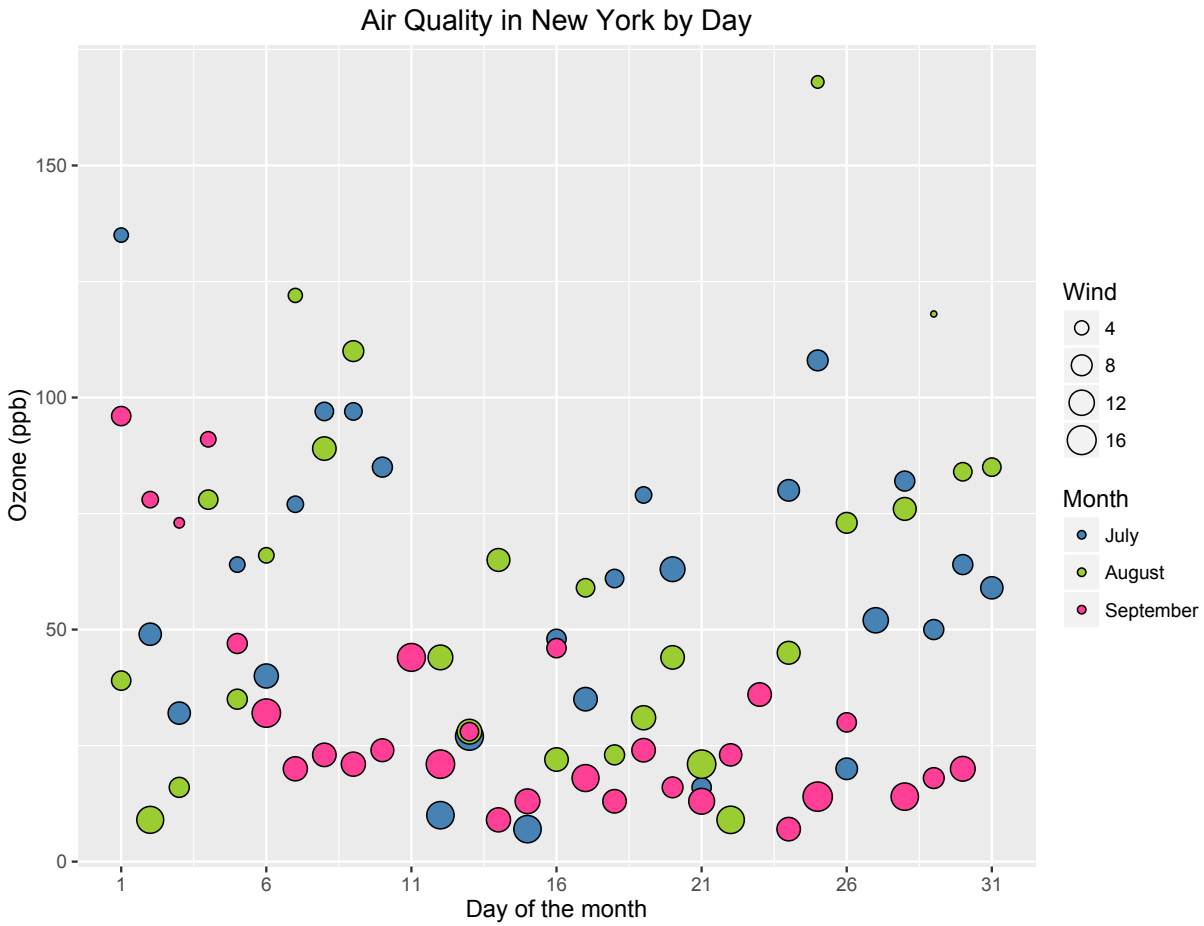
Let's now change the colours of the data points by a factor variable, `Month`.

```
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`.

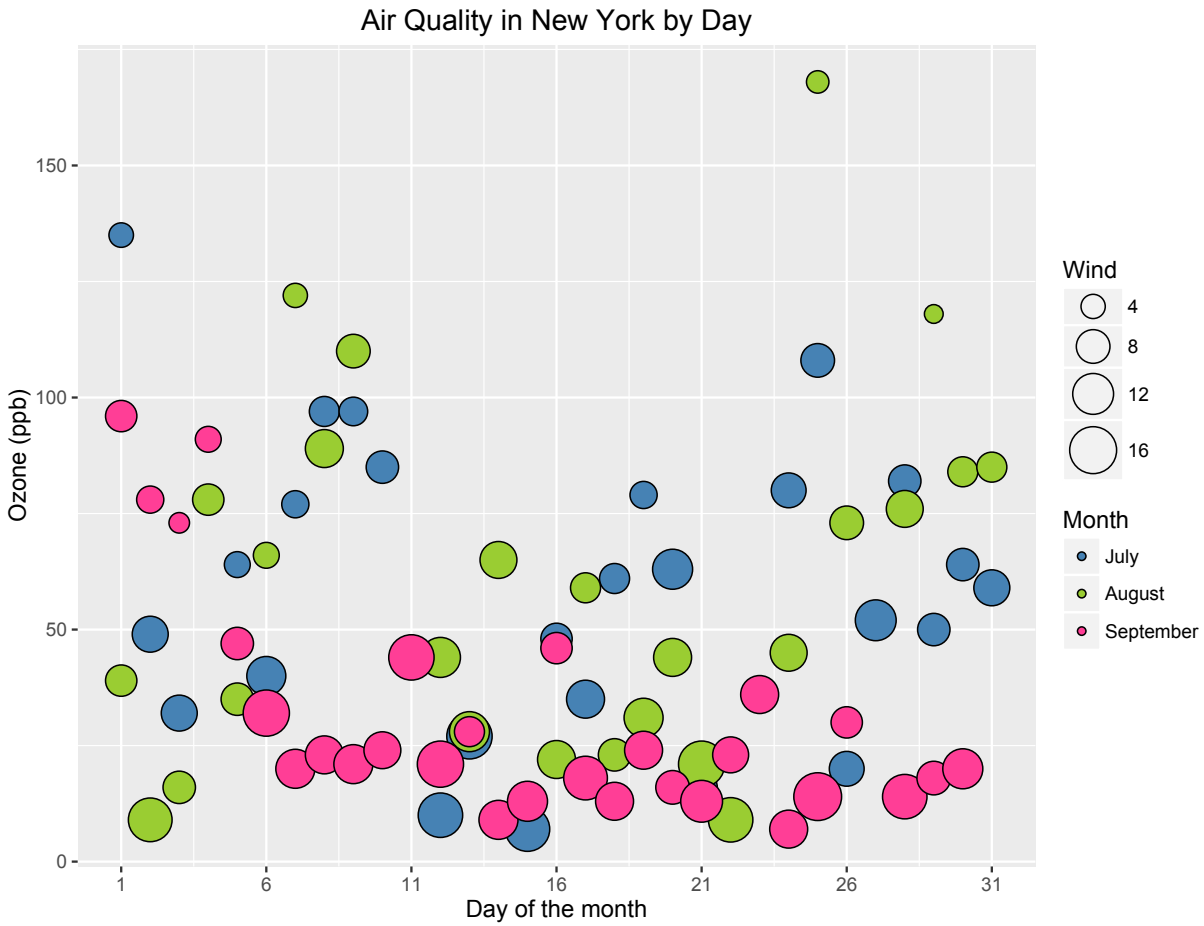
```
fill = c("steelblue", "yellowgreen", "violetred1")

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



Adjusting the size of the data points

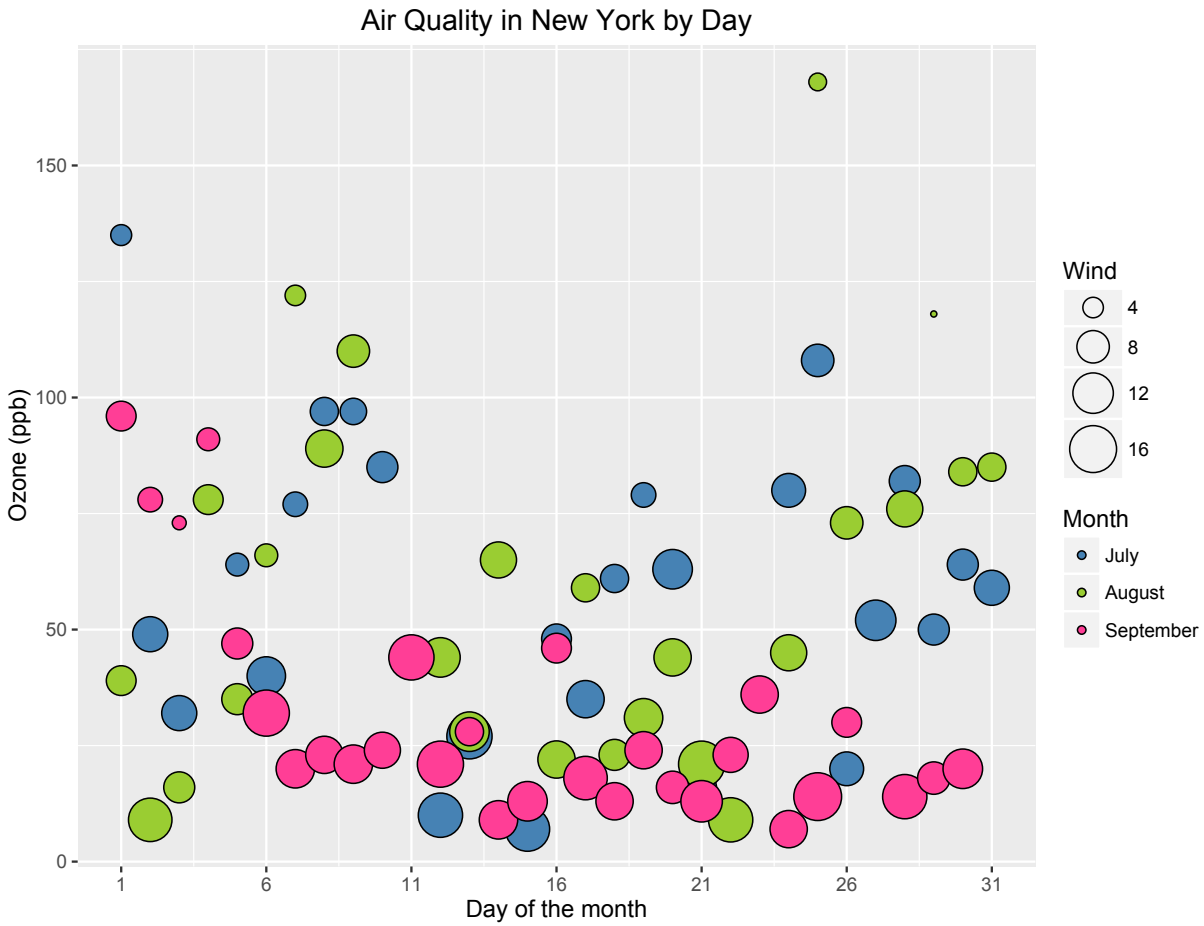
The default size of the the data points in a weighted scatterplot is mapped to the radius of the plots. If we want the data points to be proportional to the value of the weighting variable (e.g., a wind speed of 0 mph would have a value of 0), we need to use the `scale_size_area`.

```
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.

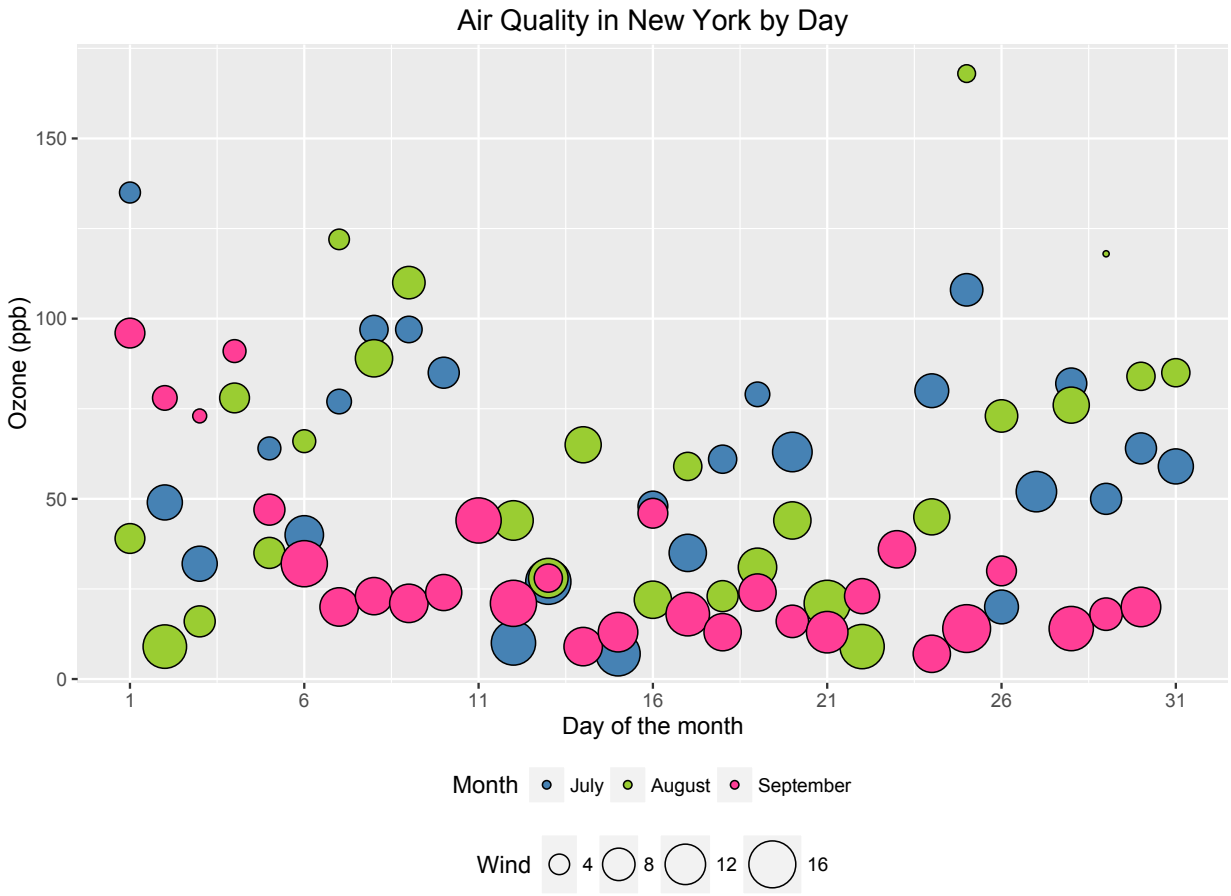
```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
      geom_point(shape = 21) +
      ggtitle("Air Quality in New York by Day") +
      labs(x = "Day of the month", y = "Ozone (ppb)") +
      scale_x_continuous(breaks = seq(1, 31, 5)) +
      scale_fill_manual(values =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



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.

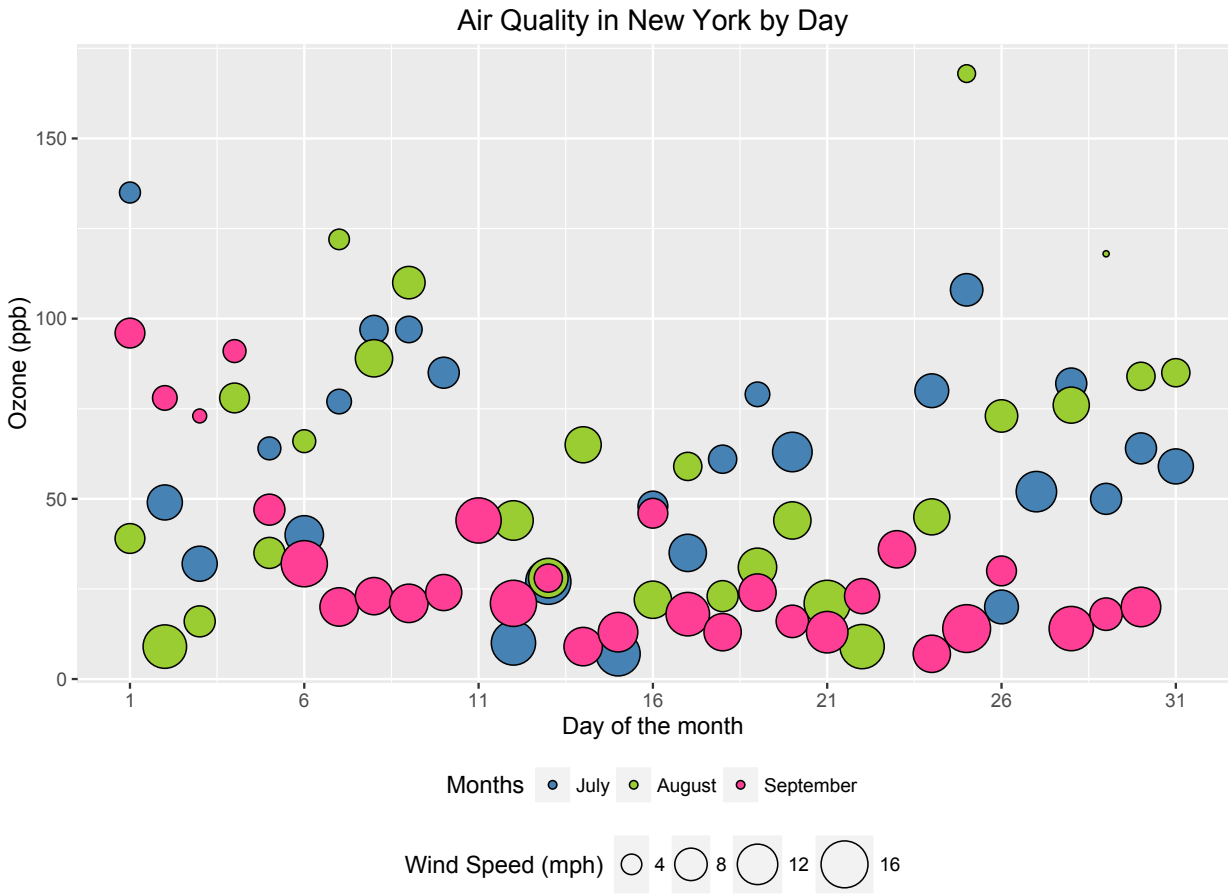
```
p6 <- p6 + theme(legend.position = "bottom", legend.direction = "horizontal")
p6
```



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.

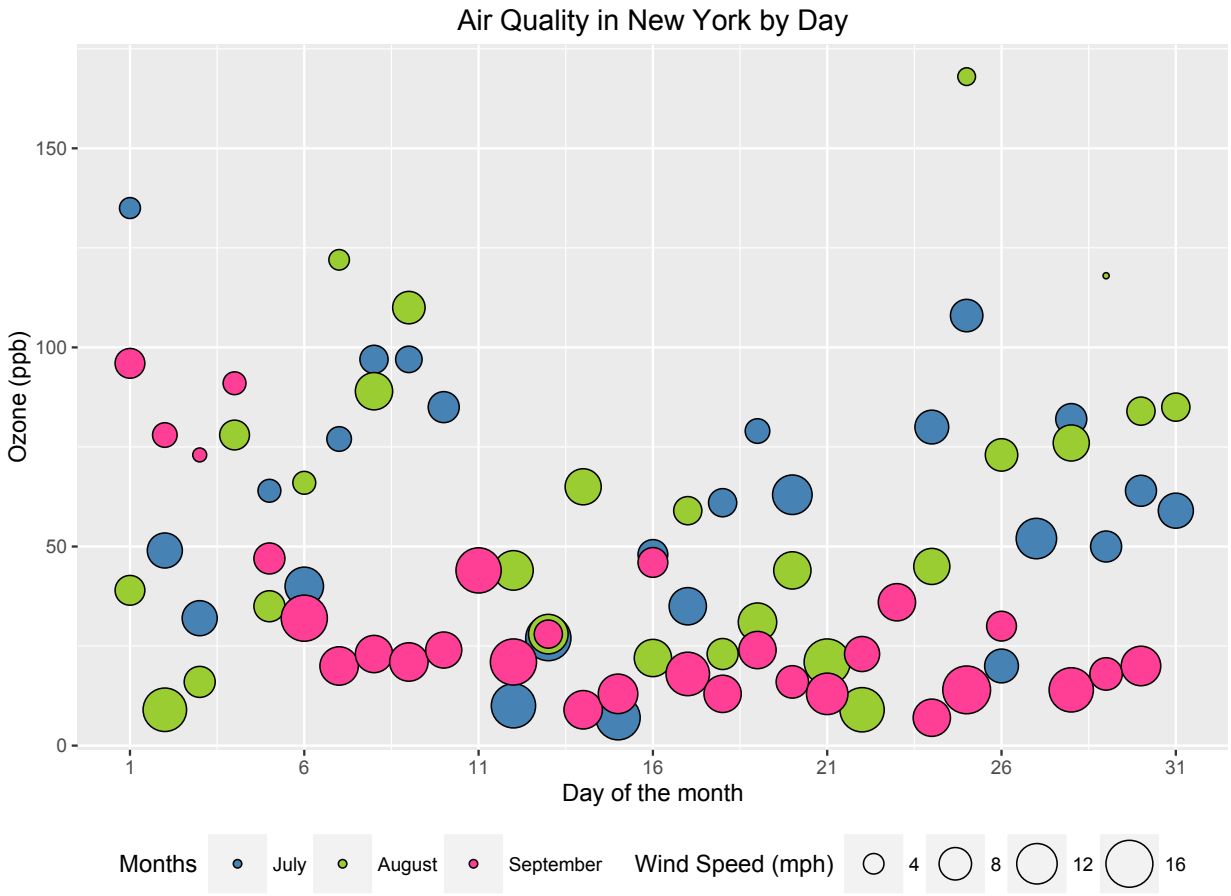
```
p6 <- p6 + labs(size = "Wind Speed (mph)", fill = "Months")
p6
```



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.

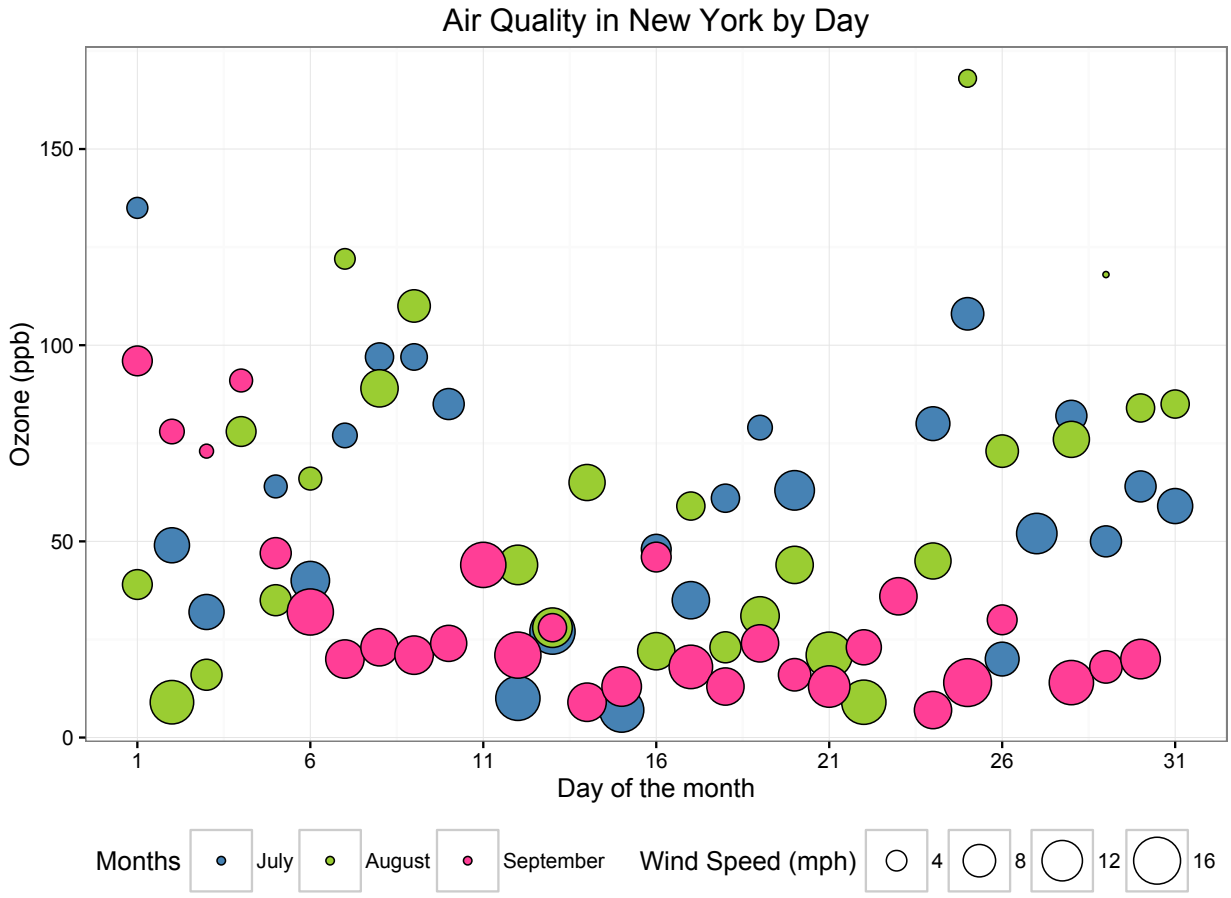
```
library(grid)
p6 <- p6 + theme(legend.box = "horizontal", legend.key.size = unit(1, "cm"))
p6
```



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.

```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) + theme_bw() +
      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(legend.position="bottom", legend.direction="horizontal",
            legend.box = "horizontal",
            legend.key.size = unit(1, "cm"))
p6
```



Creating an XKCD style chart

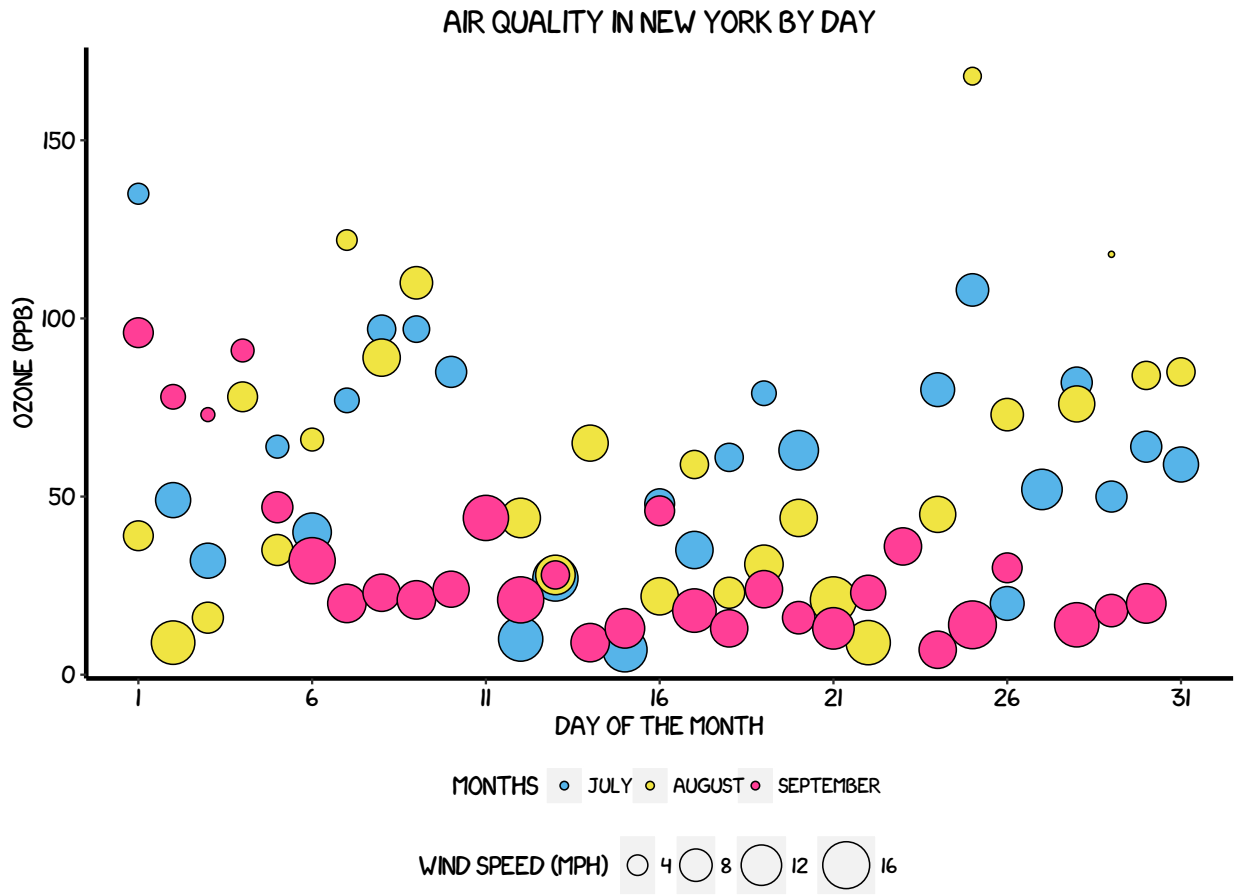
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

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

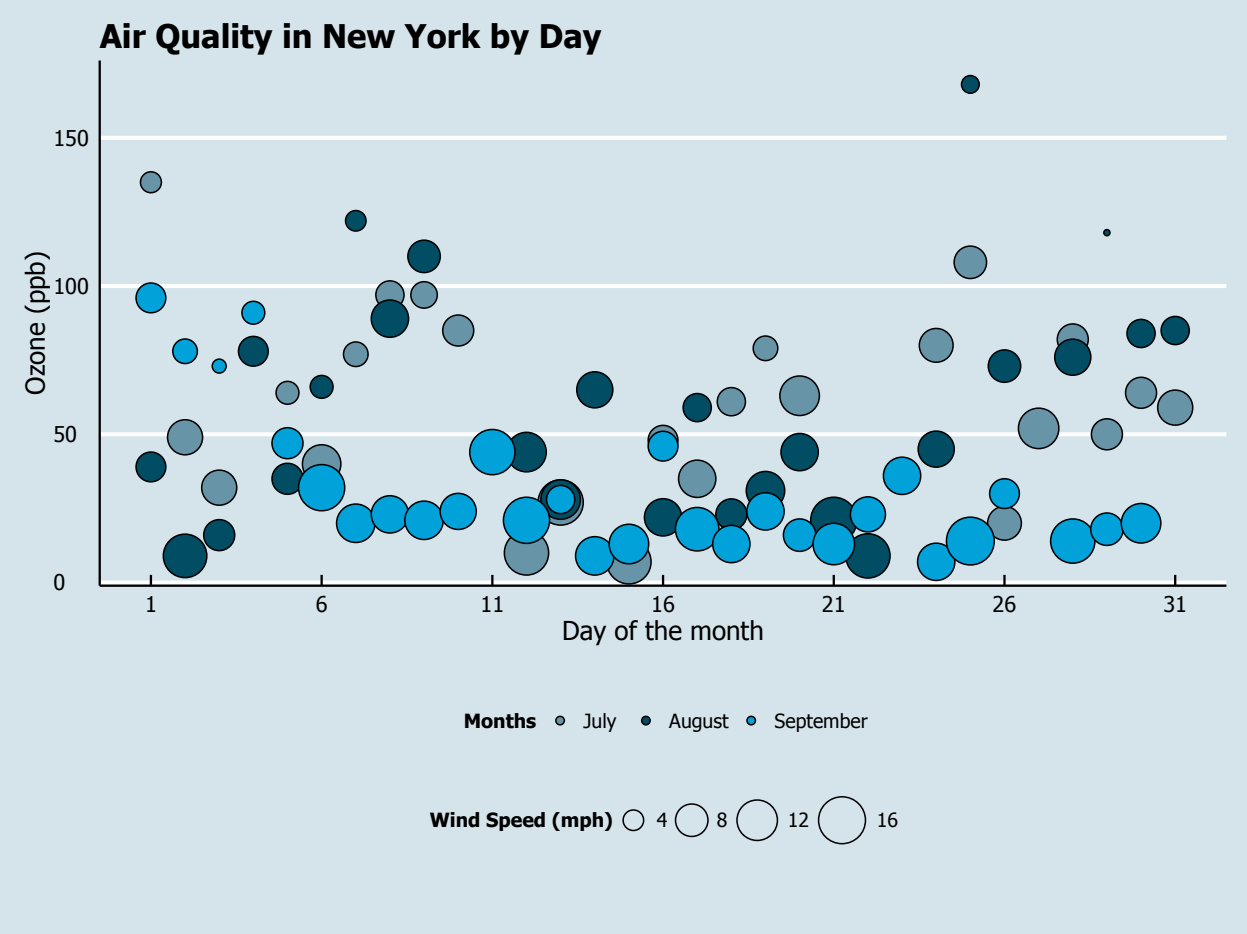
```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
      geom_point(shape = 21) +
      ggtitle("Air Quality in New York by Day") +
      labs(x = "Day of the month", y = "Ozone (ppb)",
       size = "Wind Speed (mph)", fill = "Months ") +
      scale_x_continuous(breaks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



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.

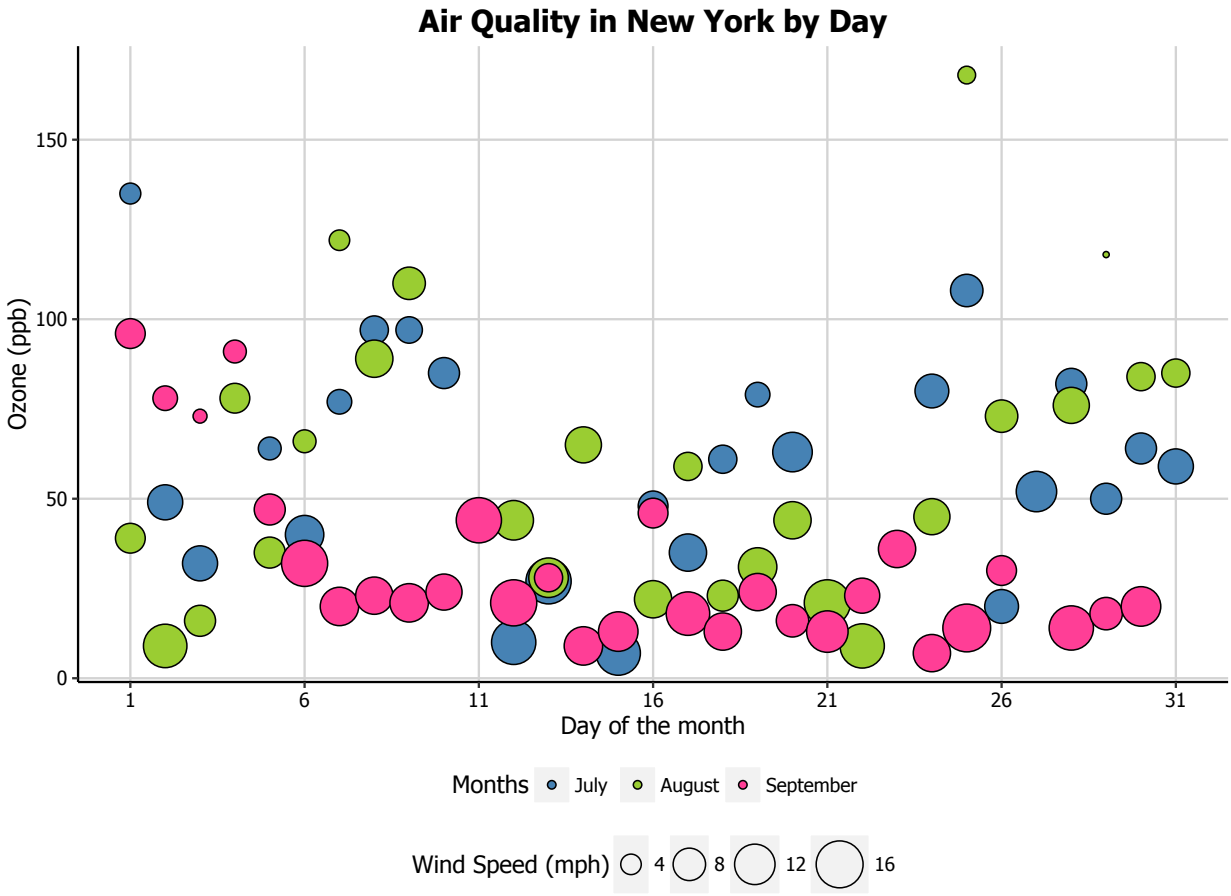
```
p6 <- ggplot(aq_trim, aes(x = Day, y = Ozone, size = Wind, fill = Month)) +
      scale_fill_economist() +
      geom_point(shape = 21) +
      ggtitle("Air Quality in New York by Day") +
      labs(x = "Day of the month", y = "Ozone (ppb)", size = "Wind Speed (mph)  ", fill = "Months ") +
      scale_x_continuous(breaks = seq(1, 31, 5)) +
      scale_size(range = c(1, 10)) +
      theme_economist() +
      theme(axis.line.x = element_line(size=.5, colour = "black"),
            axis.line.y = element_line(size=.5, colour = "black"),
            axis.title = element_text(size = 12),
            legend.position = "bottom", legend.direction = "horizontal",
            legend.text = element_text(size = 9),
            legend.title=element_text(face = "bold", size = 9),
            text = element_text(family = "Tahoma"),
            plot.title = element_text(family="Tahoma"))
p6
```



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 top of page.

```
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(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 = 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"))
p6
```

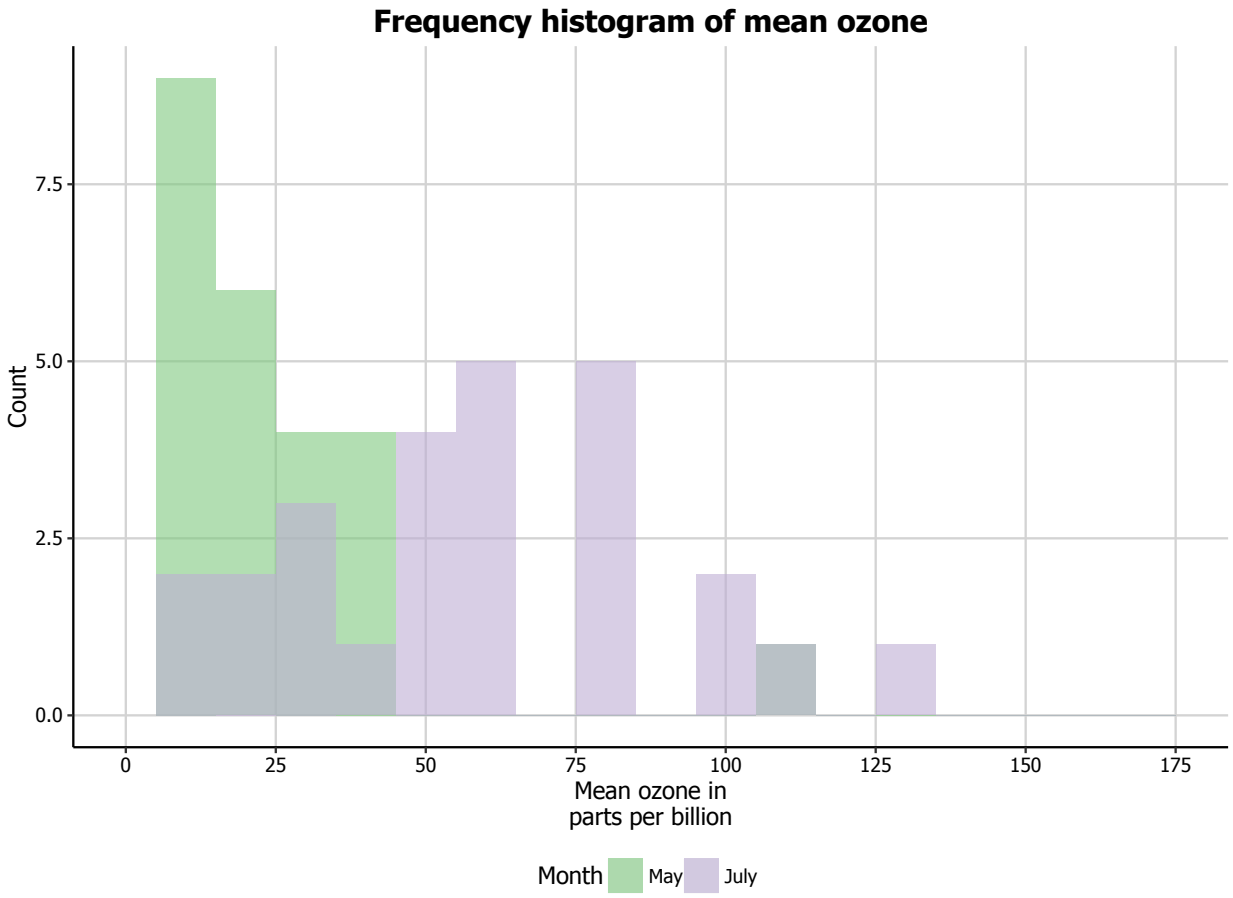


Histograms

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

```
rm(list = ls())
library(datasets)
library(ggplot2)

data(airquality)
```

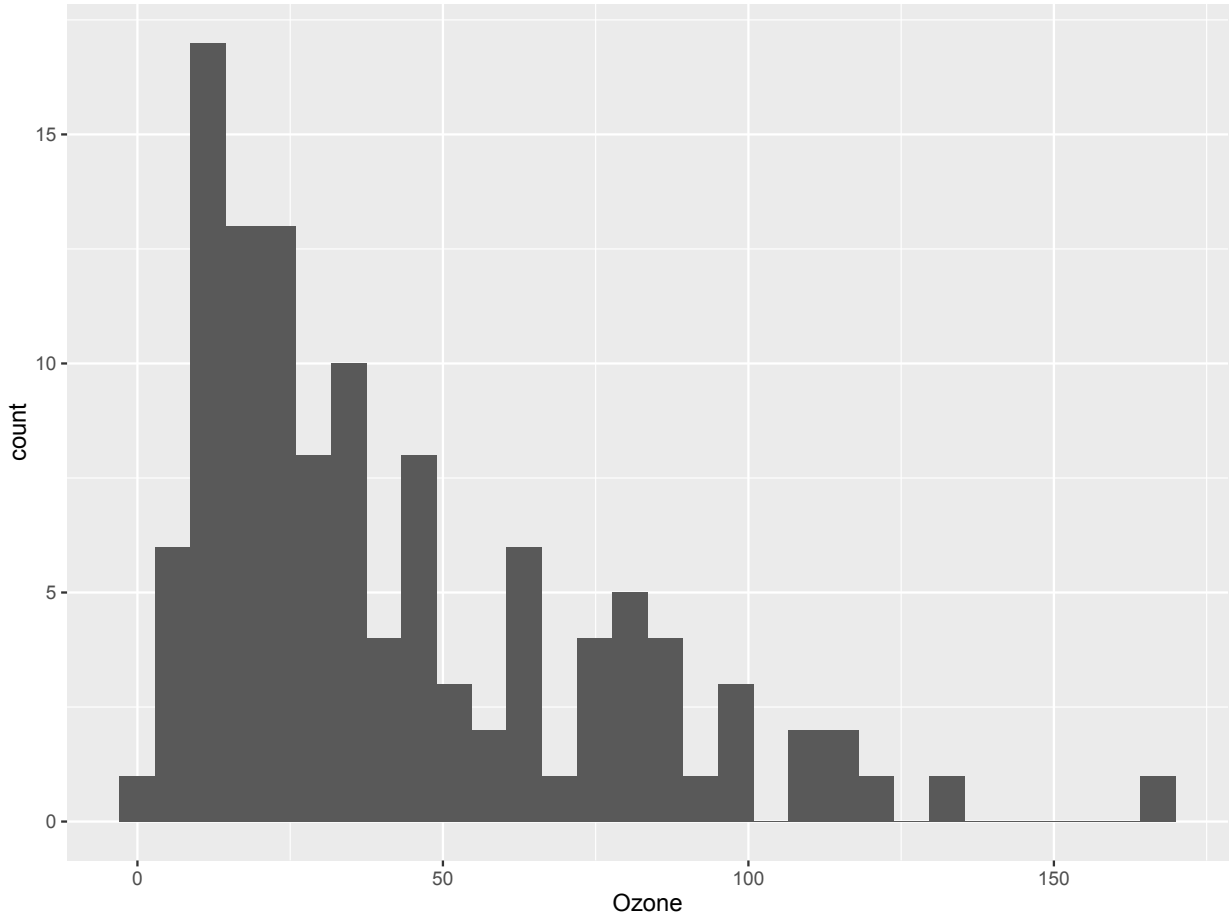
In this part, we will work towards creating the histogram below. We will take you from a basic histogram and explain all the customisations we add to the code step-by-step.



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
```

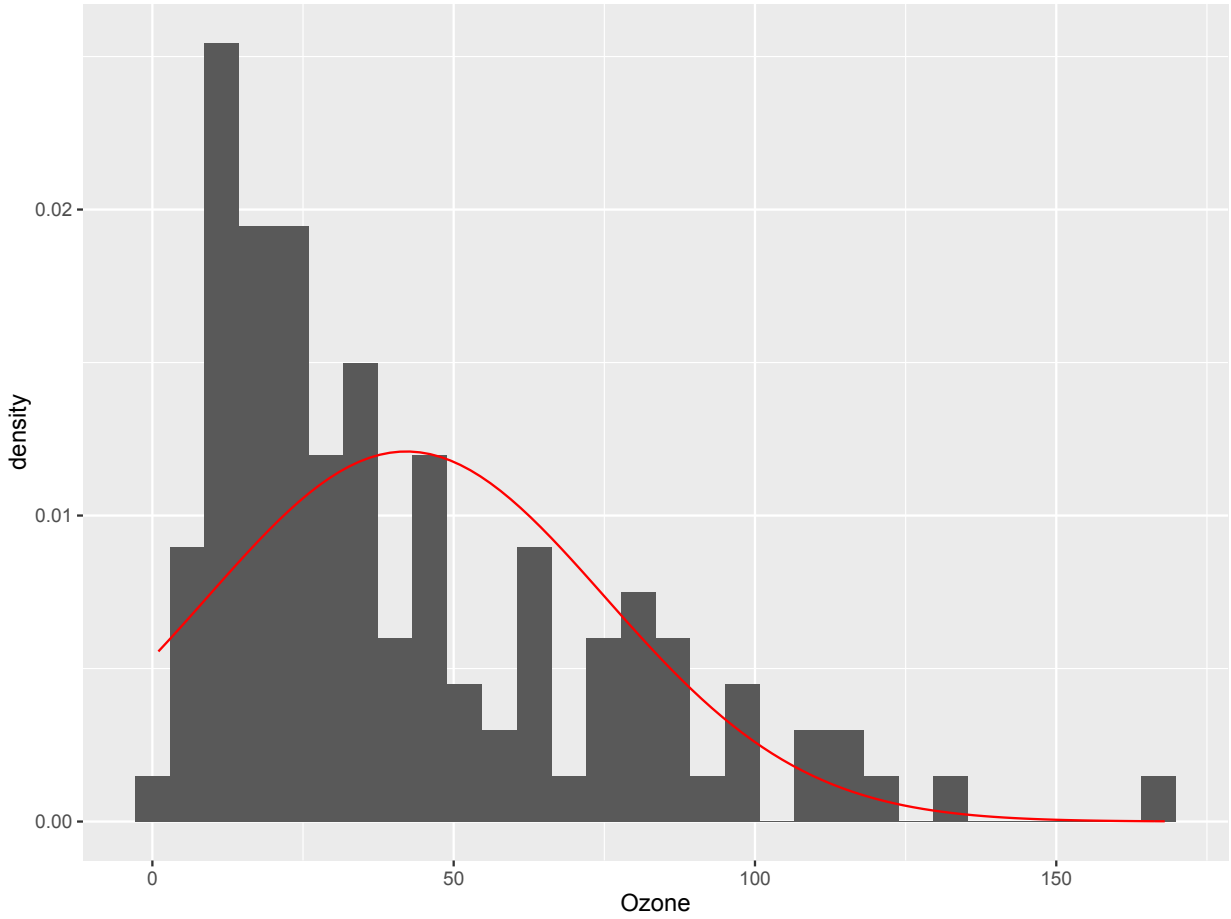


Adding a normal density curve

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.

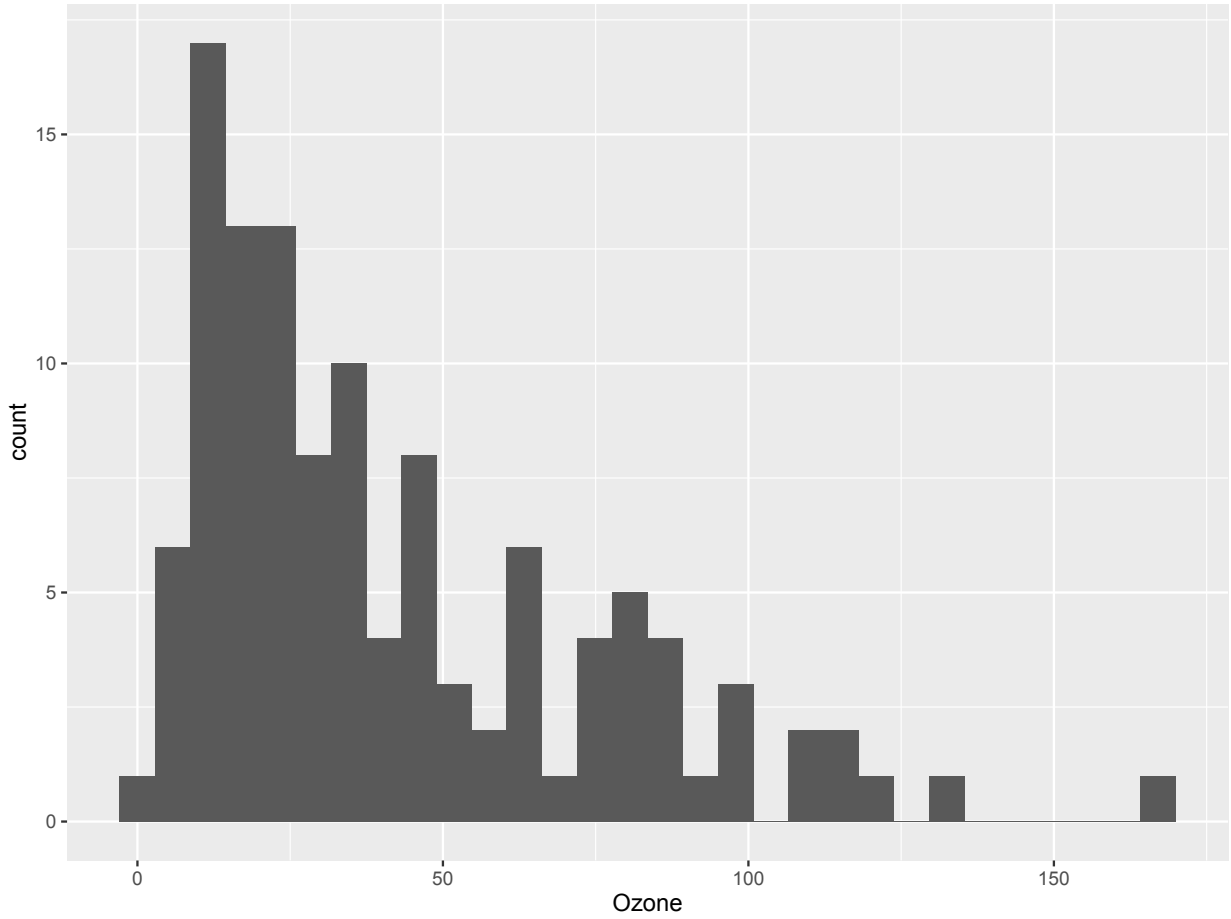
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
      geom_histogram(aes(y = ..density..)) +
      stat_function(fun = dnorm, colour = "red",
                    args = list(mean = mean(airquality$Ozone, na.rm = TRUE),
                                sd = sd(airquality$Ozone, na.rm = TRUE)))
p7
```



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`.

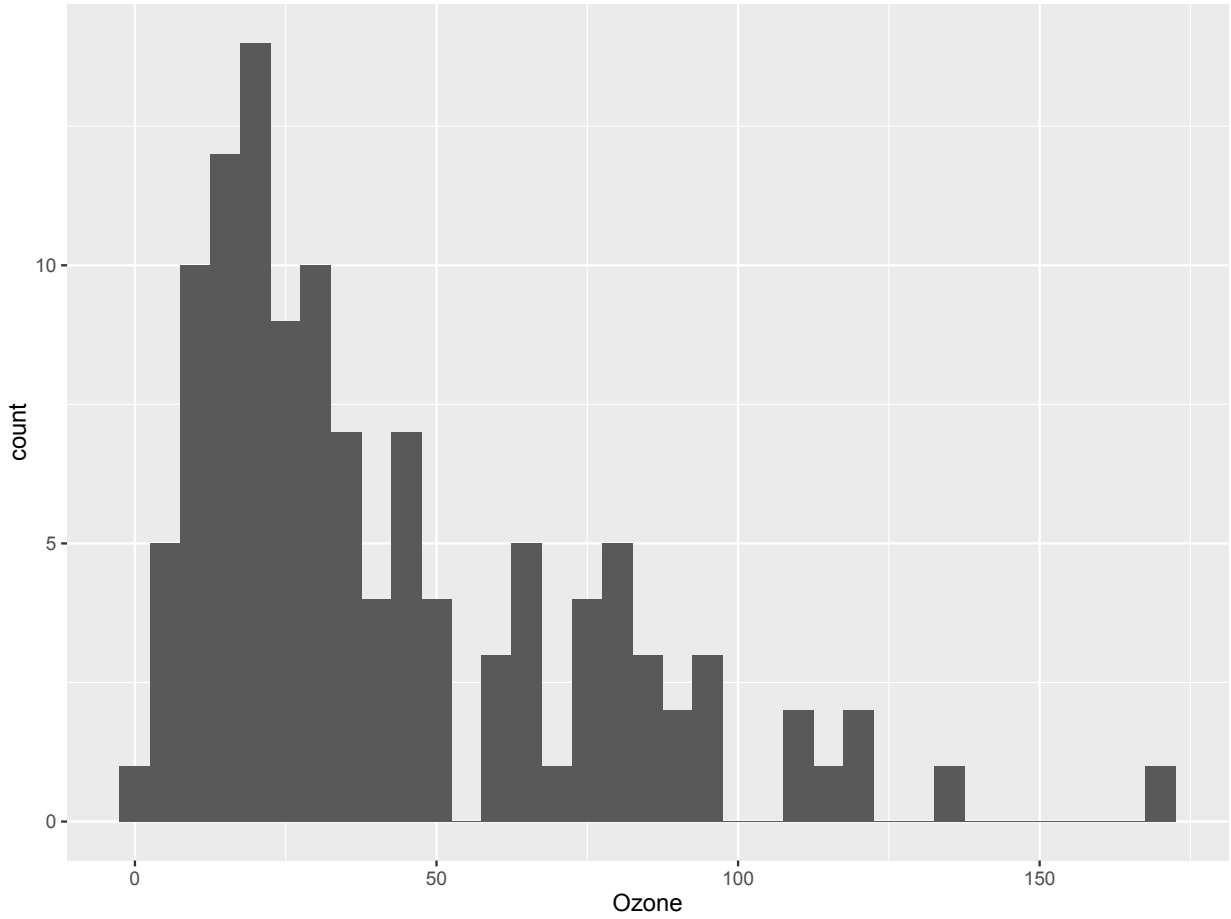
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
      geom_histogram(aes(y = ..count..))
p7
```



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
```

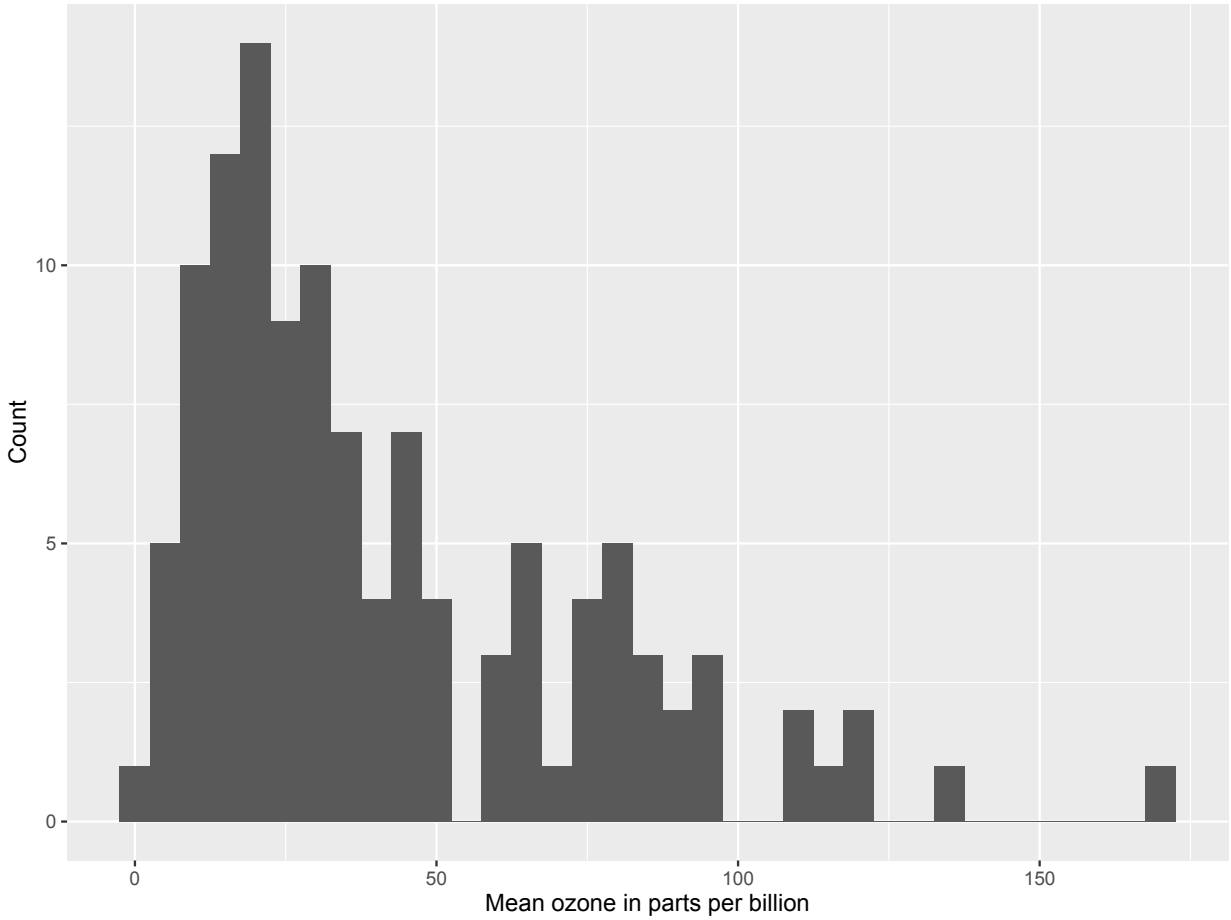


Customising axis labels

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.

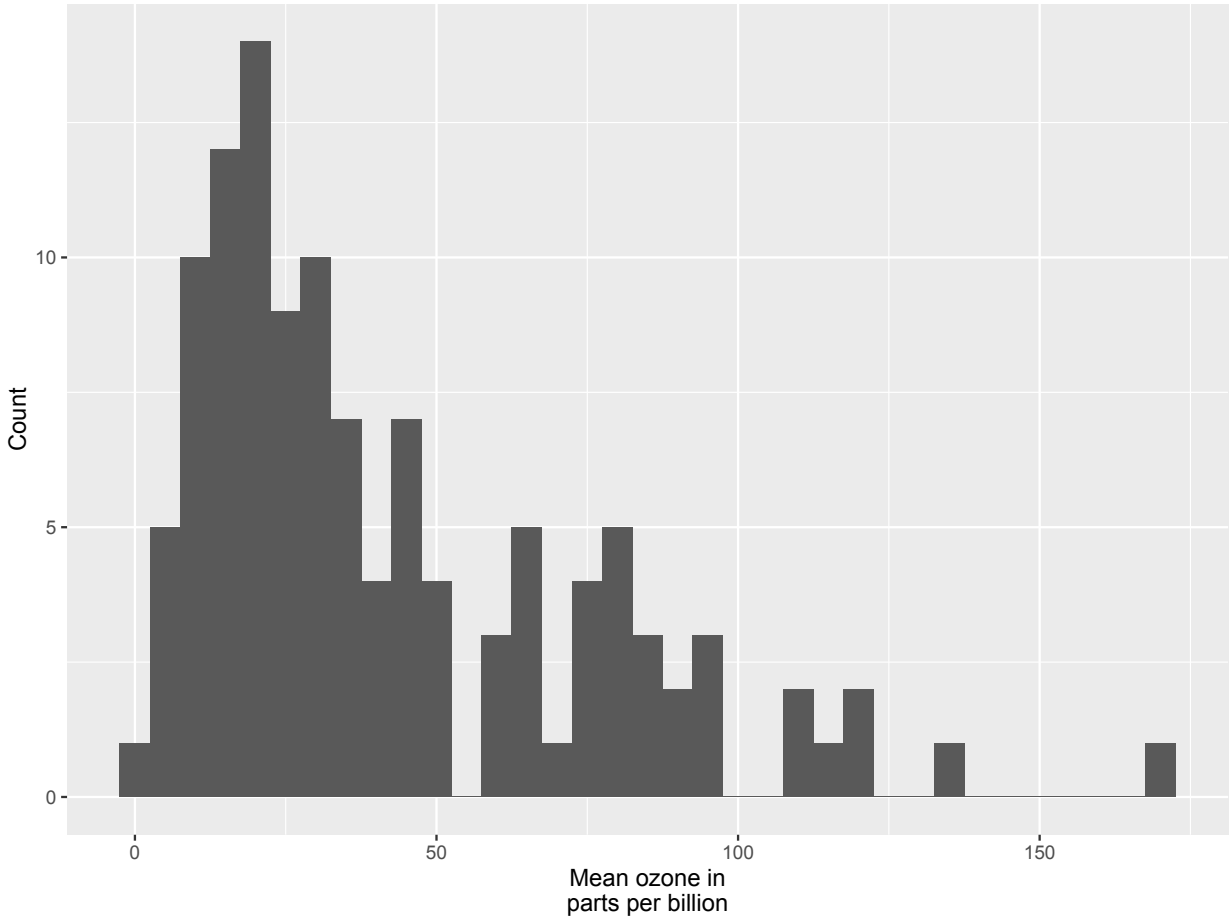
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
      geom_histogram(aes(y = ..count..), binwidth = 5) +
      scale_x_continuous(name = "Mean ozone in parts per billion") +
      scale_y_continuous(name = "Count")
p7
```



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.

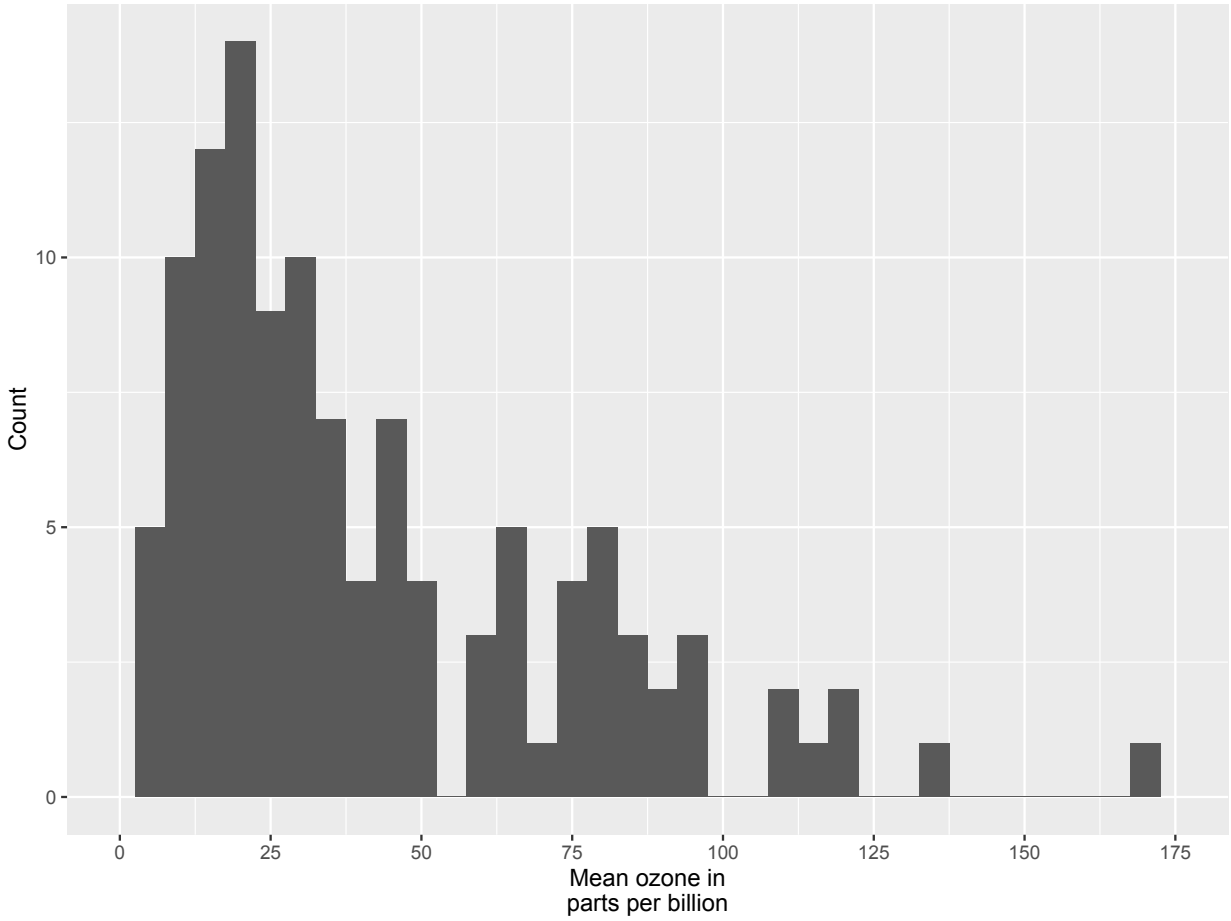
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
      geom_histogram(aes(y = ..count..), binwidth = 5) +
      scale_x_continuous(name = "Mean ozone in\nparts per billion") +
      scale_y_continuous(name = "Count")
p7
```



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`.

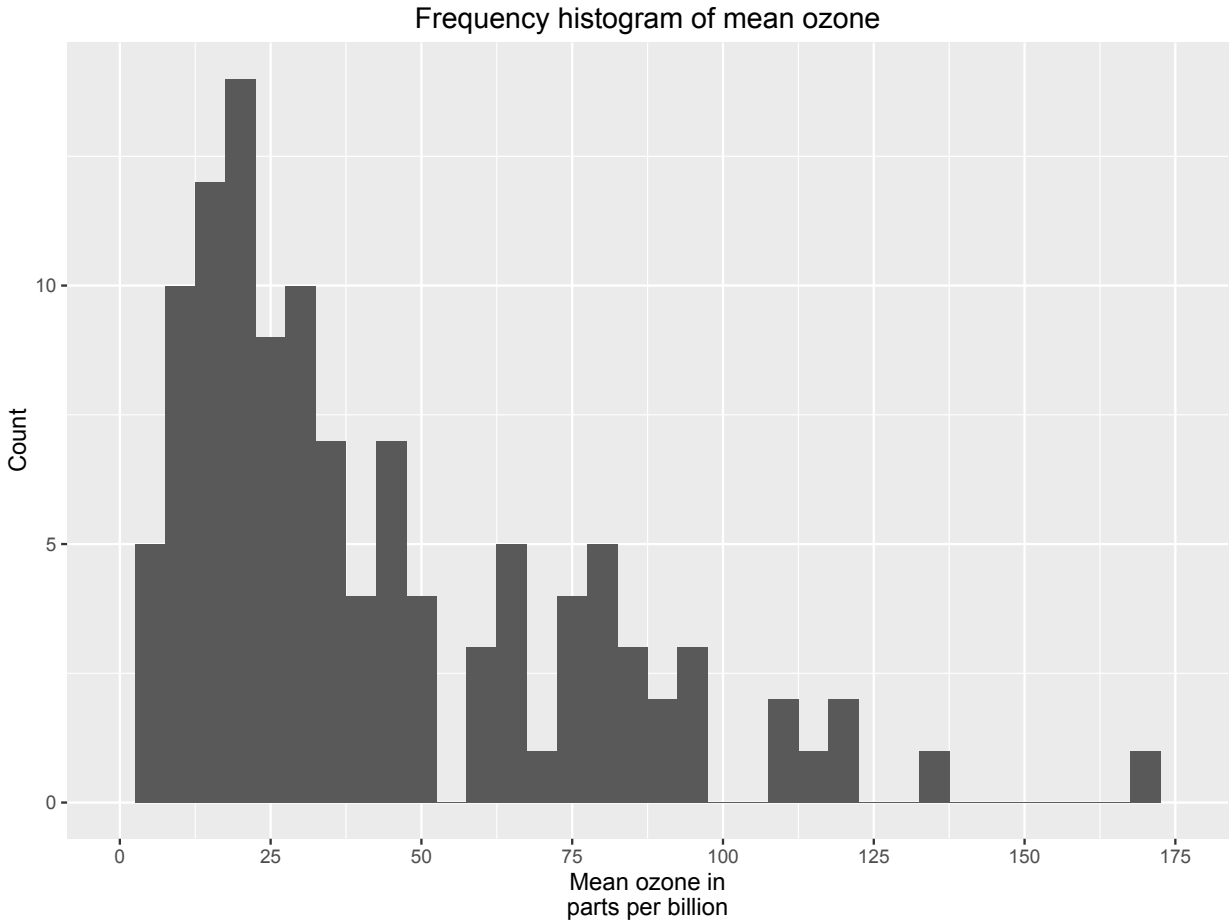
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
      geom_histogram(aes(y = ..count..), binwidth = 5) +
      scale_x_continuous(name = "Mean ozone in\nparts per billion",
                         breaks = seq(0, 175, 25),
                         limits=c(0, 175)) +
      scale_y_continuous(name = "Count")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



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
```

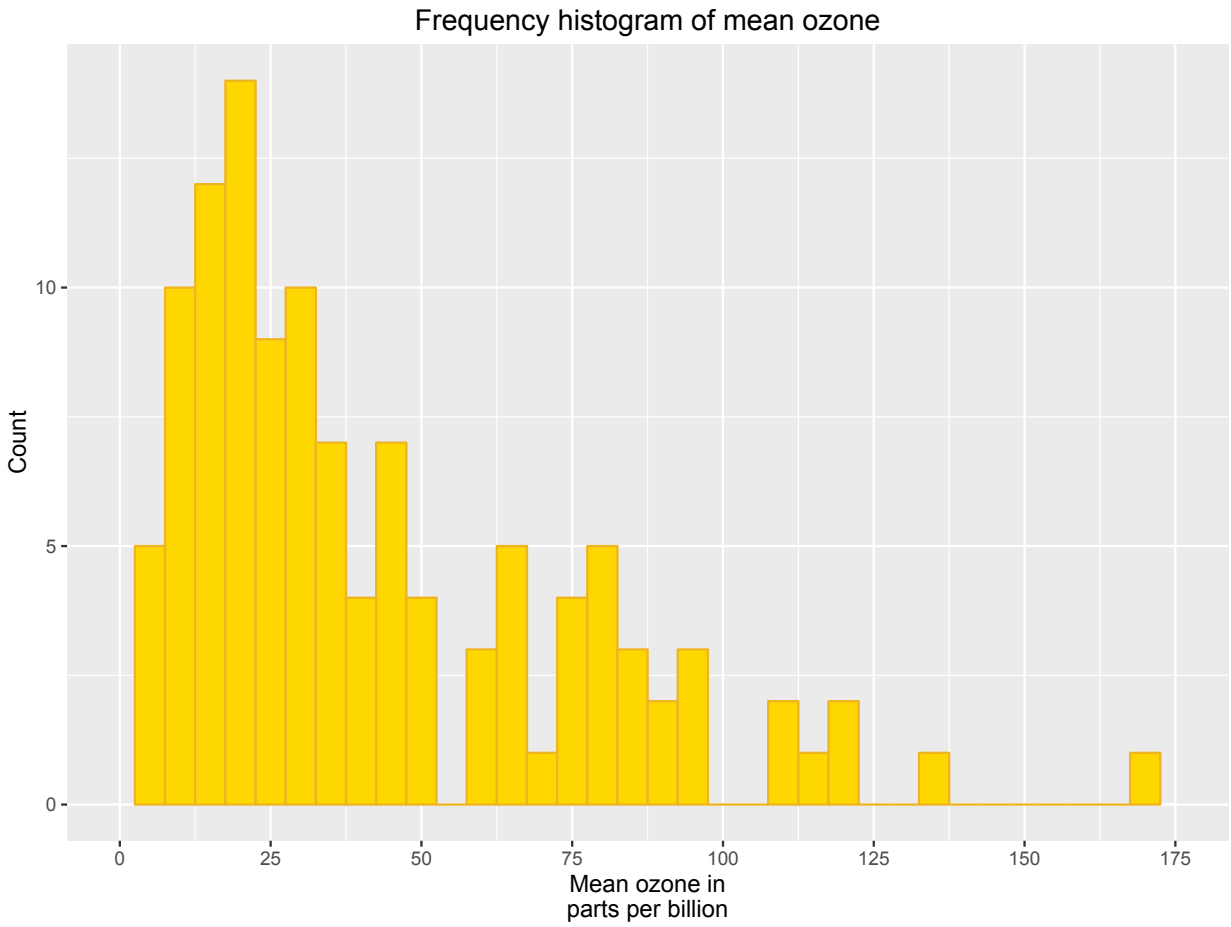


Changing the colour of the bars

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 I assigned these colours to variables outside of the plot to make it easier to change them). A list of valid colours is here.

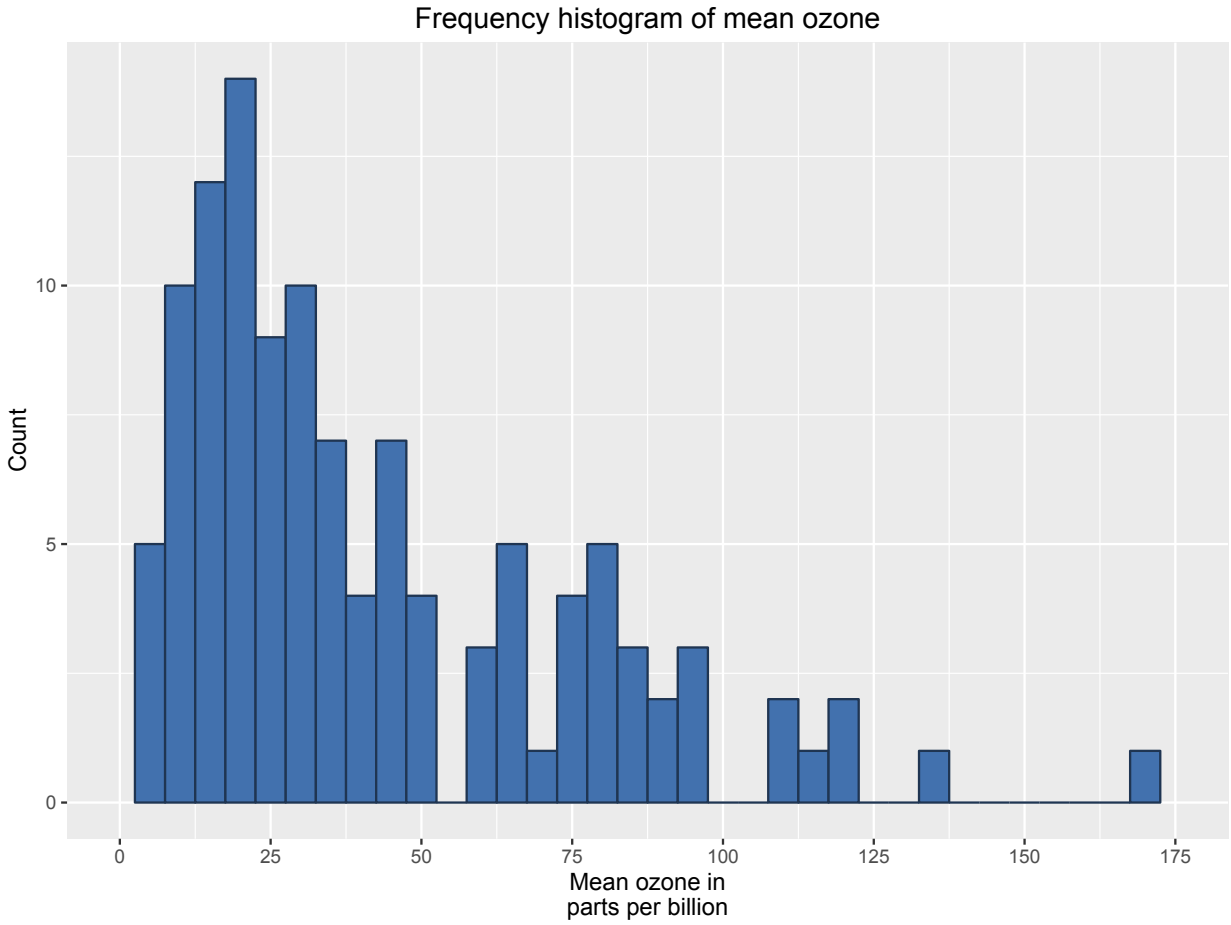
```
barfill <- "gold1"
barlines <- "goldenrod2"

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
```



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.

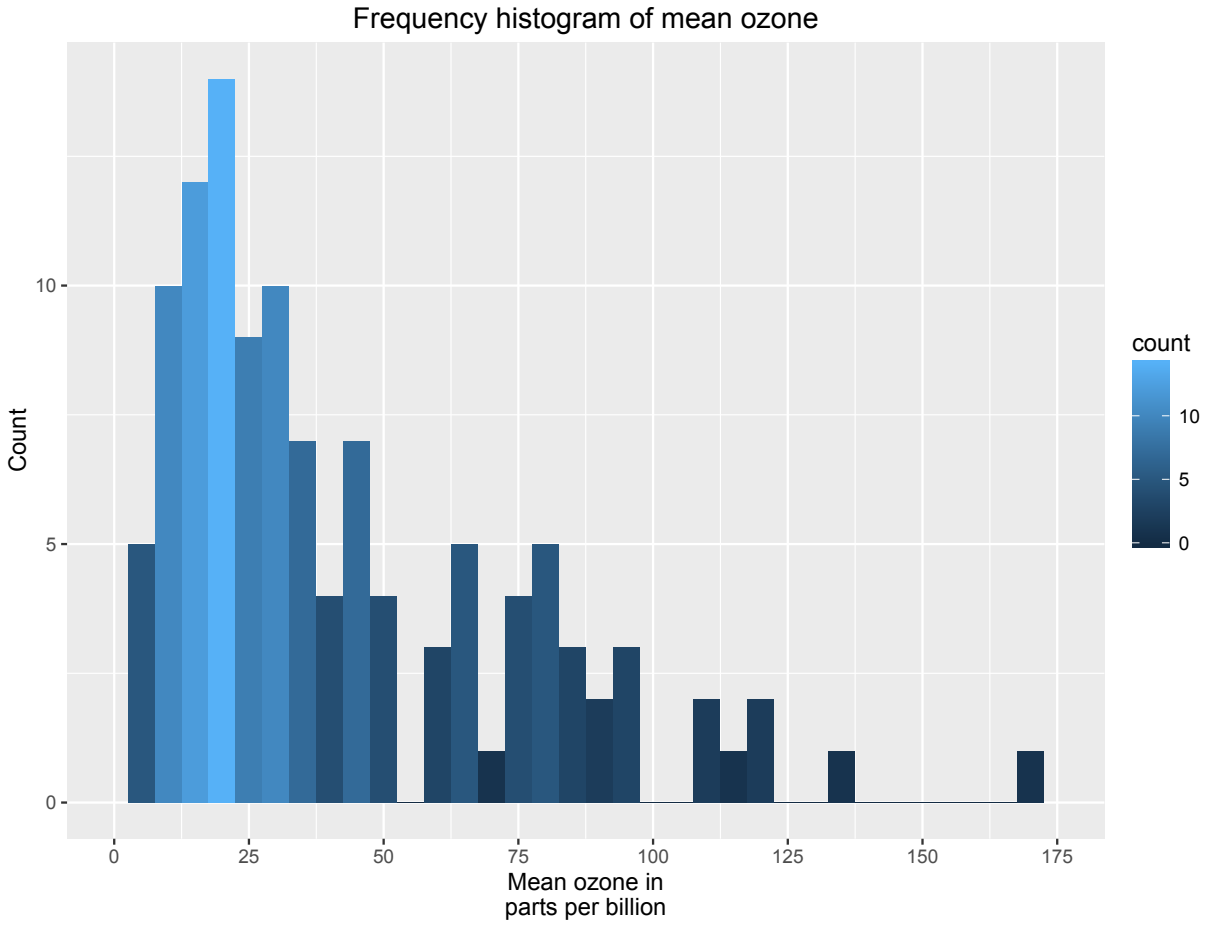
```
barfill <- "#4271AE"
barlines <- "#1F3552"

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
```



Colour gradients

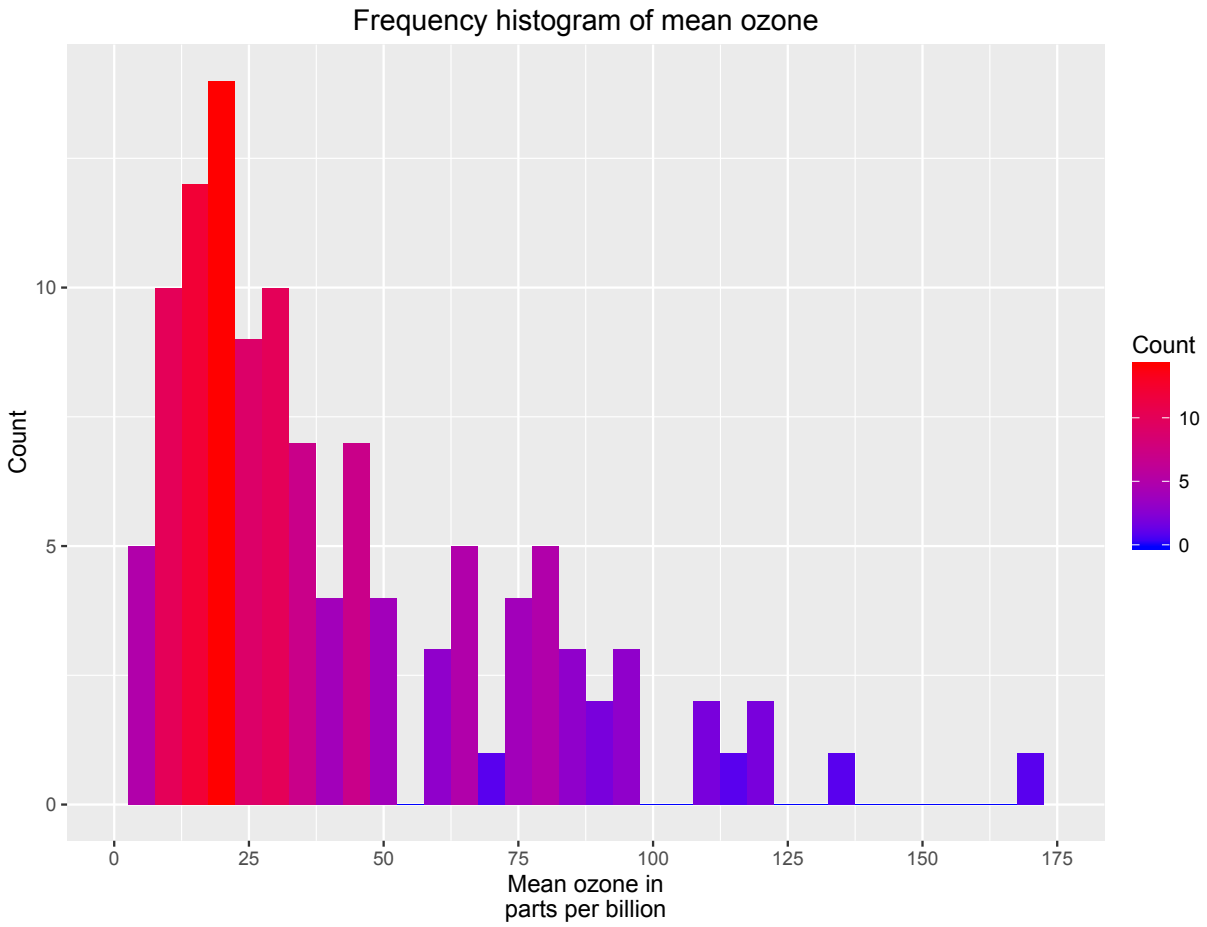
You can also add a gradient to your colour scheme that varies according to the frequency of the values. Below is the default gradient colour scheme. In order to do this, you can see we have changed the `aes(y = ..count..)` argument in `geom_histogram` to `aes(fill = ..count..)`.

```
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).

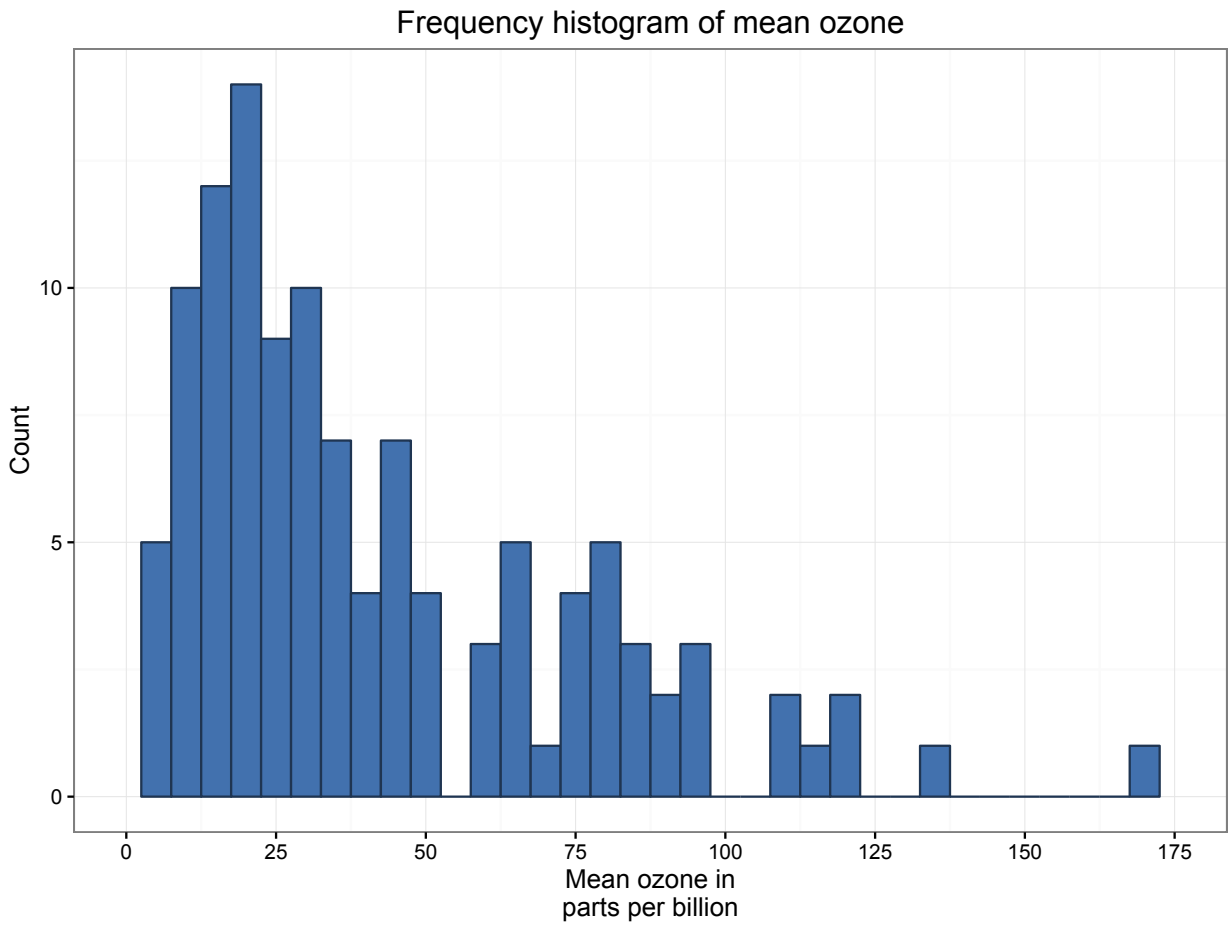
```
p7 <- ggplot(airquality, aes(x = Ozone)) +
      geom_histogram(aes(fill = ..count..), binwidth = 5) +
      scale_x_continuous(name = "Mean ozone in\nparts per billion",
                         breaks = seq(0, 175, 25),
                         limits=c(0, 175)) +
      scale_y_continuous(name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



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.

```
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_bw()
p7
```



Creating an XKCD style chart

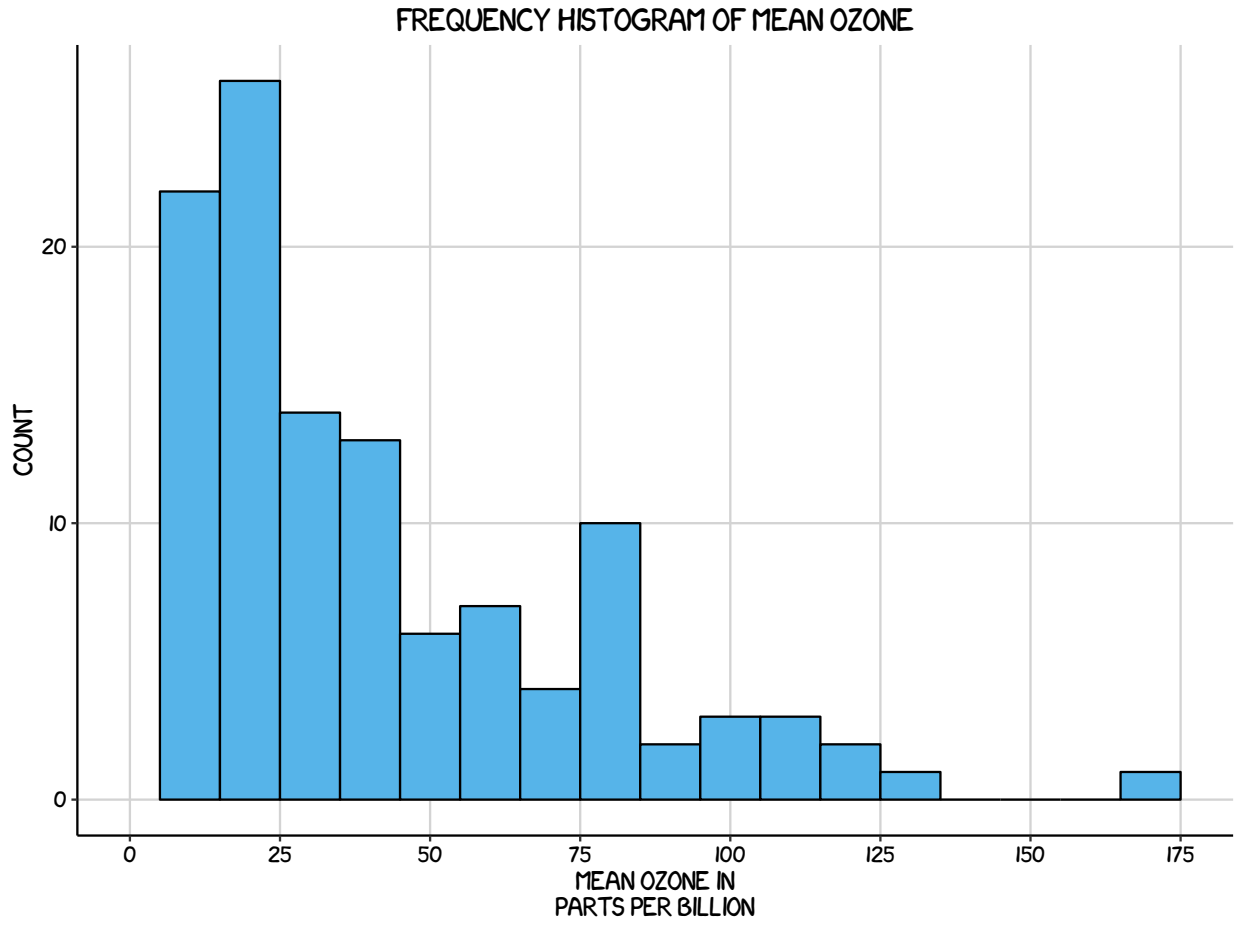
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
              dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

```
p7 <- ggplot(airquality, aes(x = Ozone)) +
      geom_histogram(aes(y = ..count..), binwidth = 10,
```

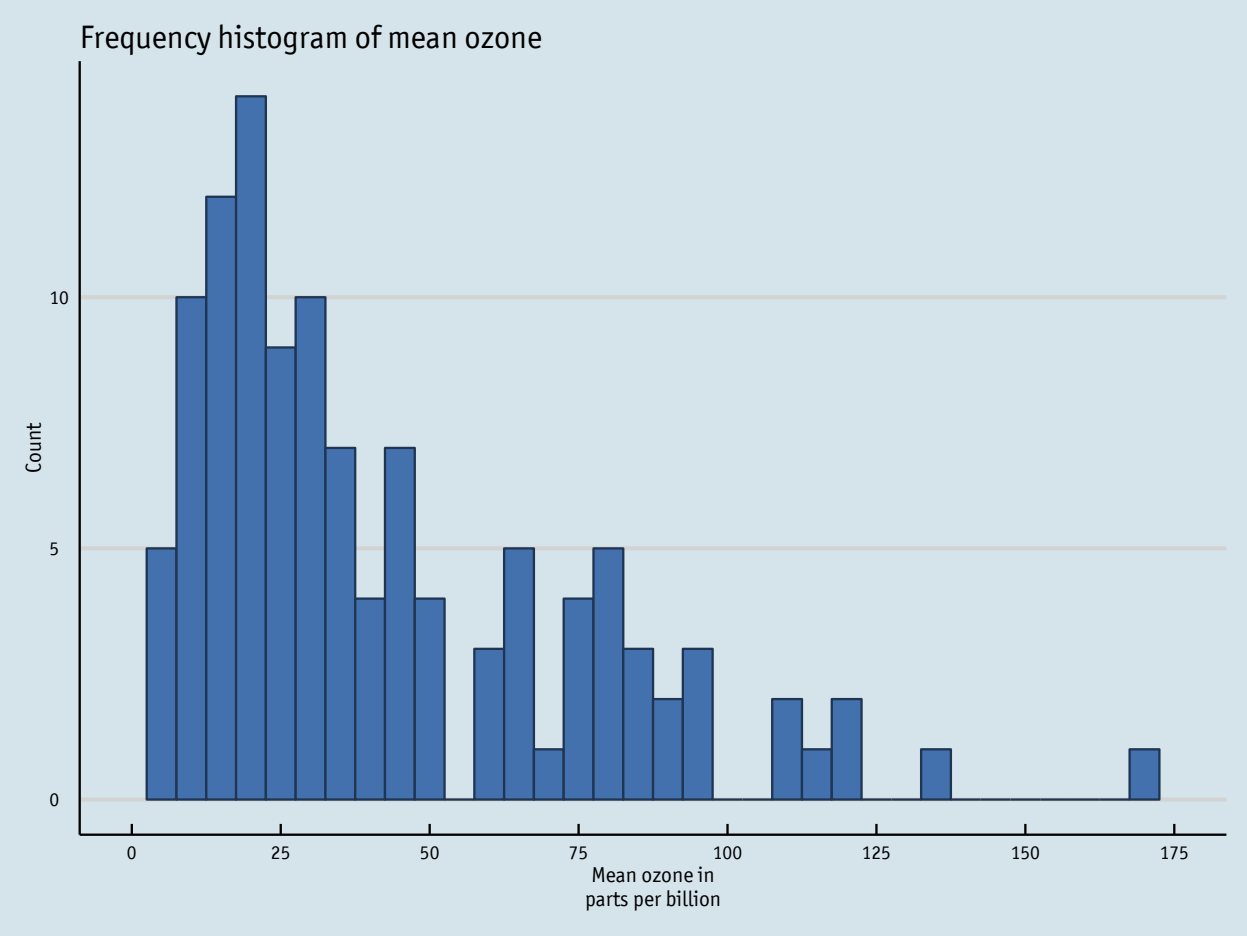
```
                    colour = "black", fill = "#56B4E9") +
      scale_x_continuous(name = "Mean ozone in\nparts per billion",
                         breaks = seq(0, 175, 25),
                         limits=c(0, 175)) +
      scale_y_continuous(name = "Count") +
      ggtitle("Frequency histogram 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 = 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"))
p7
```



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.

```
library(ggthemes)

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_economist() +
      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 = 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 = "OfficinaSanITC-Book"),
            text=element_text(family="OfficinaSanITC-Book"))
p7
```

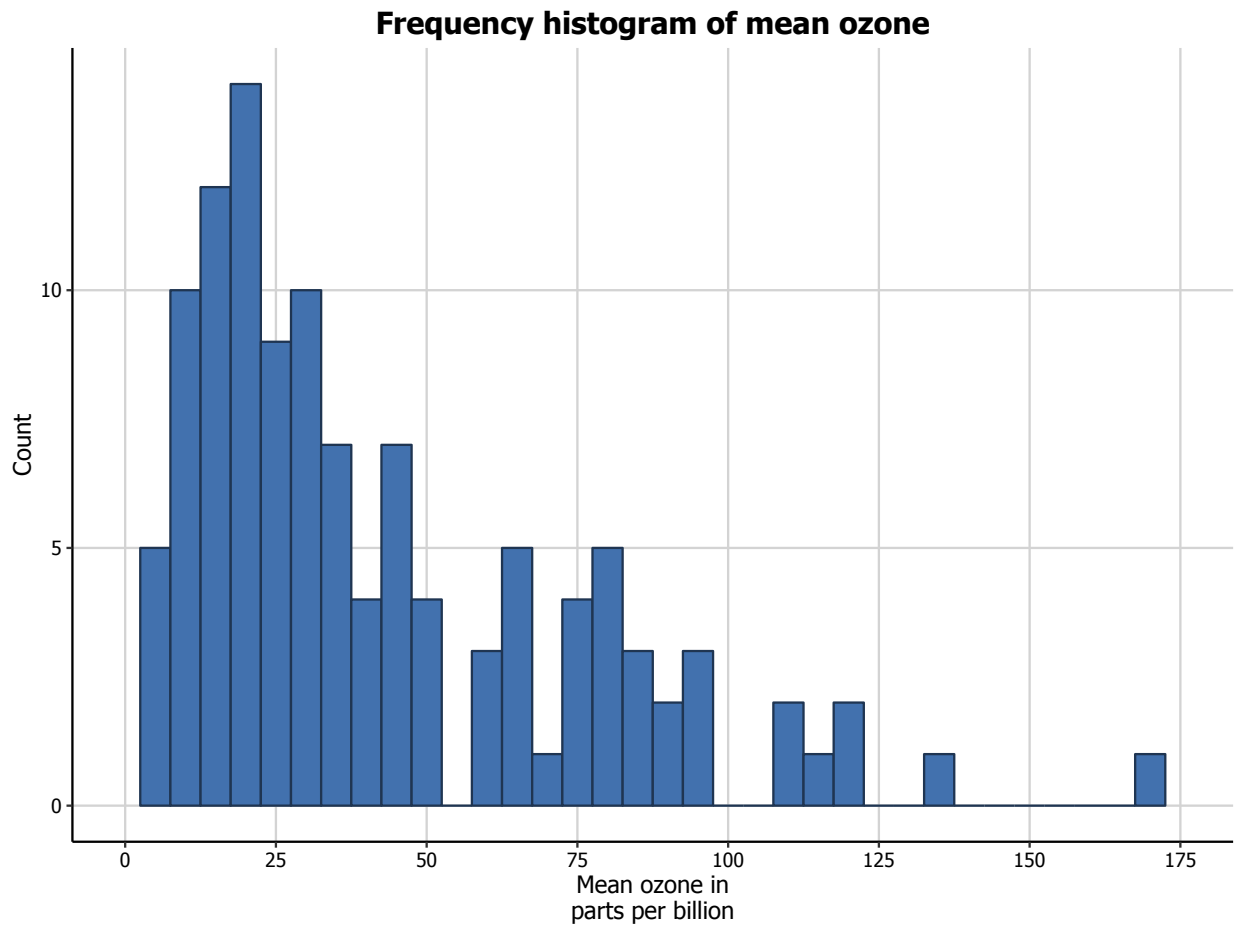


Creating your own theme

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.

```
library(grid)

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(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 = 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"))
p7
```

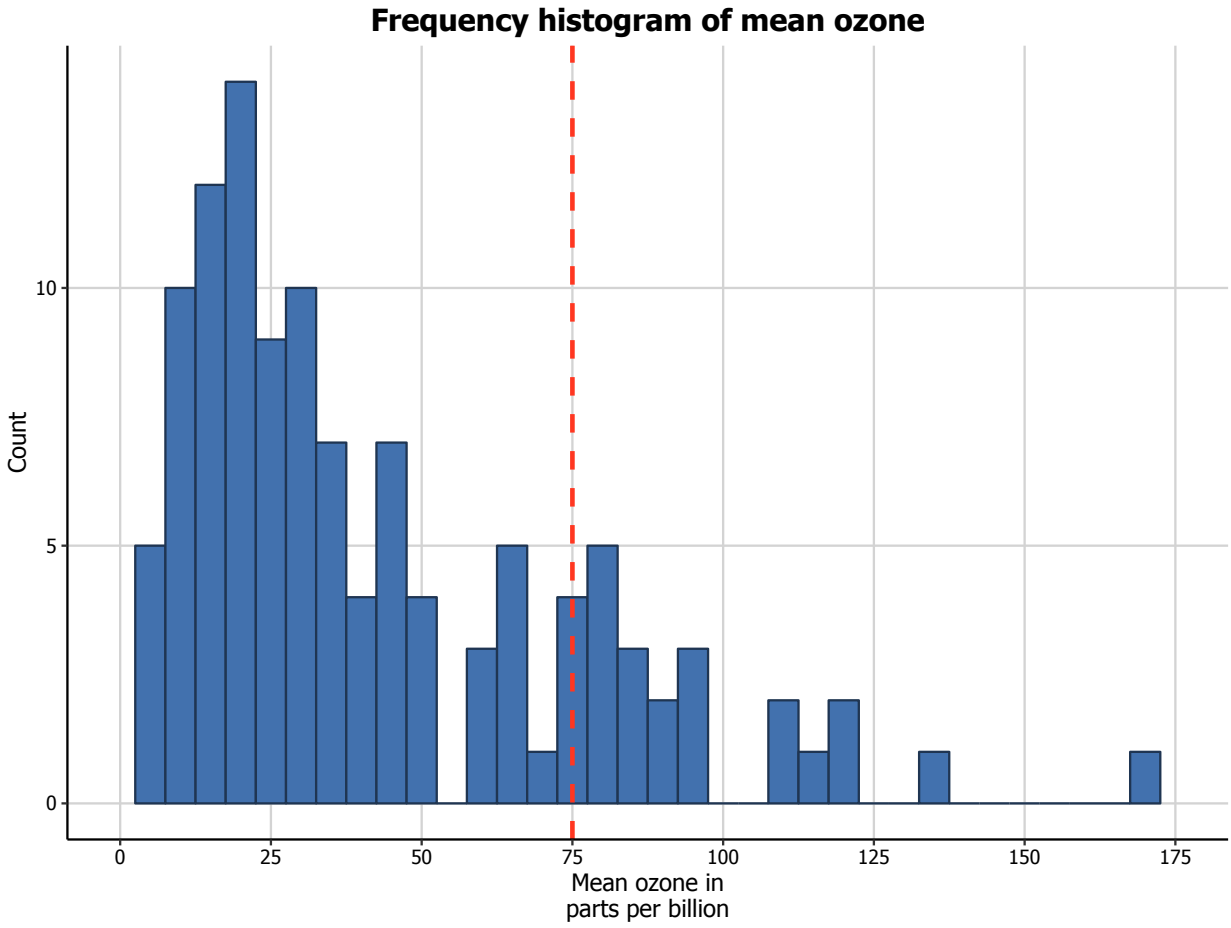


Adding lines

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(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 = 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"))
p7
```



Multiple histograms

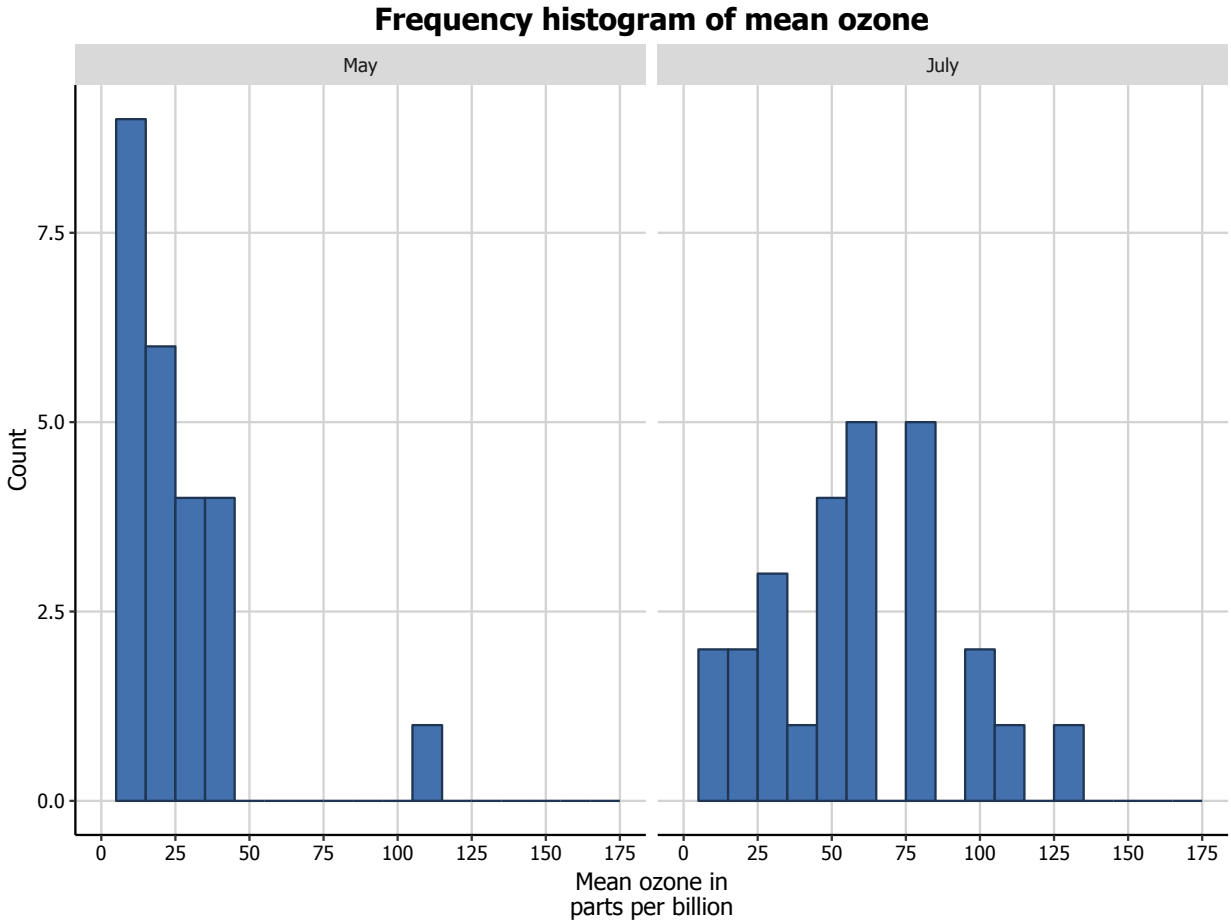
You can also easily create multiple histograms by the levels of another variable. There are two options, in separate (panel) plots, or in the same plot.

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(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 = 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"))
p7
```

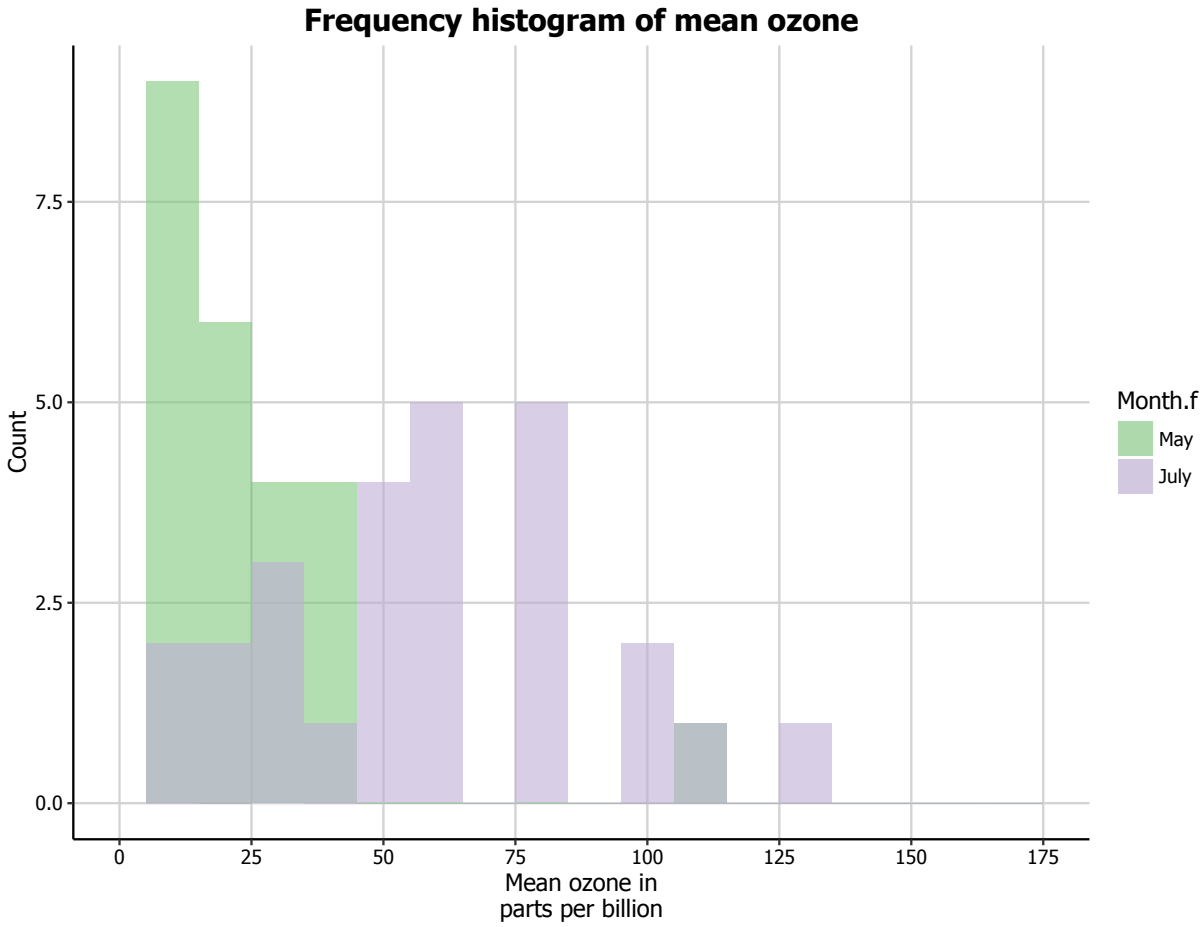


In the same plot

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 = 0.6`. 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.

```
library(RColorBrewer)

p7 <- ggplot(airquality_trimmed, aes(x = Ozone, fill = Month.f)) +
      geom_histogram(aes(y = ..count..), binwidth = 10,
                     position="identity", alpha=0.6) +
      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(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 = 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"))
p7
```

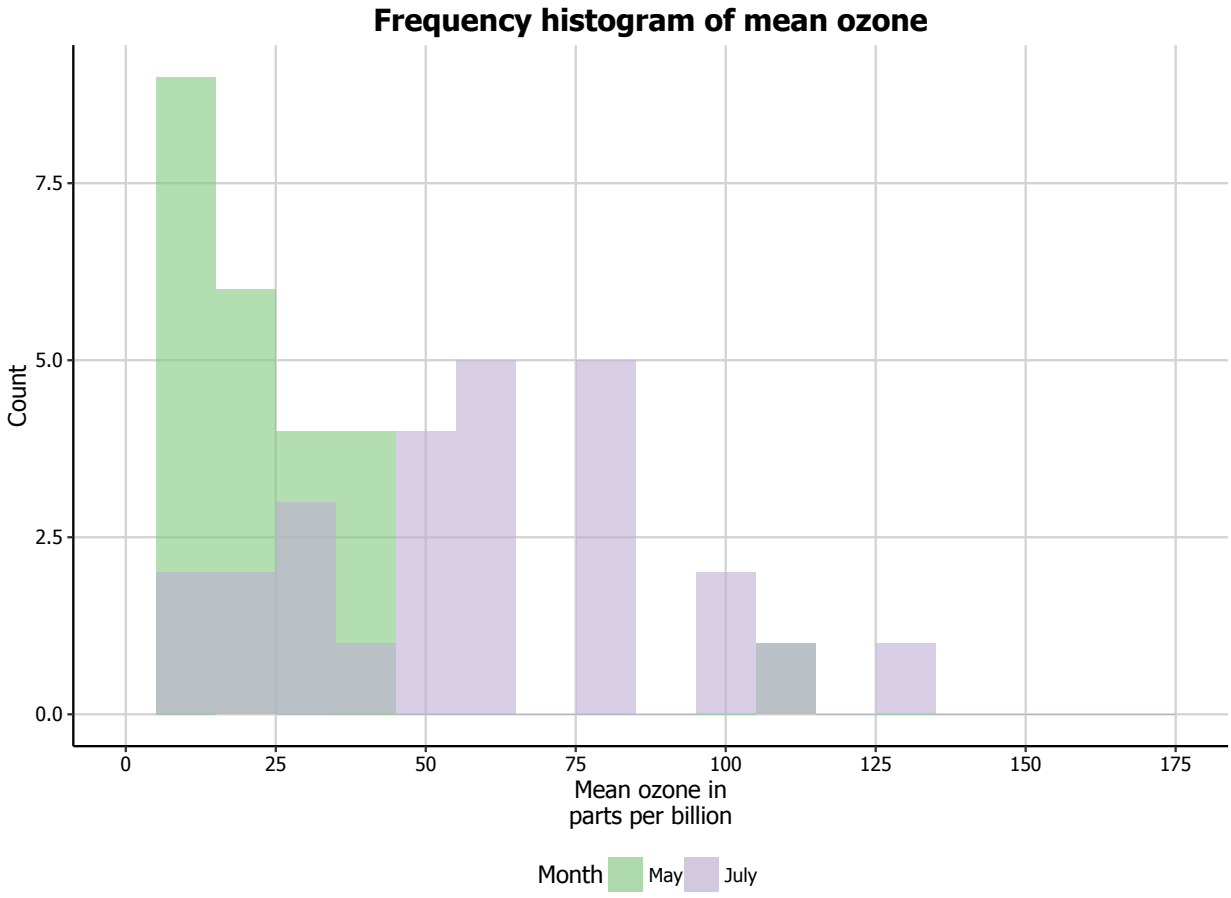


Formatting the legend

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.

```
p7 <- ggplot(airquality_trimmed, aes(x = Ozone, fill = Month.f)) +
      geom_histogram(aes(y = ..count..), binwidth = 10,
                     position="identity", alpha=0.6) +
      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(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 = 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"))
p7
```



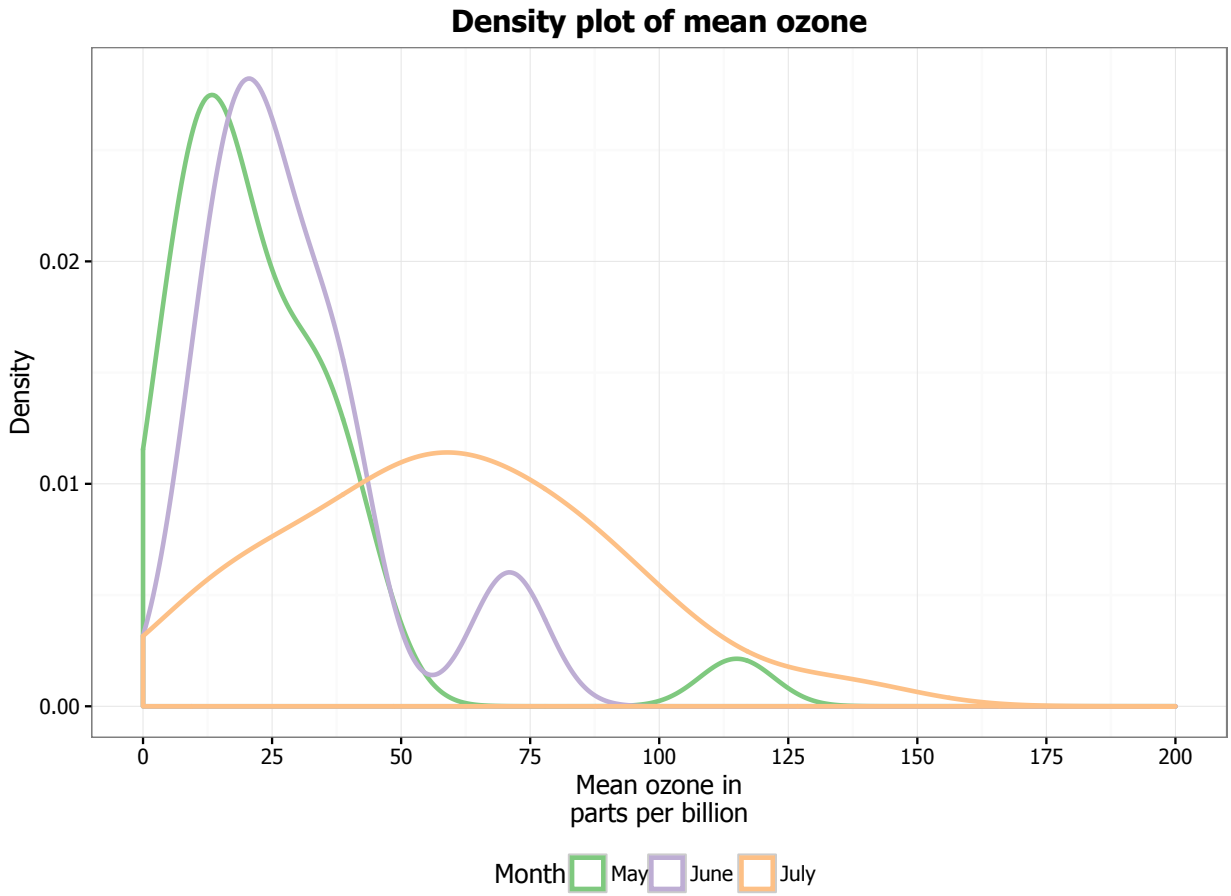
Density Plots

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

```
rm(list = ls())
library(datasets)
library(ggplot2)
```

```
data(airquality)
```

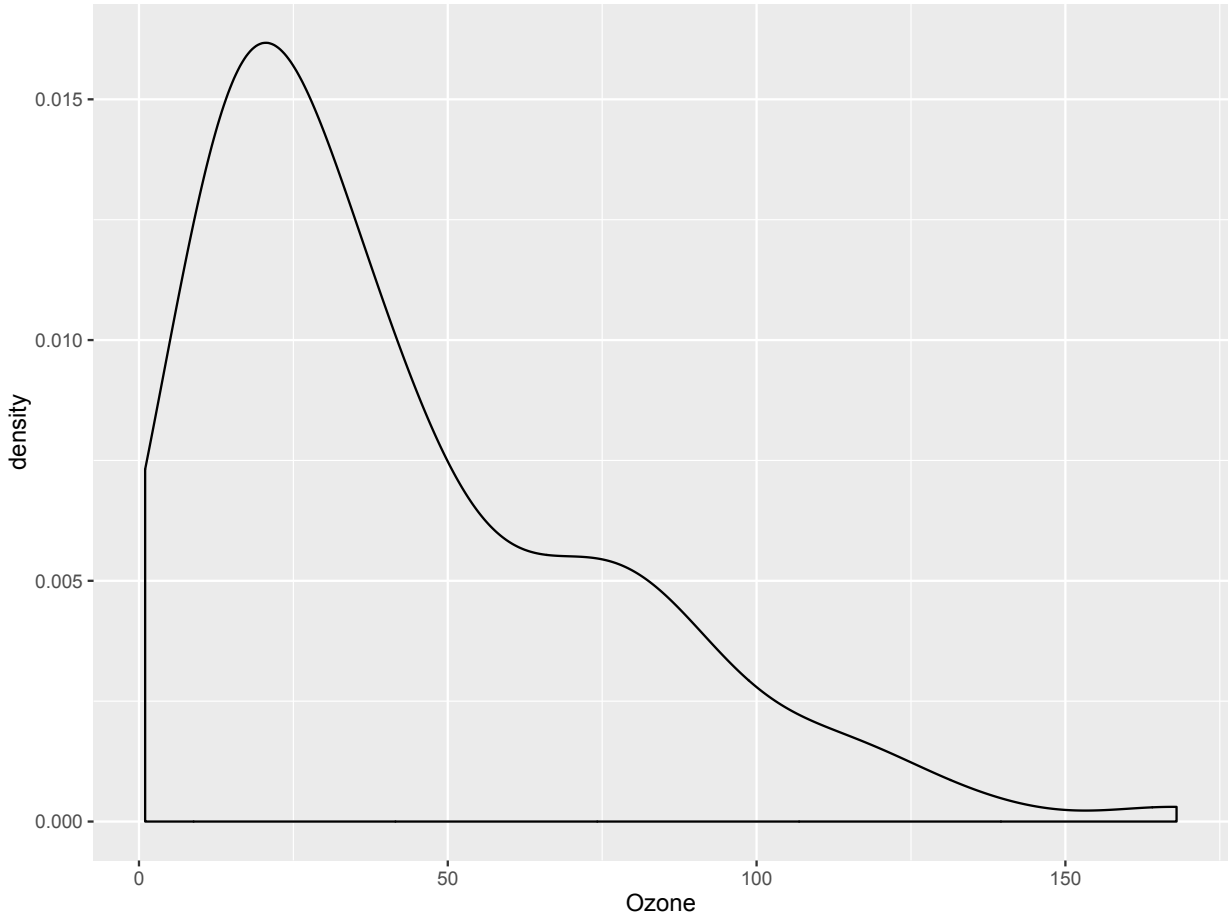
In this part, we will work towards creating the density plot below. We will take you from a basic density plot and explain all the customisations we add to the code step-by-step.



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.

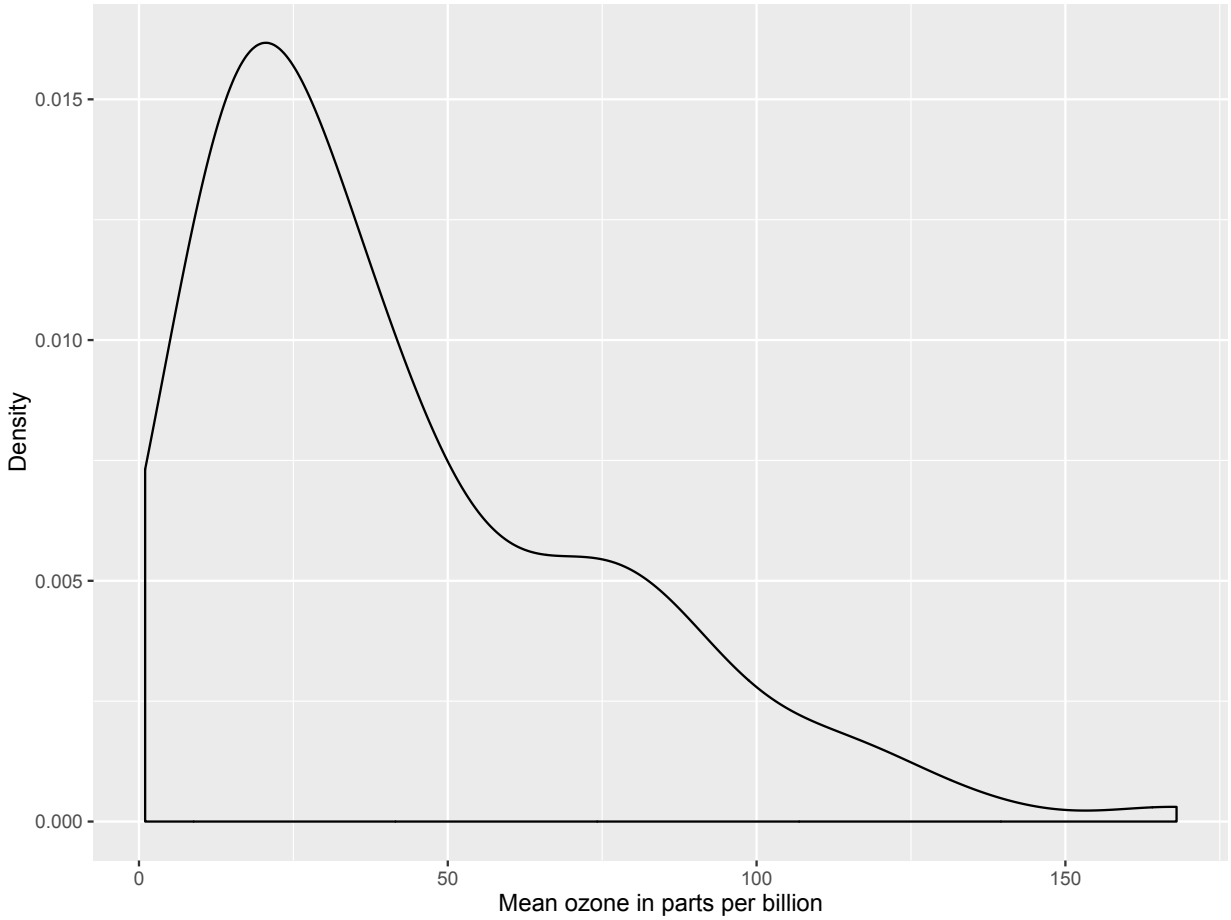
```
p8 <- ggplot(airquality, aes(x = Ozone)) + geom_density()
p8
```



Customising axis labels

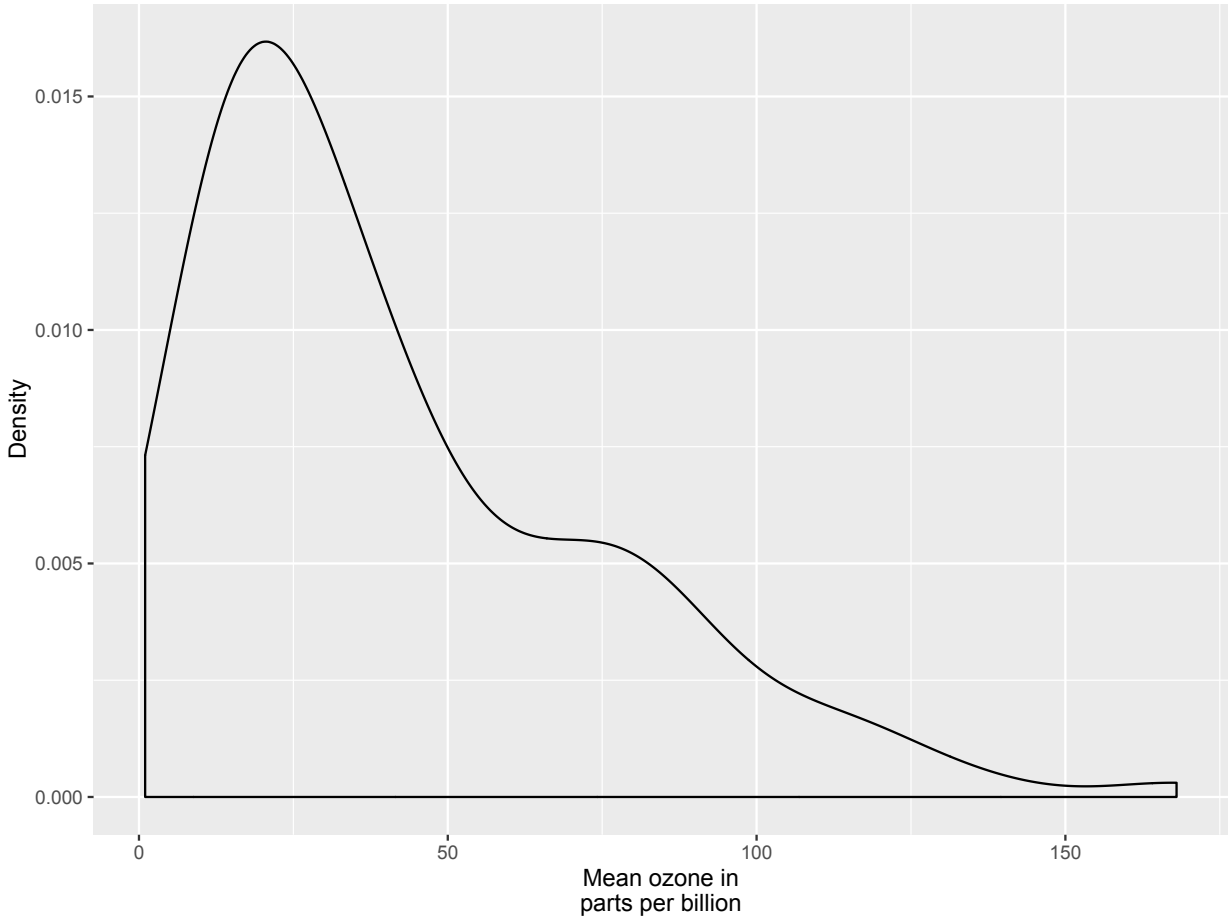
In order to change the axis labels, we have a couple of options. In this case, we have used the `scale_x_continuous` and `scale_y_continuous` options, as these have further customisation options for the axes we will use below. In each, we add the desired name to the `name` argument as a string.

```
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.

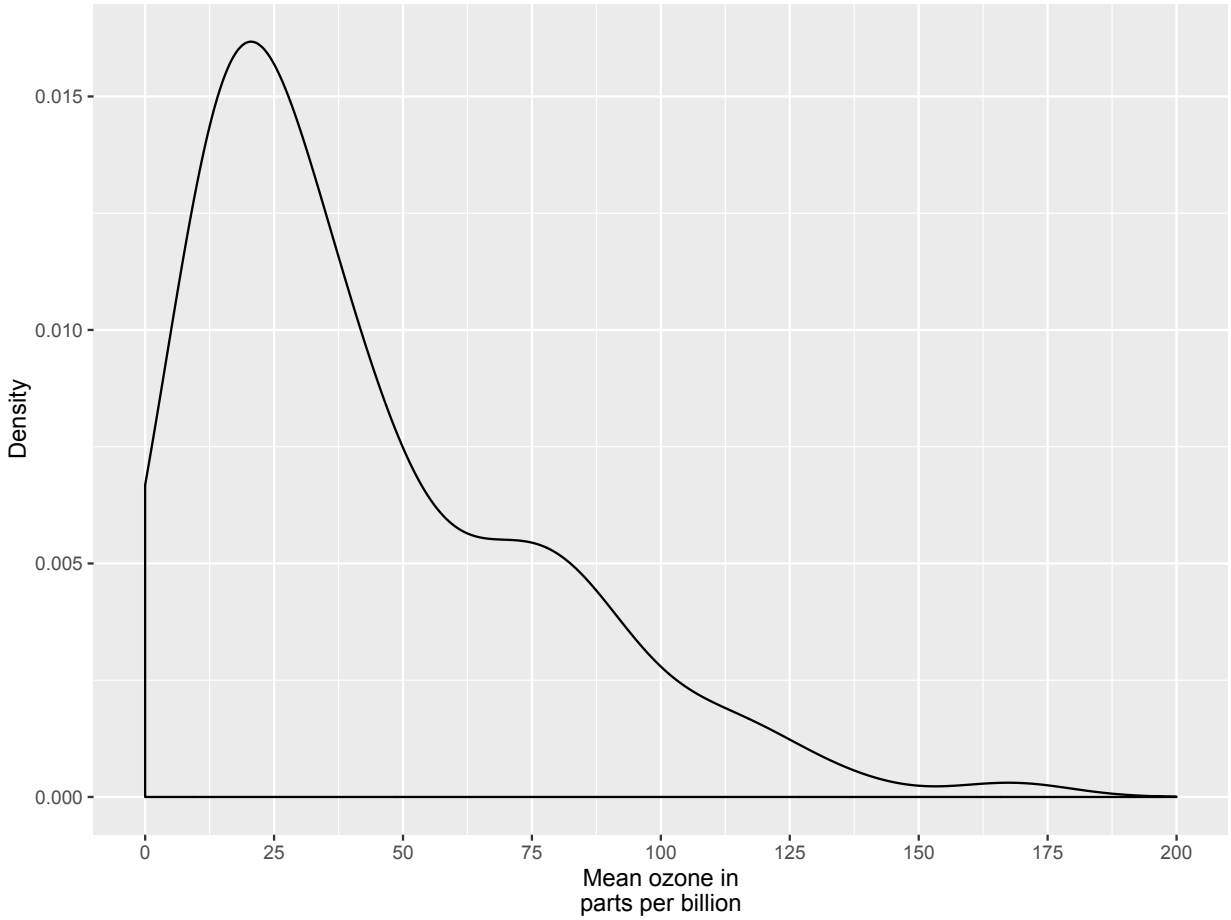
```
p8 <- p8 + scale_x_continuous(name = "Mean ozone in\nparts per billion")
p8
```



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`.

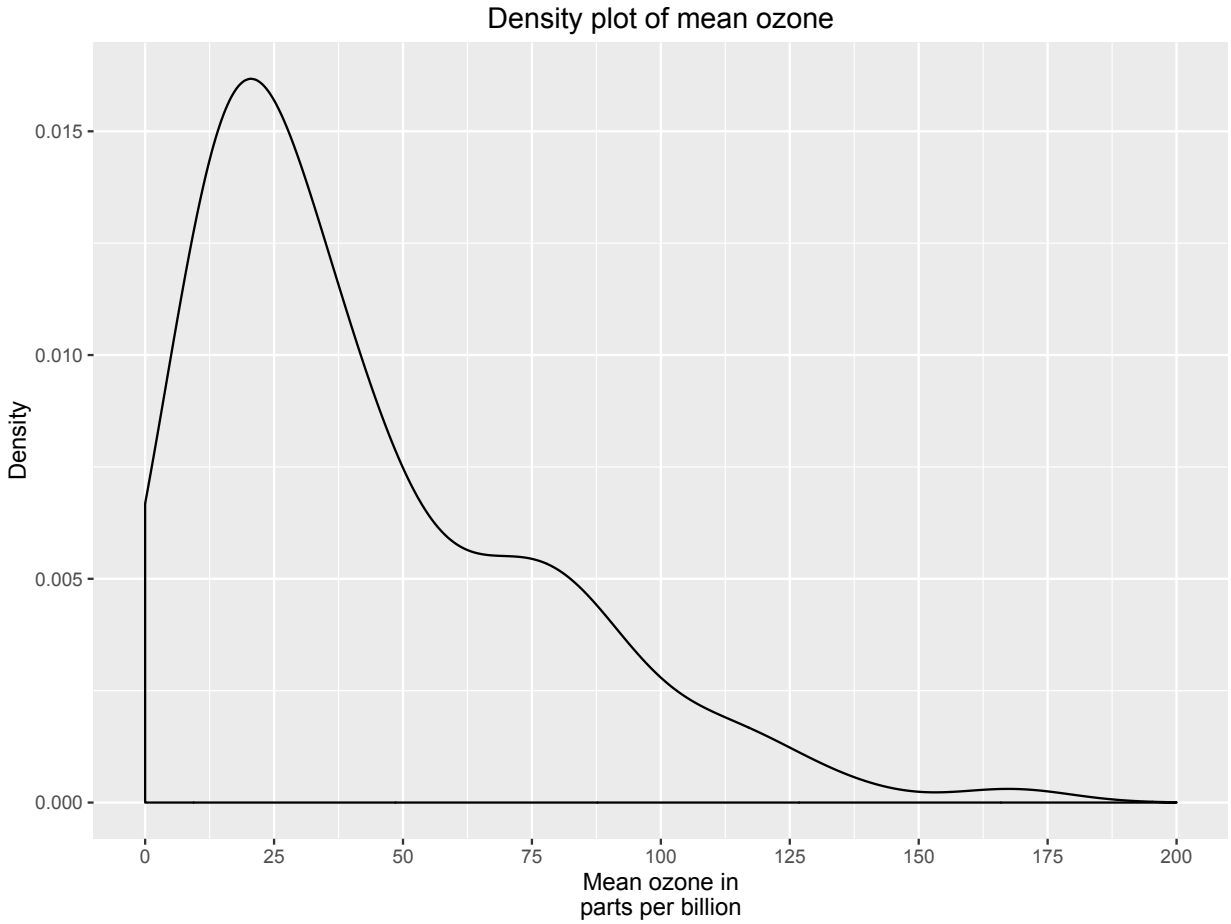
```
p8 <- p8 + scale_x_continuous(name = "Mean ozone in\nparts per billion",
                              breaks = seq(0, 200, 25),
                              limits=c(0, 200))
p8
```



Adding a title

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

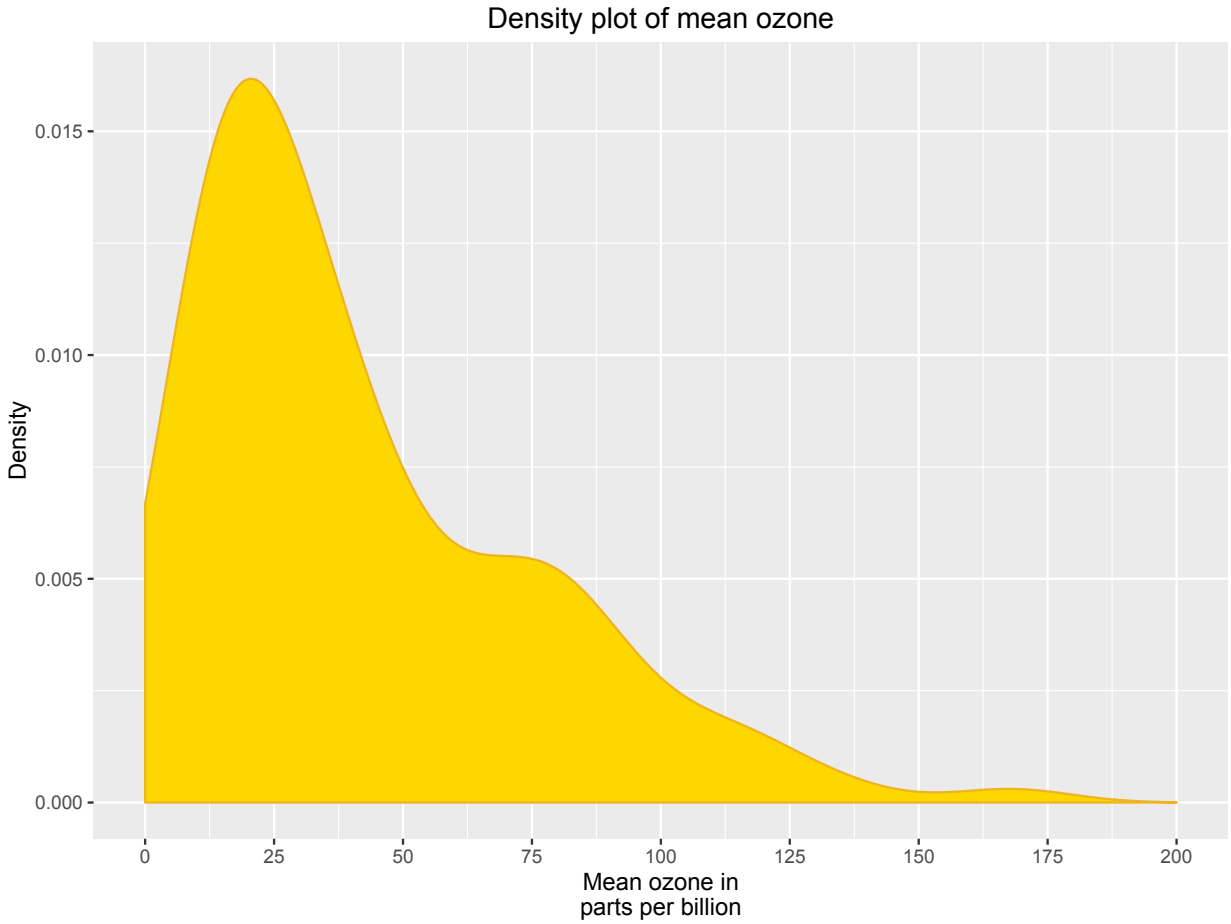
```
p8 <- p8 + ggtitle("Density plot of mean ozone")
p8
```



Changing the colour of the curves

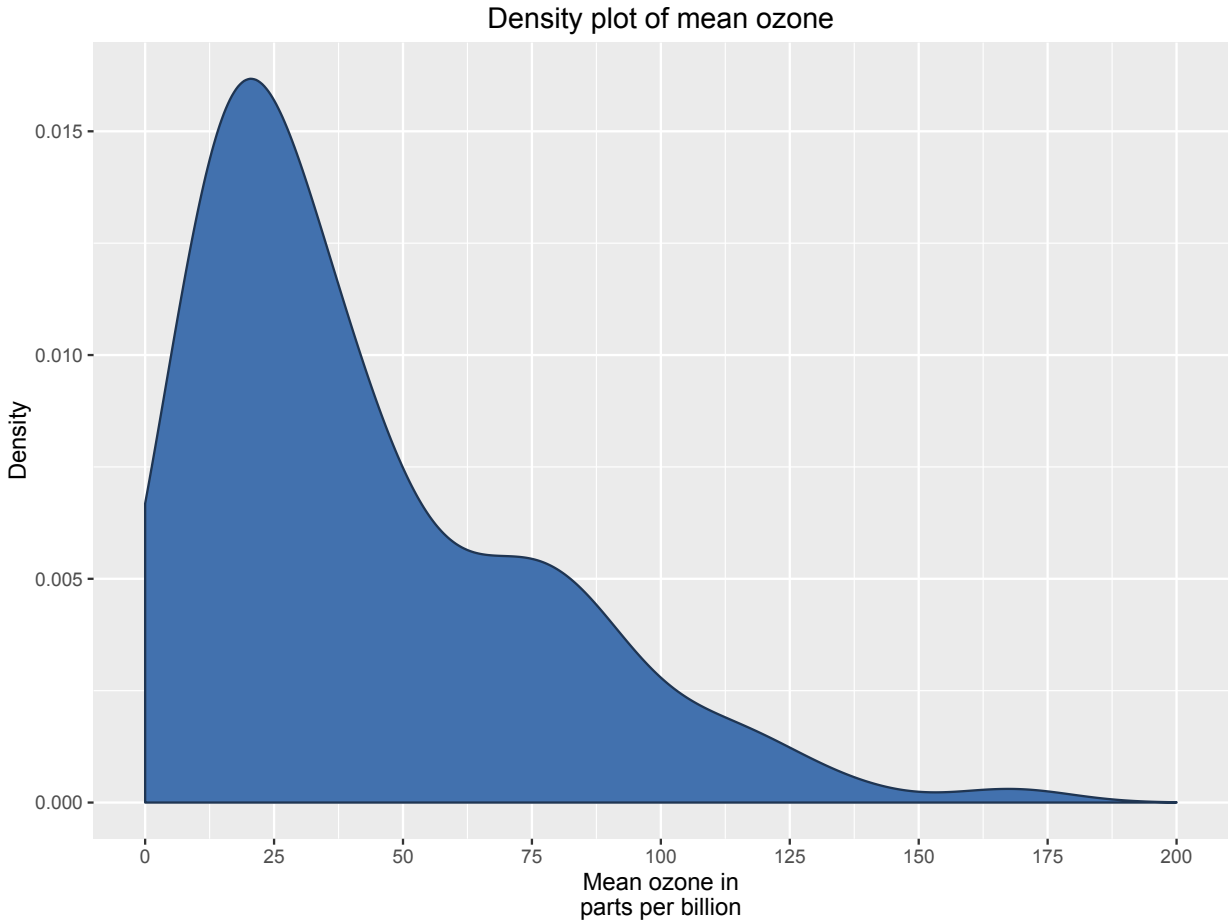
To change the line and fill colours of the density plot, we add a valid colour to the `colour` and `fill` arguments in `geom_density()` (note that I assigned these colours to variables outside of the plot to make it easier to change them). A list of valid colours is here.

```
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
```



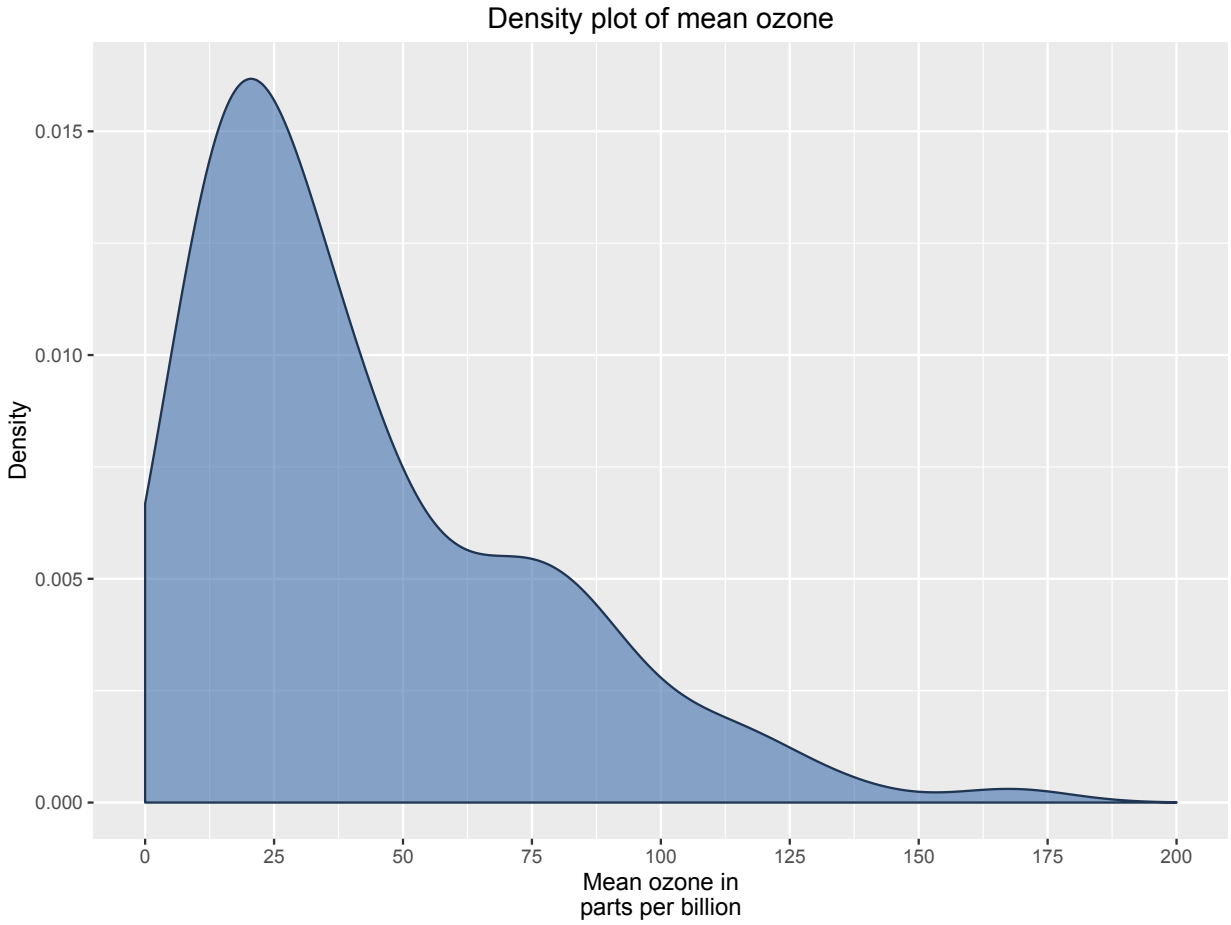
If you want to go beyond the options in the list above, you can also specify exact HEX colours by including them as a string preceded by a hash, e.g., "#FFFFFF". Below, we have called two shades of blue for the fill and lines using their HEX codes.

```
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.

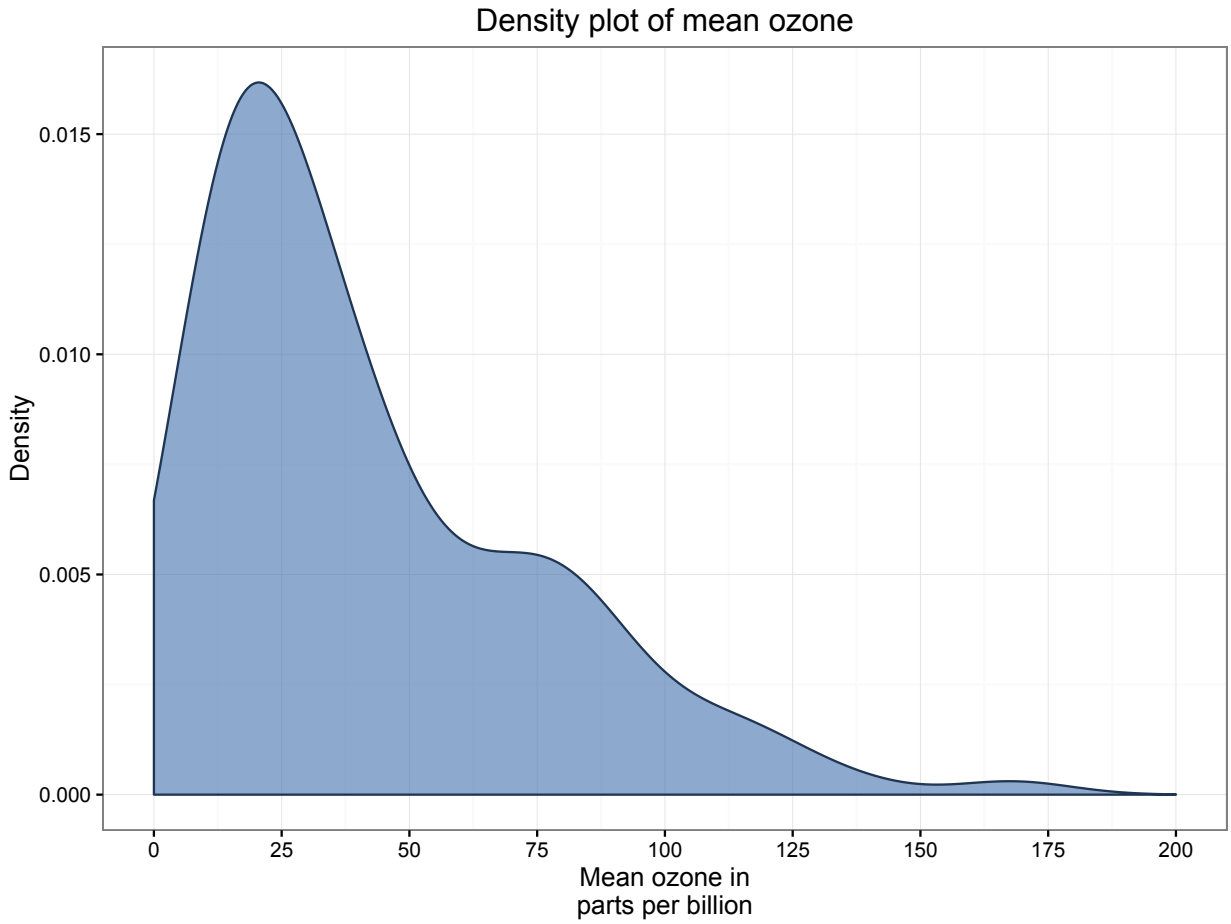
```
p8 <- ggplot(airquality, aes(x = Ozone)) +
      geom_density(fill = fill, colour = line,
                   alpha = 0.6) +
      scale_x_continuous(name = "Mean ozone in\nparts per billion",
                         breaks = seq(0, 200, 25),
                         limits=c(0, 200)) +
      scale_y_continuous(name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



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()`. 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
```



Creating an XKCD style chart

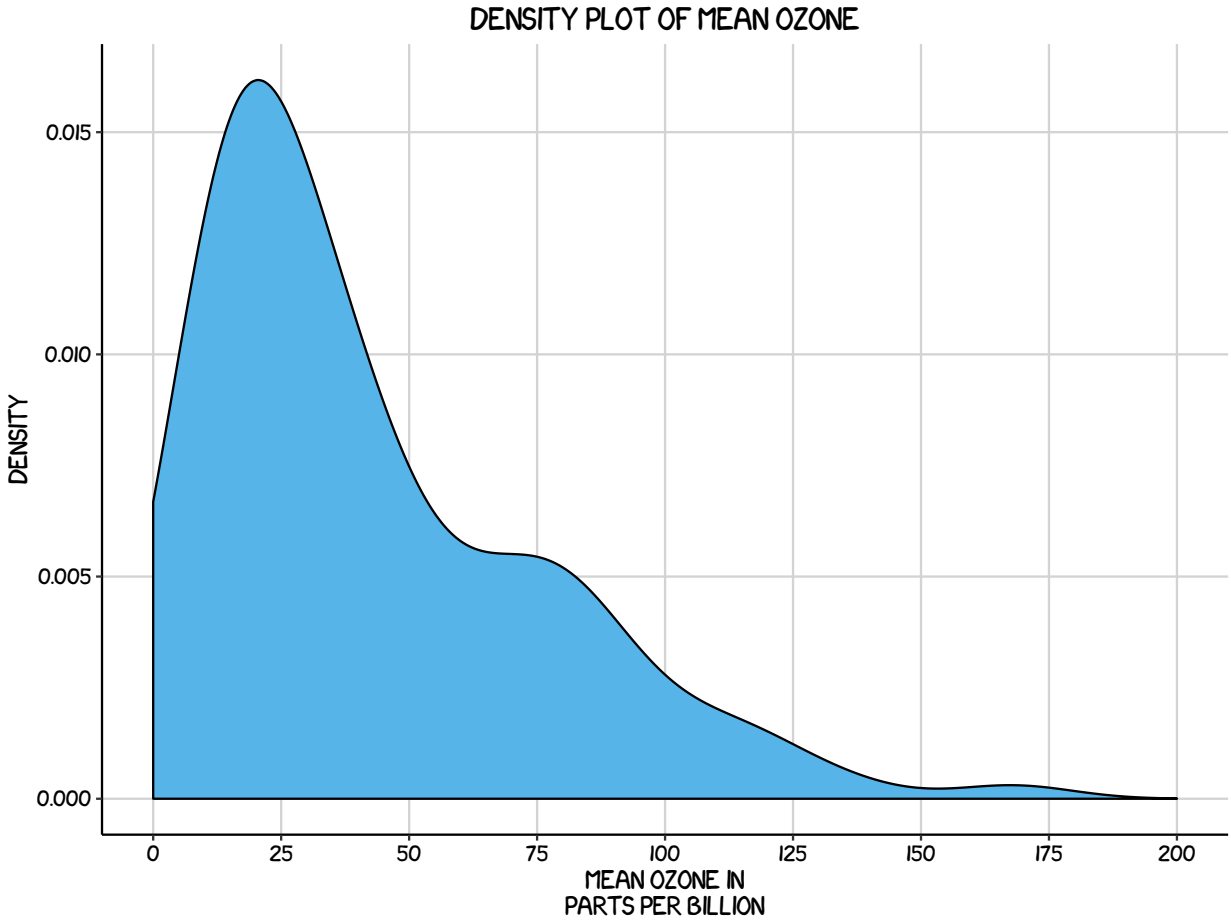
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
              dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

```
p8 <- ggplot(airquality, aes(x = Ozone)) +
      geom_density(colour = "black", fill = "#56B4E9") +
```

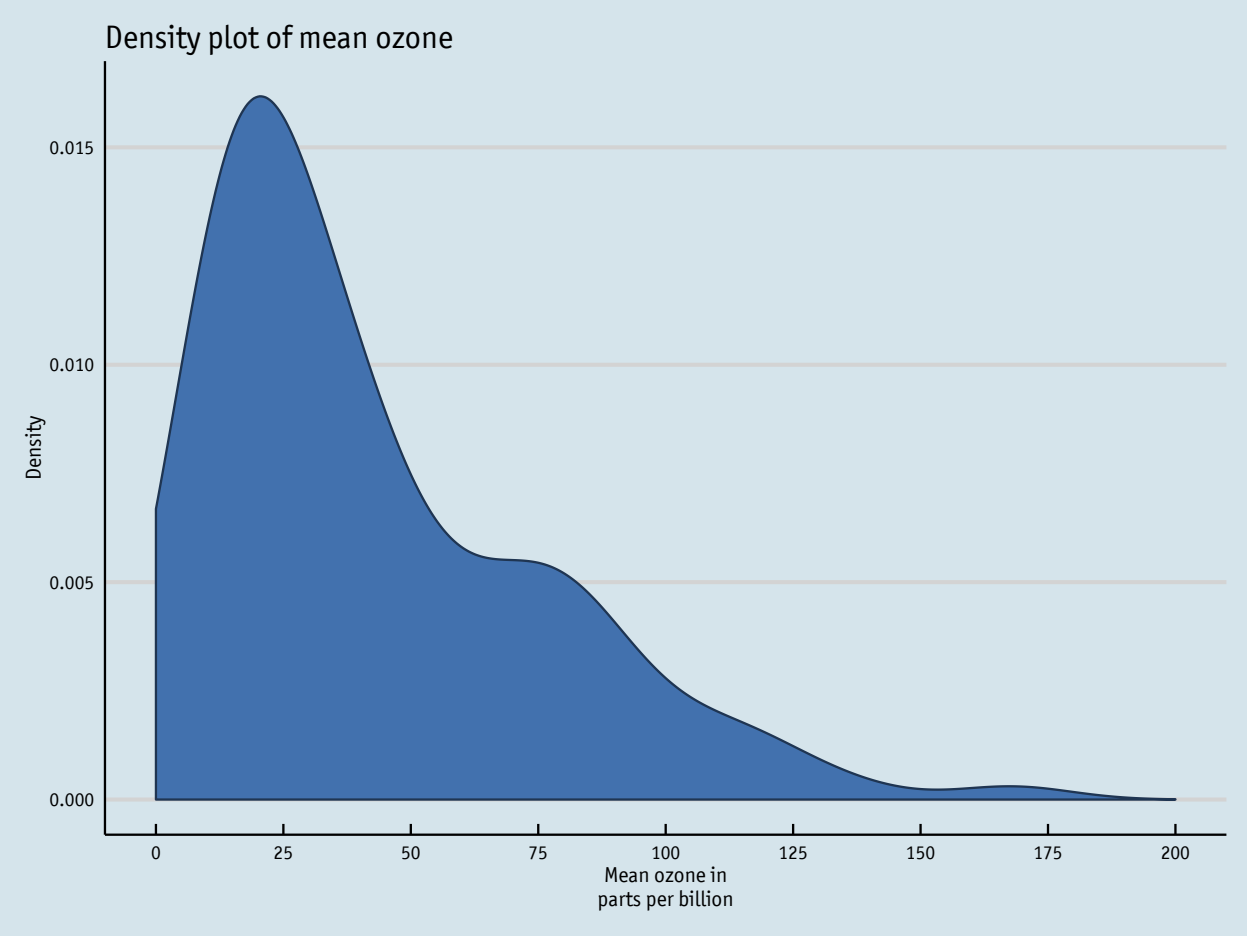
```
      scale_x_continuous(name = "Mean ozone in\nparts per billion",
                         breaks = seq(0, 200, 25),
                         limits=c(0, 200)) +
      scale_y_continuous(name = "Density") +
      ggtitle("Density plot of mean ozone") +
      theme(axis.line.x = element_line(size=.5, colour = "black"),
            axis.line.y = element_line(size=.5, colour = "black"),
            axis.text.x=element_text(colour="black", size = 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"))
p8
```



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.

```
library(ggthemes)
library(grid)

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") +
      theme_economist() +
      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 = 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 = "OfficinaSanITC-Book"),
            text=element_text(family="OfficinaSanITC-Book"))
p8
```

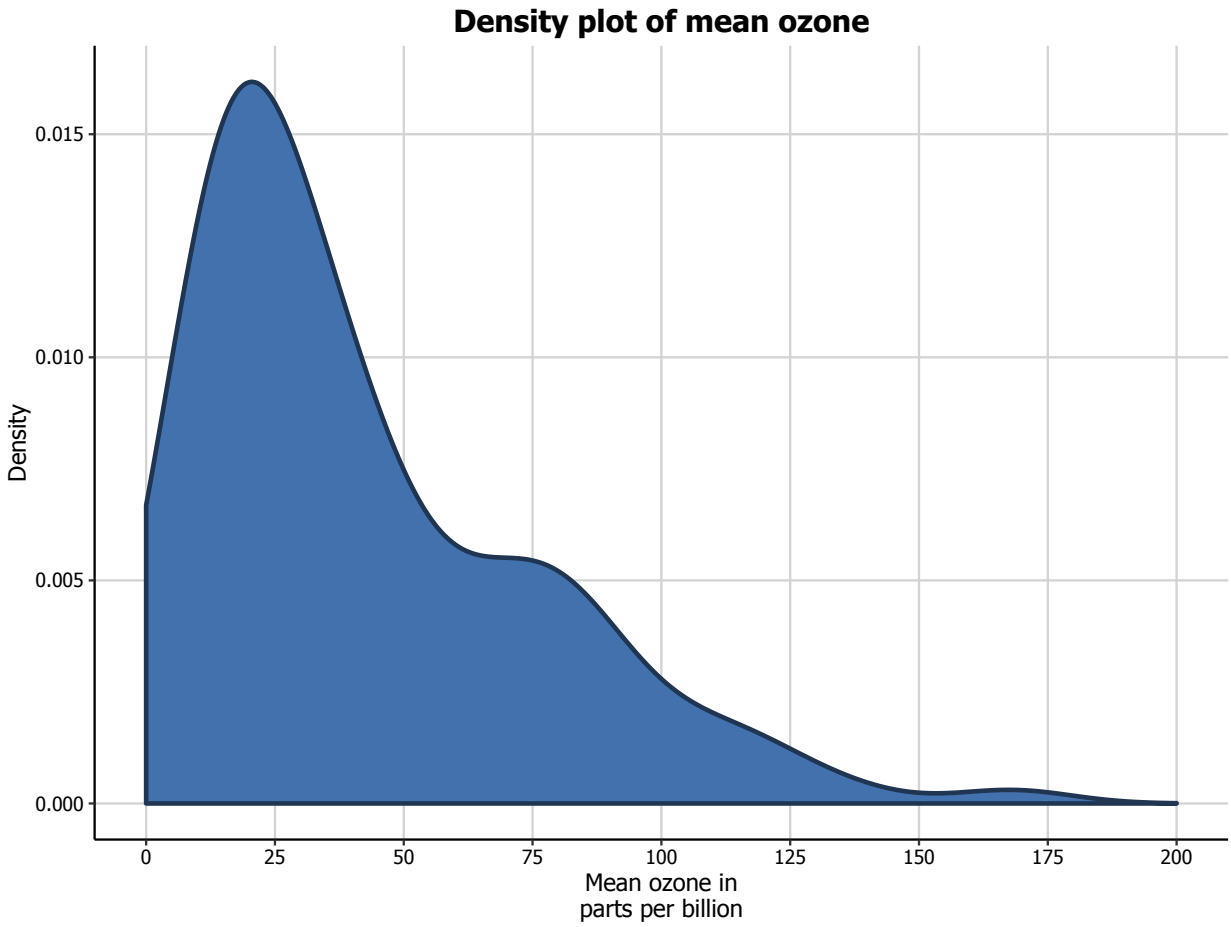


Creating your own theme

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.

```
library(grid)

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(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 = 9),
            axis.text.y=element_text(colour="black", size = 9),
            legend.position = "bottom", legend.position = "horizontal",
            panel.grid.major = element_line(colour = "#d3d3d3"),
```

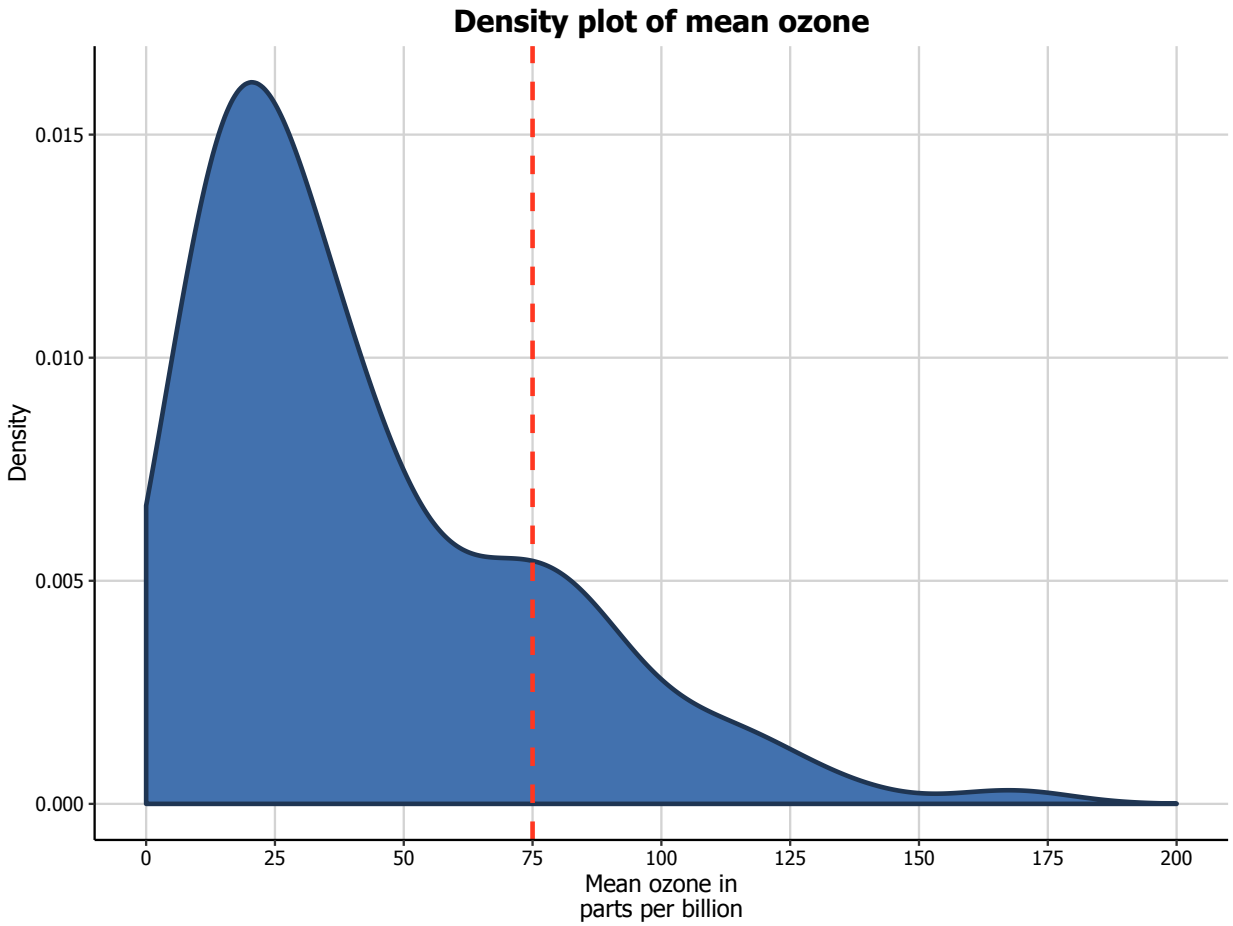
```
            panel.grid.minor = element_blank(),
            panel.border = element_blank(), panel.background = element_blank(),
            plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
            text=element_text(family="Tahoma"))
p8
```



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`.)

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

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



Multiple densities

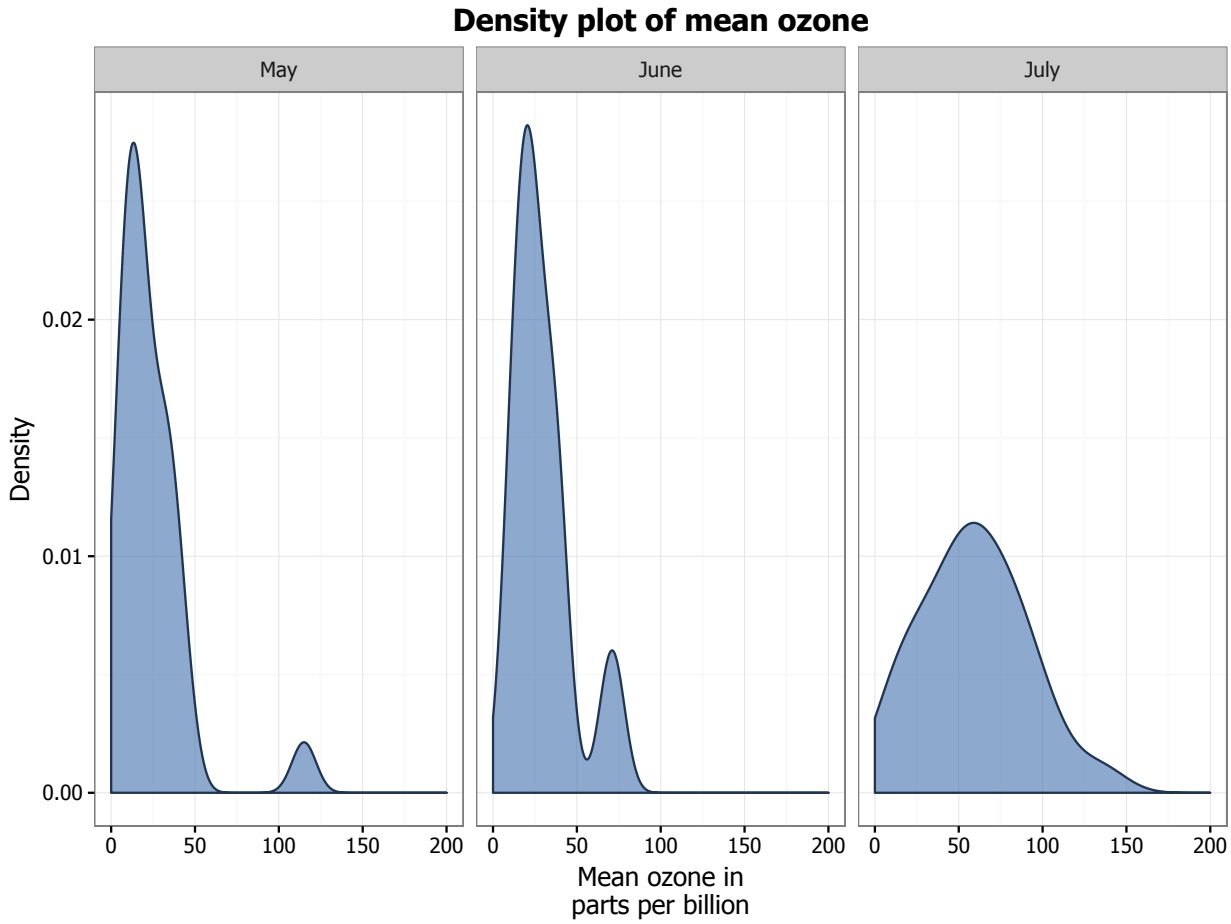
You can also easily create multiple density plots by the levels of another variable. There are two options, in separate (panel) plots, or in the same plot. There are also a couple of variations on these we'll discuss below.

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)) +
```

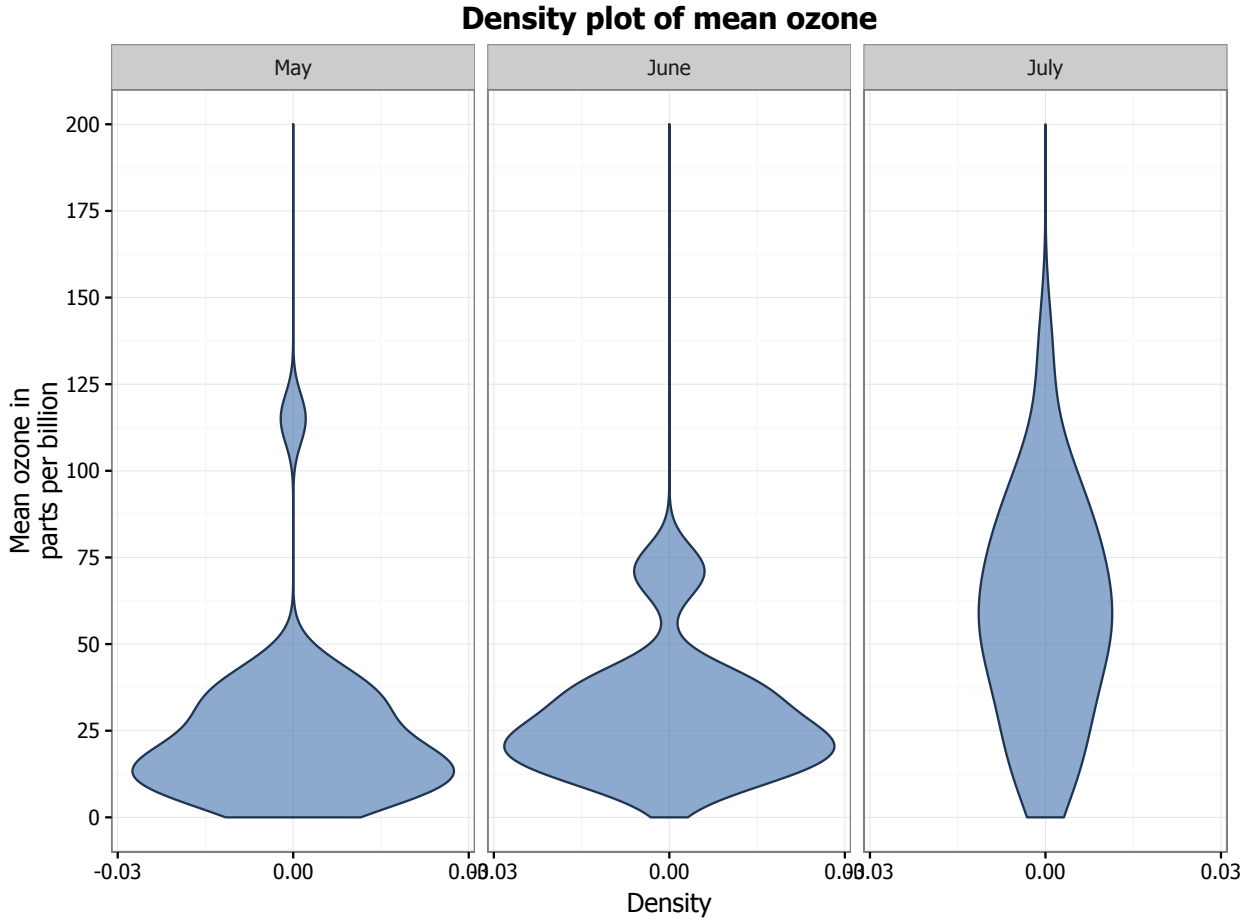
```
      scale_y_continuous(name = "Density") +
      ggtitle("Density plot of mean ozone") +
      facet_grid(. ~ Month.f) +
      theme_bw() +
      theme(plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
            text = element_text(size = 12, family = "Tahoma"))
p8
```



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)) +
```

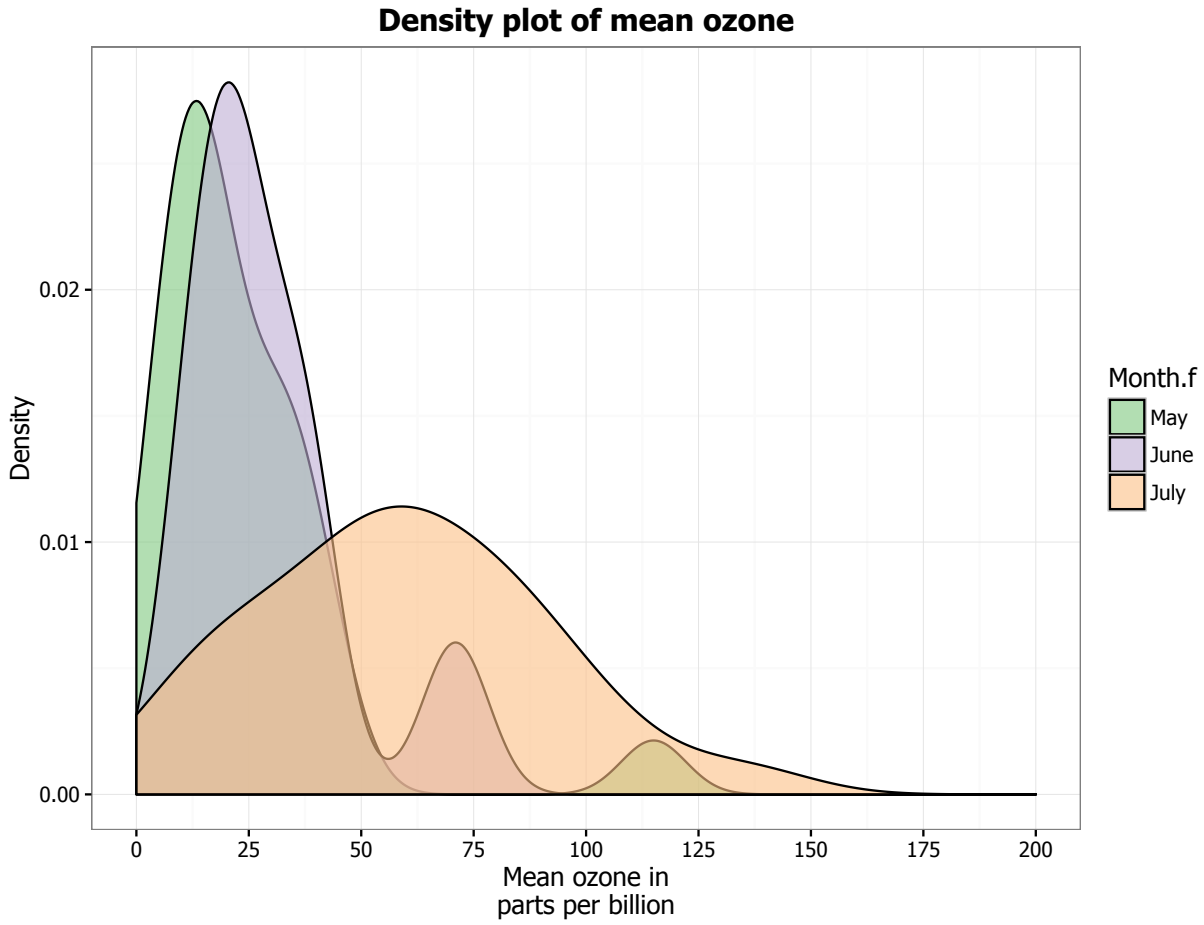
```
      scale_y_continuous(name = "Density",
                         breaks = seq(-0.03, 0.03, 0.03)) +
      ggtitle("Density plot of mean ozone") +
      facet_grid(. ~ Month.f) +
      coord_flip() +
      theme_bw() +
      theme(plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
            text = element_text(size = 12, family = "Tahoma"))
p8
```



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 packages.

```
library(RColorBrewer)

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),
```

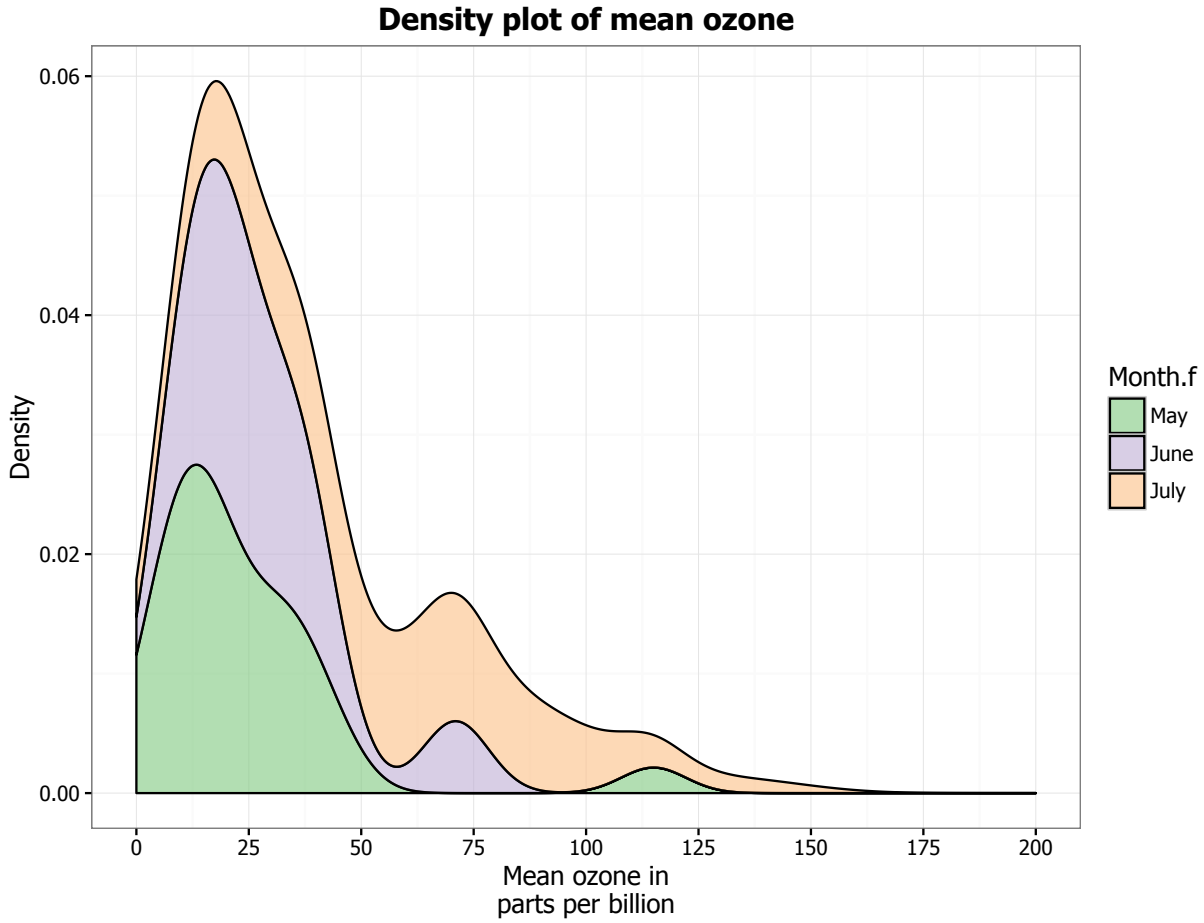
```
                        limits=c(0, 200)) +
      scale_y_continuous(name = "Density") +
      ggtitle("Density plot of mean ozone") +
      scale_fill_brewer(palette="Accent") +
      theme_bw() +
      theme(plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
            text = 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`.

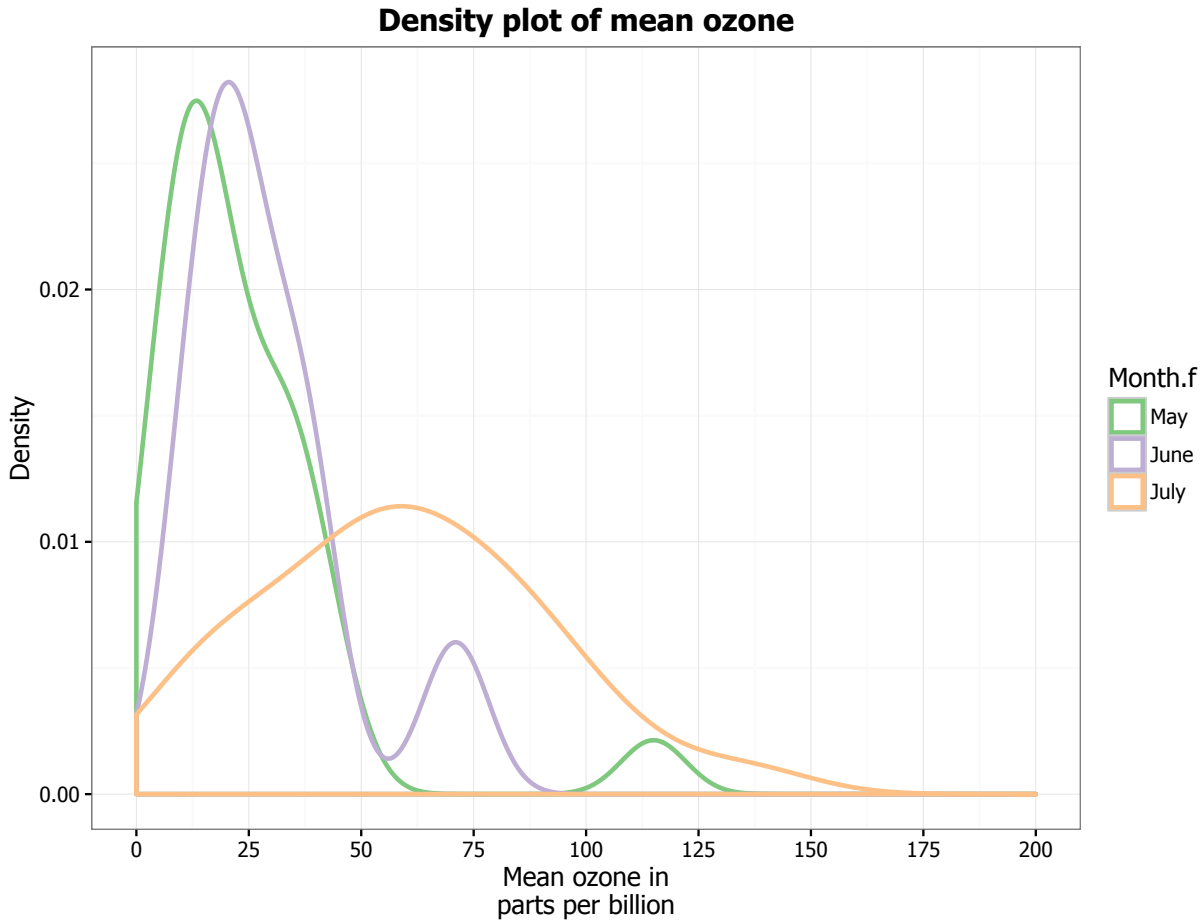
```
p8 <- ggplot(airquality_trimmed, aes(x = Ozone, fill = Month.f)) +
      geom_density(position = "stack", alpha = 0.6) +
      scale_x_continuous(name = "Mean ozone in\nparts per billion",
                         breaks = seq(0, 200, 25),
                         limits=c(0, 200)) +
      scale_y_continuous(name = "Density") +
      ggtitle("Density plot of mean ozone") +
      scale_fill_brewer(palette="Accent") +
      theme_bw() +
      theme(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()`.

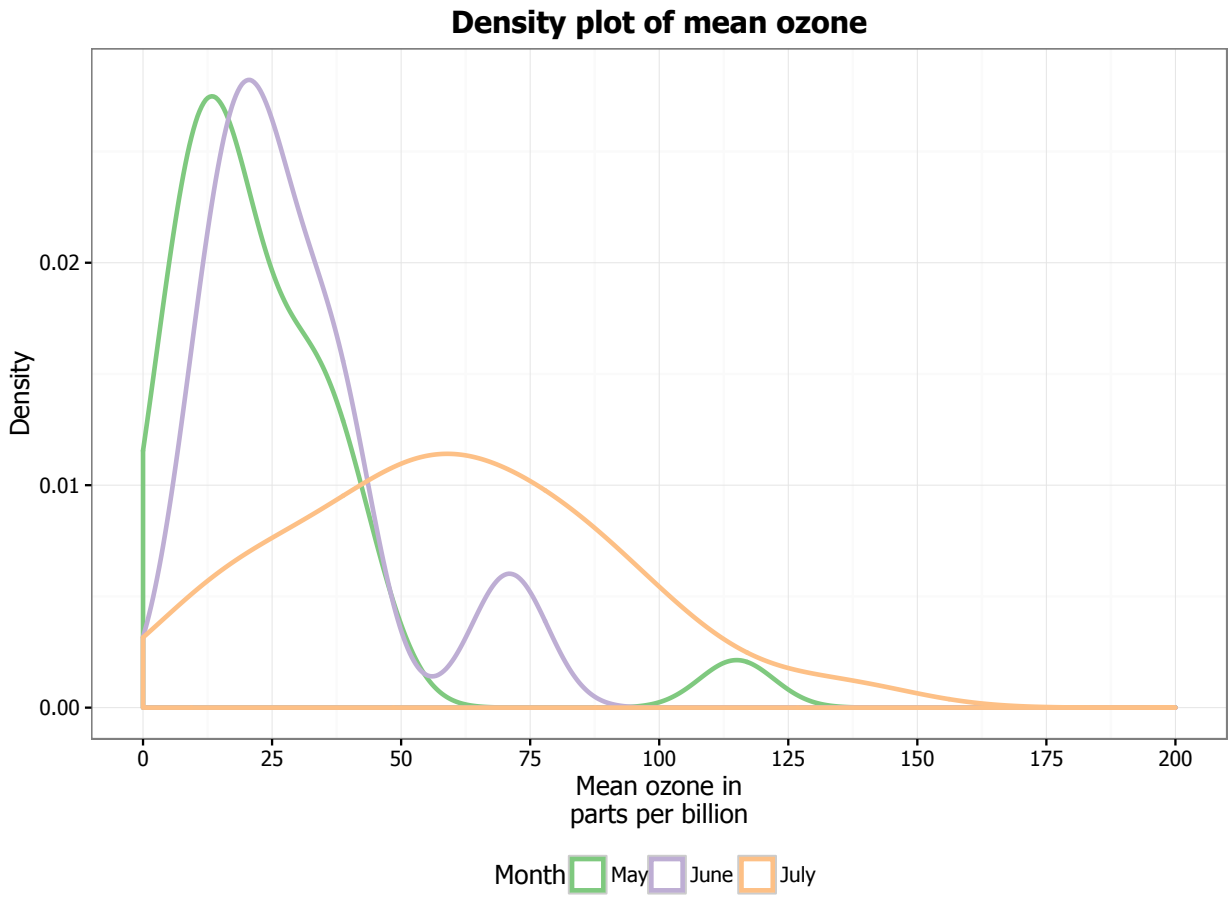
```
p8 <- ggplot(airquality_trimmed, aes(x = Ozone, colour = Month.f)) +
      geom_density(position="identity", fill = NA, size = 1) +
      scale_x_continuous(name = "Mean ozone in\nparts per billion",
                         breaks = seq(0, 200, 25),
                         limits=c(0, 200)) +
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



Formatting the legend

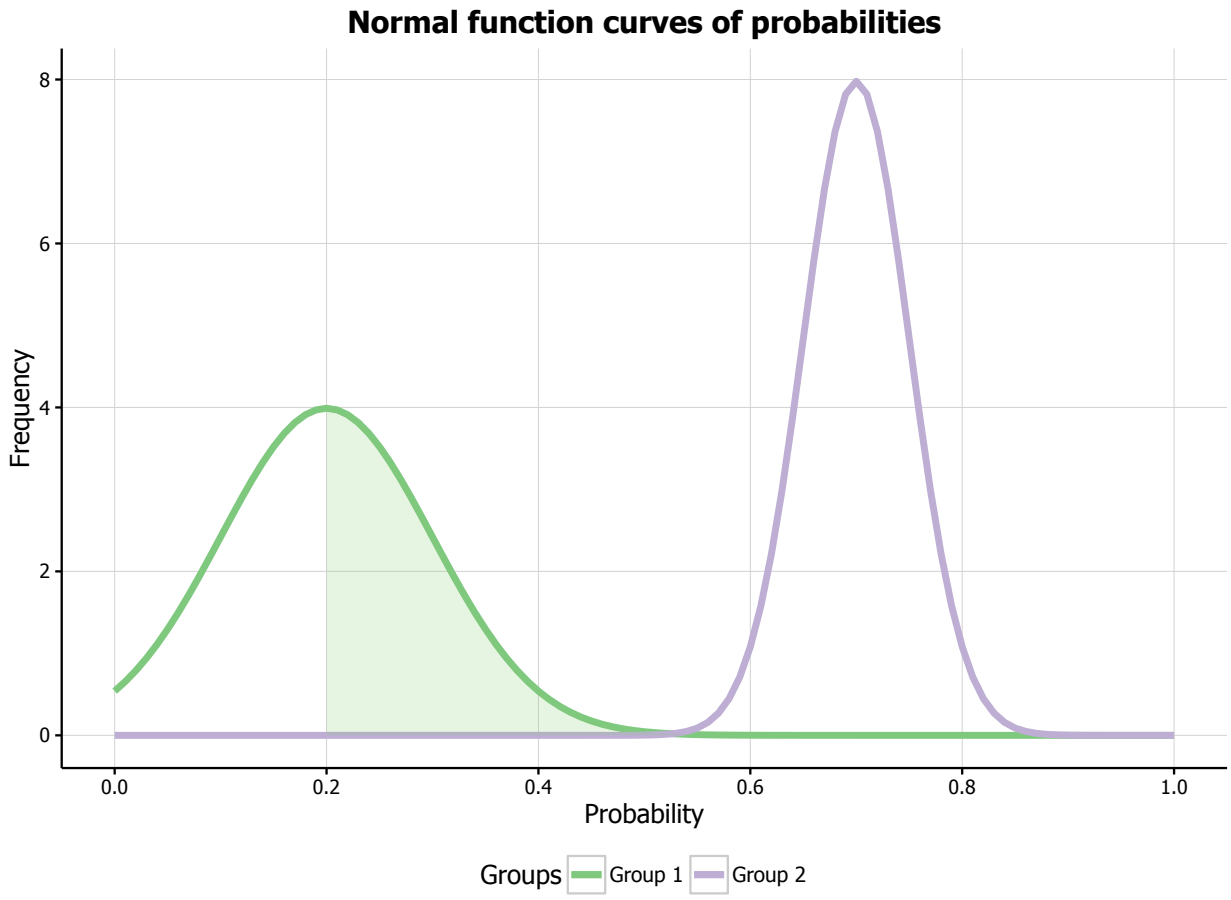
Finally, we can format the legend. Firstly, we can change the position by adding the `legend.position = "bottom"` argument to the `theme` option, which moves the legend under the plot. Secondly, we can fix the title by adding the `labs(fill="Month")` option to the plot.

```
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",
            plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
            text = element_text(size = 12, family = "Tahoma"))
p8
```



Function Plots

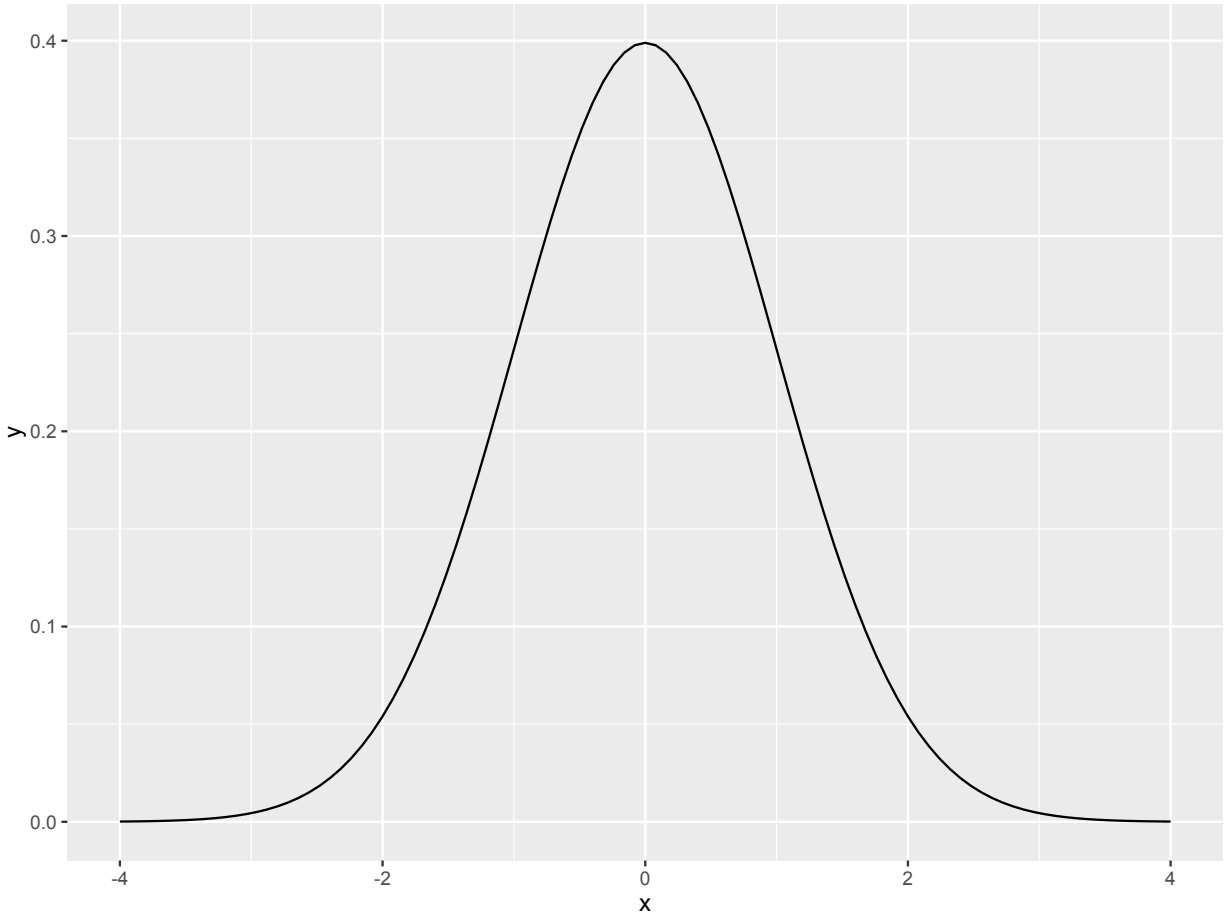
In this part, we will work towards creating the function plot below. We will take you from a basic function plot and explain all the customisations we add to the code step-by-step.



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.

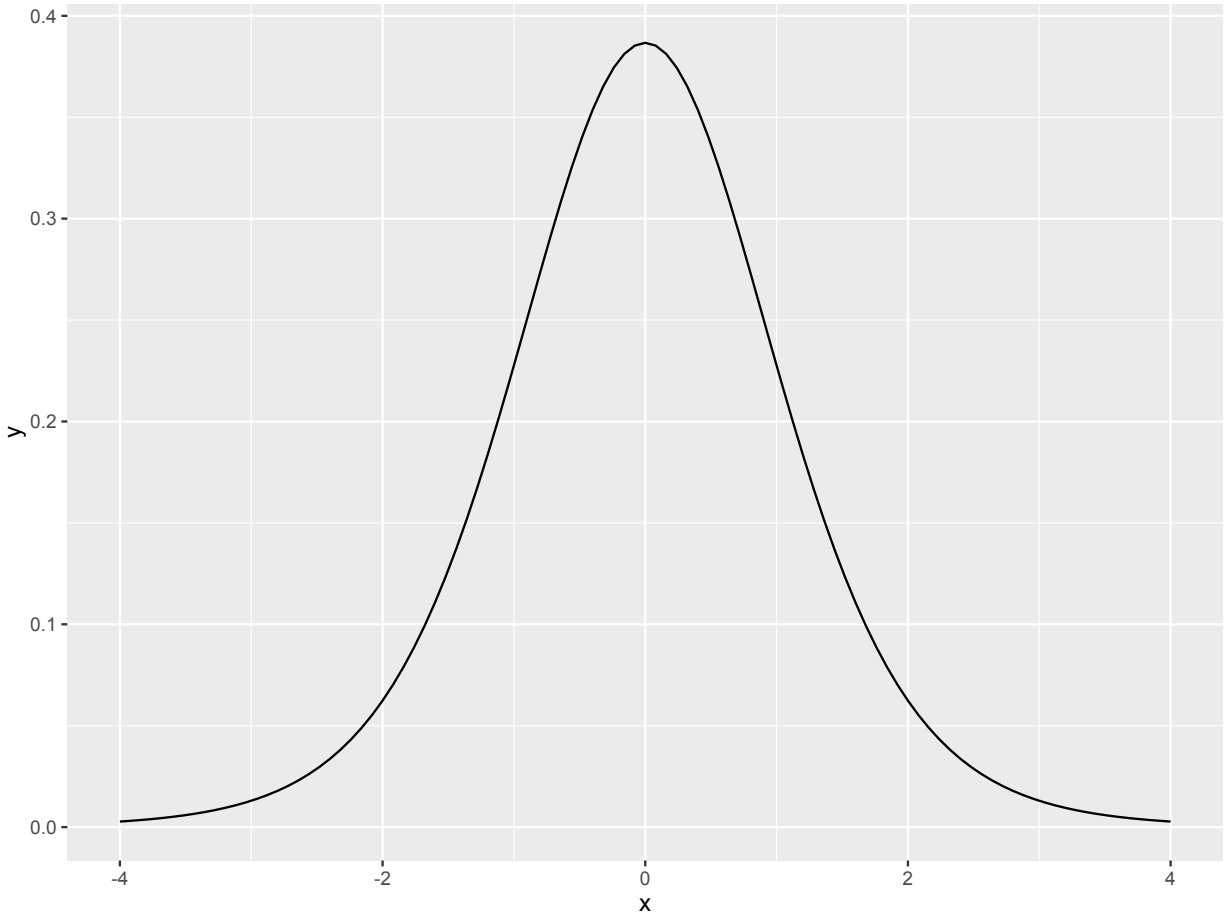
```
p9 <- ggplot(data.frame(x = c(-4, 4)), aes(x = x)) +
      stat_function(fun = dnorm)
p9
```



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.

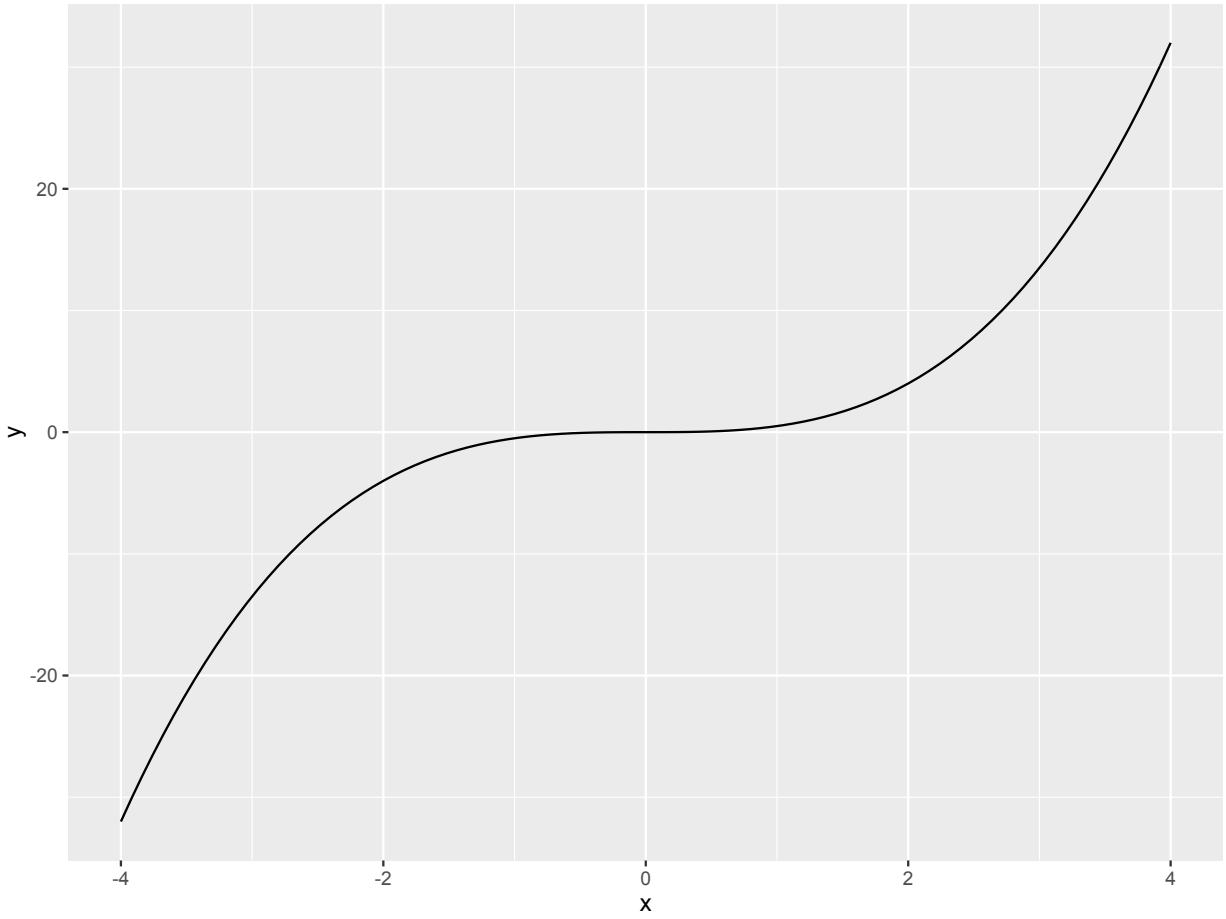
```
p9 <- ggplot(data.frame(x = c(-4, 4)), aes(x = x)) +
      stat_function(fun = dt, args = list(df = 8))
p9
```



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:

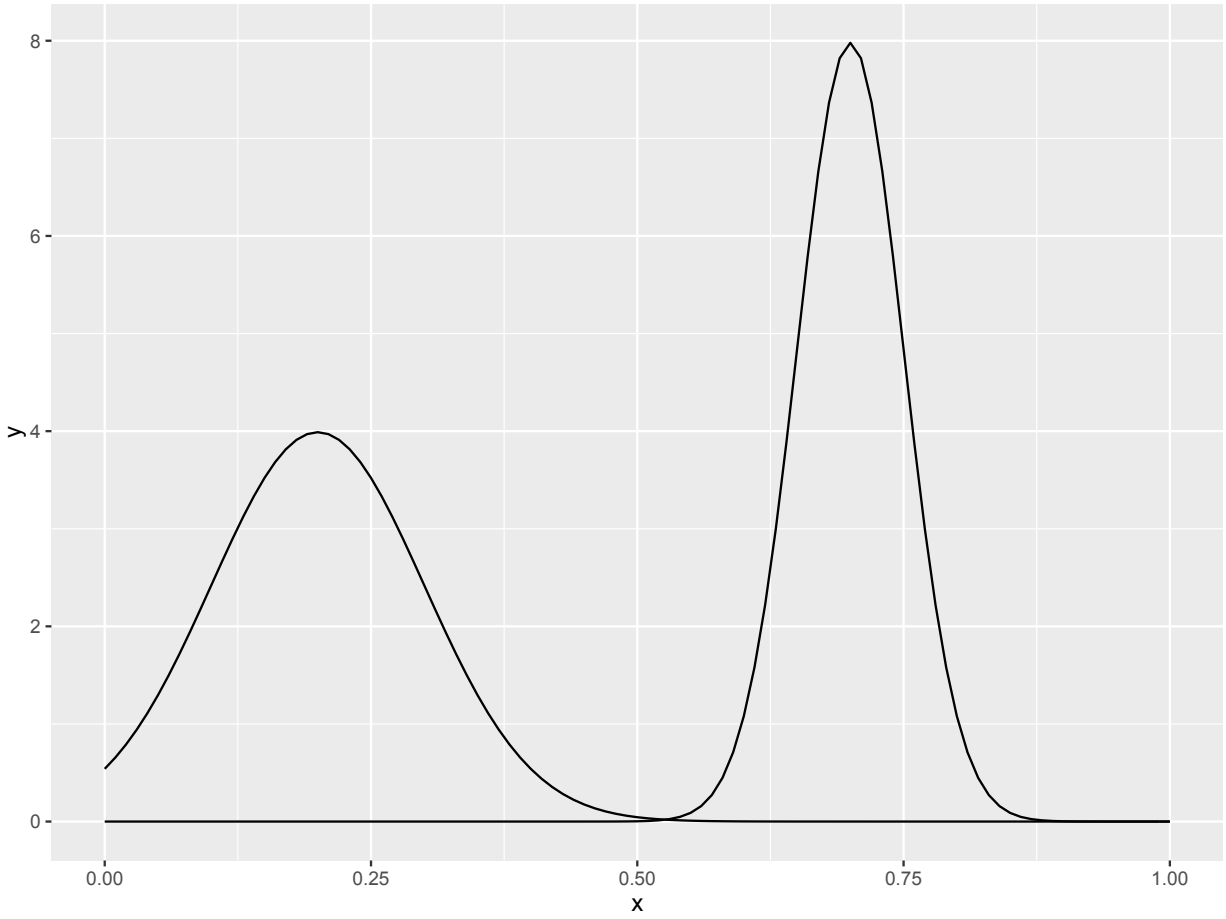
```
cubeFun <- function(x) {
    x^3 * 0.5
}

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



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.

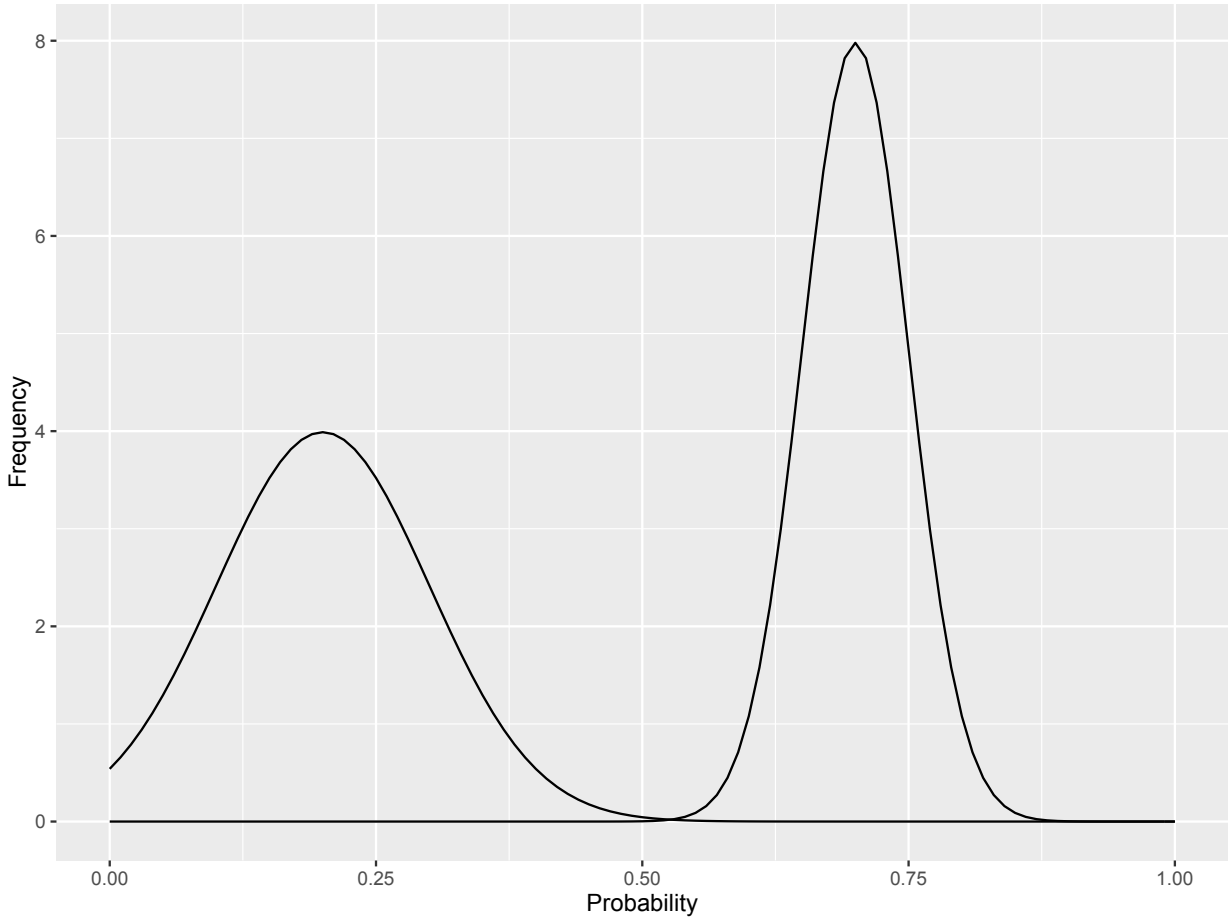
```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
      stat_function(fun = dnorm, args = list(0.2, 0.1)) +
      stat_function(fun = dnorm, args = list(0.7, 0.05))
p9
```



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.

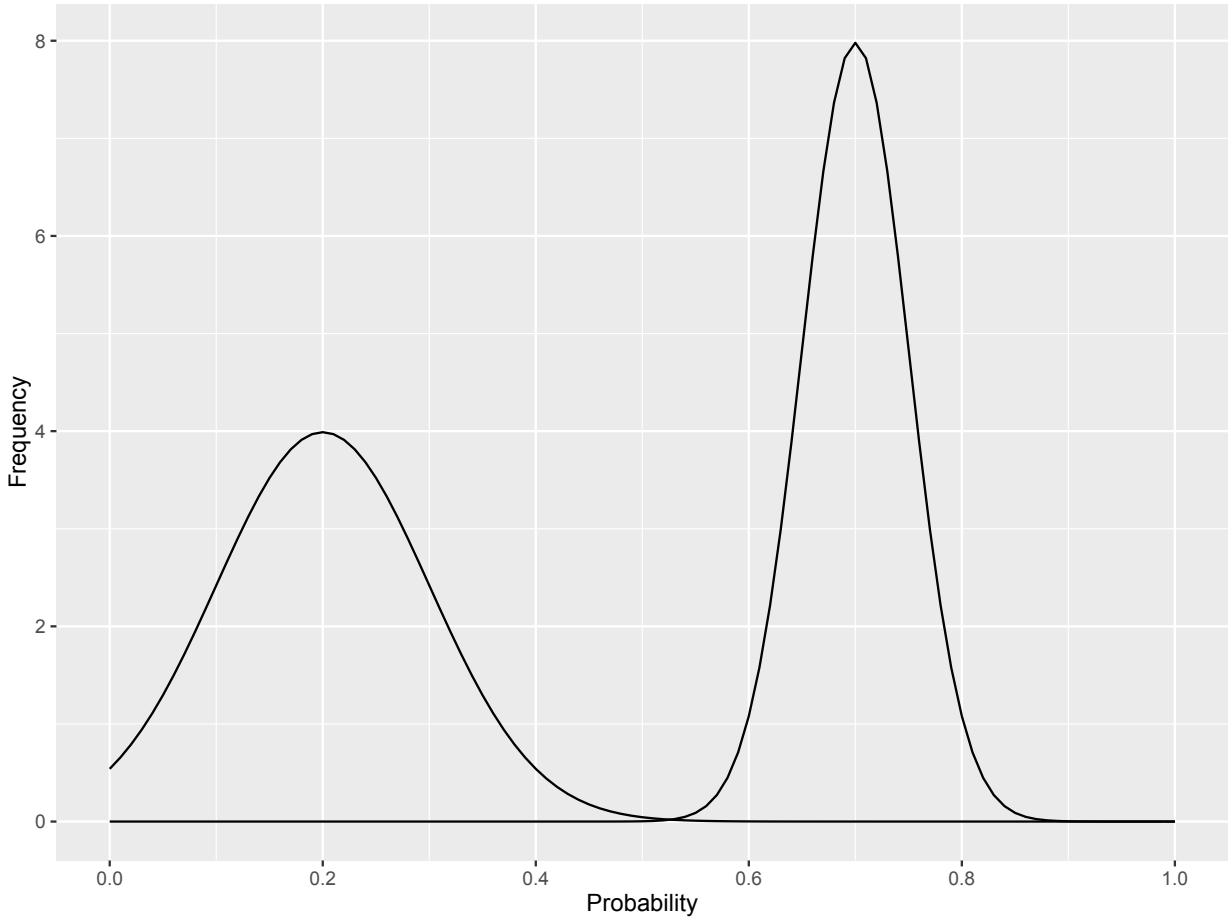
```
p9 <- p9 + scale_x_continuous(name = "Probability") +
      scale_y_continuous(name = "Frequency")
p9
```



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`.

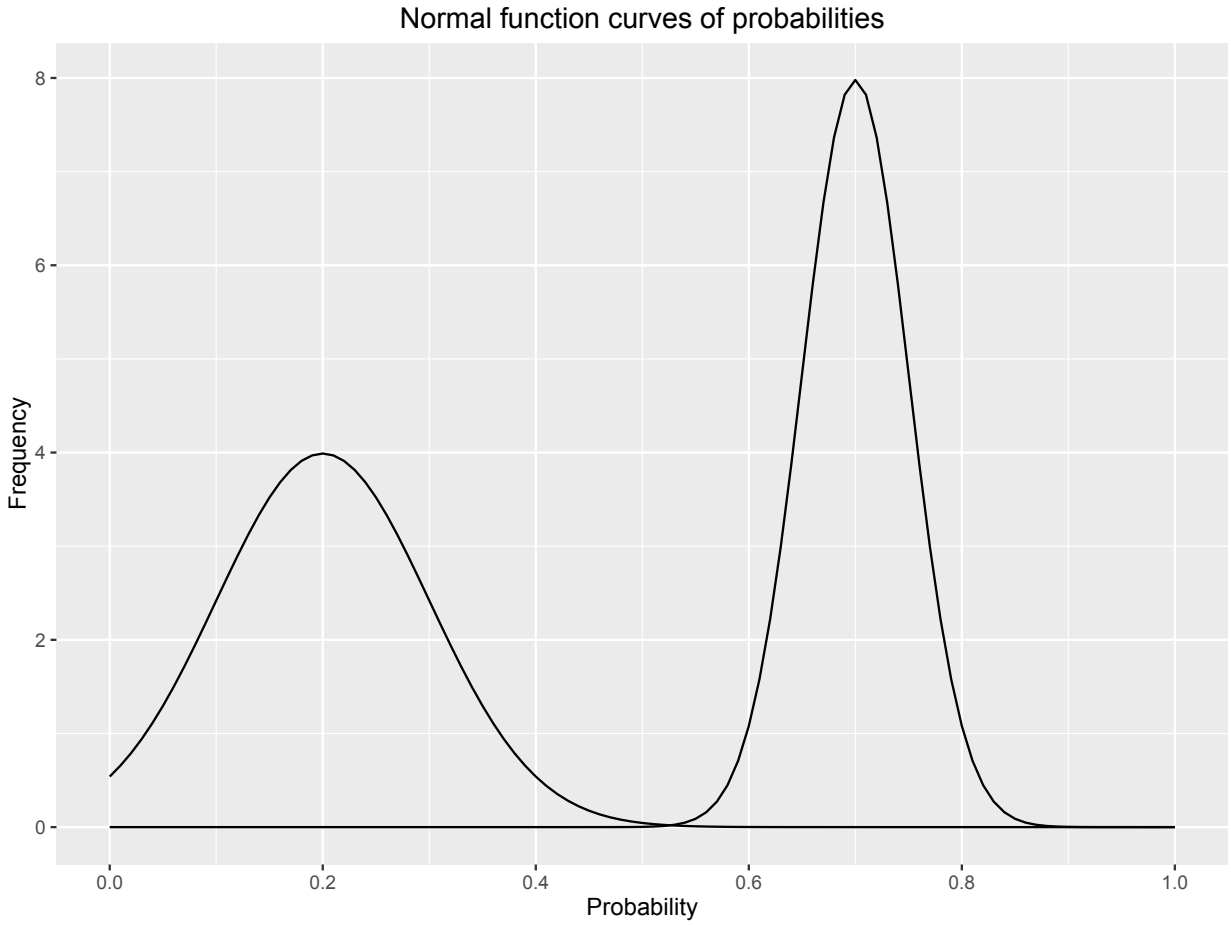
```
p9 <- p9 + scale_x_continuous(name = "Probability",
                              breaks = seq(0, 1, 0.2),
                              limits=c(0, 1)) +
      scale_y_continuous(name = "Frequency")
p9
```



Adding a title

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

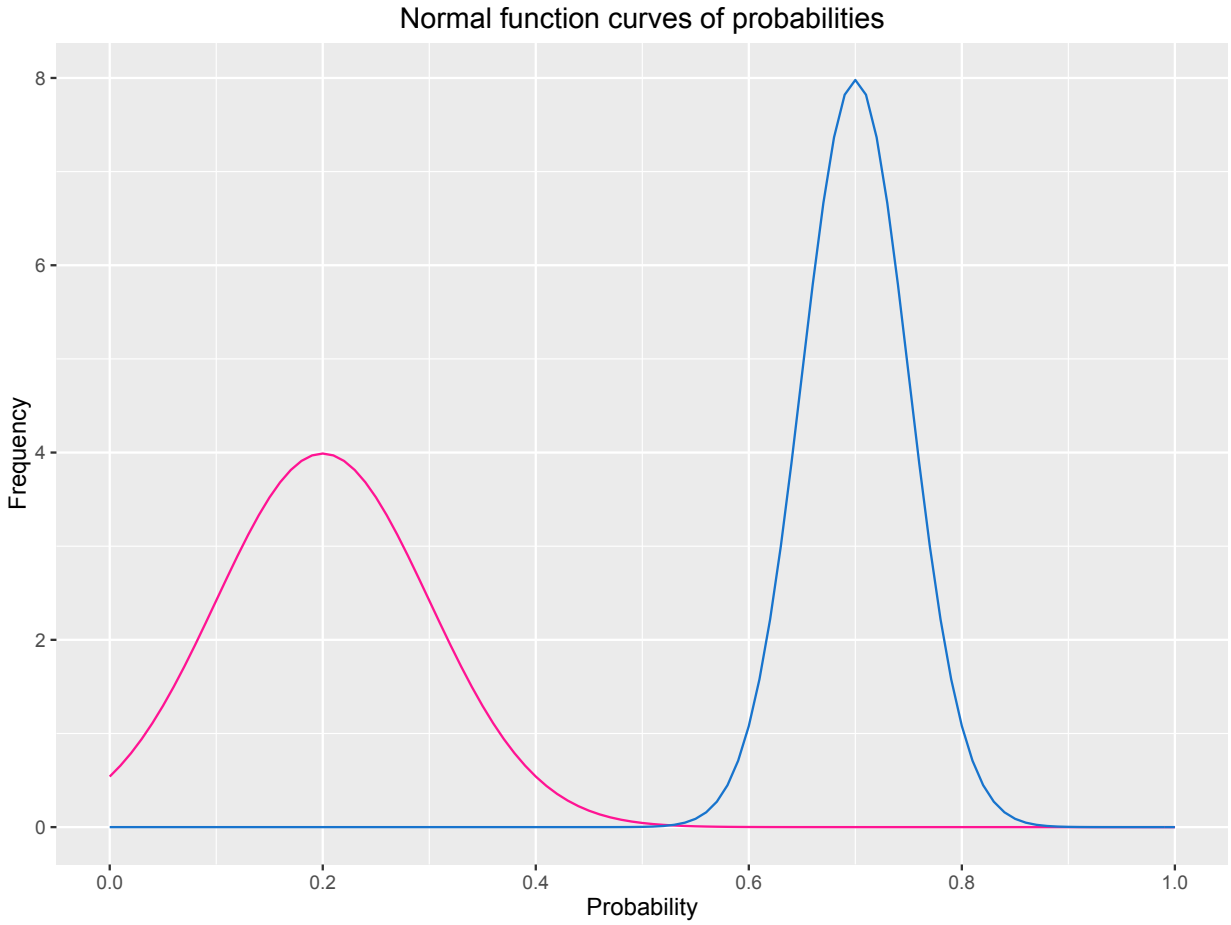
```
p9 <- p9 + ggtitle("Normal function curves of probabilities")
p9
```



Changing the colour of the curves

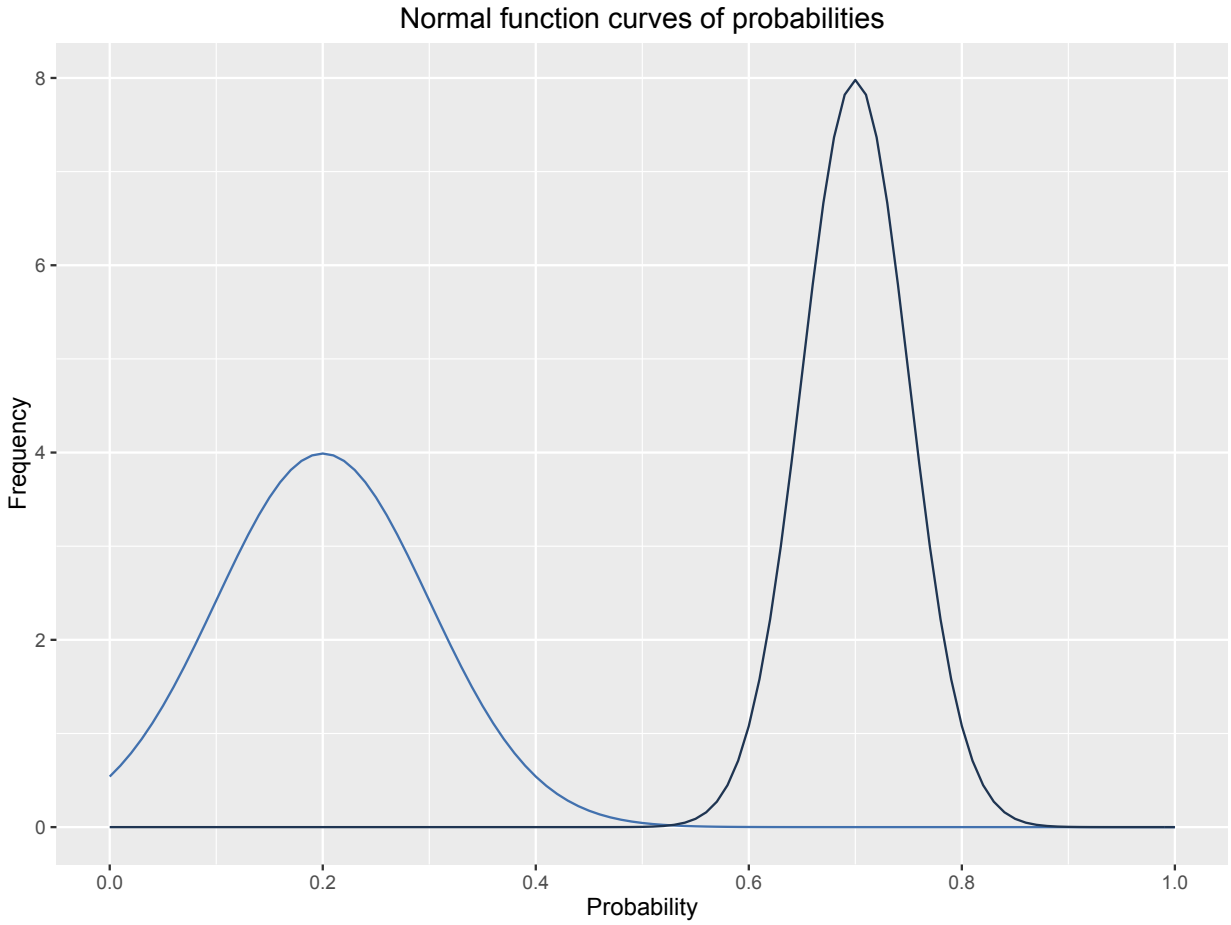
To change the line colours of the curves, we add a valid colour to the `colour` arguments in `stat_function`. A list of valid colours is here.

```
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.

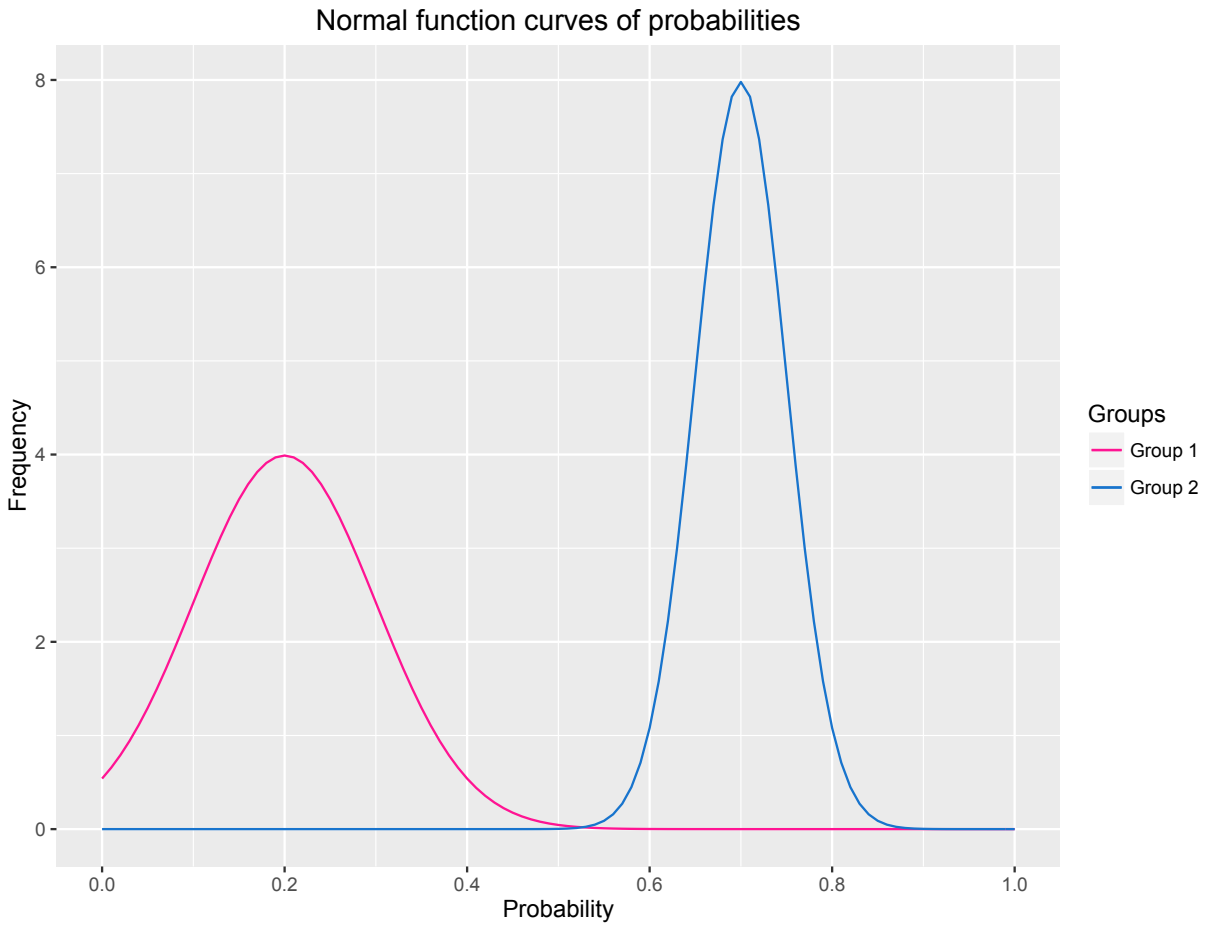
```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
        stat_function(fun = dnorm, args = list(0.2, 0.1),
                      colour = "#4271AE") +
        stat_function(fun = dnorm, args = list(0.7, 0.05),
                      colour = "#1F3552") +
        scale_x_continuous(name = "Probability",
                           breaks = seq(0, 1, 0.2),
                           limits=c(0, 1)) +
        scale_y_continuous(name = "Frequency") +
        ggtitle("Normal function curves of probabilities")
p9
```



Adding a legend

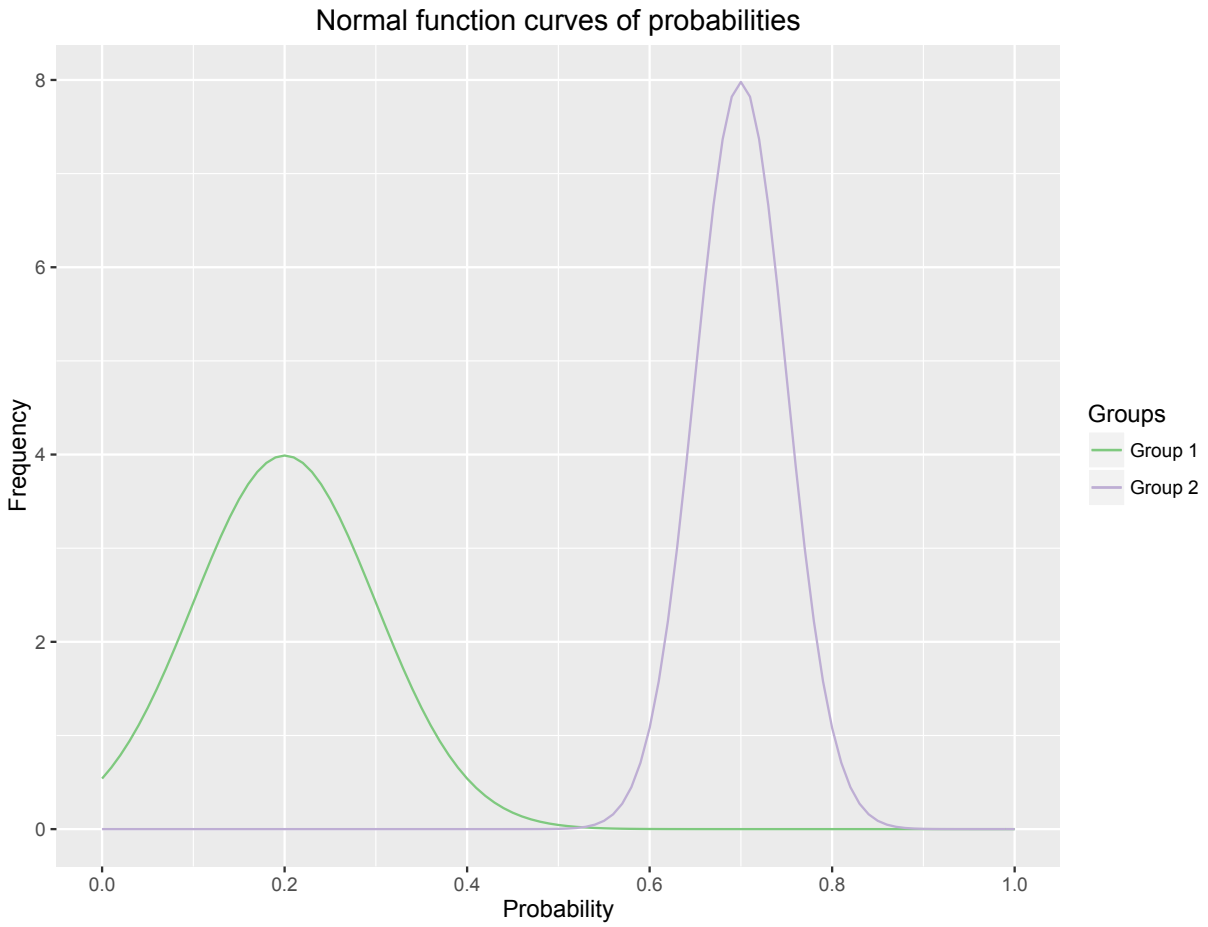
As we have added two separate commands to plot the two function curves, ggplot does not automatically recognise that it needs to create a legend. We can make a legend by swapping out the `colour` argument in each of the `stat_function` commands for `aes(colour = )`, and assigning it the name of the group.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

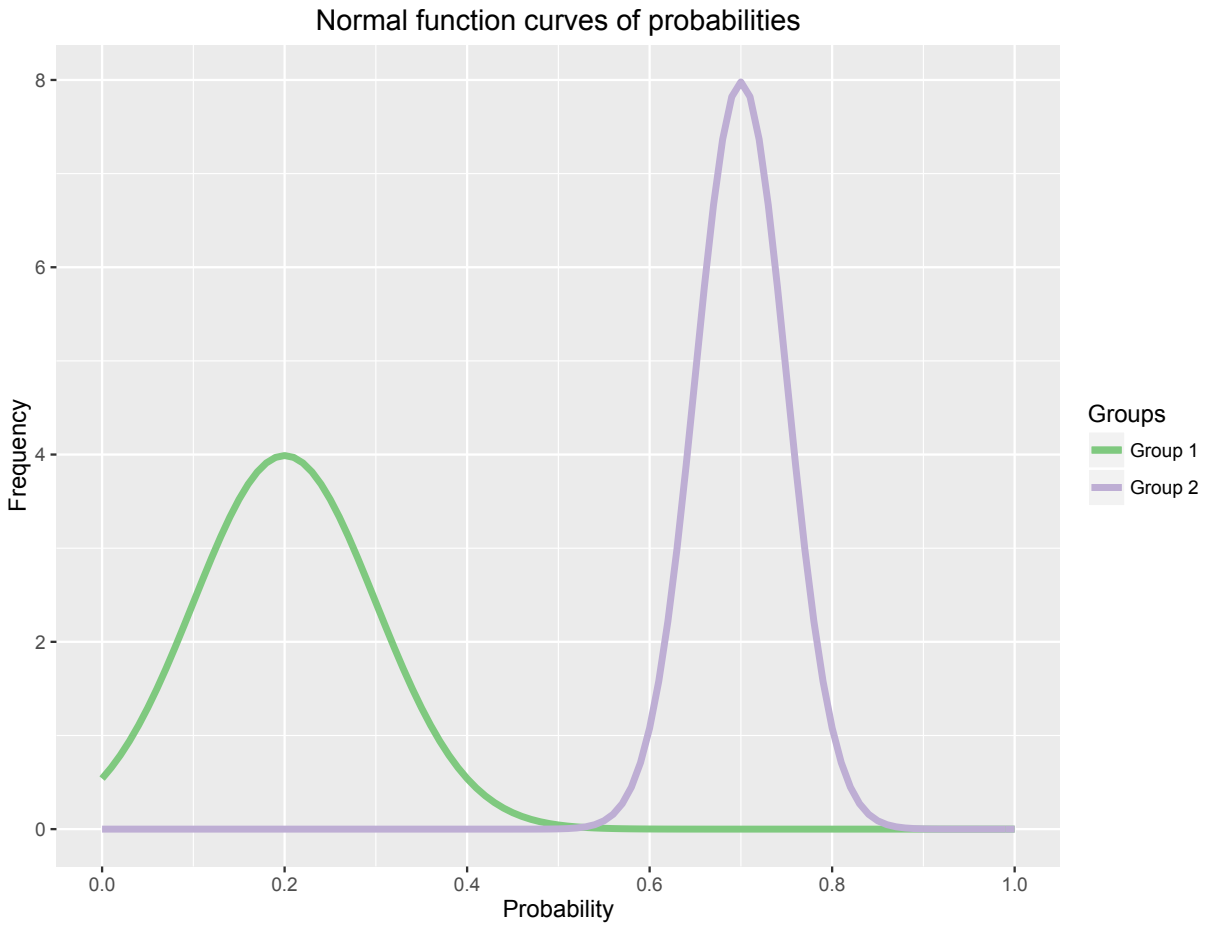
```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
        stat_function(fun = dnorm, args = list(0.2, 0.1),
                      aes(colour = "Group 1")) +
        stat_function(fun = dnorm, args = list(0.7, 0.05),
                      aes(colour = "Group 2")) +
        scale_x_continuous(name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



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.

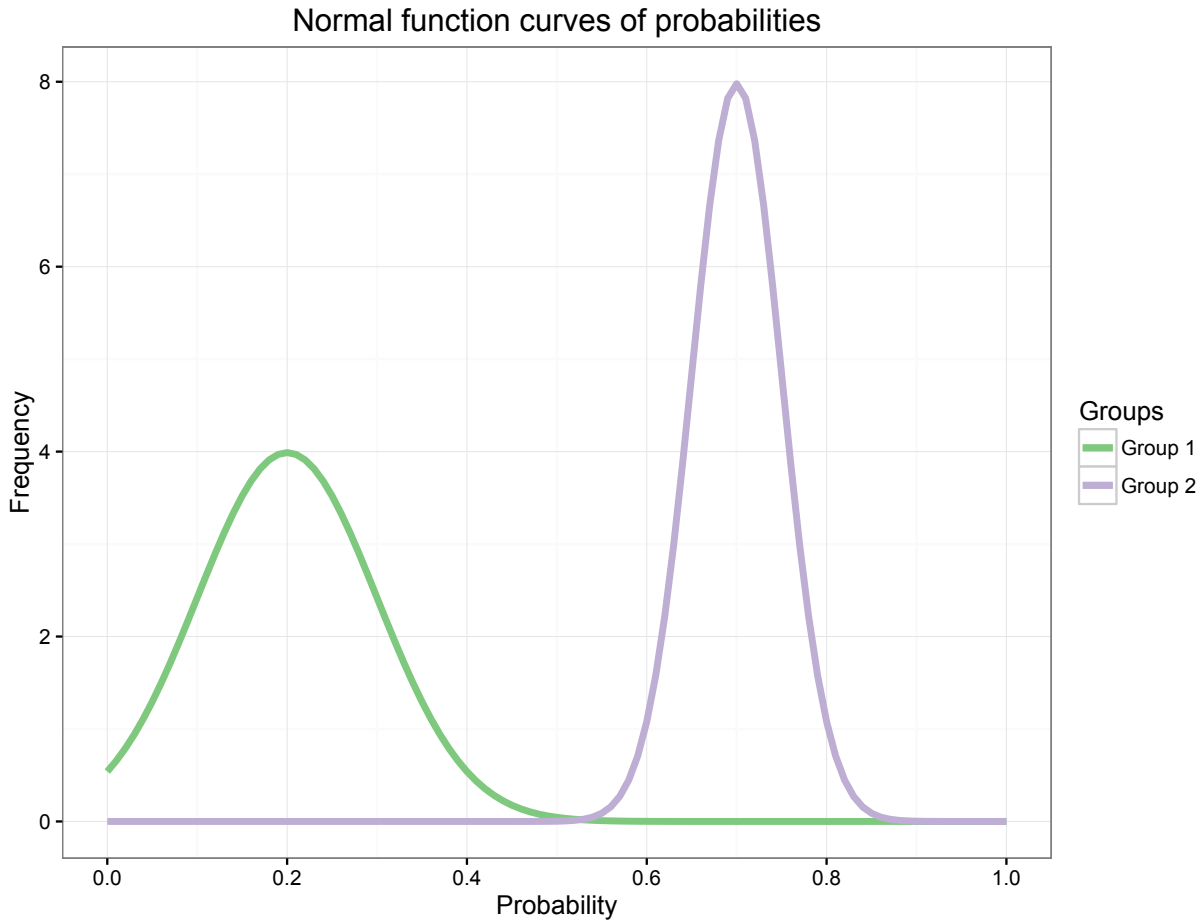
```
p9 <- ggplot(data.frame(x = c(0, 1)), aes(x = x)) +
        stat_function(fun = dnorm, args = list(0.2, 0.1),
                      aes(colour = "Group 1"), size = 1.5) +
        stat_function(fun = dnorm, args = list(0.7, 0.05),
                      aes(colour = "Group 2"), size = 1.5) +
     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



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.

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



Creating an XKCD style chart

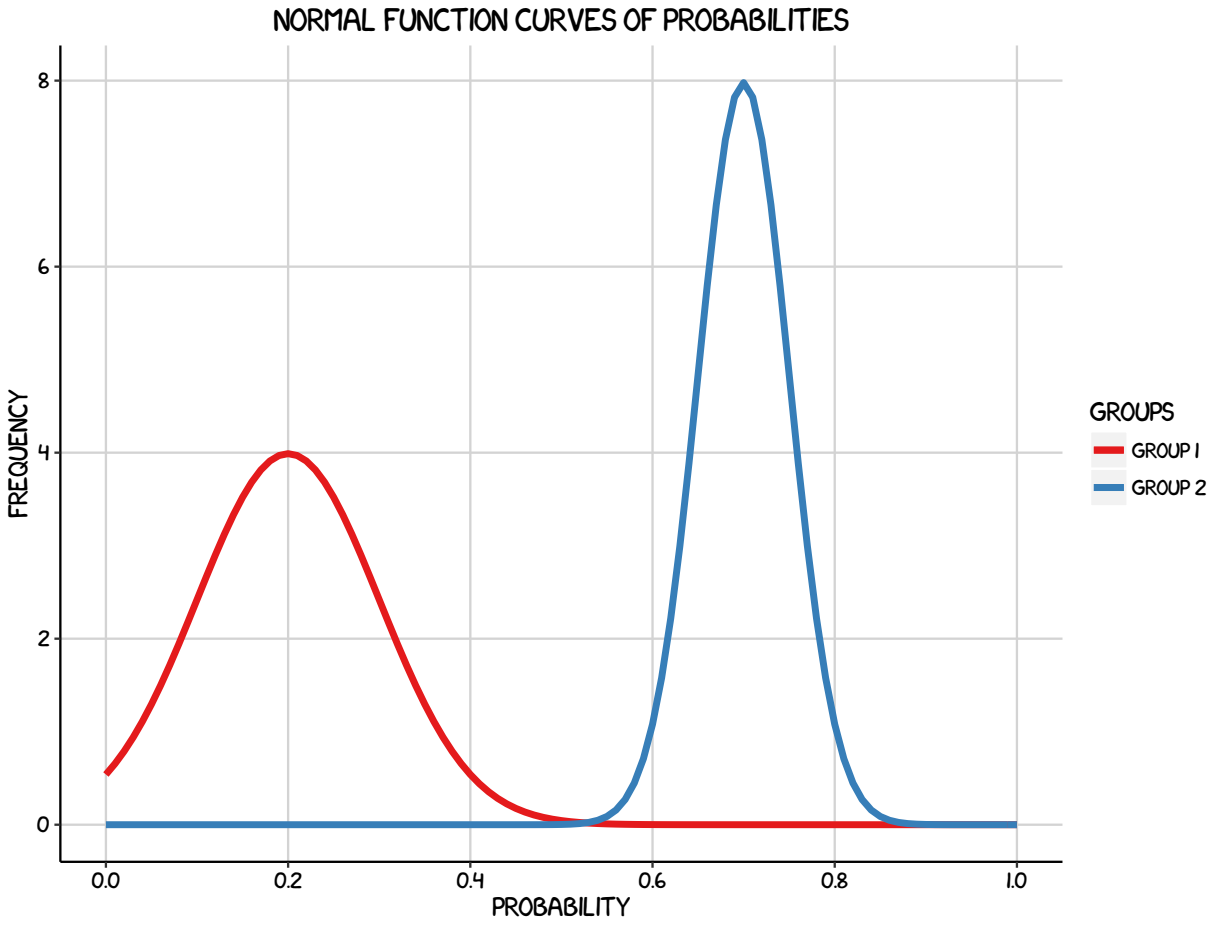
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
              dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

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

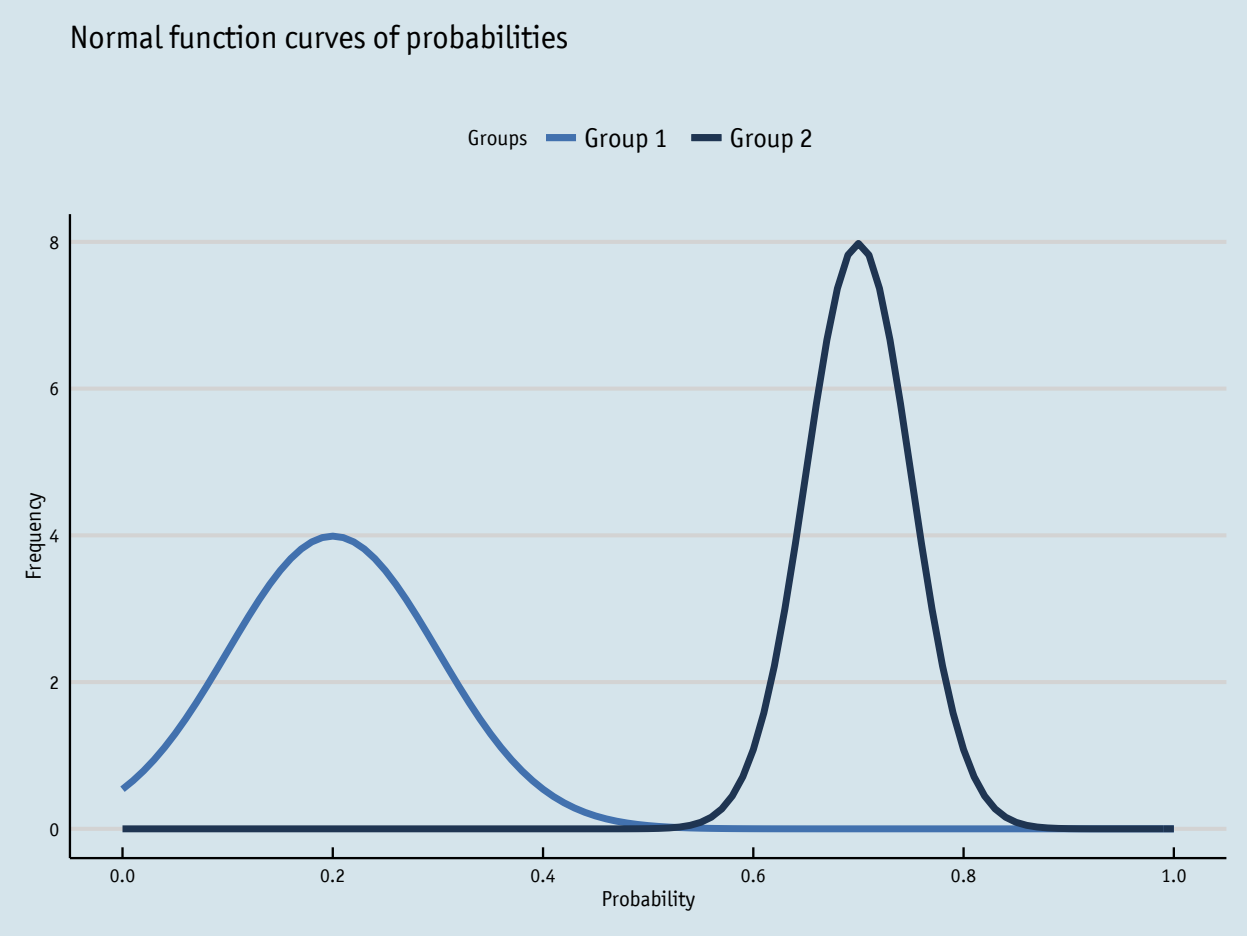
```
                  aes(colour = "Group 1"), size = 1.5) +
      stat_function(fun = dnorm, args = list(0.7, 0.05),
                  aes(colour = "Group 2"), size = 1.5) +
      scale_x_continuous(name = "Probability",
                         breaks = seq(0, 1, 0.2),
                         limits=c(0, 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 = 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"))
p9
```



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.

```
library(ggthemes)

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() +
      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 = 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 = "OfficinaSanITC-Book"),
            text=element_text(family="OfficinaSanITC-Book"))
p9
```

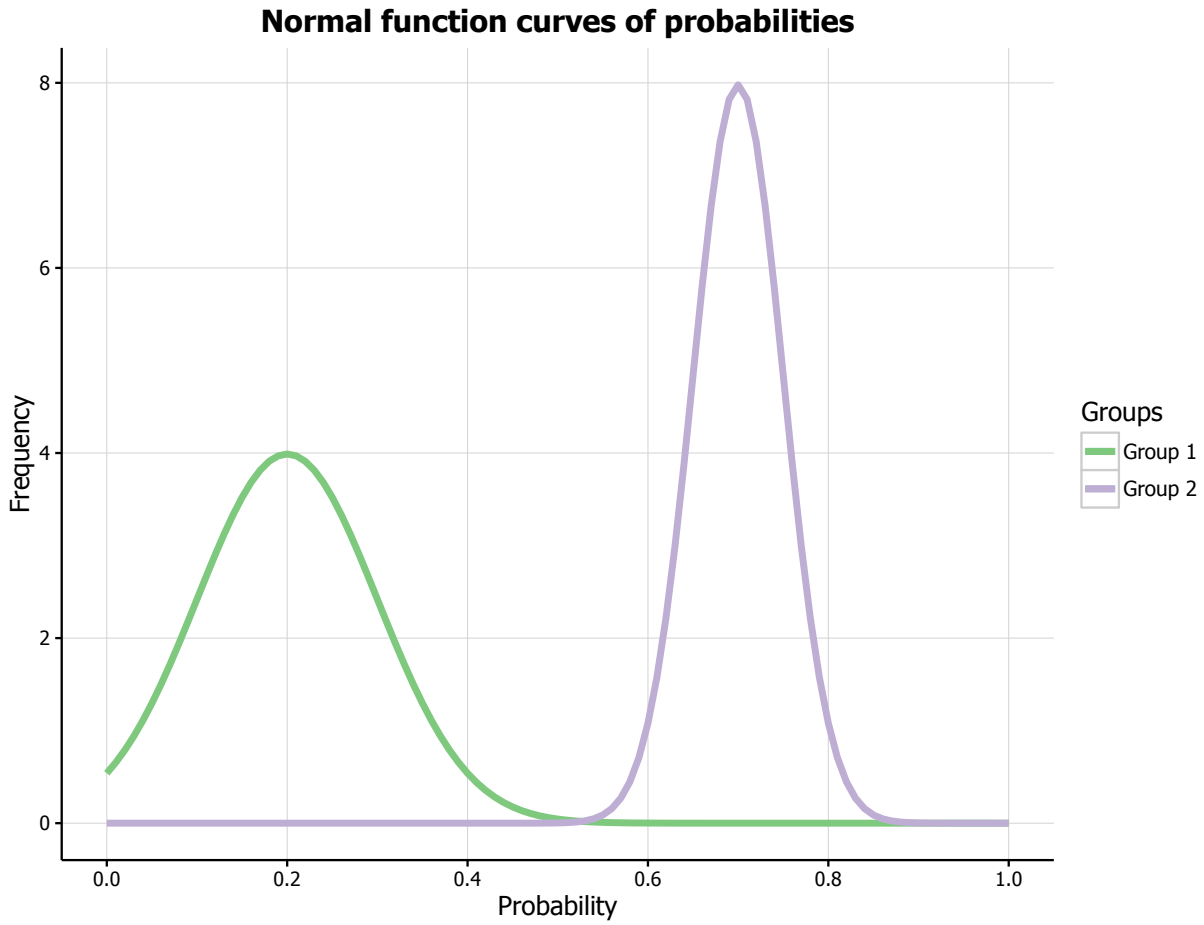


Creating your own theme

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.

```
library(grid)

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(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 = 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
```

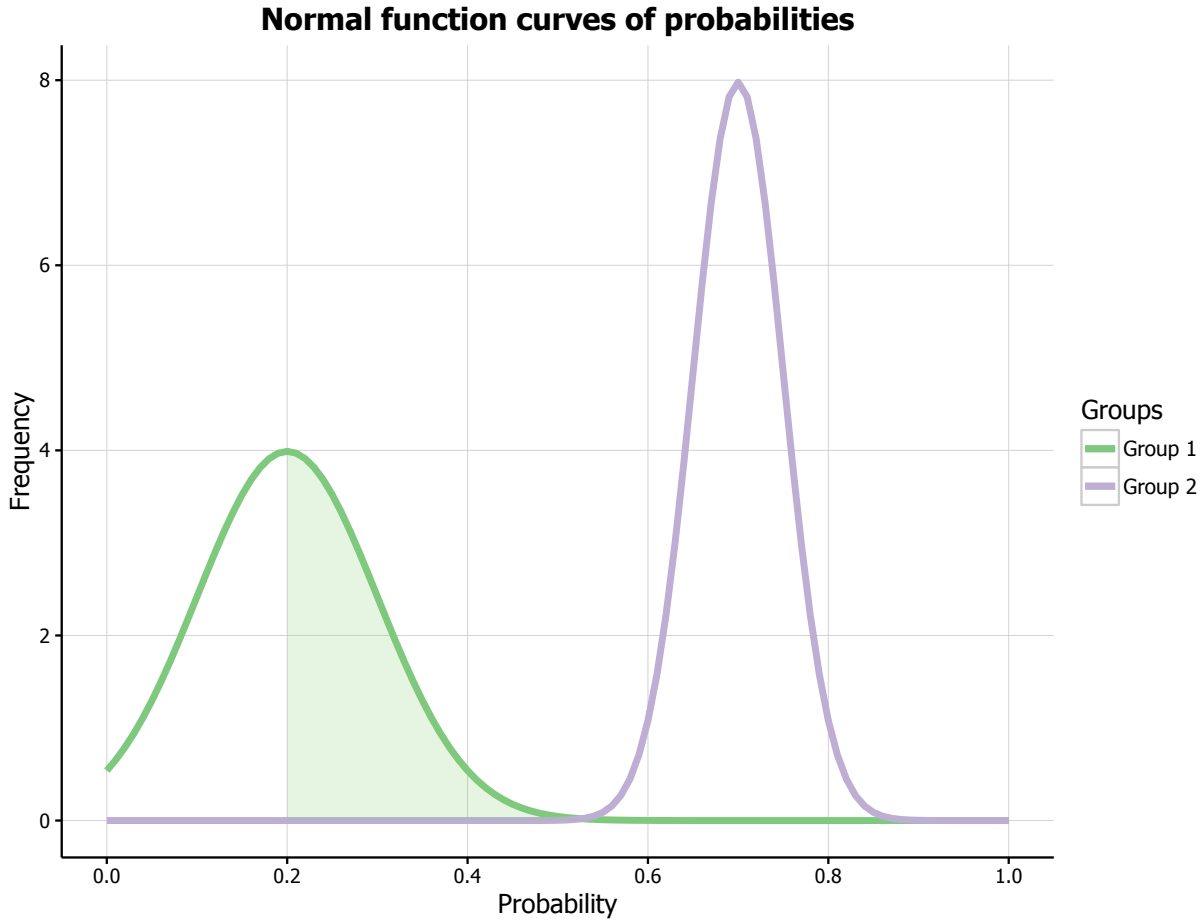


Adding areas under the curve

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
```

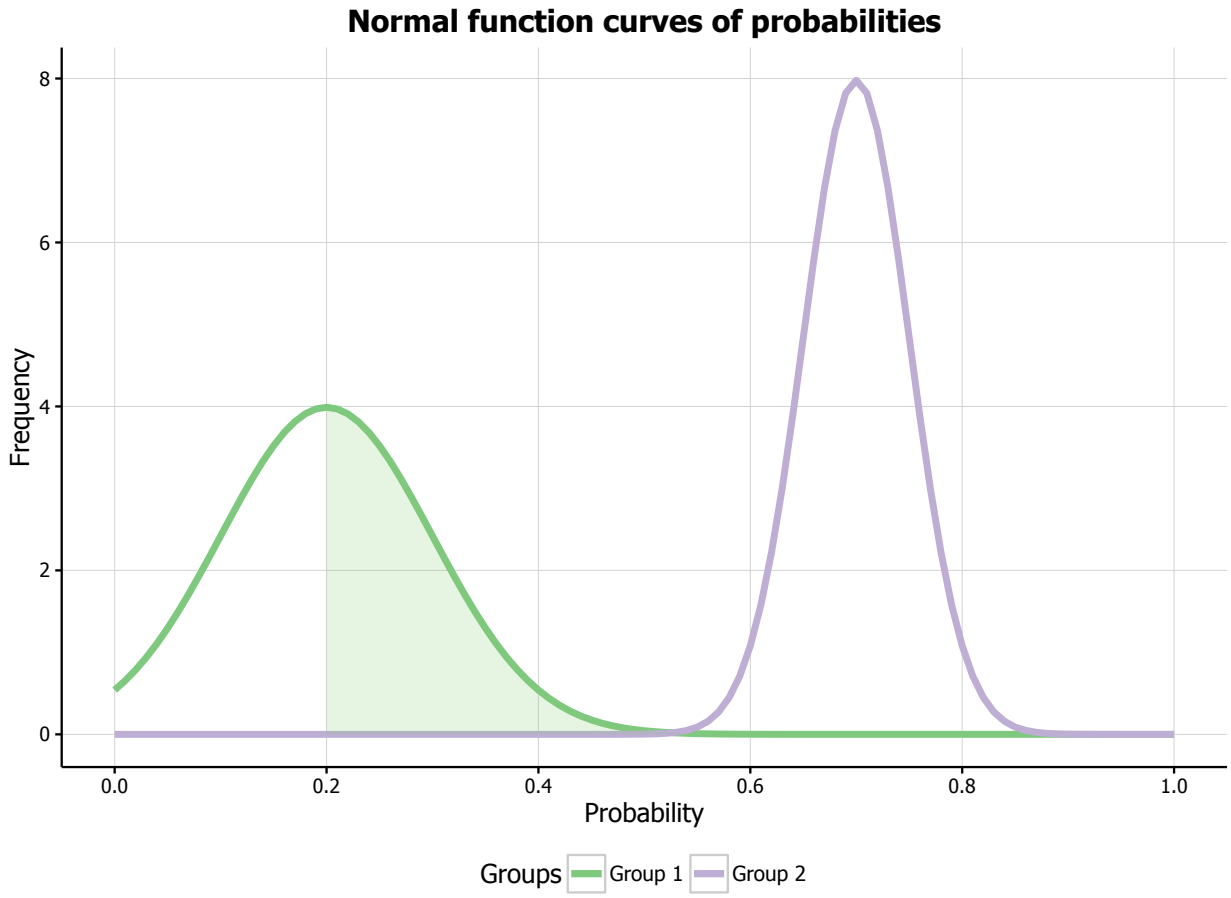


Formatting the legend

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.

```
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(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 = 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
```



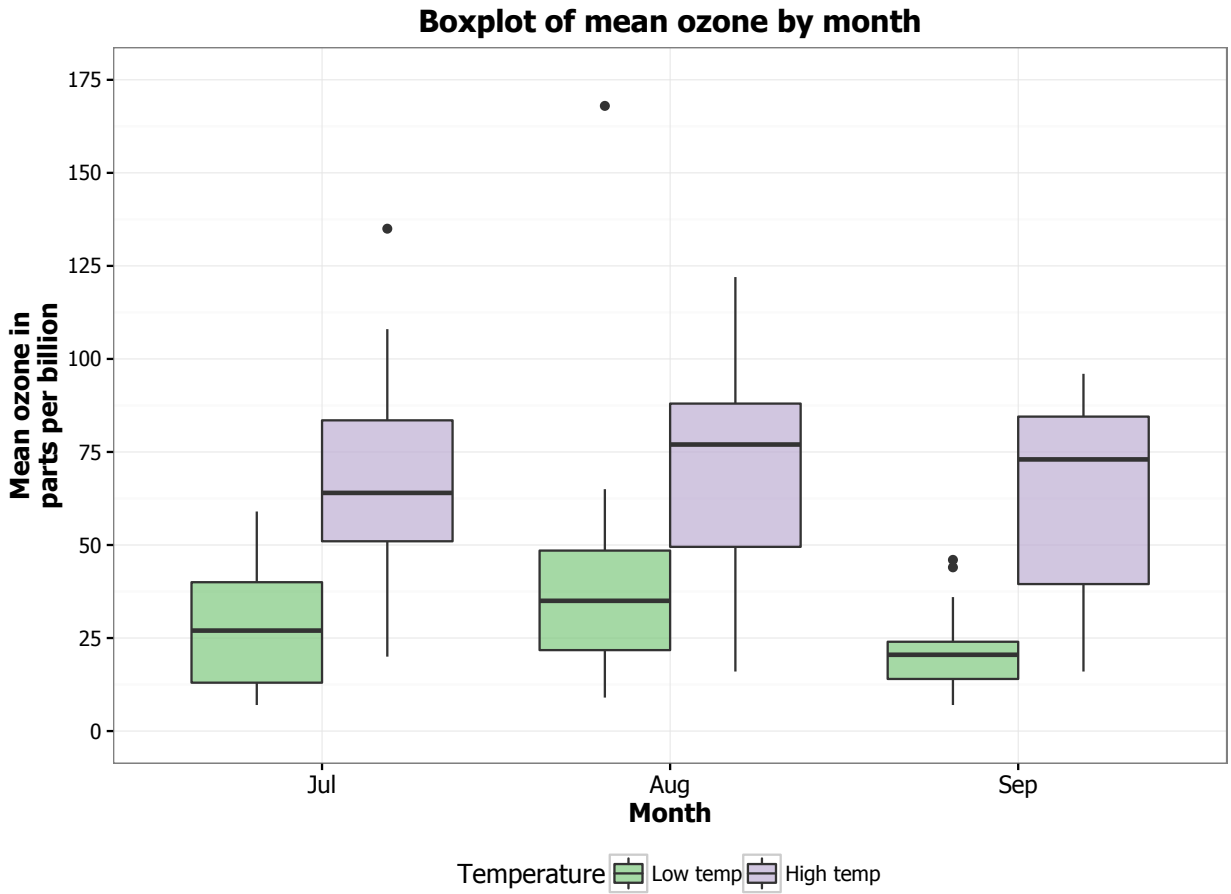
Boxplots

The first thing to do is load in the data, as below. We'll convert `Month` into a labelled factor in order to use it as our grouping variable.

```
rm(list = ls())
library(datasets)
```

```
library(ggplot2)

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

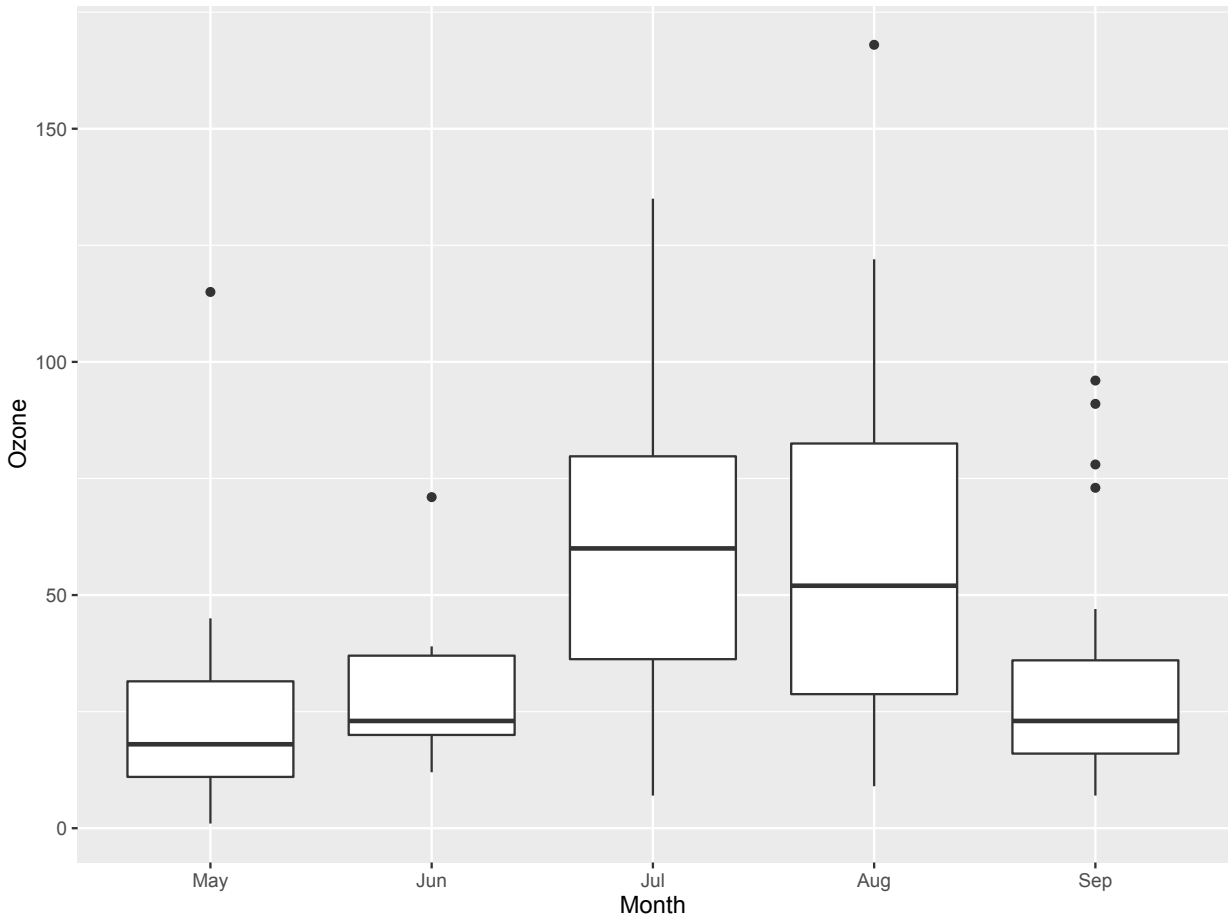
In this part, we will work towards creating the boxplot below. We will take you from a basic boxplot and explain all the customisations we add to the code step-by-step.



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.

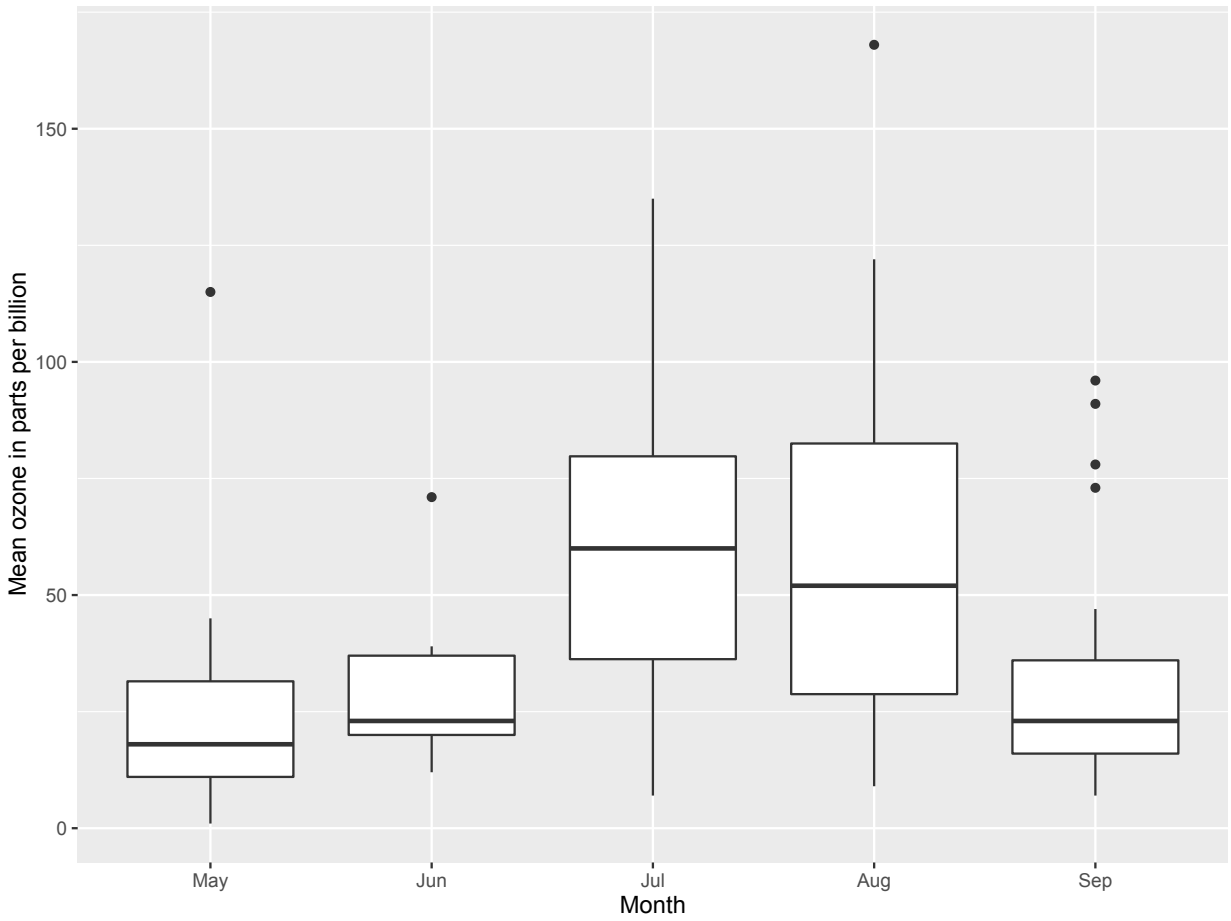
```
p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
        geom_boxplot()
p10
```



Customising axis labels

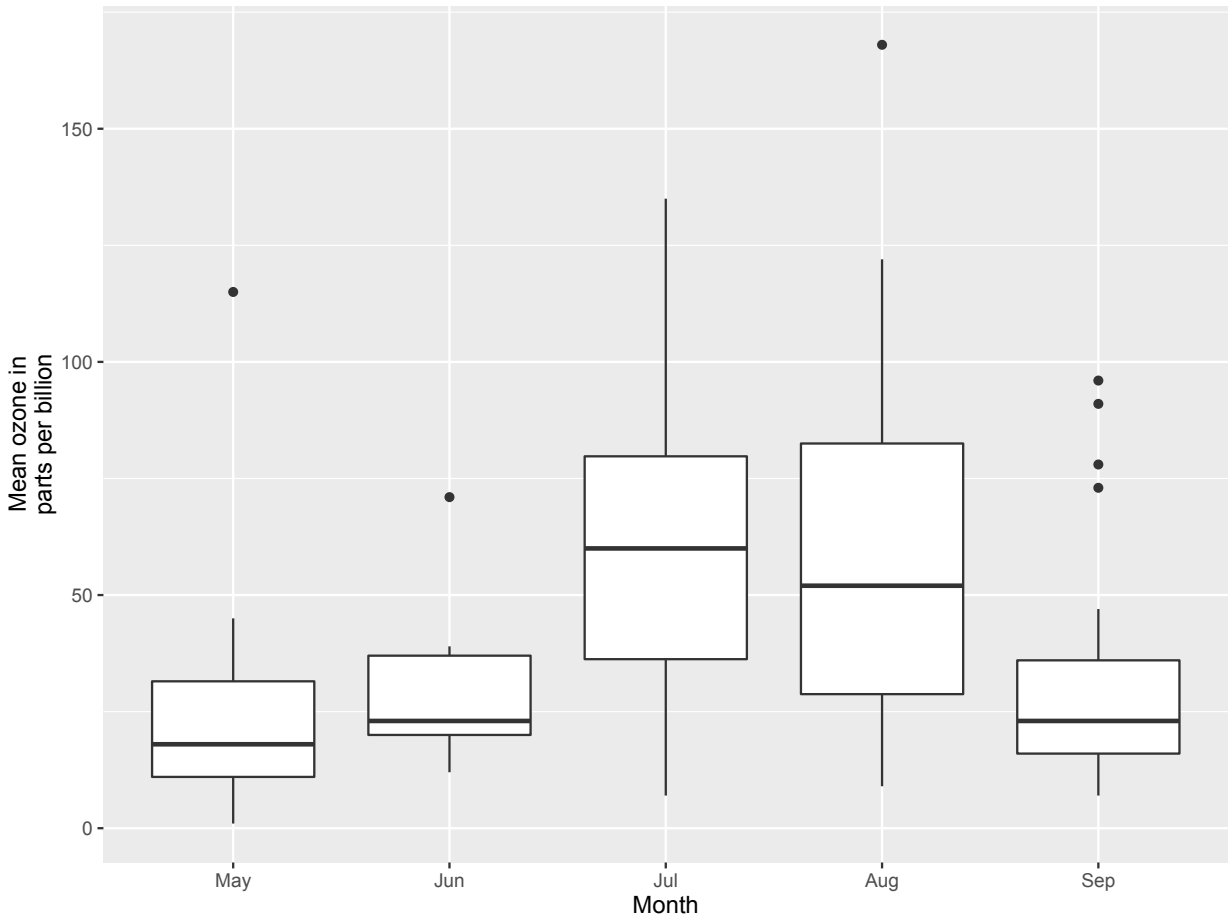
In order to change the axis labels, we have a couple of options. In this case, we have used the `scale_x_discrete` and `scale_y_continuous` options, as these have further customisation options for the axes we will use below. In each, we add the desired name to the `name` argument as a string.

```
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.

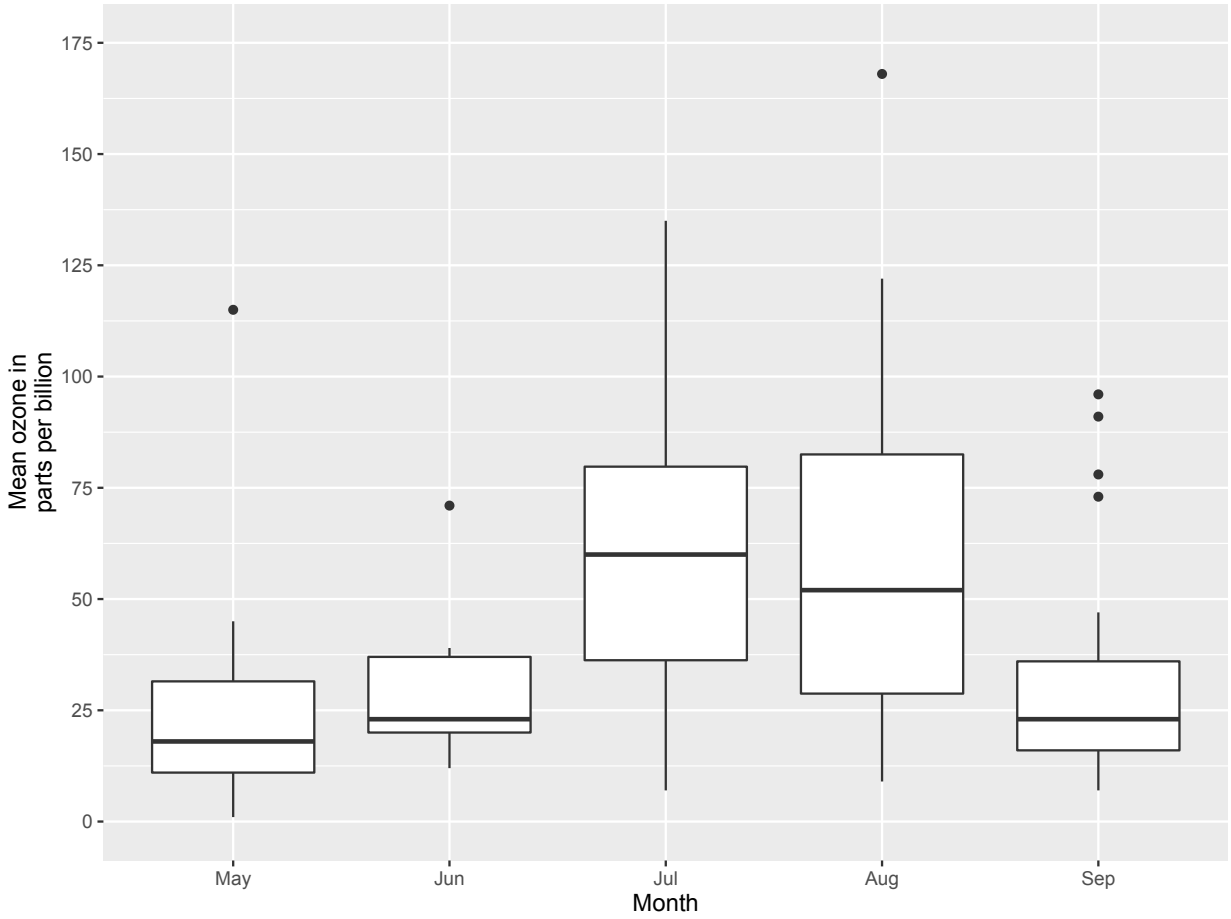
```
p10 <- p10 + scale_y_continuous(name = "Mean ozone in\nparts per billion")
p10
```



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`.

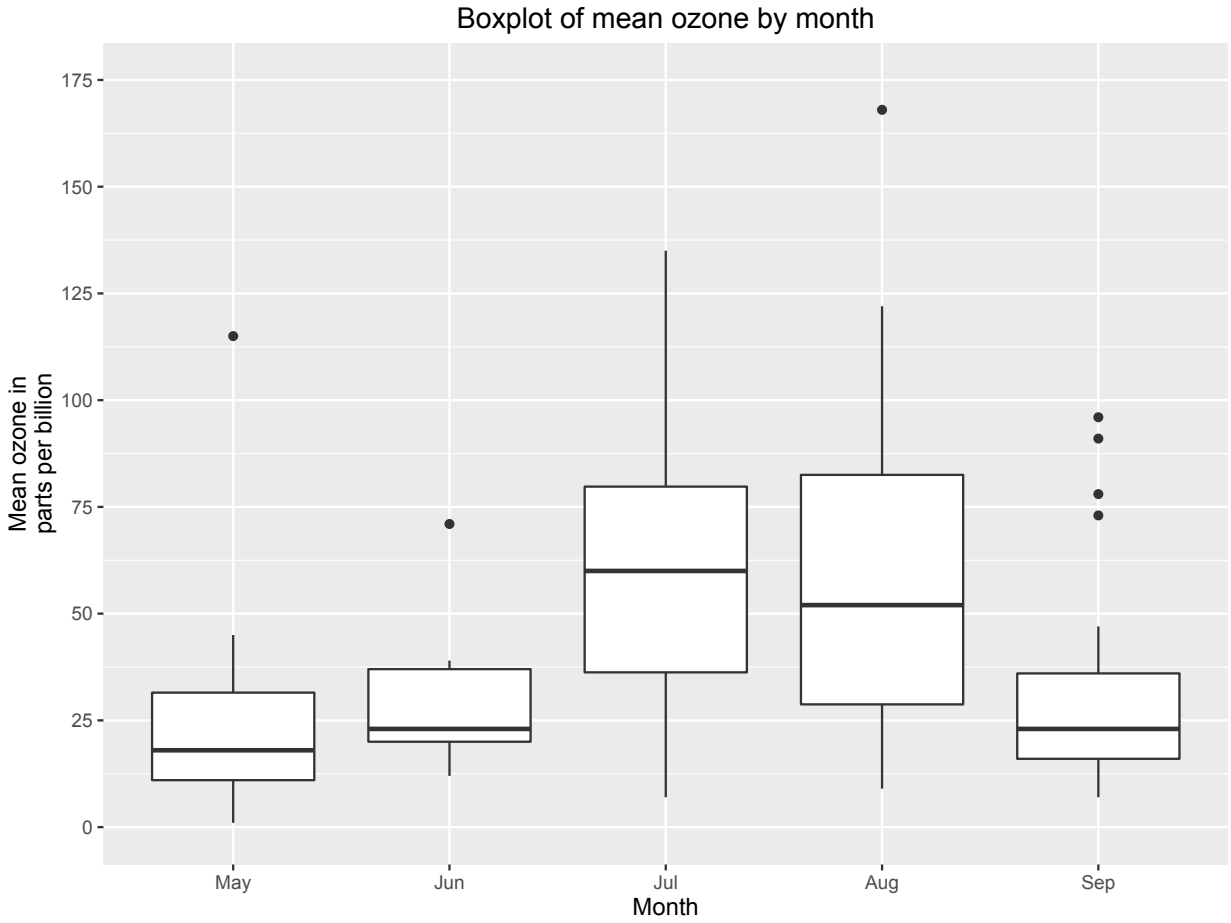
```
p10 <- p10 + scale_y_continuous(name = "Mean ozone in\nparts per billion",
                                breaks = seq(0, 175, 25),
                                limits=c(0, 175))
p10
```



Adding a title

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

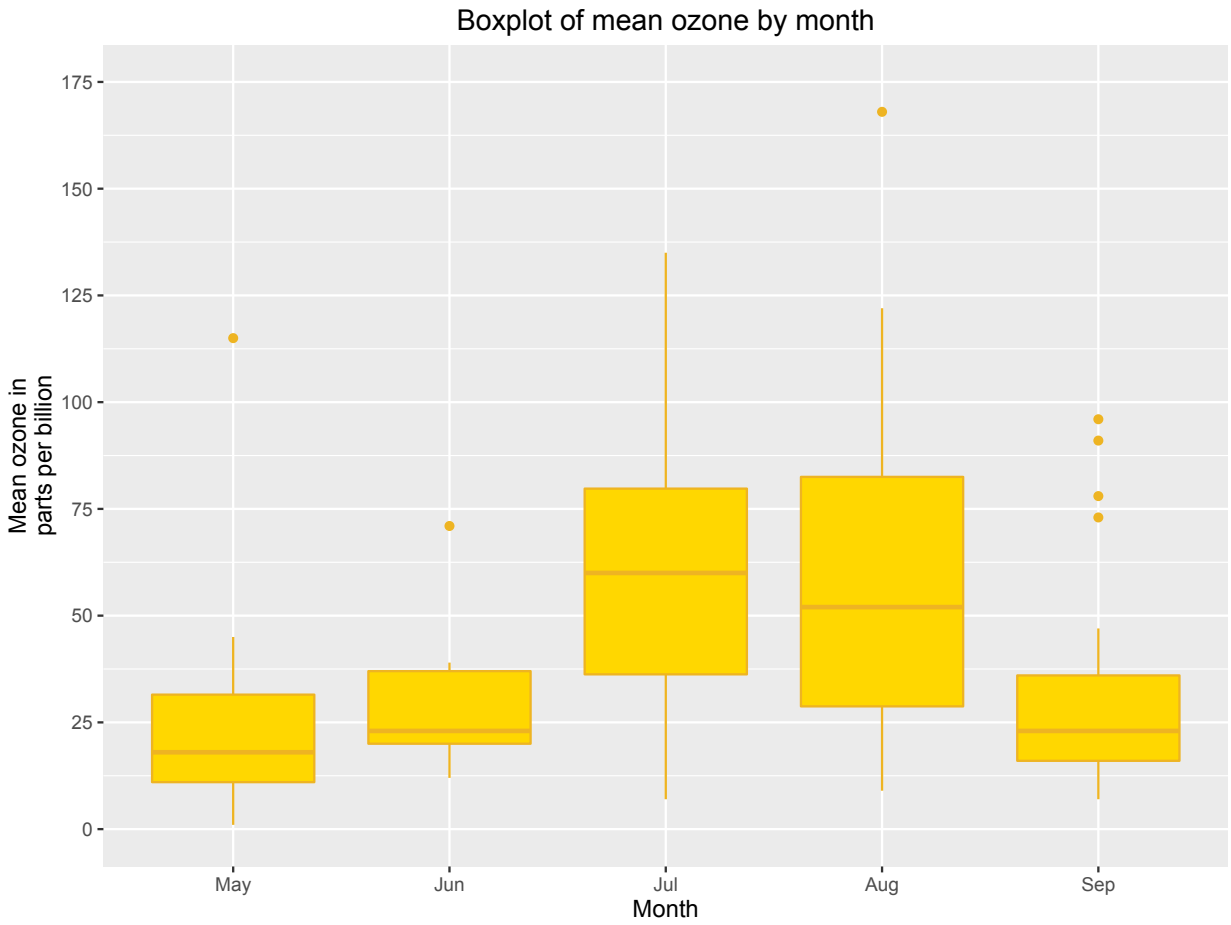
```
p10 <- p10 + ggtitle("Boxplot of mean ozone by month")
p10
```



Changing the colour of the boxes

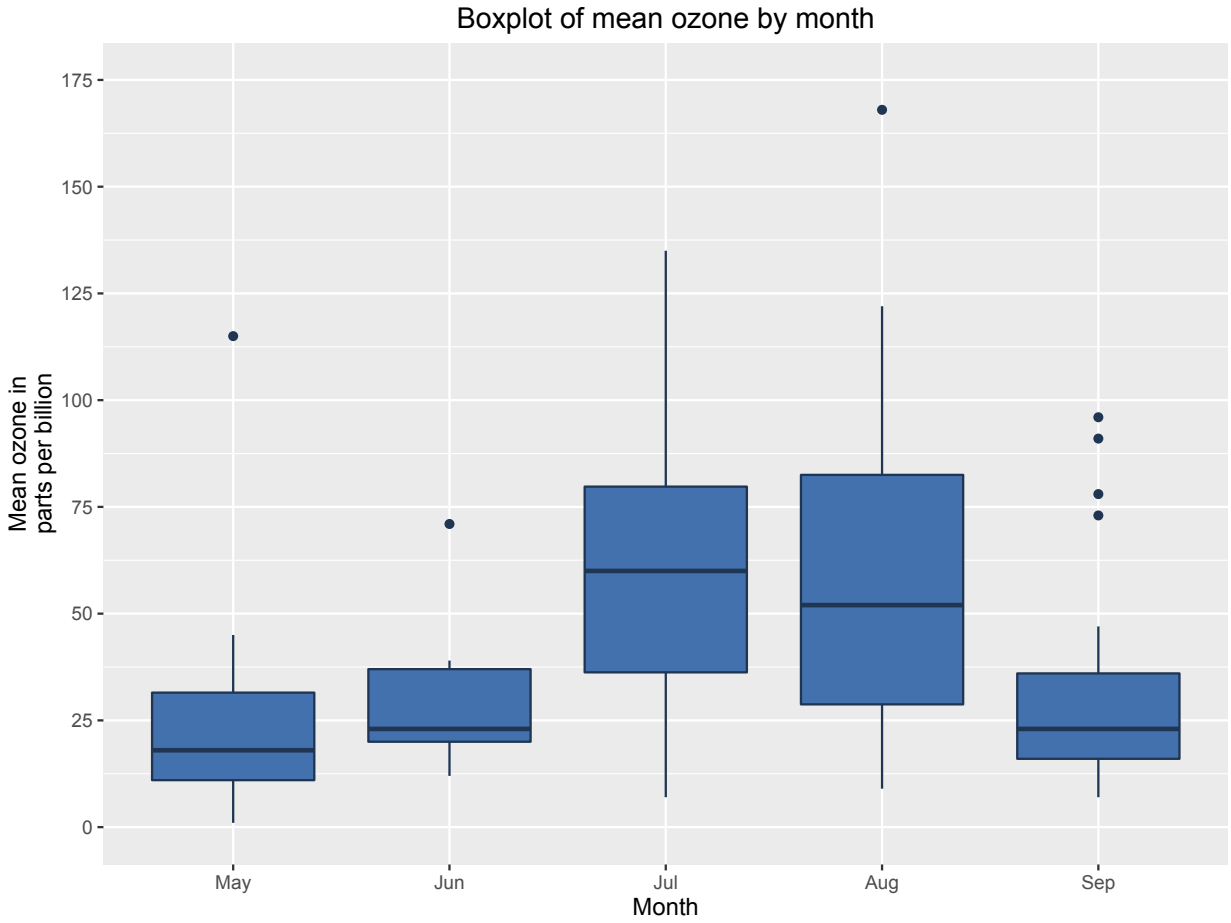
To change the line and fill colours of the box plot, we add a valid colour to the `colour` and `fill` arguments in `geom_boxplot()` (note that we assigned these colours to variables outside of the plot to make it easier to change them). A list of valid colours is here.

```
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
```



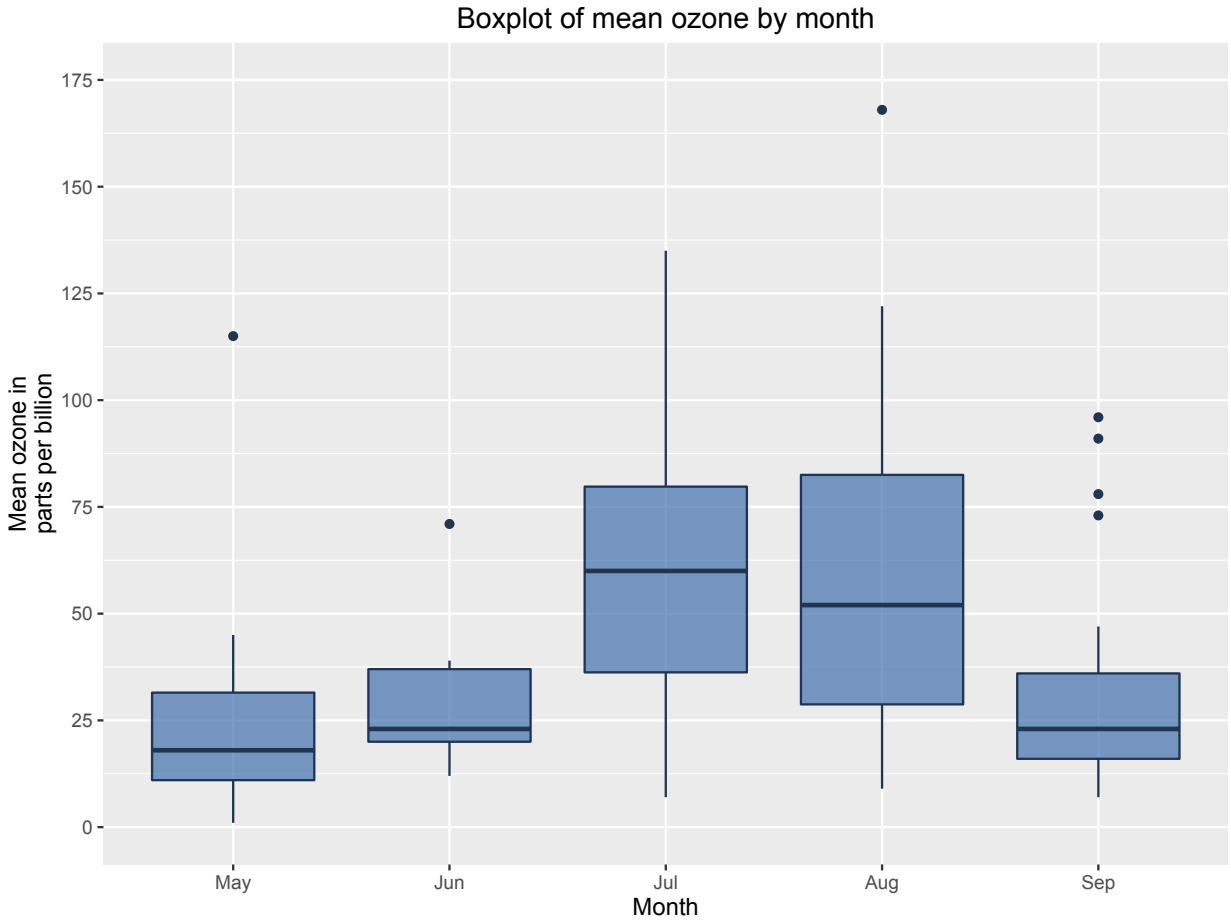
If you want to go beyond the options in the list above, you can also specify exact HEX colours by including them as a string preceded by a hash, e.g., "#FFFFFF". Below, we have called two shades of blue for the fill and lines using their HEX codes.

```
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
```



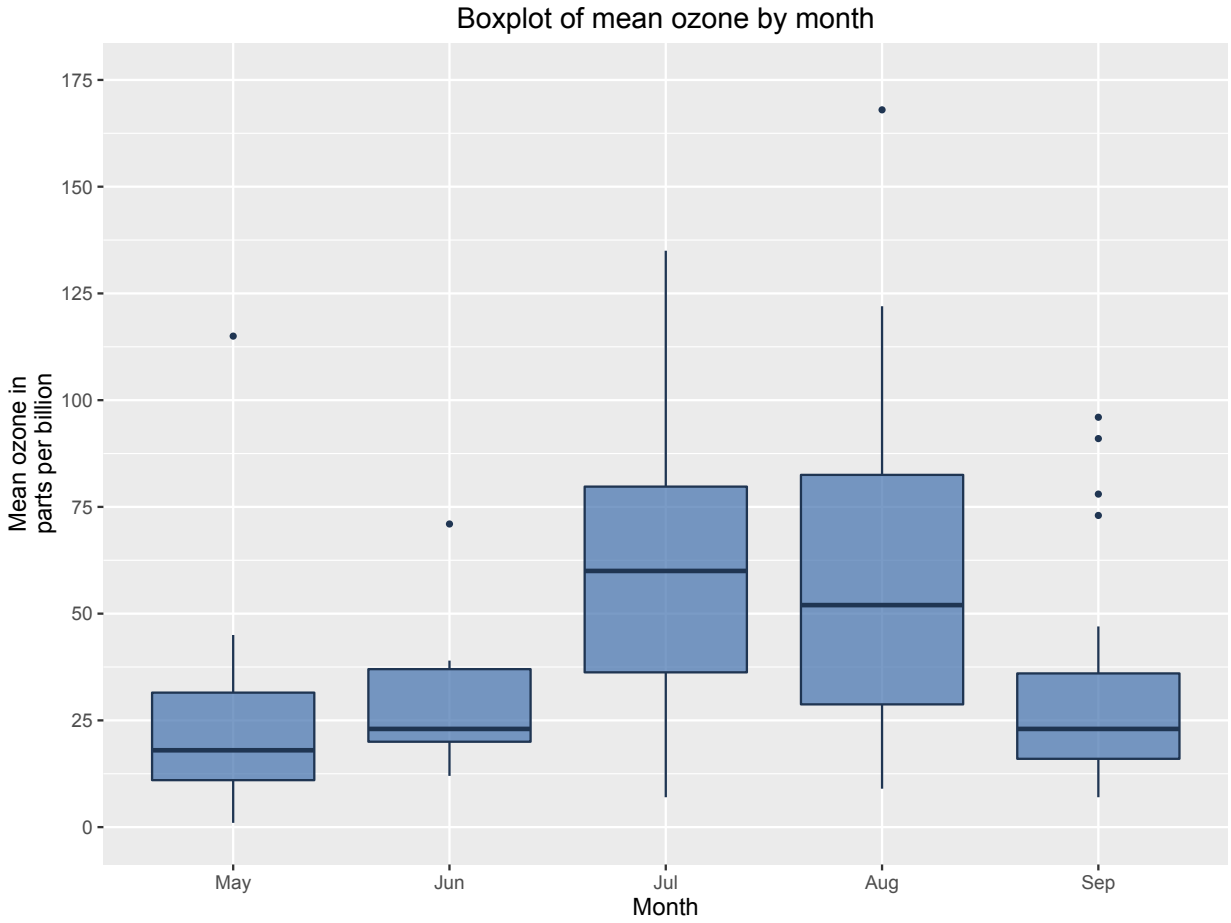
You can also specify the degree of transparency in the box fill area using the argument `alpha` in `geom_boxplot`. This ranges from 0 to 1.

```
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"`).

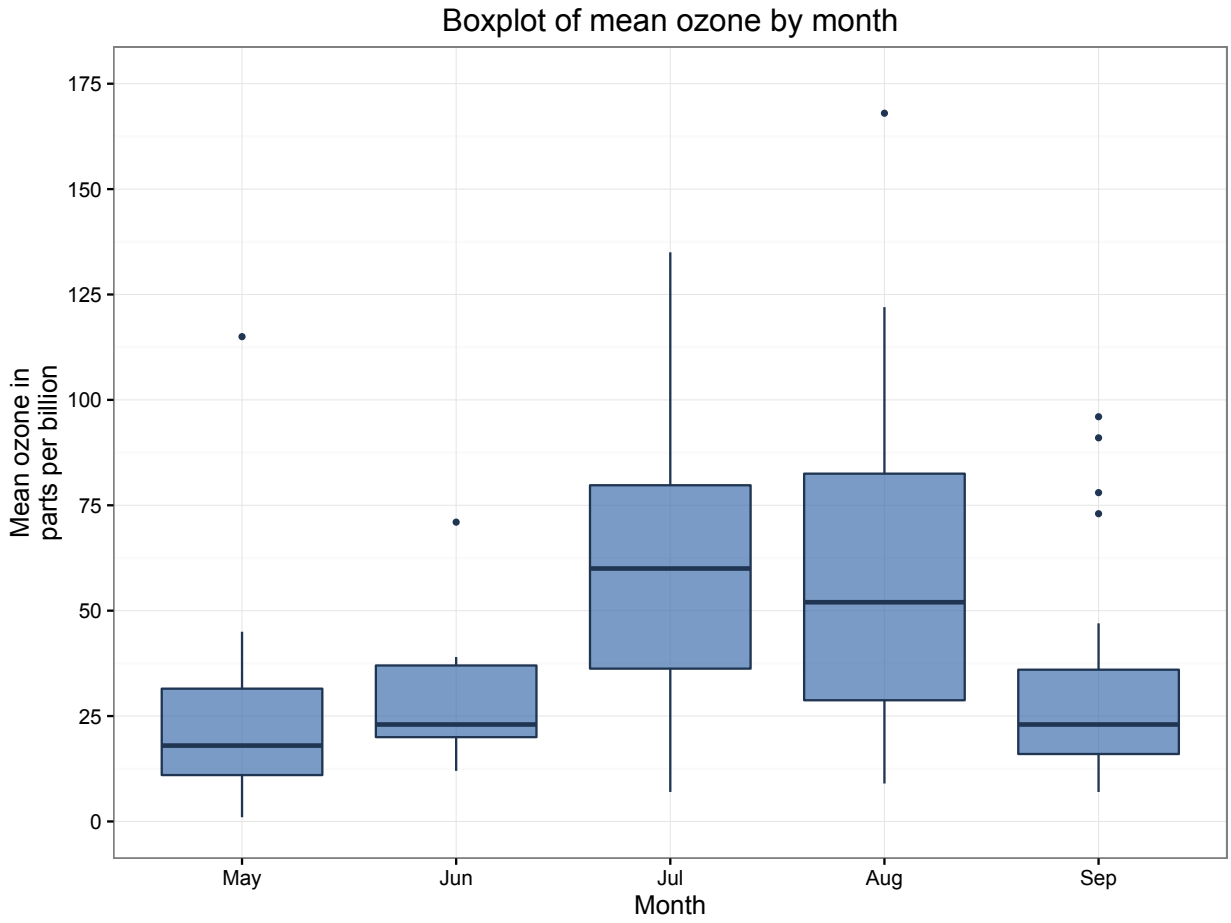
```
p10 <- ggplot(airquality, aes(x = Month, y = Ozone)) +
        geom_boxplot(fill = fill, colour = line, alpha = 0.7,
                     outlier.colour = "#1F3552", outlier.shape = 20) +
        scale_y_continuous(name = "Mean ozone in\nparts per billion",
                           breaks = seq(0,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



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()`. 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
```



Creating an XKCD style chart

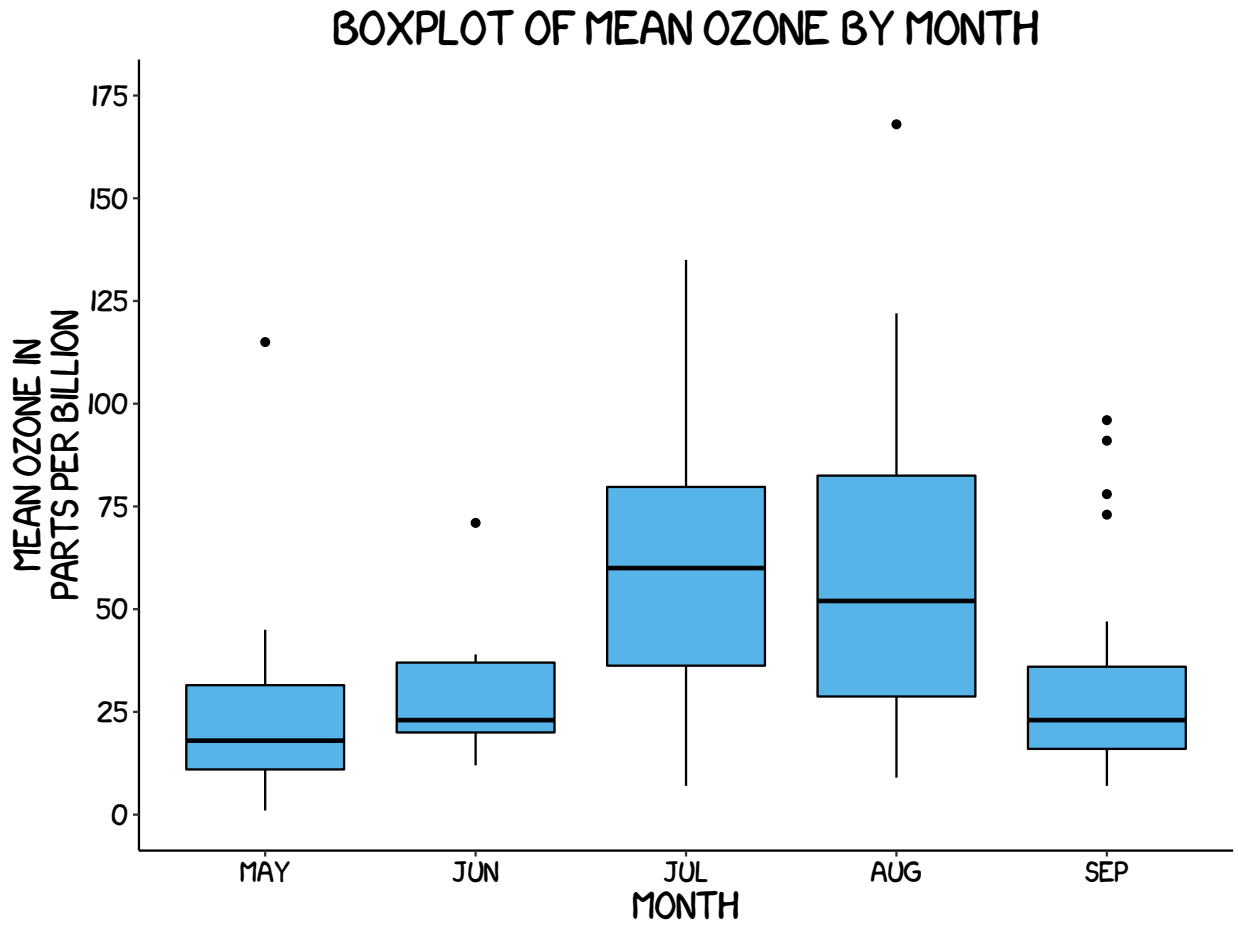
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
              dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

```
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 = 0.5, colour = "black"),
              axis.line.y = element_line(size = 0.5, colour = "black"),
              axis.line = element_line(size=1, colour = "black"),
              panel.grid.major = element_blank(),
              panel.grid.minor = element_blank(),
              panel.border = element_blank(),
              panel.background = element_blank(),
              plot.title=element_text(size = 20, family="xkcd-Regular"),
              text=element_text(size = 16, family="xkcd-Regular"),
              axis.text.x=element_text(colour="black", size = 12),
              axis.text.y=element_text(colour="black", size = 12))
p10
```

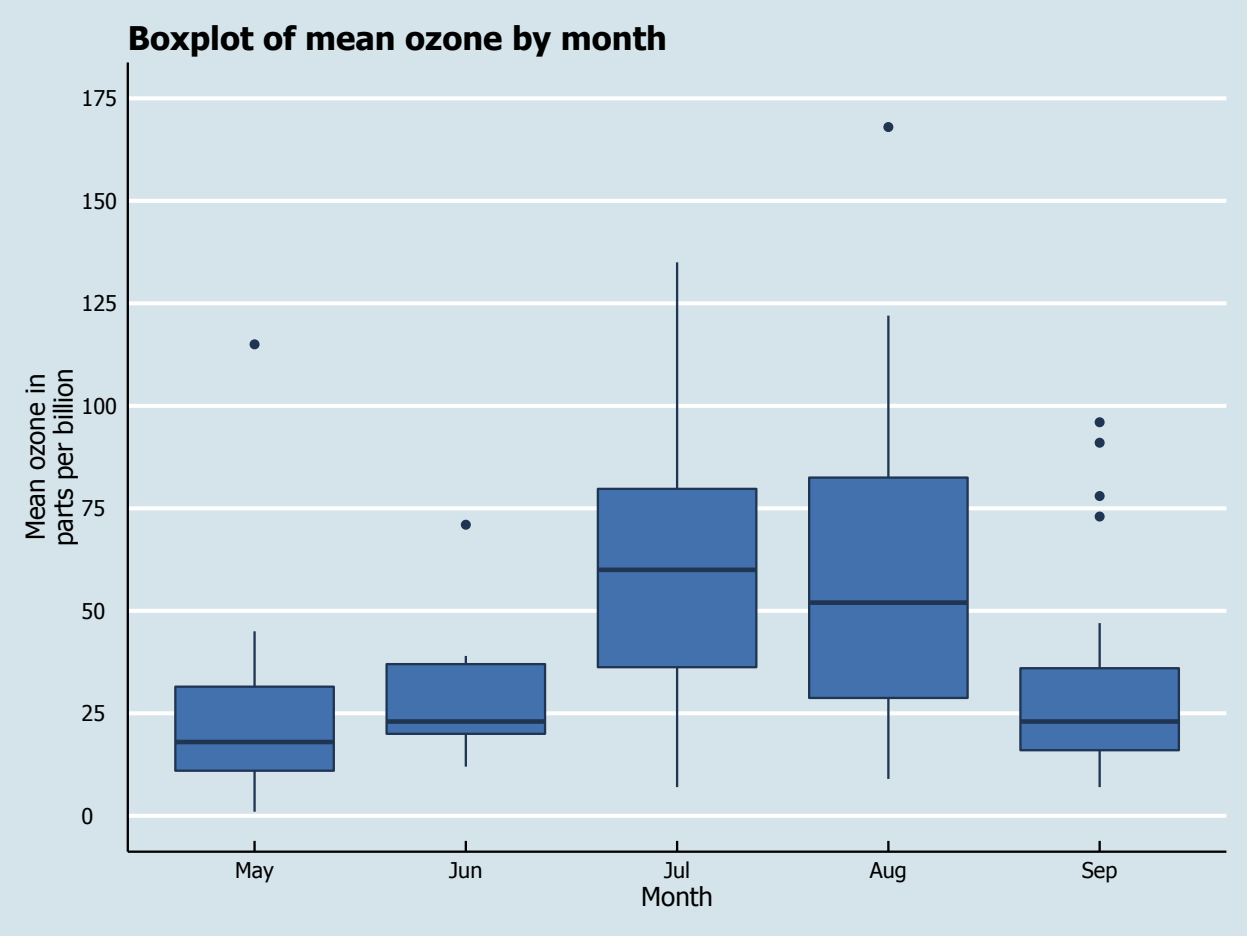


Using 'The Economist' theme

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.

```
library(ggthemes)
library(grid)

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") +
        theme_economist() +
        theme(axis.line.x = element_line(size = 0.5, colour = "black"),
              axis.line.y = element_line(size = 0.5, colour = "black"),
              legend.position = "bottom", legend.direction = "horizontal",
              legend.box = "horizontal",
              legend.key.size = unit(1, "cm"),
              plot.title = element_text(family="Tahoma"),
              text = element_text(family = "Tahoma"),
              axis.title = element_text(size = 12),
              legend.text = element_text(size = 9),
              legend.title=element_text(face = "bold", size = 9))
p10
```

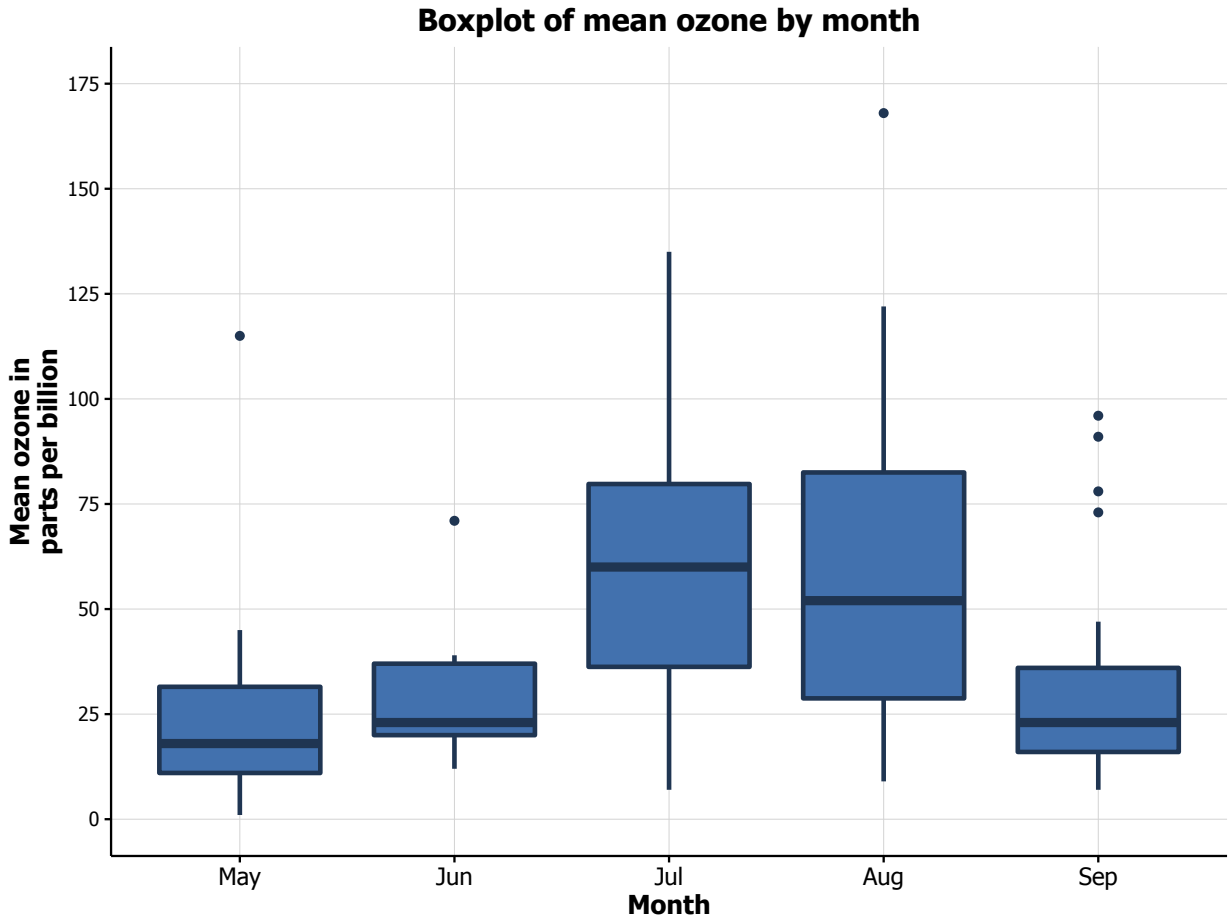


Creating your own theme

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.

```
library(grid)

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(axis.line.x = element_line(size = 0.5, colour = "black"),
              axis.line.y = element_line(size = 0.5, colour = "black"),
              panel.grid.major = element_line(colour = "#d3d3d3"),
              panel.grid.minor = element_blank(),
```

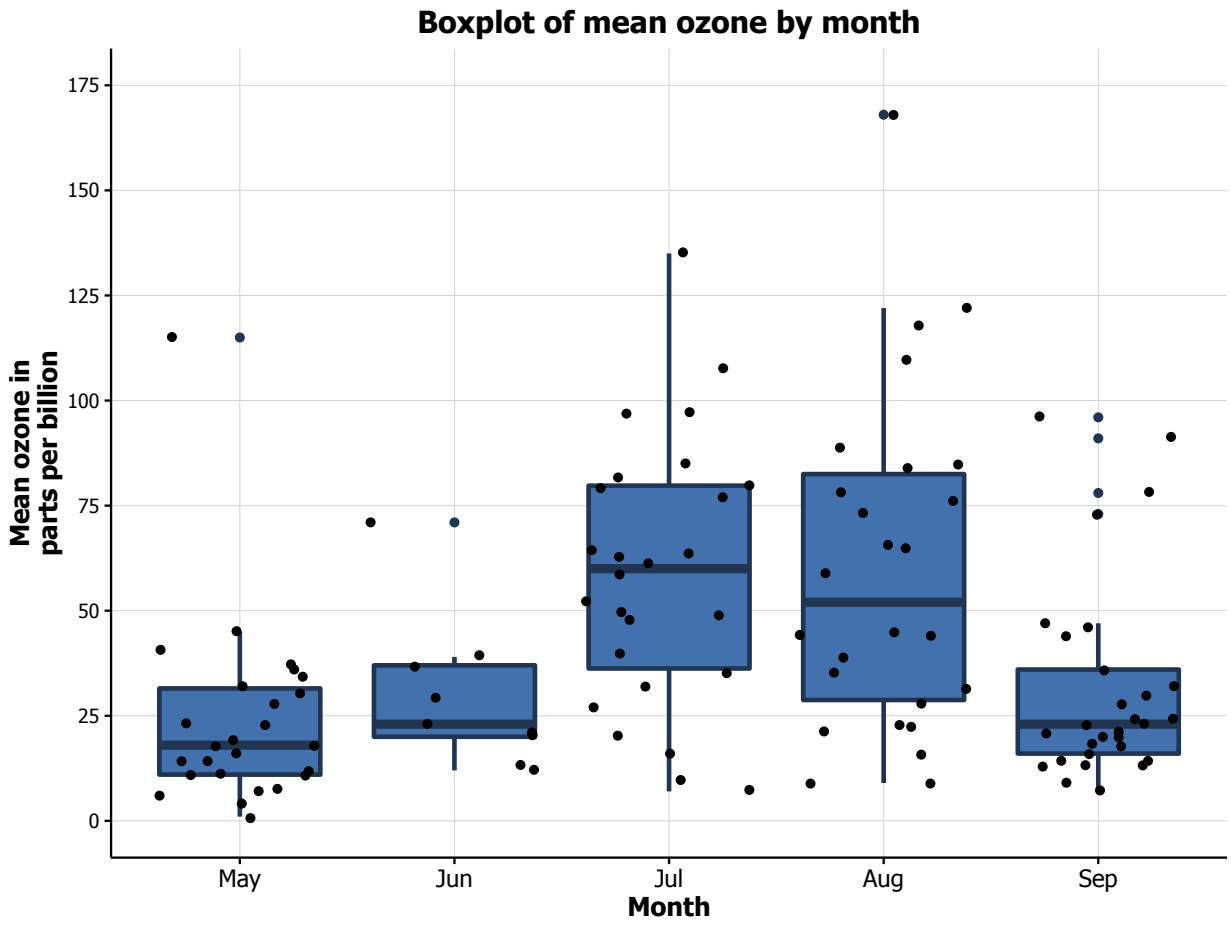
```
            panel.border = element_blank(), panel.background = element_blank(),
            plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
            text=element_text(family = "Tahoma"),
            axis.title = element_text(face="bold"),
            axis.text.x = element_text(colour="black", size = 11),
            axis.text.y = element_text(colour="black", size = 9))
p10
```



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
```

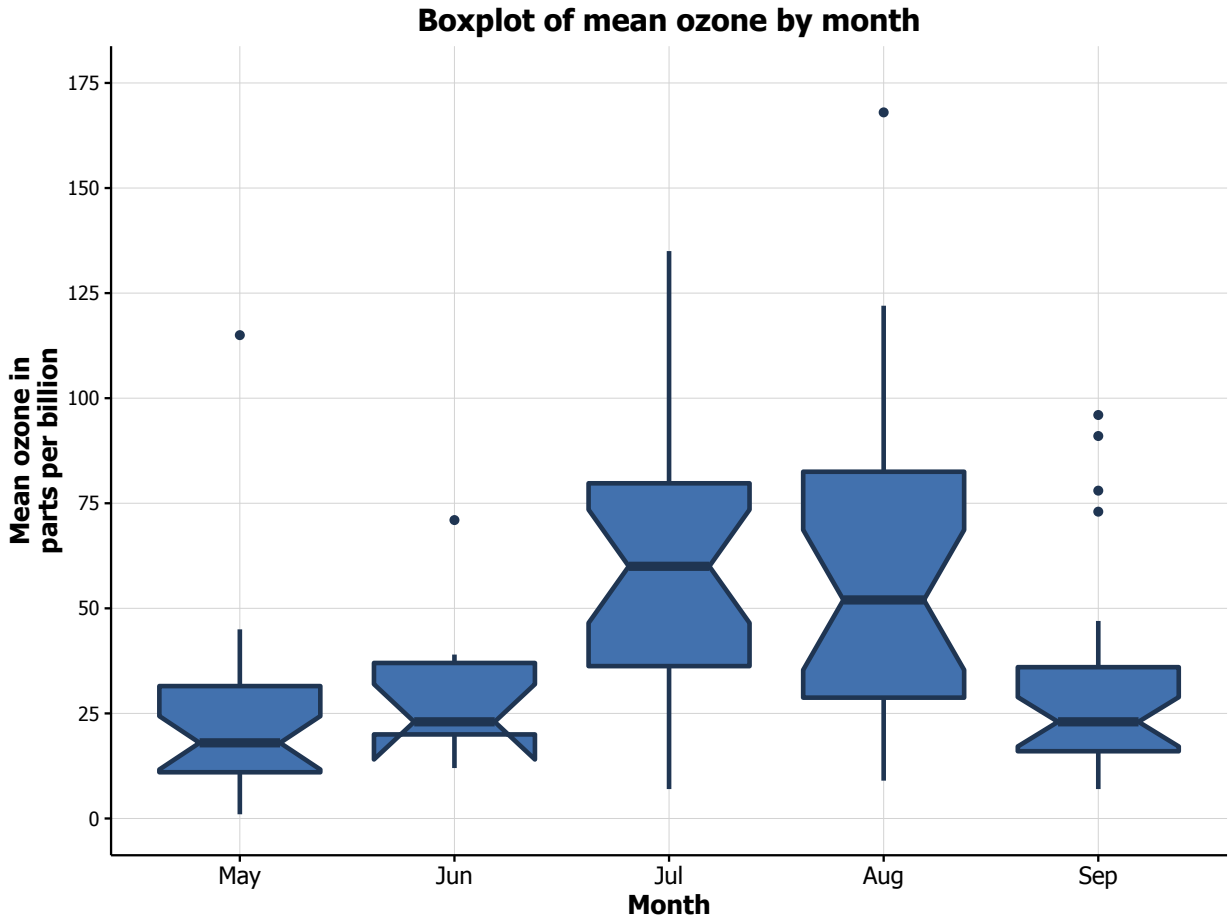


We can see that June has a pretty small sample, indicating that information based on this group may not be very reliable.

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(axis.line.x = element_line(size = 0.5, colour = "black"),
              axis.line.y = element_line(size = 0.5, colour = "black"),
              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
```



Grouping by another variable

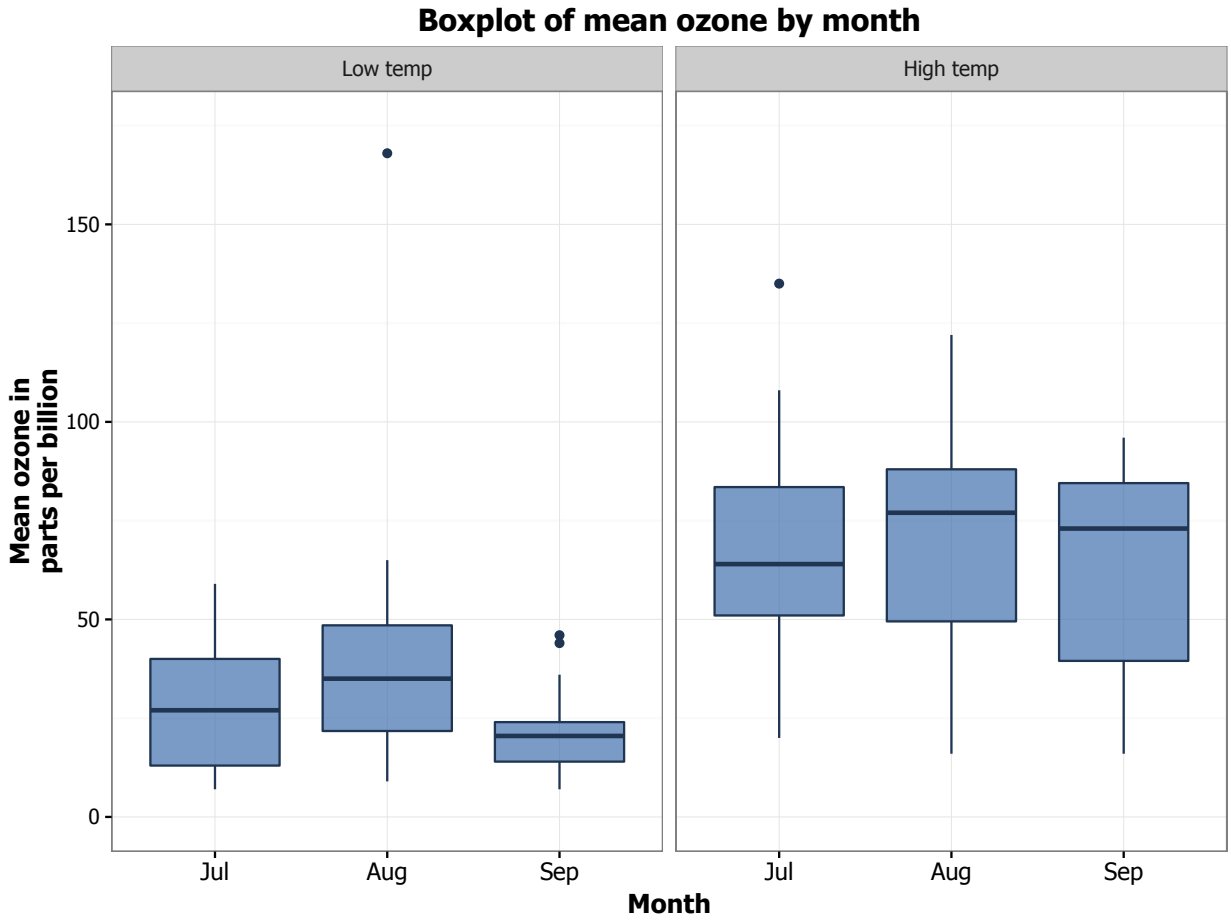
You can also easily group box plots by the levels of another variable. There are two options, in separate (panel) plots, or in the same plot.

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)) +
```

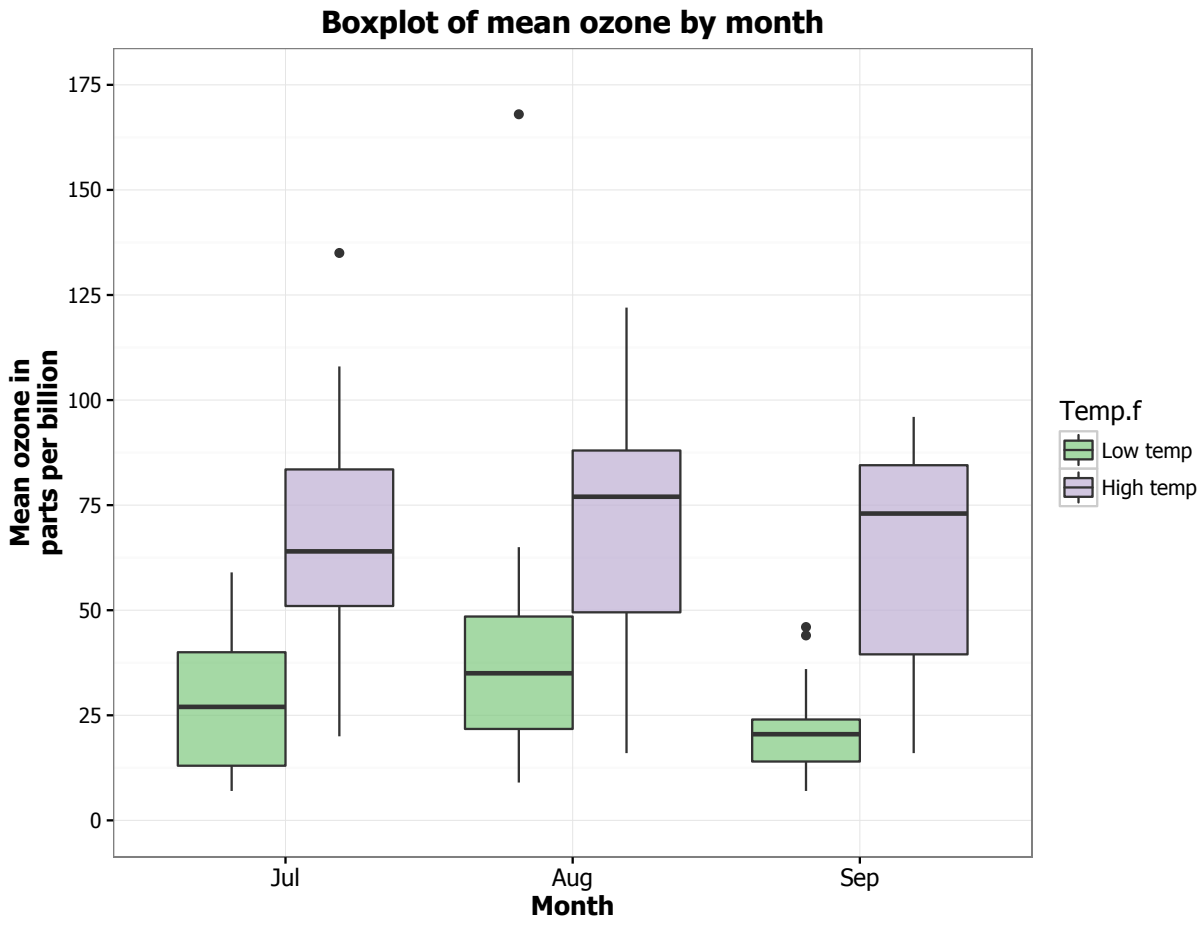
```
        geom_boxplot(fill = fill, colour = line,
                     alpha = 0.7) +
        scale_y_continuous(name = "Mean ozone in\nparts per billion",
                           breaks = seq(0, 175, 50),
                           limits=c(0, 175)) +
        scale_x_discrete(name = "Month") +
        ggtitle("Boxplot of mean ozone by month") +
        theme_bw() +
        theme(plot.title = element_text(size = 14, family = "Tahoma", face = "bold"),
              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
```



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.

```
library(RColorBrewer)
```

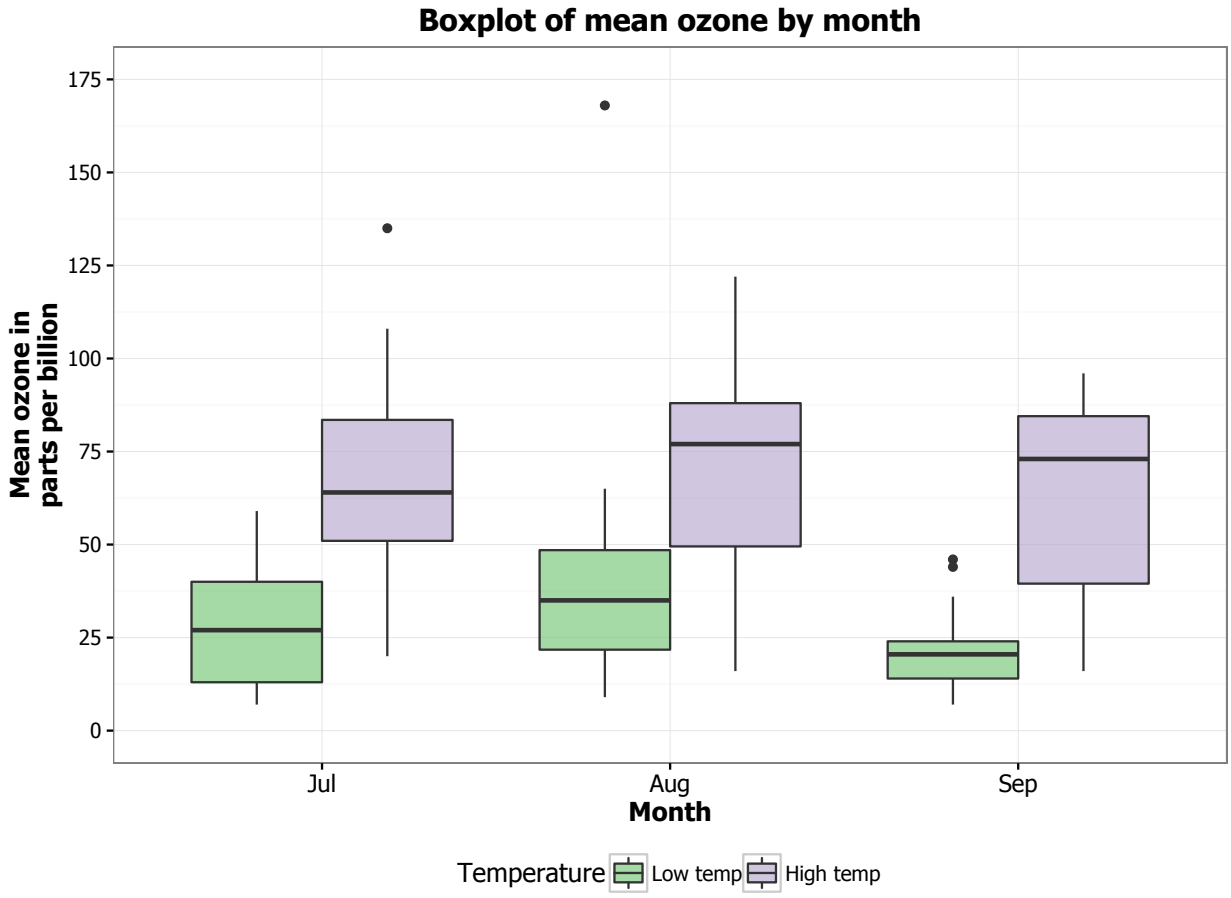
```
p10 <- ggplot(airquality_trimmed, aes(x = Month, y = Ozone, fill = Temp.f)) +
        geom_boxplot(alpha=0.7) +
        scale_y_continuous(name = "Mean ozone in\nparts per billion",
                           breaks = seq(0, 175, 25),
                           limits=c(0, 175)) +
        scale_x_discrete(name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



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.

```
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"),
              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
```



Linear regression

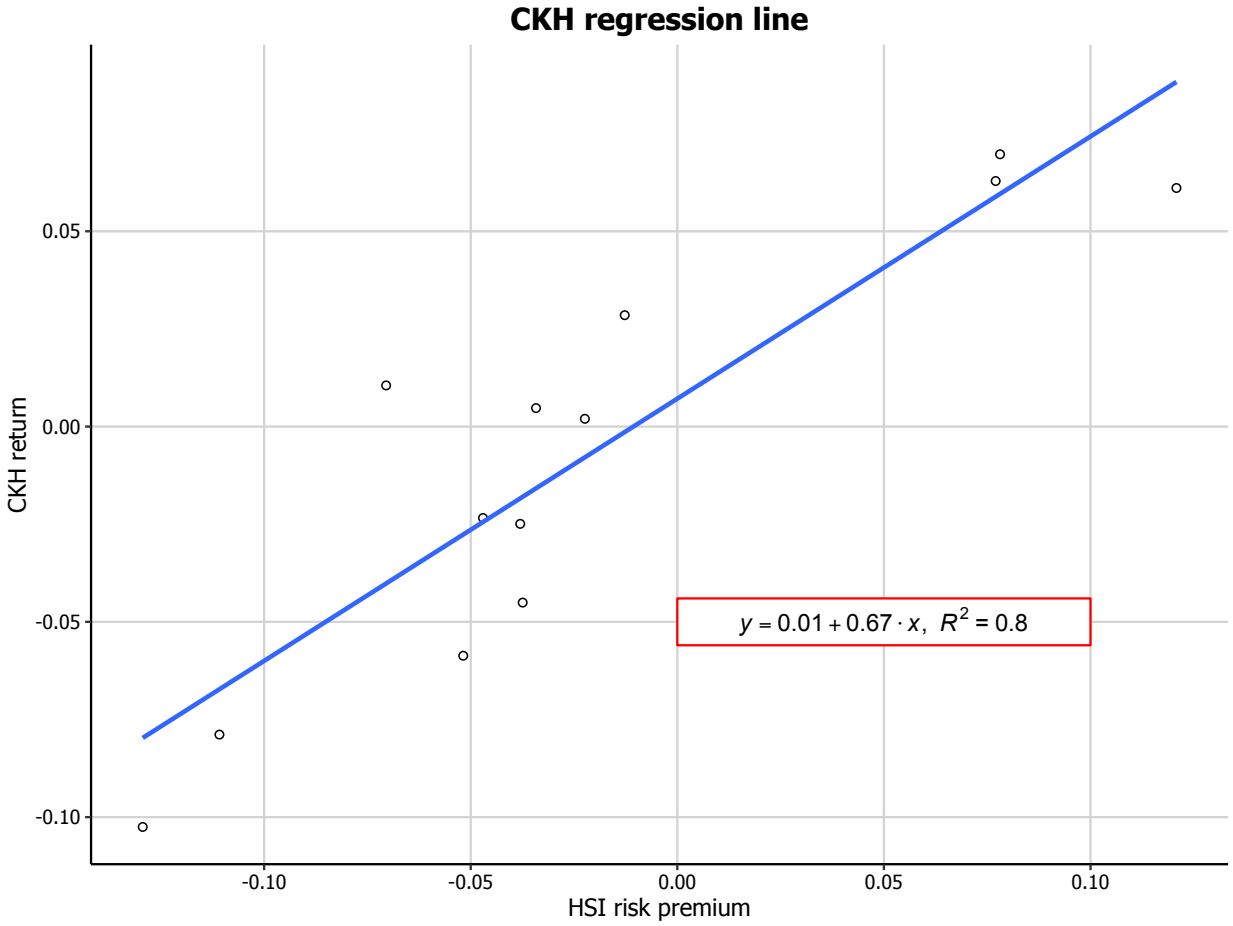
This is the eleventh tutorial in a series on using `ggplot2` I am creating with Mauricio Vargas Sepúlveda. In this tutorial we will demonstrate some of the many options the `ggplot2` package.

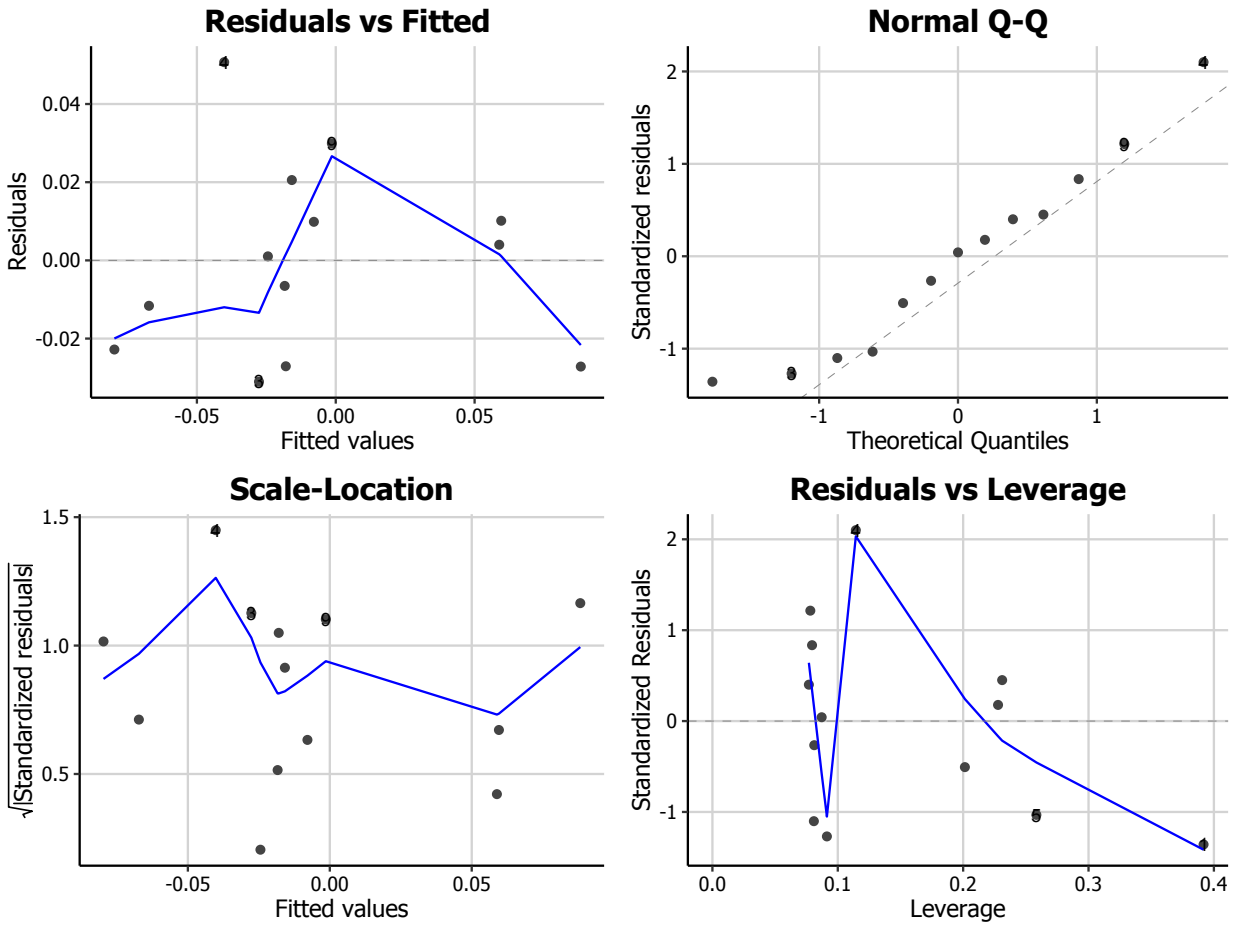
This post will be much more than showing you how to create regression plots. Here we are extracting, cleaning and processing financial data from Quandl.

Before going ahead, we strongly suggest to create a Quandl account in order to obtain an API key that allows you to download data without restrictions. It is even possible to log into Quandl using Github or Linkedin accounts. Quandl's website has complete instruction 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 asset is exactly as risky as the market portfolio). More on the CAPM model can be read here, but in this tutorial we will focus on plotting.

In this tutorial, 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 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(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
```

Trend line plot

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 $\beta_i = \frac{\sigma_{i,m}}{\sigma_m^2}$.

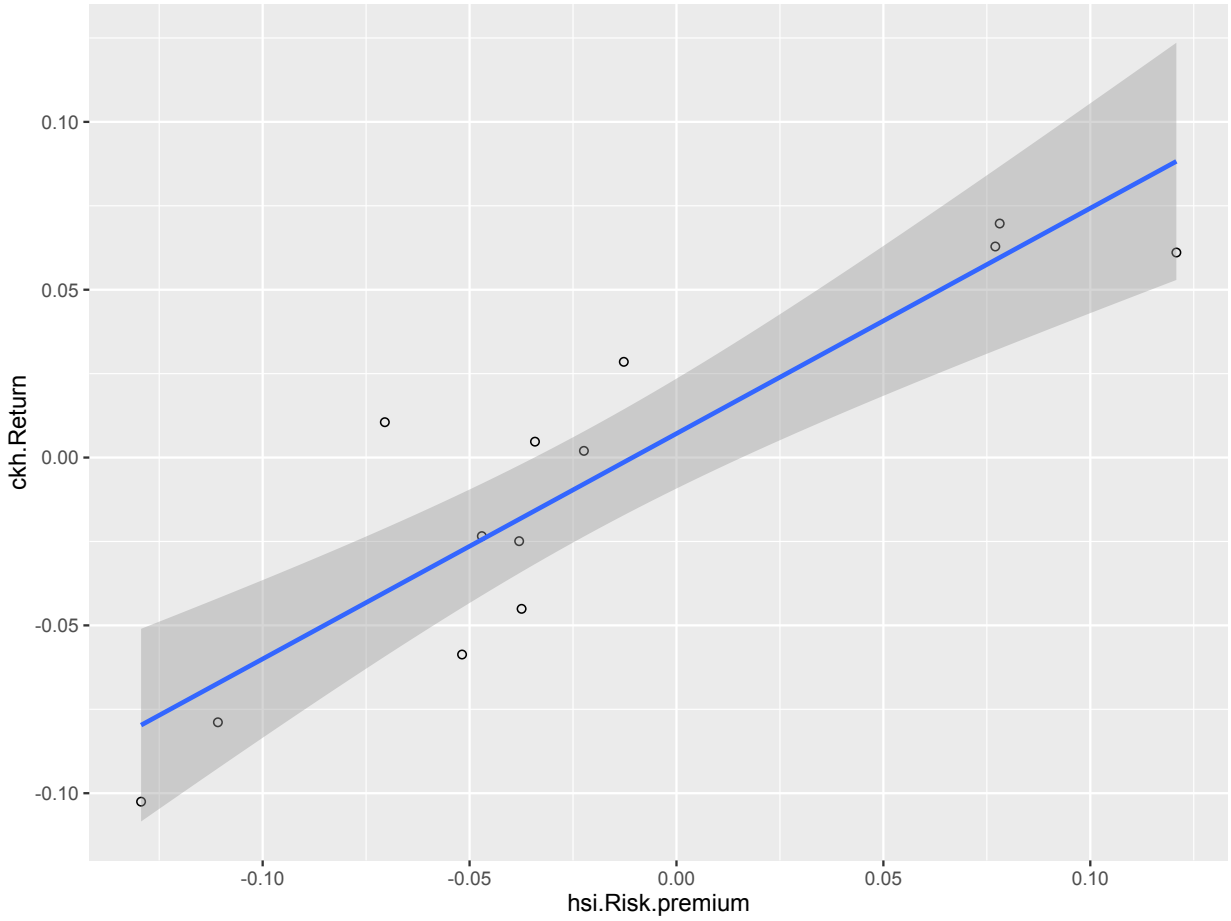
```
hsi.ckh.returns$hsi.Risk.free <- usa.risk.free + hsi.risk.premium
hsi.ckh.returns$hsi.Risk.premium <- hsi.ckh.returns$hsi.Return - hsi.ckh.returns$hsi.Risk.free
hsi.Return.vector <- as.vector(hsi.ckh.returns$hsi.Return)
ckh.Return.vector <- as.vector(hsi.ckh.returns$ckh.Return)
cov.hsi.ckh <- cov(ckh.Return.vector,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

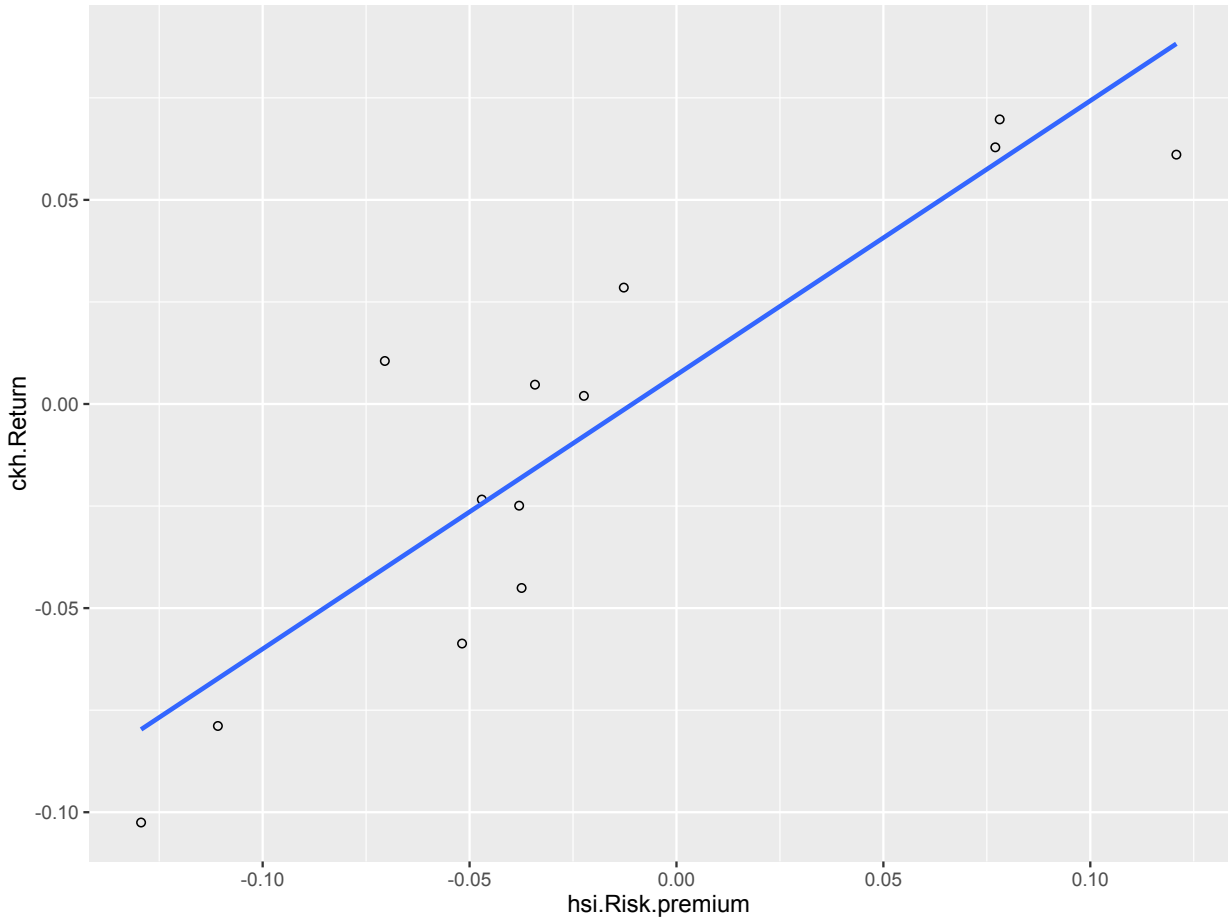
Up to this point we have all we need to plot regressions. We will start with a basic regression plot.

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



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

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

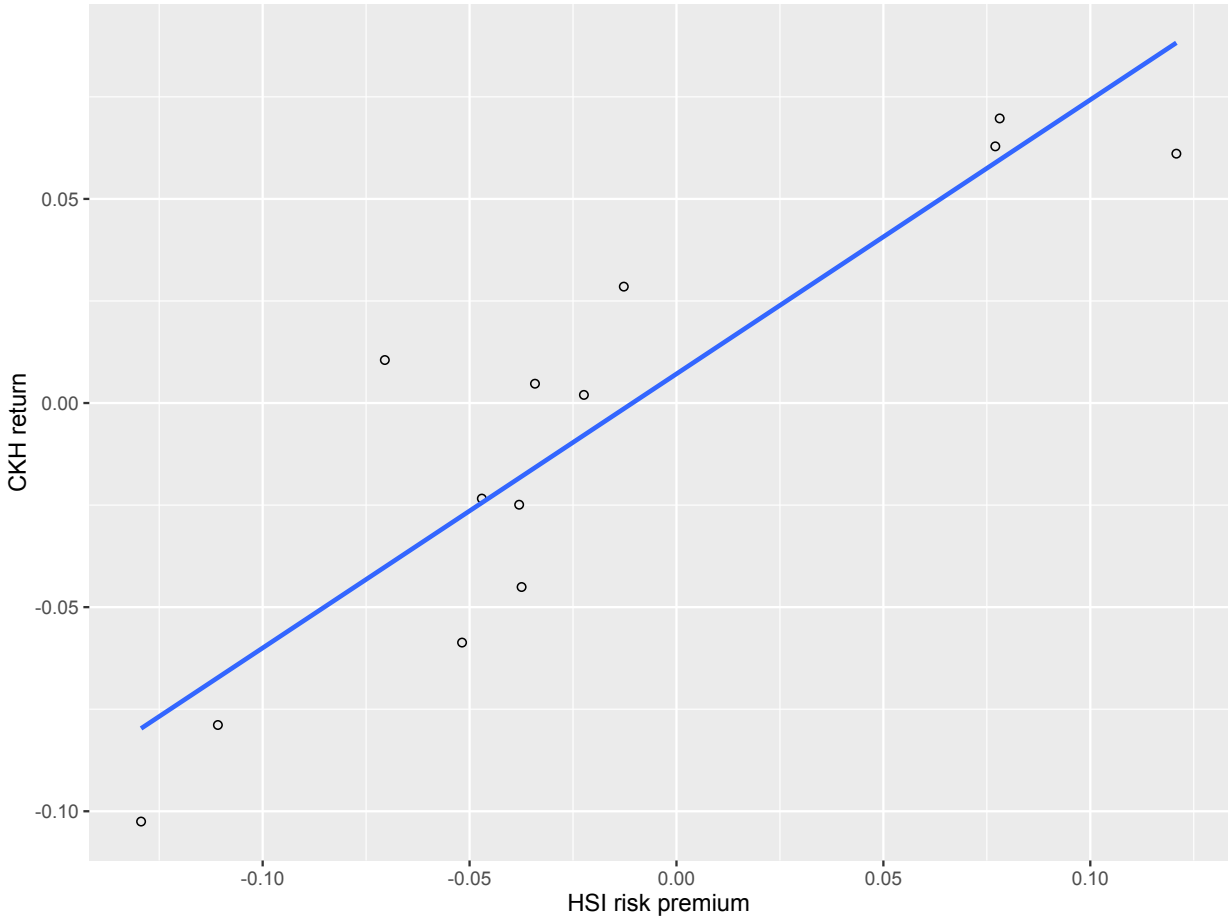


Before continuing it is a good idea to fix the axis labels and add a title.

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.

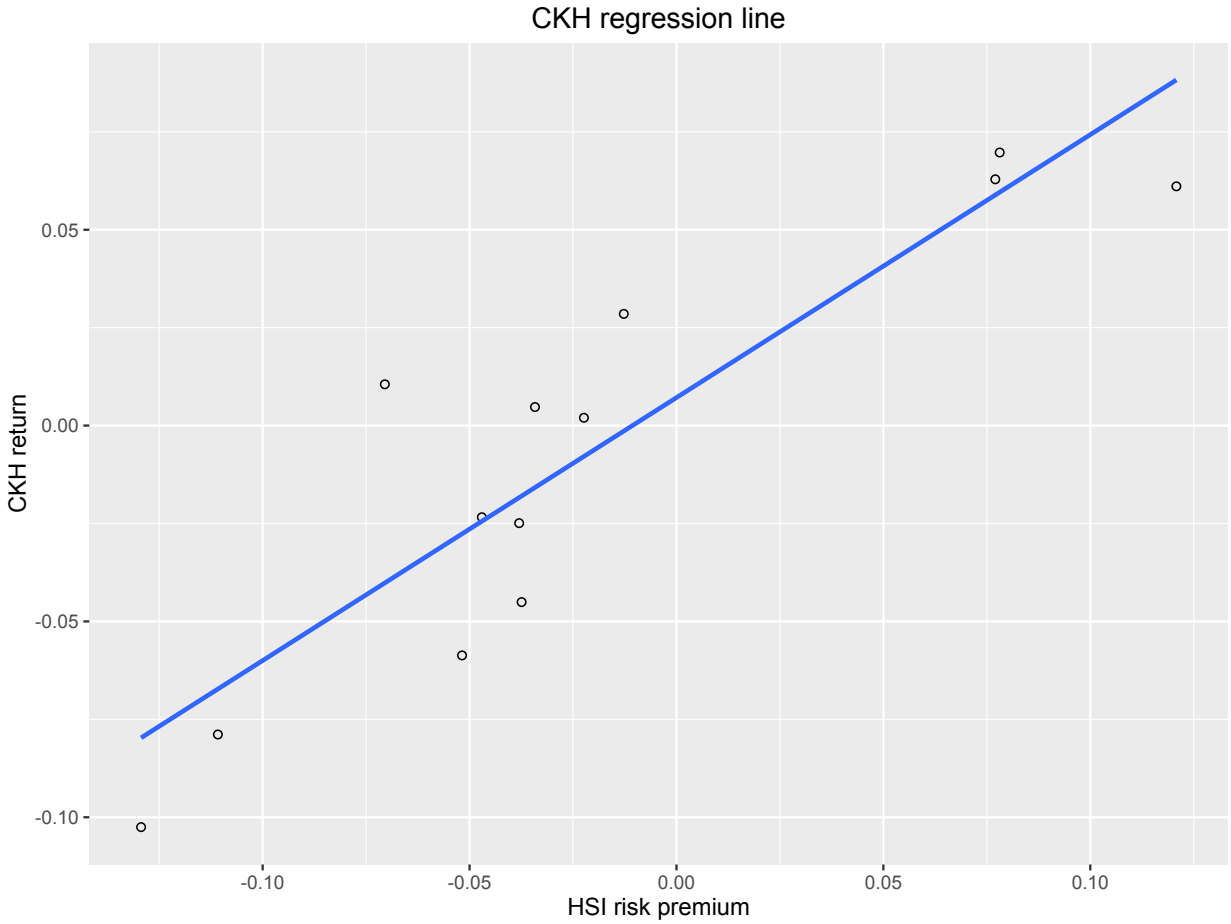
```
p11 <- p11 + scale_x_continuous(name = "HSI risk premium") +
      scale_y_continuous(name = "CKH return")
p11
```



Adding a title

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

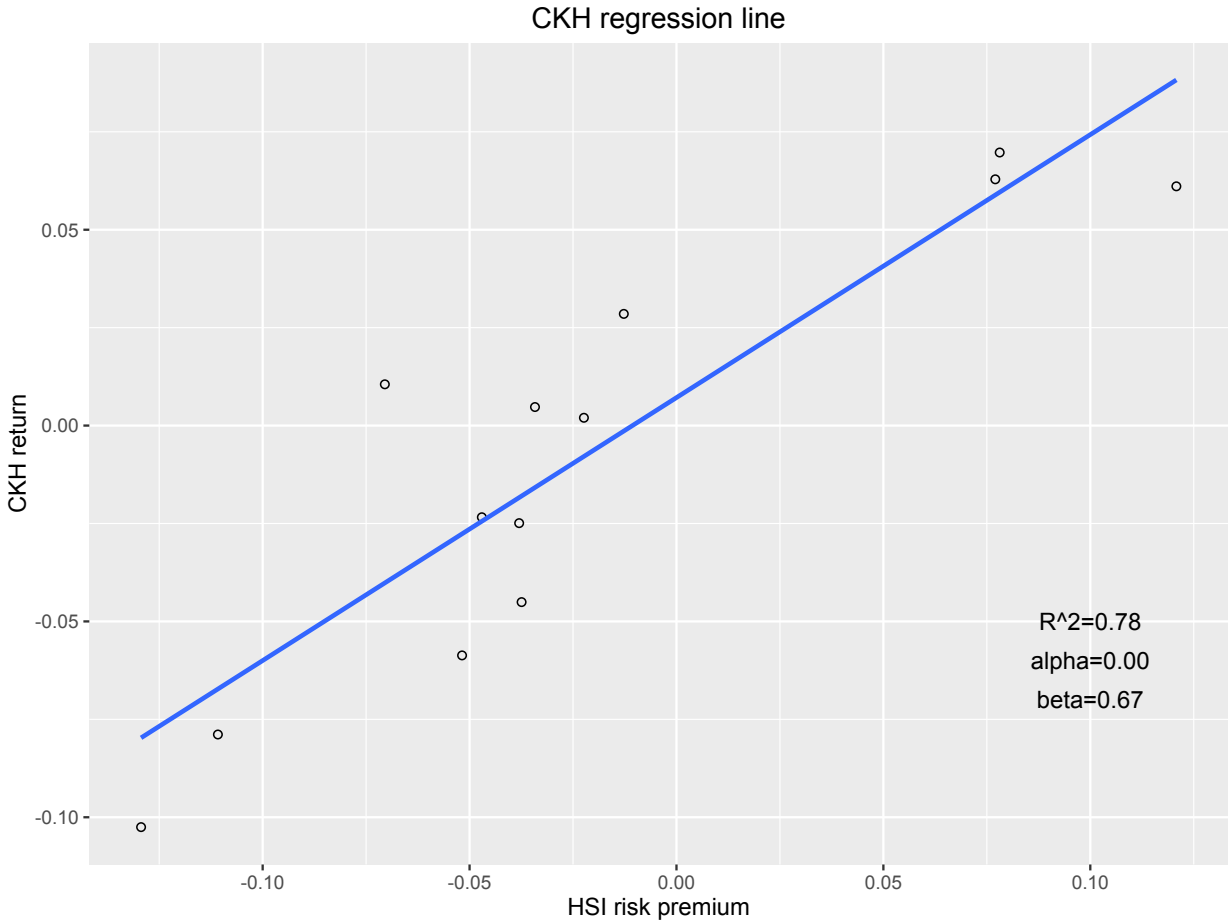
```
p11 <- p11 + ggtitle("CKH regression line")
p11
```



Including regression coefficients

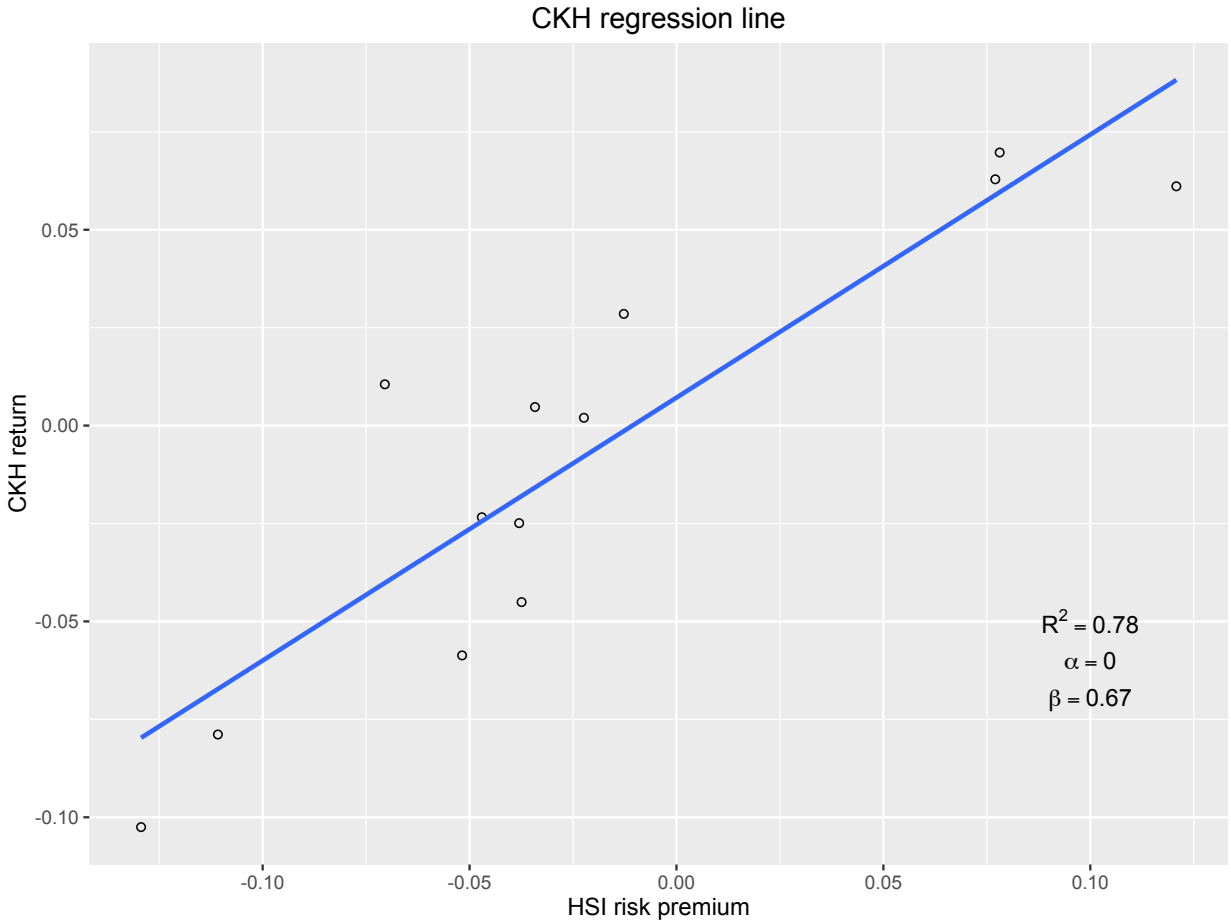
We can also include more information about the regression line itself. It would be interesting to show R^2 and regression coefficients within the plot.

```
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
```



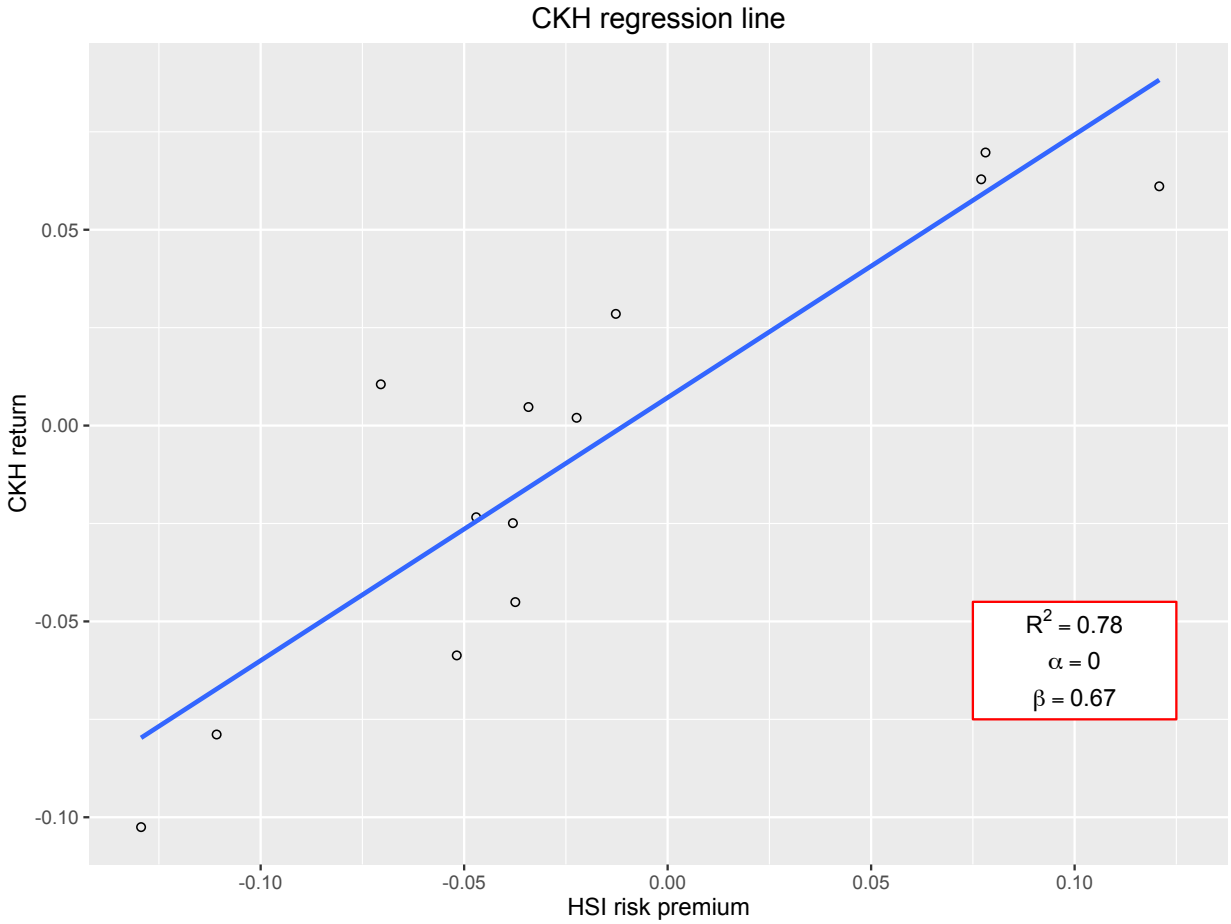
Another option would be to add greek letters and exponents.

```
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
```



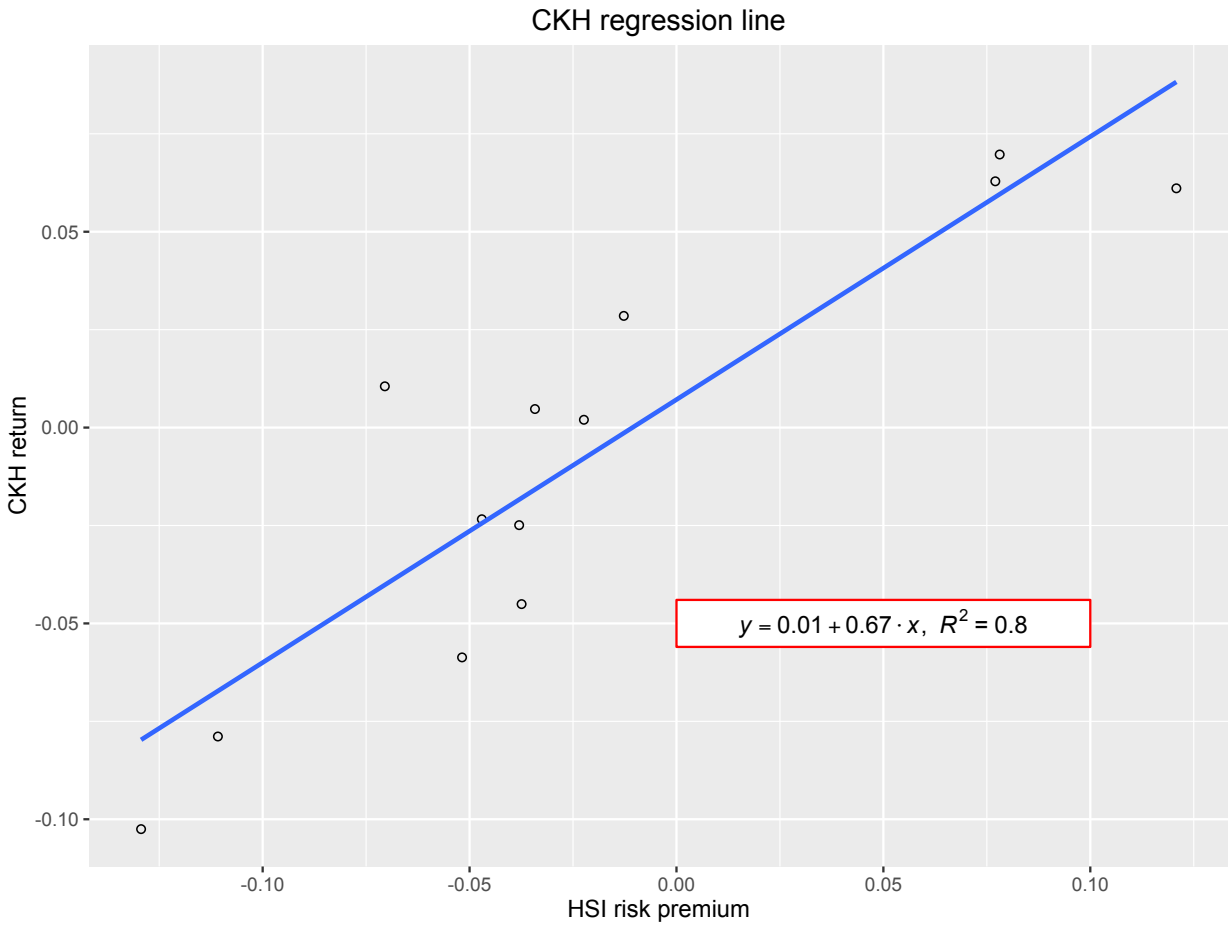
To make the coefficients more clear we will add some elements to increase visibility.

```
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="
      annotate("text", x=0.1, y=-0.05, label = "R^2 == 0.78", parse=T) + annotate("text", x=0.1, y=-0.0
      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.

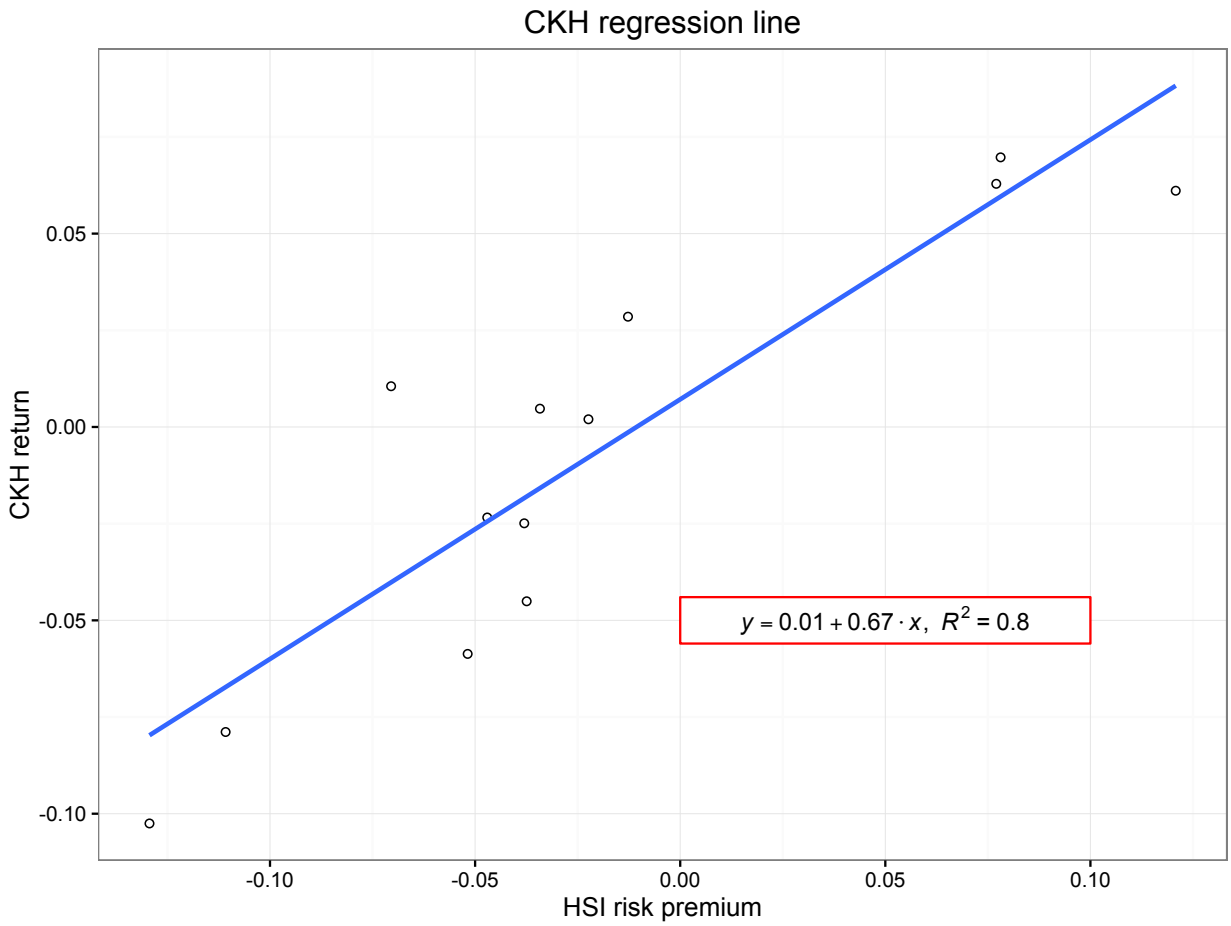
```
equation = function(x) {
  lm_coef <- list(a = round(coef(x)[1], digits = 2),
                  b = round(coef(x)[2], digits = 2),
                  r2 = round(summary(x)$r.squared, digits = 2));
  lm_eq <- substitute(italic(y) == a + b %.% italic(x)*","~~italic(R)^2~"="~r2,lm_coef)
  as.character(as.expression(lm_eq));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

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



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()`. 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
```



Creating an XKCD style chart

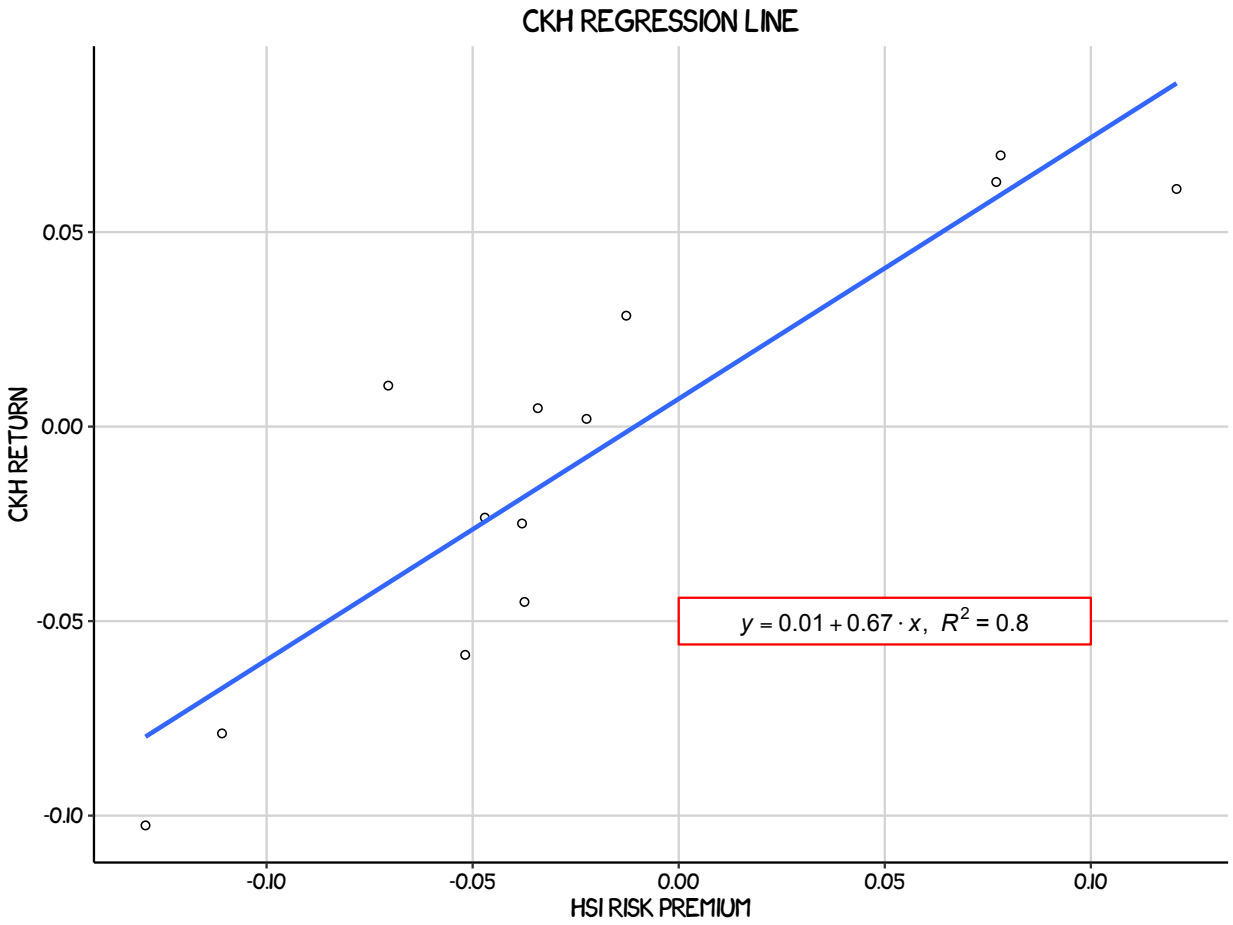
Of course, you may want to create your own themes as well. `ggplot2` allows for a very high degree of customisation, including allowing you to use imported fonts. Below is an example of a theme Mauricio was able to create which mimics the visual style of XKCD. In order to create this chart, you first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

```
library(extrafont)

download.file("http://simonsoftware.se/other/xkcd.ttf",
              dest="xkcd.ttf", mode="wb")
system("mkdir ~/.fonts")
system("cp xkcd.ttf  ~/.fonts")
font_import(paths = "~/.fonts", pattern="[X/x]kcd")
fonts()
loadfonts()
```

You can then create your graph:

```
p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) + geom_point(shape=1) + geom_smoo
      ggtitle("CKH regression line") +
```

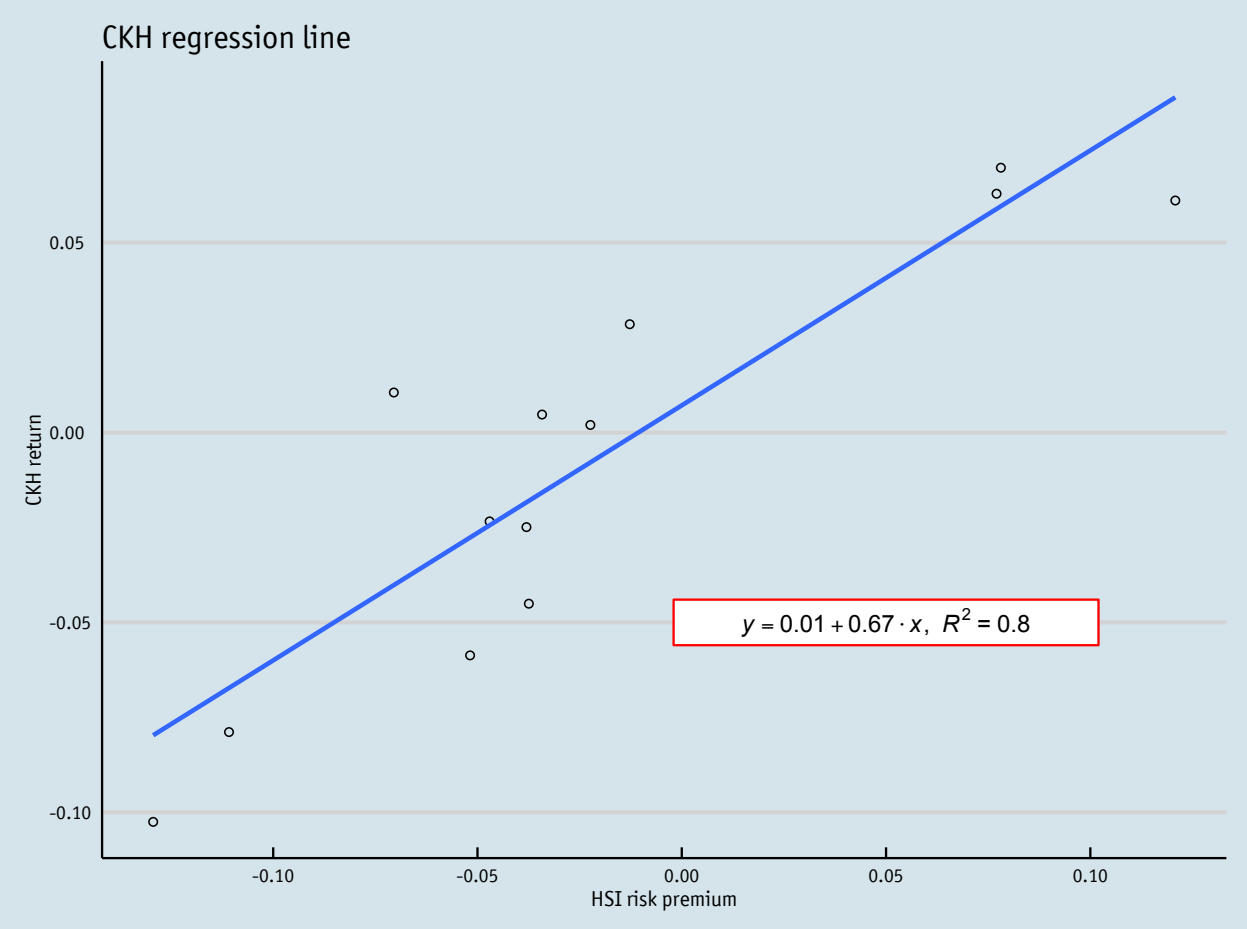
```
      scale_x_continuous(name = "HSI risk premium") +
      scale_y_continuous(name = "CKH return") +
      annotate("rect", xmin = 0.00, xmax = 0.1, ymin = -0.056, ymax = -0.044, fill="white", colour="red
      annotate("text", x = 0.05, y = -0.05, label = equation(fit), parse = TRUE) +
      theme(axis.line.x = element_line(size=.5, colour = "black"),
            axis.line.y = element_line(size=.5, colour = "black"),
            axis.text.x=element_text(colour="black", size = 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"))
p11
```



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.

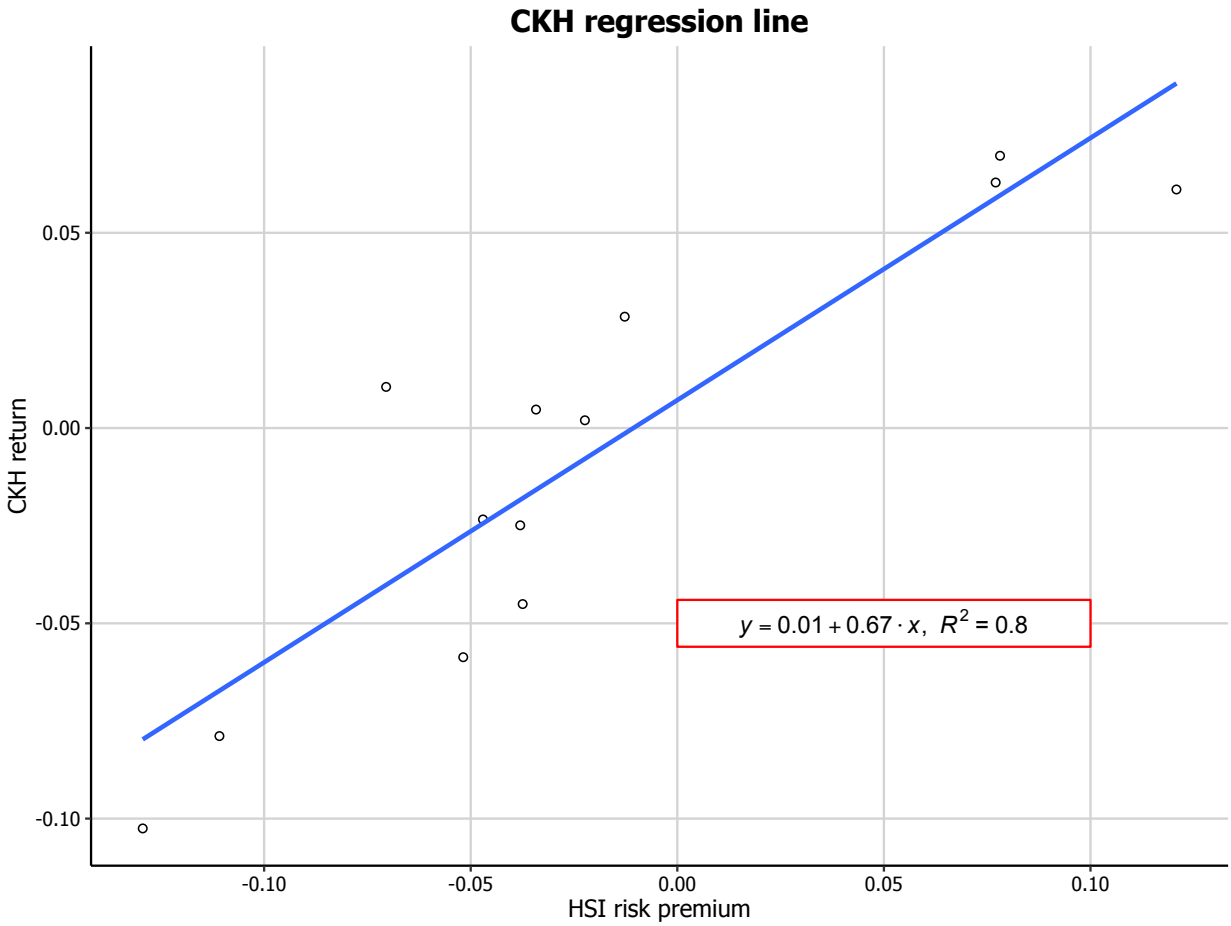
```
library(ggthemes)
library(grid)

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

p11 <- ggplot(hsi.ckh.returns, aes(x=hsi.Risk.premium, y=ckh.Return)) + geom_point(shape=1) + geom_smoo
      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=
      annotate("text", x = 0.05, y = -0.05, label = equation(fit), parse = TRUE) +
      theme_economist() +
      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 = 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 = "OfficinaSanITC-Book"),
            text=element_text(family="OfficinaSanITC-Book"))
p11
```



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.

```
library(grid)

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

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

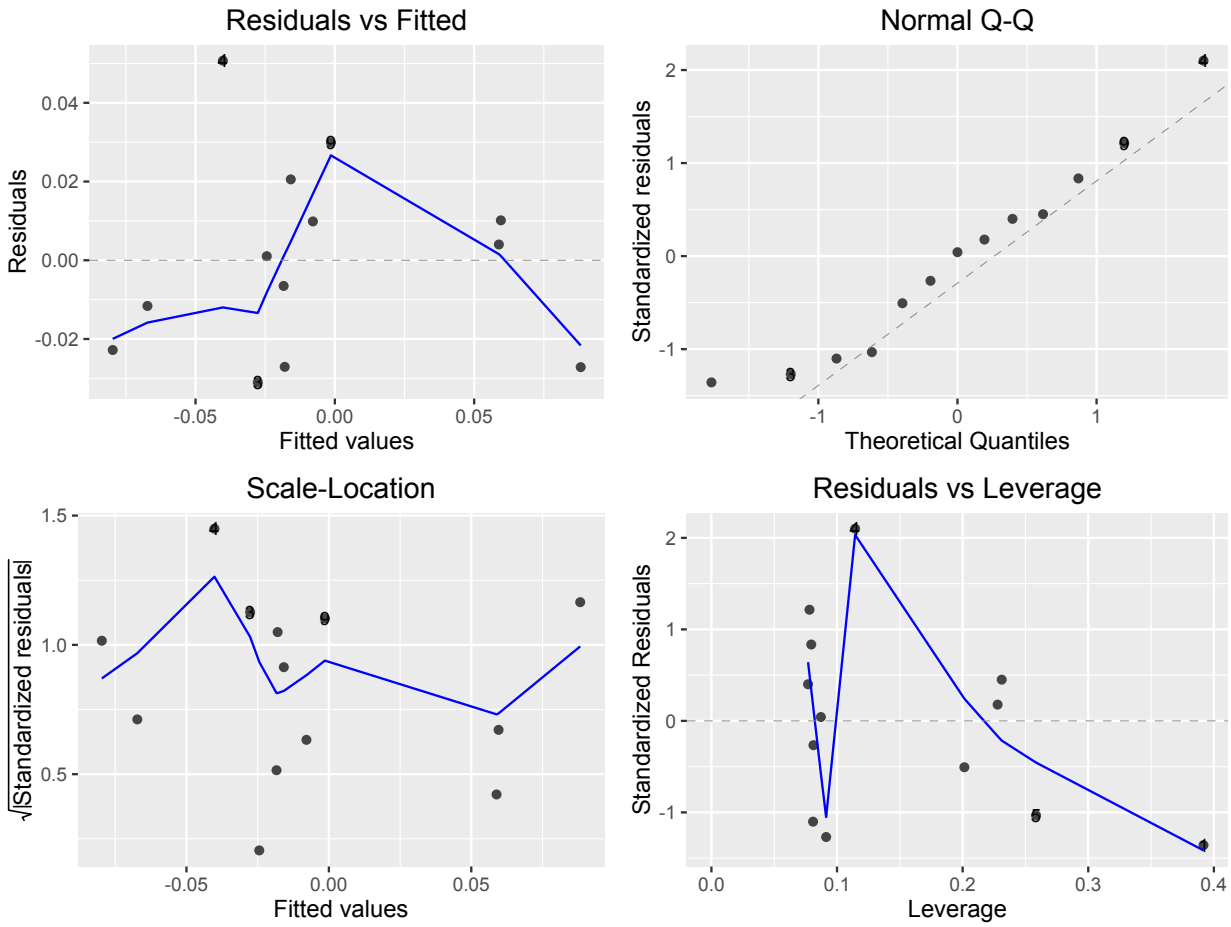


Regression diagnostics plots

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.

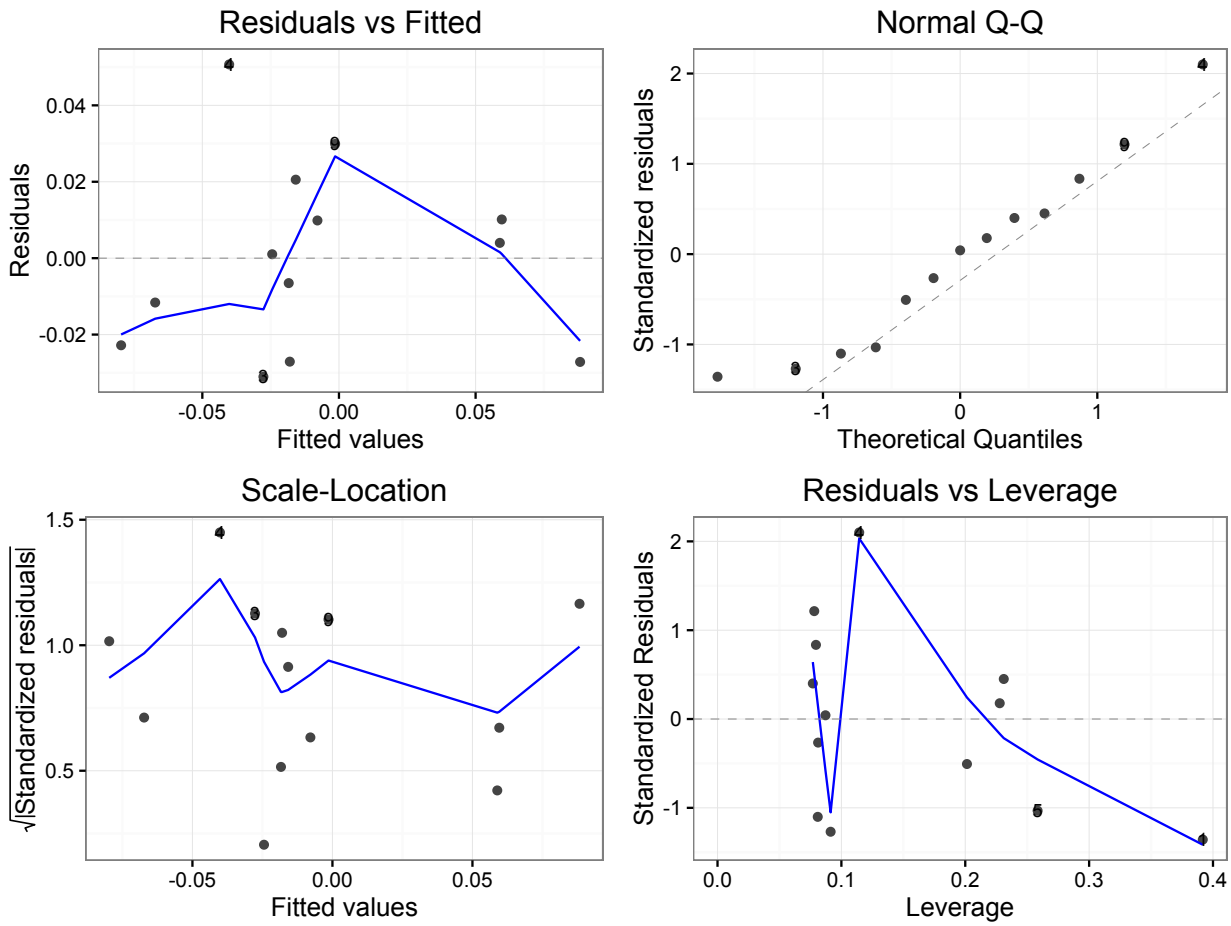
```
# install.packages("ggfortify")
library(ggfortify)
autoplot(fit, label.size = 3)
```



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()`.

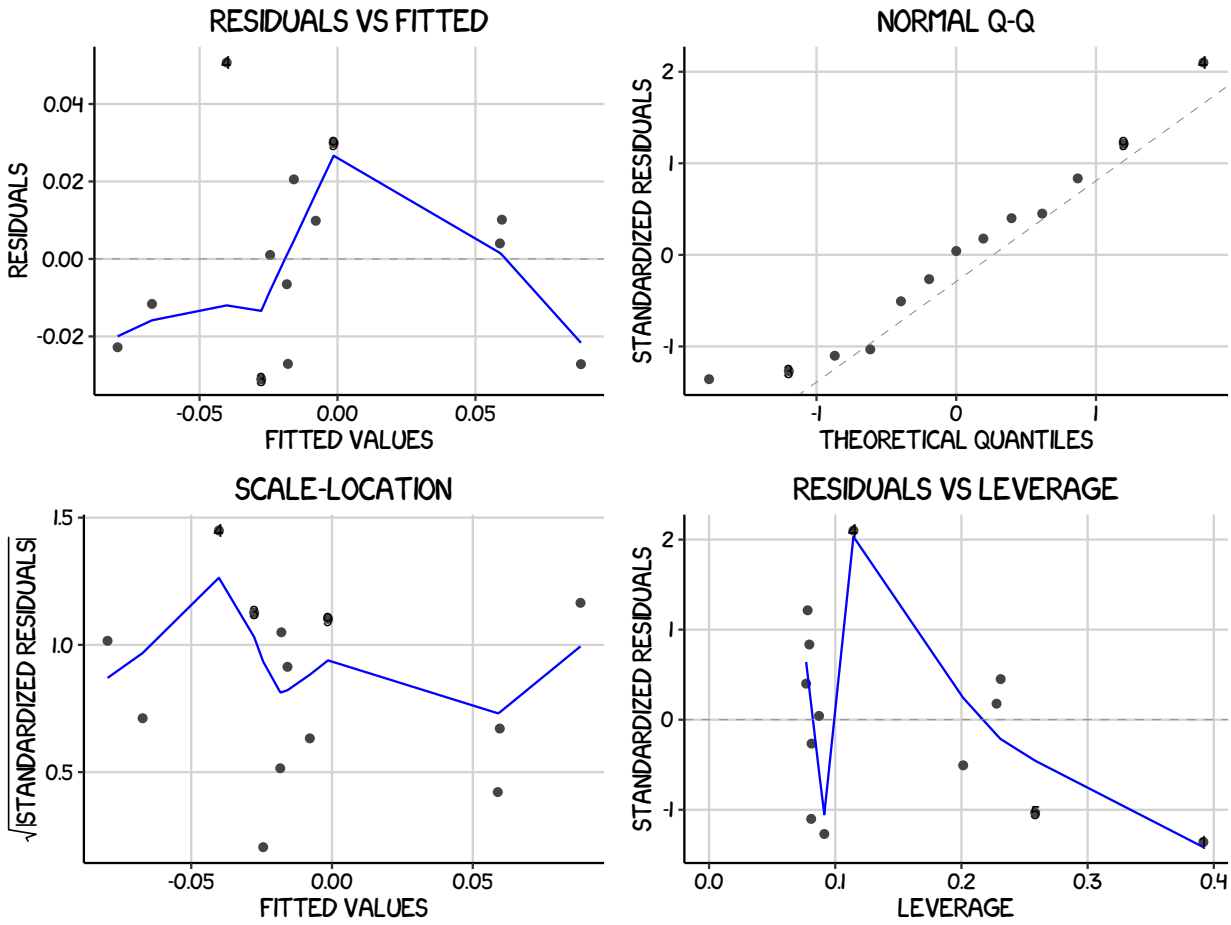
```
autoplot(fit, label.size = 3) + theme_bw()
```



Creating an XKCD style chart

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

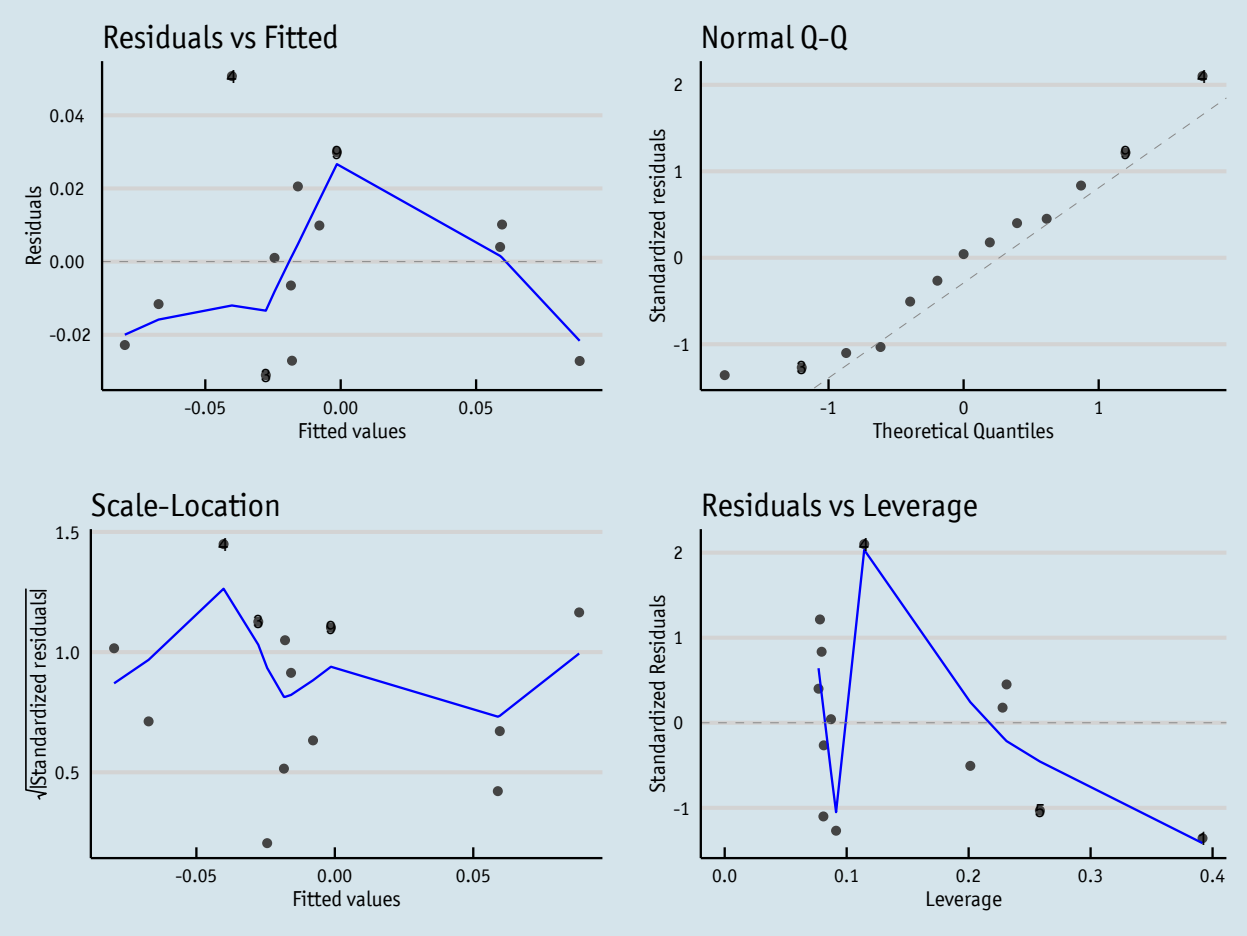
```
autoplot(fit, label.size = 3) + theme(axis.line.x = element_line(size=.5, colour = "black"),
            axis.line.y = element_line(size=.5, colour = "black"),
            axis.text.x=element_text(colour="black", size = 9),
            axis.text.y=element_text(colour="black", size = 9),
            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"))
```



Using 'The Economist' theme

And now the Economist theme.

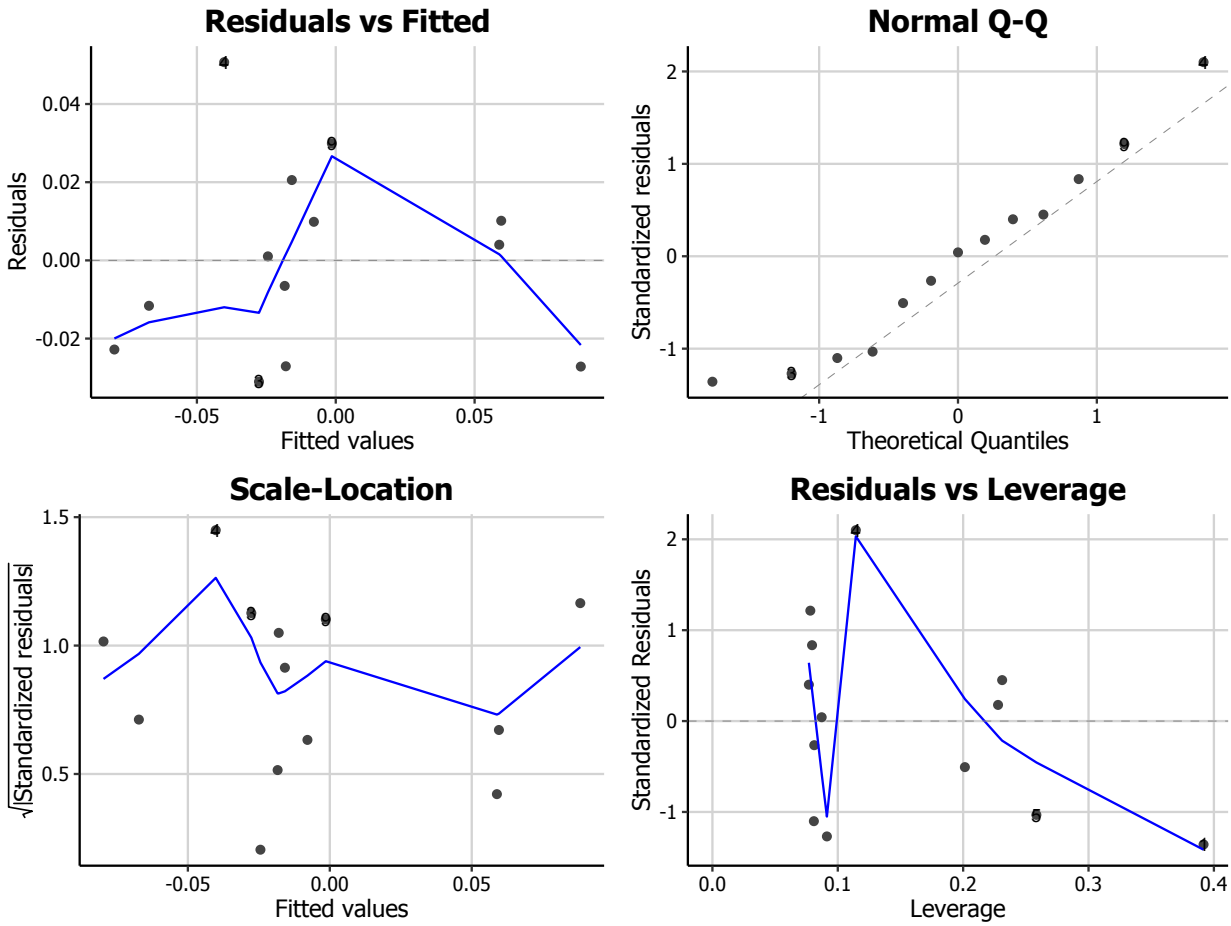
```
autoplot(fit, label.size = 3) + theme_economist() +
      theme(axis.line.x = element_line(size=.5, colour = "black"),
            axis.line.y = element_line(size=.5, colour = "black"),
            axis.text.x=element_text(colour="black", size = 9),
            axis.text.y=element_text(colour="black", 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 = "OfficinaSanITC-Book"),
            text=element_text(family="OfficinaSanITC-Book"))
```



Creating your own theme

Finally, we can also fully customise the diagnostic plots to match our regression plot simply by applying all of the same theme options.

```
autoplot(fit, label.size = 3) + 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 = 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
0.04
0.02
0.00
-0.02
-0.05
0.00
0.05
Normal Q-Q
Standardized residuals
Theoretical Quantiles
2
1
0
-1
-1
0
1
Scale-Location
√|Standardized residuals|
Fitted values
1.5
1.0
0.5
-0.05
0.00
0.05
Residuals vs Leverage
Standardized Residuals
Leverage
2
1
0
-1
0.0
0.1
0.2
0.3
0.4