Data Visualisation with GG-Plot

Johnny Lo

Feb 2023

Contents

1 Introduction to GG-Plot

ggplot2 is a powerful and flexible R package, implemented by Hadley Wickham, for producing elegant graphics piece by piece. **ggplot2** has become a popular package for data visualization, in particular for creating publication-ready plots.

Let us now load the **tidyverse** package, which includes **ggplot2**. Furthermore, install and load the **ggpubr** package, which has some useful additional functions and themes.

```
#install.packages(c("tidyverse","timeDate","ggpubr"))  #De-comment the code to install
library(tidyverse); library(ggpubr)
```

ggplot2 functions typically require data in 'long' format, i.e. a column for every dimension, and a row for every observations. For example, if you need to plot two features on the same axes, then the corresponding data have to be combined by stacking. Having well structured data will save time when making figures with **ggplot2**.

The gg in **ggplot2** means Grammar of Graphics, a graphic concept which describes plots by using "grammar". According to the **ggplot2** concept, a plot can be divided into different fundamental parts: Plot = Data + Aesthetics + Geometry:

1. Data: a data frame
2. Aesthetics: used to indicate the **x** and **y** variables. It can be also used to control the color, the size and the shape of points, etc…
3. Geometry: corresponds to the type of graphics (histogram, box plot, line plot, ..)

In R, the basic template to build a gg-plot is:

```
ggplot(data=<DATA>, mapping=aes(<MAPPINGS>)) + <GEOM_FUNCTION>()
```

The great thing with **ggplot2** is that we can continuously add layers (e.g. changing title, axes labels, font size, grids, and etc) to an existing plot by using the + sign. We can even overlay it with another geometry.

Note that anything that you specify for **DATA** and **MAPPING** in *ggplot(.)* can be seen by any geometry layer that you later add. In other words, these are global plot settings. Specifying them within the geometry function **GEOM_FUNCTION** ensures that these settings are local to that geometry.

In this workshop, you will learn to generate barcharts, histograms, boxplots, scatter and line plots using **ggplot2**.

2 Barchart

Barcharts are an effective way to visualise categorical and (short finite range) discrete data.

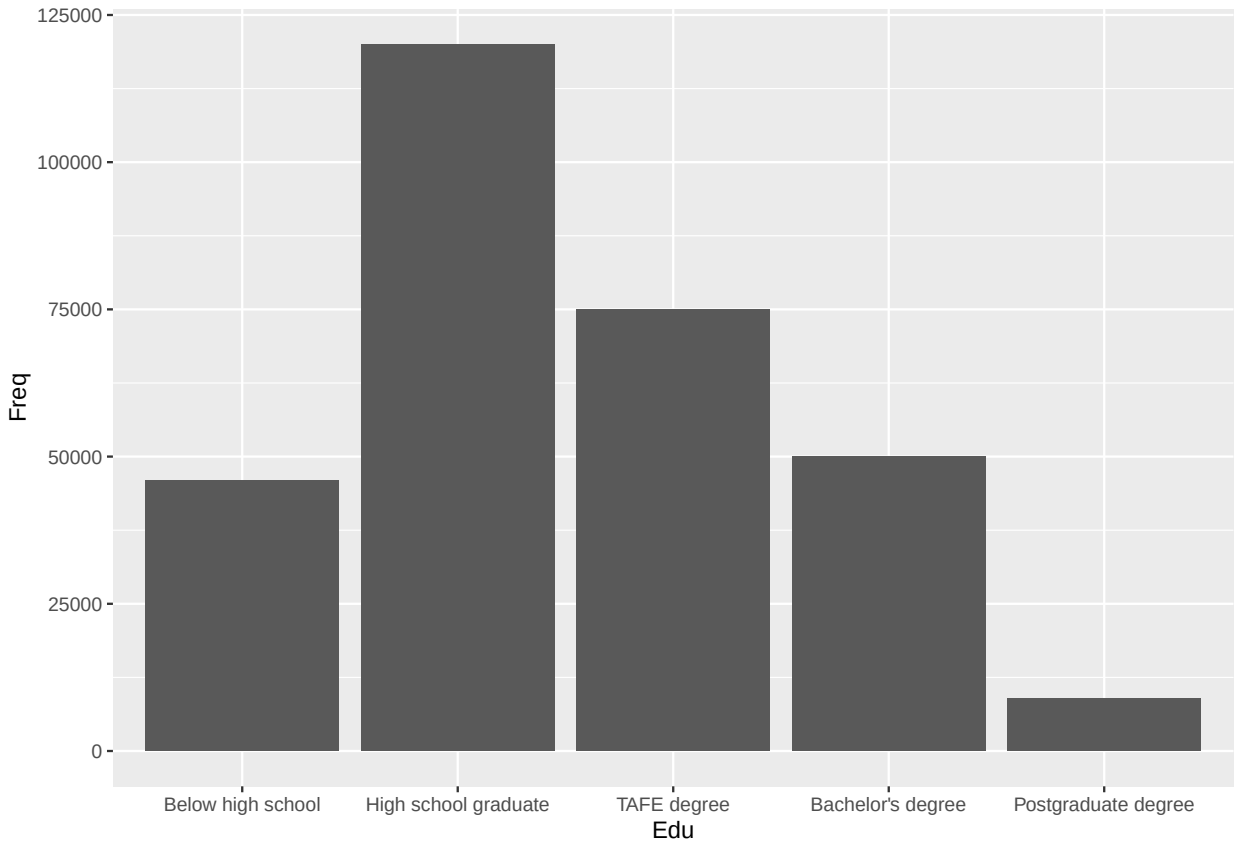
Let us begin with the example from the Module 1 notes where 300,000 randomly sampled people were asked about their highest level of education attained.

```
Edu <- c("Below high school","High school graduate",
         "TAFE degree","Bachelor's degree","Postgraduate degree")  #Highest education level
Freq <- c(46000,120000,75000,50000,9000) #Corresponding frequency
```

```
Perc <- Freq/sum(Freq)*100;  #Corresponding percentage

#Create the data frame
df.edu <- data.frame(Edu,Freq,Perc)

#Reorganise the factor levels for the variable Edu.
df.edu$Edu <- factor(Edu,levels=c("Below high school","High school graduate",
                                  "TAFE degree","Bachelor's degree","Postgraduate degree"))
View(df.edu)  #View the data frame
```

To start, we will build a basic barchart. As you will see, the default colour and overall design are pretty dull.

```
ggplot(df.edu,aes(x=Edu,y=Freq))+
  geom_bar(stat="identity")
```

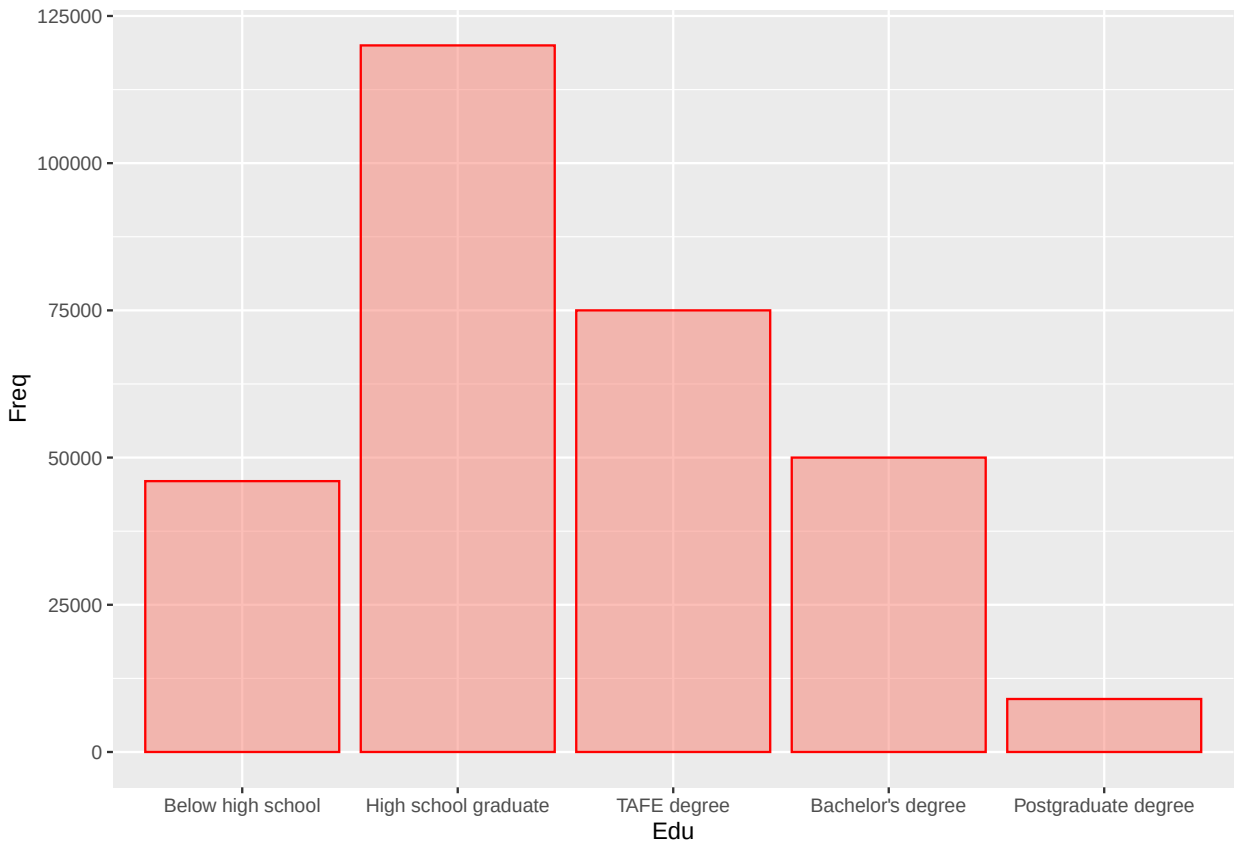


Exercise: Describe the above barchart.

Enter text here

Let us add colours to the bars.

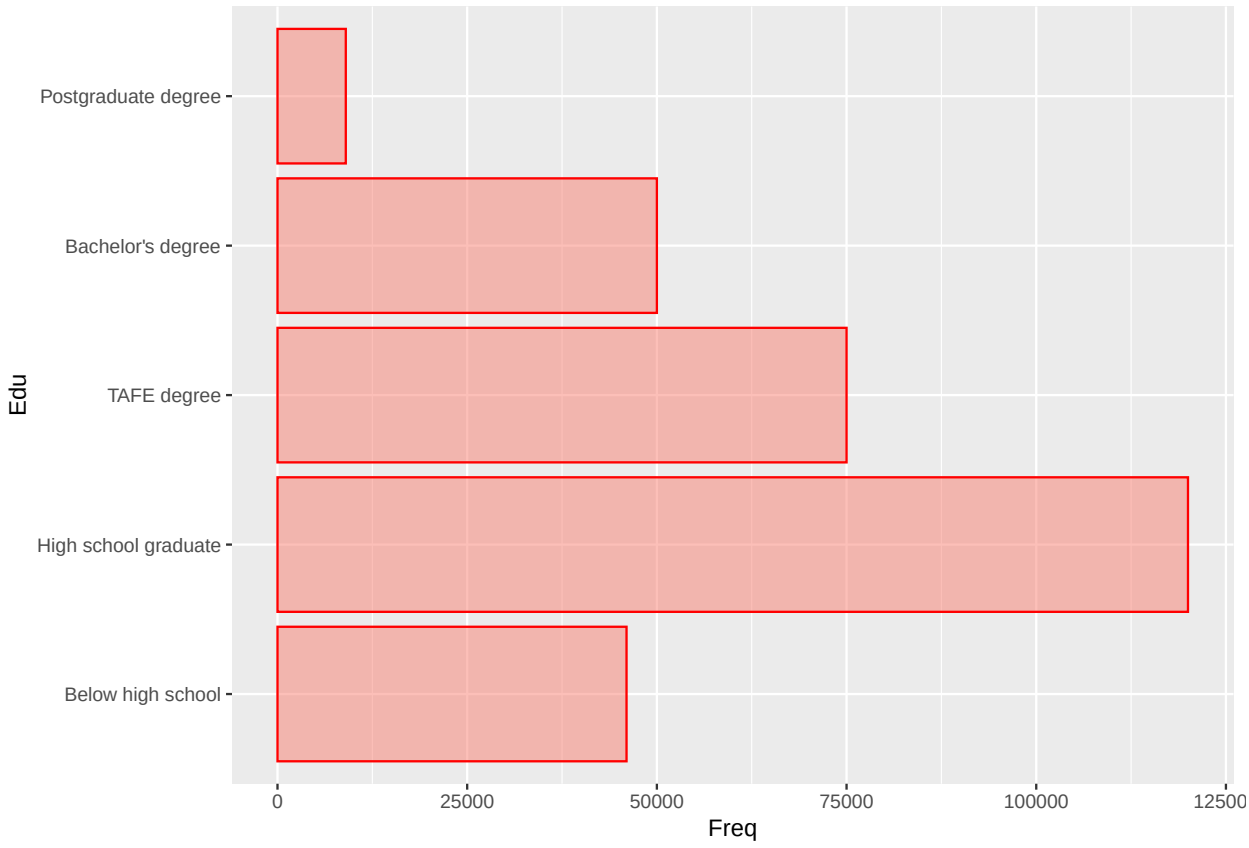
```
ggplot(df.edu,aes(x=Edu,y=Freq))+
  geom_bar(stat="identity",
           colour="red",  #Colour of the border of the bars
           fill="salmon", #Colour inside the bars
           alpha=0.5)  #Transparency of the fill colour from 0 to 1
```



You can click here to see a list of colours in R.

We can also flip the barchart so that the bars are displayed horizontally using the *coord_flip(.)* function.

```
ggplot(df.edu,aes(x=Edu,y=Freq))+
  geom_bar(stat="identity",
           colour="red",  #Colour of the border of the bars
           fill="salmon", #Colour inside the bars
           alpha=0.5) + #Transparency of the fill colour from 0 to 1
  coord_flip()
```



Note that as everything in R is an object, so we can assign a plot to a variable and call it to then display the plot. This makes it easier and cleaner when when we want to add layers to a plot.

Let assign the previous plot (with vertical bars) to the variable **g1**.

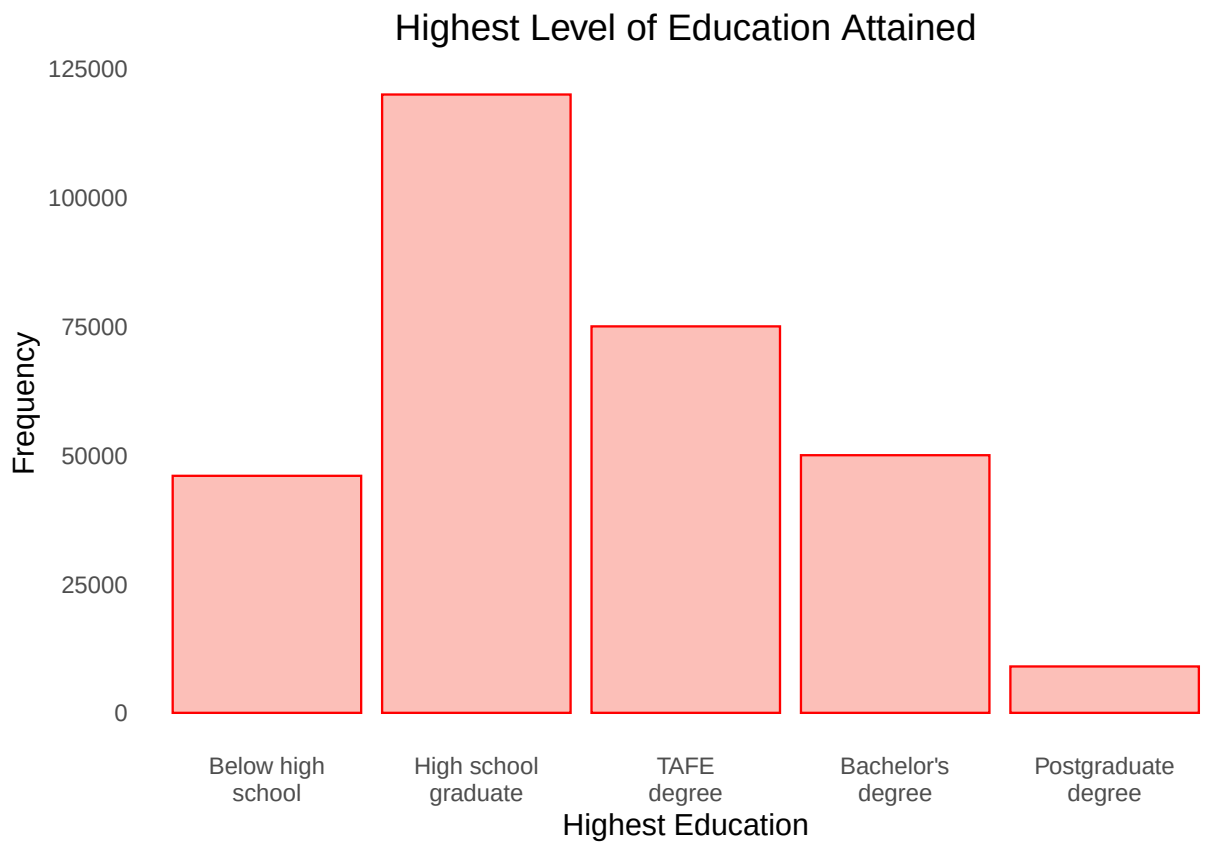
```
g1 <- ggplot(df.edu,aes(x=Edu,y=Freq))+
        geom_bar(stat="identity",
                 colour="red",  #Colour of the border of the bars
                 fill="salmon", #Colour inside the bars
                 alpha=0.5)  #Transparency of the fill colour from 0 to 1
```

We will now add some layers to the plot. In particular, we will:

1. Apply a built-in theme to clean up the plot,

2. Add and centre the plot title,
3. Change the labels for the x and y axes,
4. Remove the grid lines, and
5. Modify the x-axis tick labels.

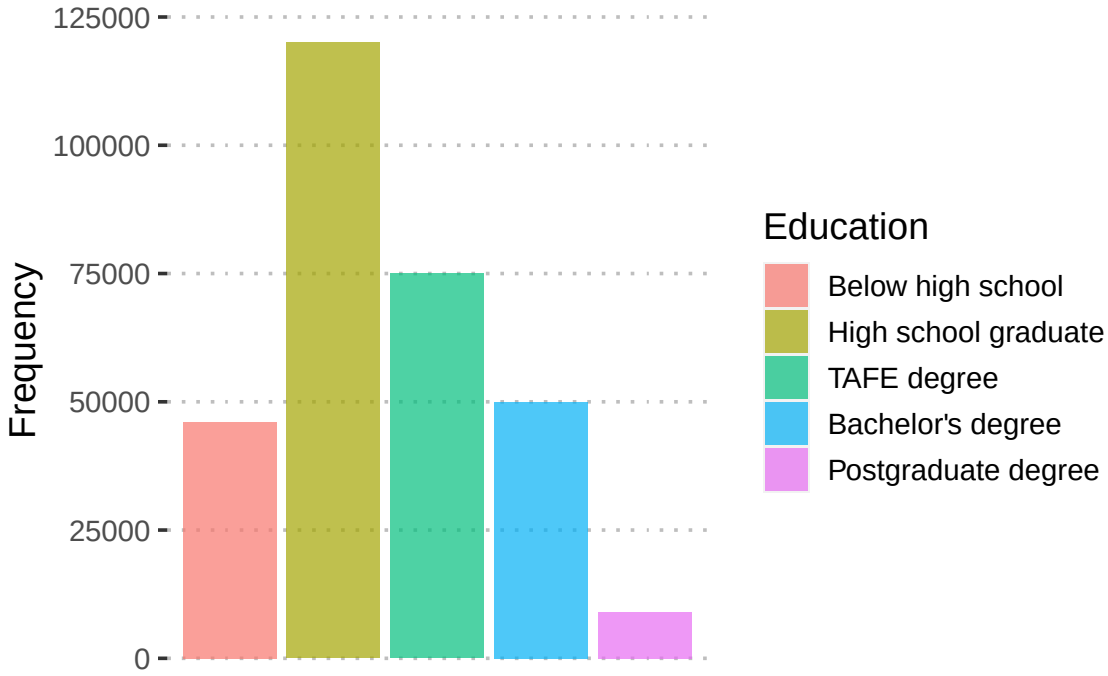
```
g1.clean <- g1 +
              theme_minimal(base_size=14) +  #Apply a built-in theme.
              ggtitle("Highest Level of Education Attained") +  #Add a title
              theme(plot.title = element_text(hjust = 0.5),  #Centre the title
                    panel.grid = element_blank()) +   #Remove the grid lines
              ylab("Frequency") +  #Change the y-axis label, i.e. vertical axis
              xlab("Highest Education") + #Change the x-axis label, i.e. horizontal axis
              #Add a carriage return using '\n' to the labels so that they fit the space
              scale_x_discrete(labels=c("Below high\nschool","High school\ngraduate",
                                        "TAFE\ndegree","Bachelor's\ndegree",
                                        "Postgraduate\ndegree")); g1.clean
```



You can click here to find more information relating to modifying the components of a gg-plot using *theme(.)*.

We can alternatively colour code the bars by specifying the **fill** parameter in *aes(.)*. Doing so will also result in a legend. Here we will colour code the bars according to their education level.

```
g2 <- ggplot(df.edu,aes(x=Edu,y=Freq,fill=Edu))+
        geom_bar(stat="identity",alpha=0.7)
g2.clean <- g2 +
              theme_pubclean(base_size=14) +  #Apply a theme
              ylab("Frequency") +  #Change the y-axis label
              labs(x="",  #Remove x-axis label
                   fill="Education") +  #Change the legend title
              theme(axis.text.x=element_blank(),  #Remove the tick labels
                    axis.ticks.x=element_blank(),  #Remove the tick marks
                    legend.position="right")   #Shif the legend to the right-hand
g2.clean
```



==========

The remainder of this section (pp. 7-12) is optional

==========

For the next example, let us generate some random data for the number of herrings caught at Busselton jetty from morning (6-9 am) and evening (5-8 pm) over a span of 150 days. Here, we note that the maximum number of fishers surveyed is 6 in any sampling frame, i.e. discrete data.

```
set.seed(2)

#Total number of fish caught
```

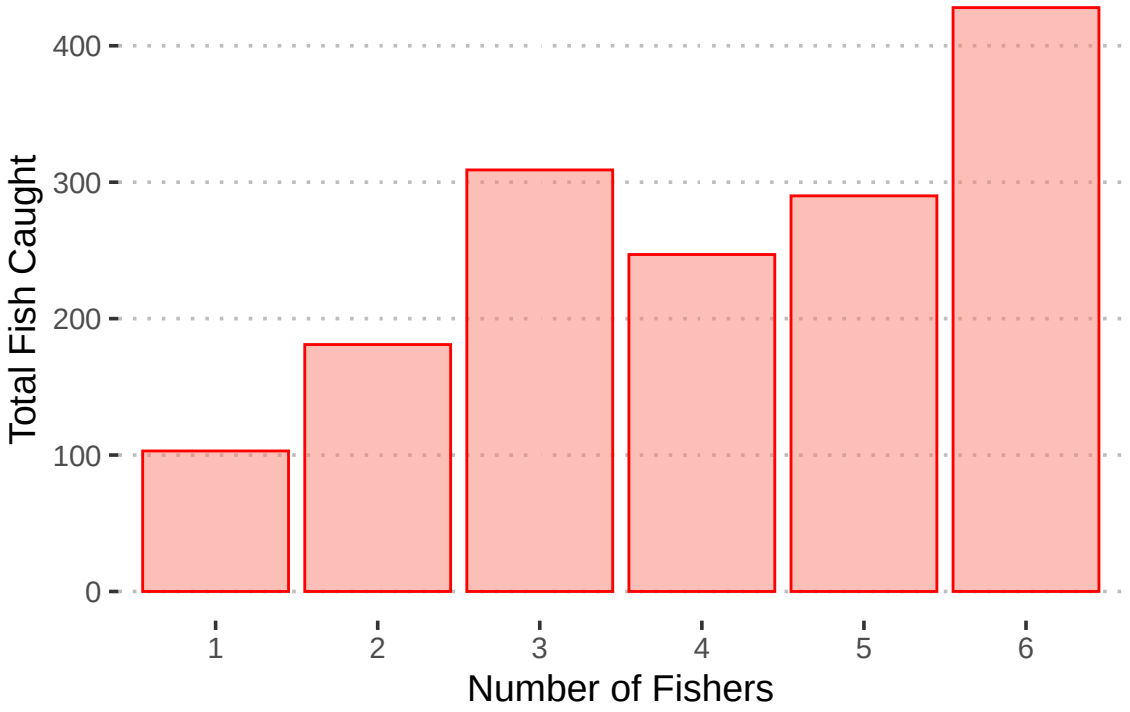
```
fish_caught <- sample(0:15, #List of values to sample from
                      size = 300,  #Number of random values
                      replace=TRUE,  #Sampling with replacement
                      prob=c(0.11,0.1,0.1,0.09,0.09,0.09,0.09,0.08,0.07,
                             0.04,0.04,0.04,0.03,0.01,0.01,0.01)) #probability weights

#Number of fishers at the time of sampling
num_fisher <- sample(1:6,size=300,replace=TRUE,
                     prob=c(0.06,0.11,0.16,0.17,0.22,0.28))

#Time of day
ToD <- factor(rep(c("Morning","Evening"),each=150),levels=c("Morning","Evening"));

#Create the data frame. Note that num_fisher is converted to a factor.
df.fish <- data.frame(num_fisher=as.factor(num_fisher),ToD,fish_caught);
View(df.fish)
```

Next, we are going to aggregate the data and find the total catch according to the number of fishers.

```
Tot_fishers <- aggregate(fish_caught~num_fisher,data=df.fish,FUN=sum);
View(Tot_fishers)
```

Now, we are ready to plot.

```
g3 <- ggplot(Tot_fishers,aes(x=num_fisher,y=fish_caught))+
        geom_bar(stat="identity",
                 colour="red",  #Colour of the border of the bars
                 fill="salmon", #Colour inside the bars
                 alpha=0.5)  #Transparency of the fill colour from 0 to 1

#Clean up the plot
g3.clean <- g3 +
              theme_pubclean(base_size=14) + #This theme is from ggpubr
              ylab("Total Fish Caught") +
              xlab("Number of Fishers"); g3.clean
```



Exercise: Describe the above plot.

Enter text here

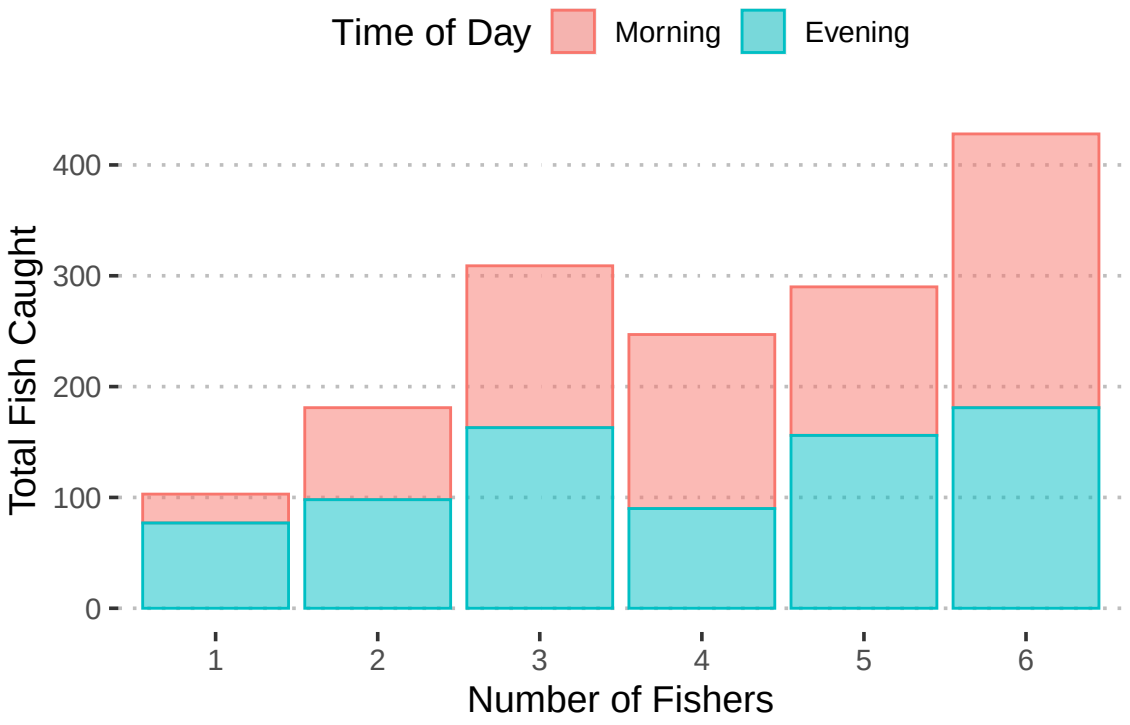
We can also aggregate total catch by number of fishers **and** time of day,

```
Tot_fishers.ToD <- aggregate(fish_caught~num_fisher+ToD,data=df.fish,FUN=sum);
View(Tot_fishers.ToD)
```

and plot total catch by both factors in a **stacked** barchart.

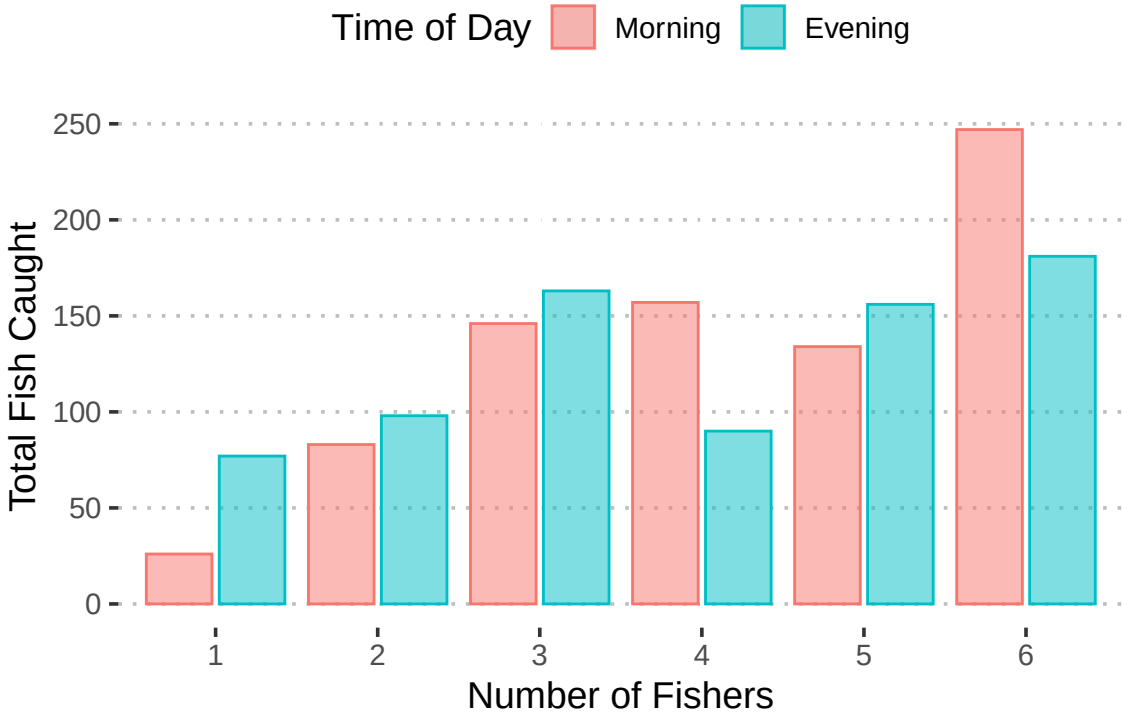
```
g4 <- ggplot(Tot_fishers.ToD,aes(x=num_fisher,y=fish_caught,fill=ToD,colour=ToD))+
        geom_bar(stat="identity",alpha=0.5)

#Clean up the plot
g4.clean <- g4 +
              theme_pubclean(base_size=14) +
```

```
              ylab("Total Fish Caught") +
              xlab("Number of Fishers") +
              labs(fill="Time of Day",colour="Time of Day")  #Changing the legend title
g4.clean
```



Alternatively, we can plot the bars side by side by adding the **position** argument in the *geom_bar(.)* function.

```
g5 <- ggplot(Tot_fishers.ToD,aes(x=num_fisher,y=fish_caught,fill=ToD,colour=ToD))+
        geom_bar(stat="identity",alpha=0.5,
                 position = position_dodge2(width=3,padding=0.1))

#Clean up the plot
g5.clean <- g5 +
              theme_pubclean(base_size=14) +
              ylab("Total Fish Caught") +
              xlab("Number of Fishers") +
              labs(fill="Time of Day",colour="Time of Day"); g5.clean
```



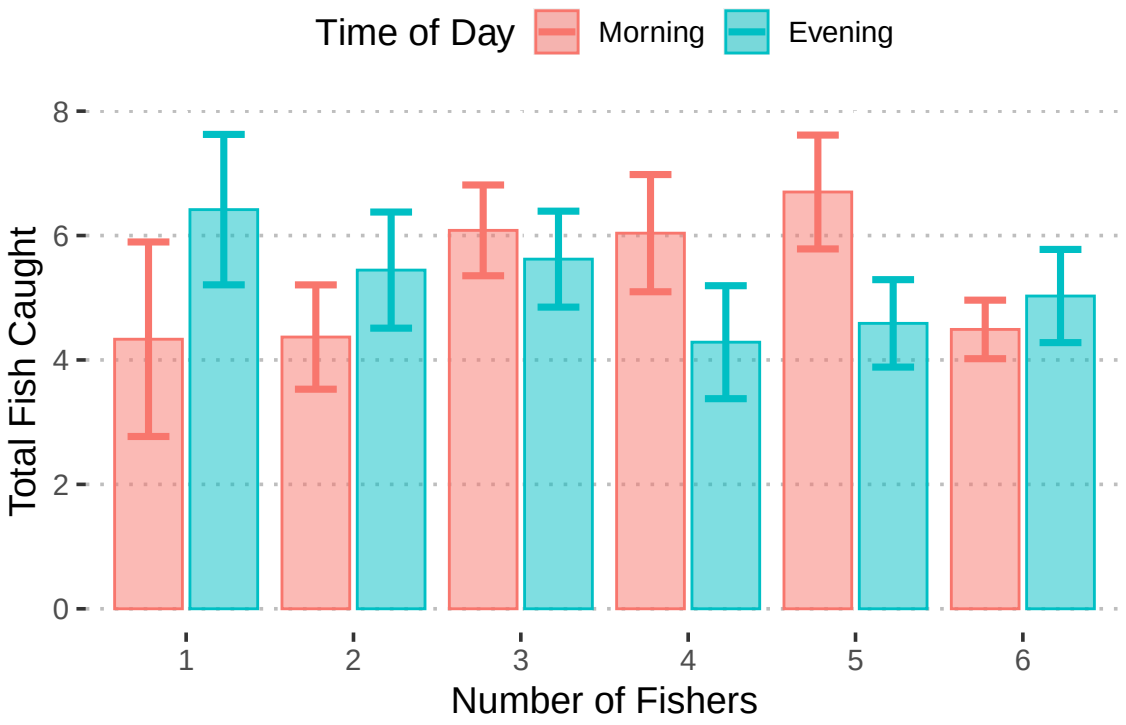
Next, let us generate a barchart for the **mean** catch instead of total catch. Furthermore, We will add error bars to the plot. First we need to calculate the mean catch and the standard error of the mean (i.e. $SE = \frac{sd}{\sqrt{n}}$).

```
#Function to calculate standard error of the mean
SE <- function(x)
{
  SE <- sd(x)/sqrt(length(x))
}

mean_catch <- aggregate(fish_caught~num_fisher+ToD,data=df.fish,FUN=mean); #Mean catch
SE_catch <- aggregate(fish_caught~num_fisher+ToD,data=df.fish,FUN=SE);  #SE of mean

#Calculate the lower bound (LB) and upper bound (UB) of the error bars and add to data frame
mean_catch$LB <- mean_catch[,3] - SE_catch[,3]
mean_catch$UB <- mean_catch[,3] + SE_catch[,3]

View(mean_catch)  #View the data frame
```

Now we can plot the mean catch along with the error bars using the *geom_errorbar(.)* function.

```
g6 <- ggplot(mean_catch,aes(x=num_fisher,y=fish_caught,fill=ToD,colour=ToD))+
        geom_bar(stat="identity",alpha=0.5,
                 position = position_dodge2(width=3,padding=0.1)) +
        geom_errorbar(aes(ymin=LB,ymax=UB,colour=ToD),width=0.5,size=1.25,
```

```
                position=position_dodge(0.9))

#Clean up the plot
g6.clean <- g6 +
              theme_pubclean(base_size=14) +
              ylab("Total Fish Caught") +
              xlab("Number of Fishers") +
              labs(fill="Time of Day",colour="Time of Day"); g6.clean
```



Exercise: Compare the barcharts for total catch and mean catch. What is the difference between the two plots and what can you conclude about the catch rate and catch total and how they vary across time and with increasing number of fishers?

Enter text here

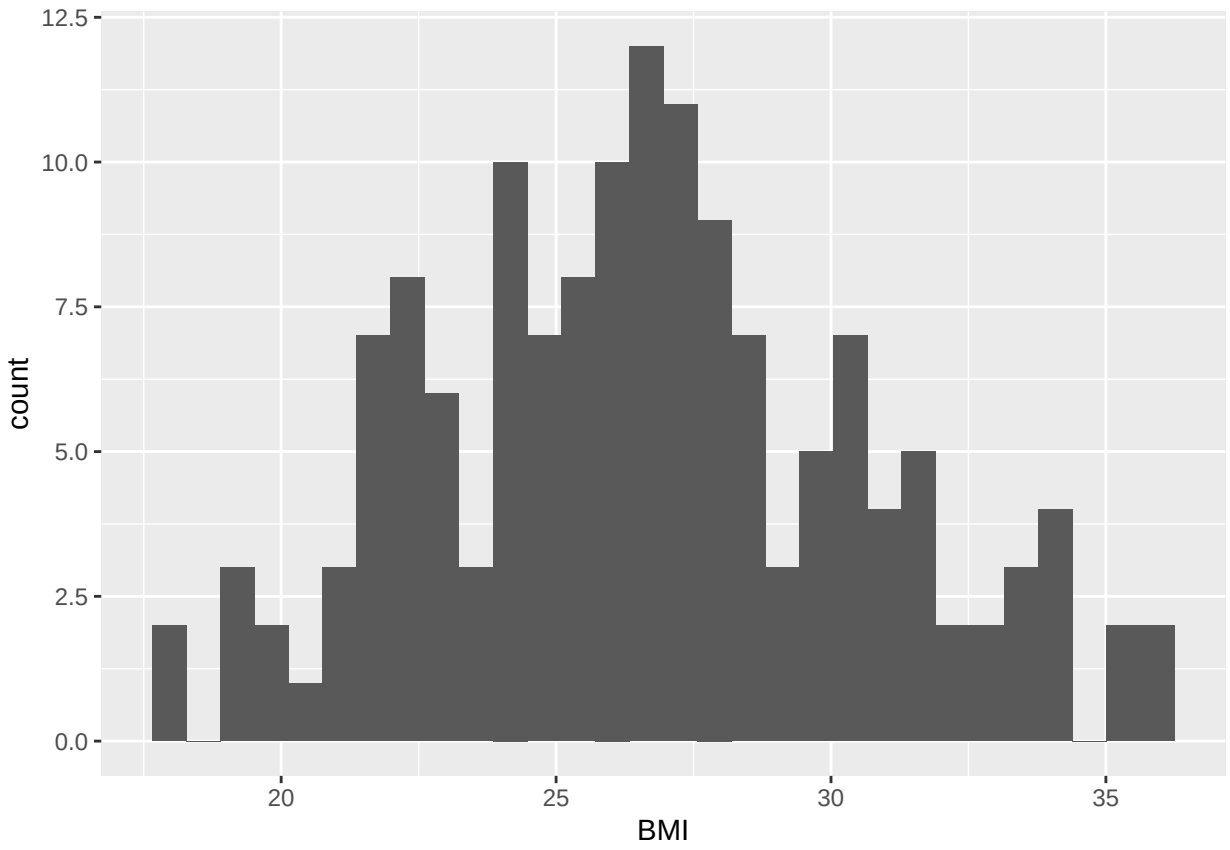
3 Histogram

A great way to visualise **continuous** and (long range) discrete data is with histograms. A histogram is similar to a barchart in that it is a frequency plot. However for the latter, which are appropriate for visualising qualitative and discrete data, the categories (along the horizontal axis) are pre-defined. With histograms, these '*categories*' are intervals (or bins), the width of which is specified by the user. The only criterion is that the intervals **must** all be the same width.

Given this, it is quite important to select the appropriate bin width for a histogram as it will impact the overall shape of the distribution. A rough guideline is to select a bin width such that each interval has at least 5 observations, and this is particular useful guide for data with $n < 50$. For larger datasets, it is more about the number of bins, and in general, 10-20 bins will often suffice. Having said this, you should always explore difference values and find one that best illustrates the data distribution.

Let us revisit the *ElderlyPopWA* dataset and create a histogram for the **BMI** variable. Load the dataset by clicking on *Import Dataset* in the **Environment** panel, or use the *read.csv(.)* command.

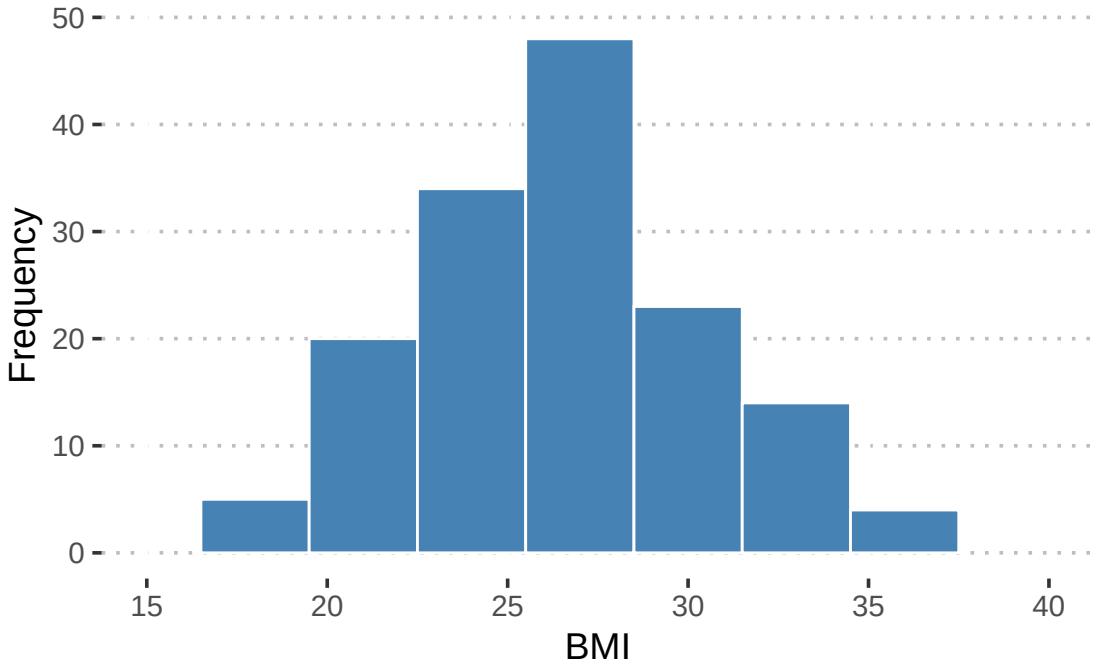
```
#Note that this location of the data file may be different on your machine
ElderlyPopWA <- read.csv("~/Desktop/ElderlyPopWA.csv",stringsAsFactors=TRUE)

ggplot(ElderlyPopWA,aes(x=BMI)) +
  geom_histogram()  #Histogram with default settings
```



If the range is large enough, *geom_histogram(.)* defaults the number of bins to 30. You should always

override this value. Let us set the bin width to be 3, add some colours and clean up the plot.

```
g7 <- ggplot(ElderlyPopWA,aes(x=BMI)) +
        geom_histogram(binwidth = 3,colour="white",fill="steelblue")

#Clean up the plot
g7.clean <- g7 +
              ylab("Frequency") +
              xlim(15,40) +  #Change the limits of the x-axis
              theme_pubclean(base_size=14); g7.clean
```



Exercise: Describe the BMI data in relation to centre, spread and shape. Similarly, plot a histogram for **Waist**, **Hip**, **Tricep**, **Arm Girth** and **Percent Body Fat** and describe the them.

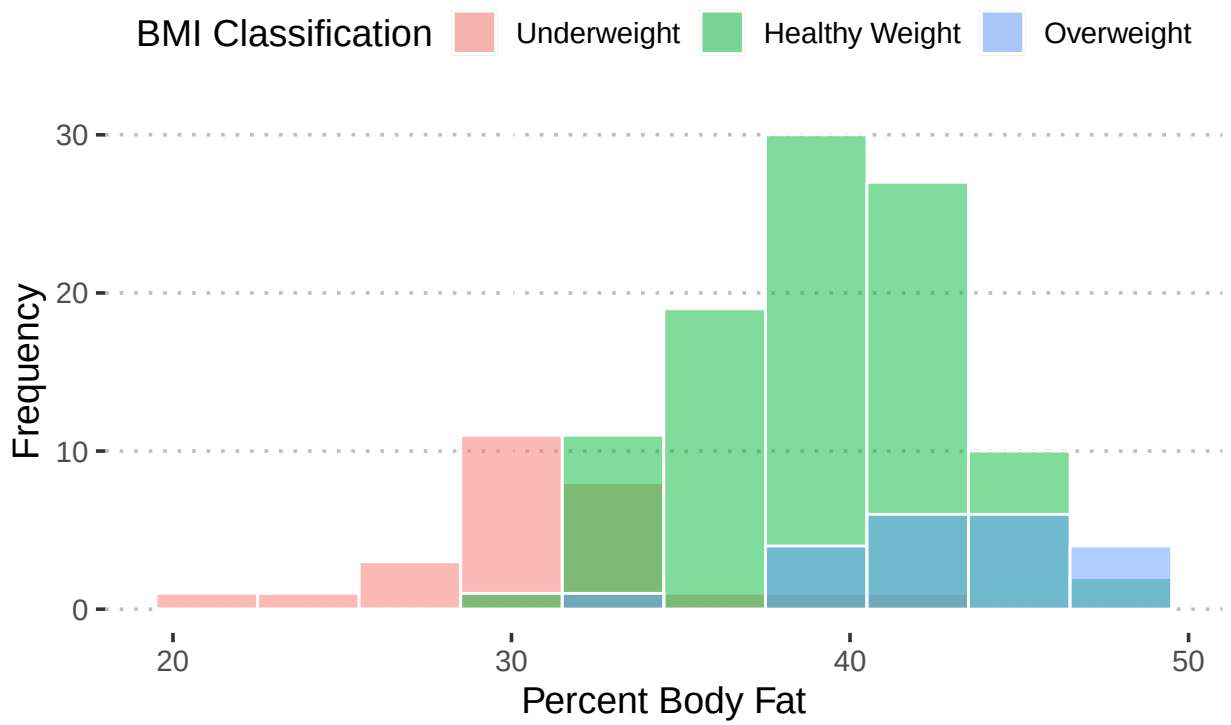
Enter text here

Let us now recreate the BMI classes as follows.

```
#Create BMI categories for the elderly female participants

mBMI <- max(ElderlyPopWA$BMI)  #maximum BMI value within the sample

ElderlyPopWA$BMI.class <- cut(ElderlyPopWA$BMI,
                              breaks=c(0,23,31,mBMI), #Set the intervals for the classes
                              labels=c("Underweight",
                                       "Healthy Weight",
                                       "Overweight")) #Add labels to the classes
```

We can plot, say the percent body fat of the individuals across all the BMI classes.

```
g8 <- ggplot(ElderlyPopWA,aes(x=Pc_Body_Fat,fill=BMI.class)) +
       geom_histogram(colour="white", #border colour of the bars
                      alpha=0.5,  #Transparency of bars
                      binwidth=3,  #width of the bins
                      position="identity")  #default is stacked. Can be "identity" or "dodge"

#Clean up the plot
g8.clean <- g8 +
             theme_pubclean(base_size=14) +
             labs(x="Percent Body Fat", y="Frequency",fill="BMI Classification"); g8.clean
```

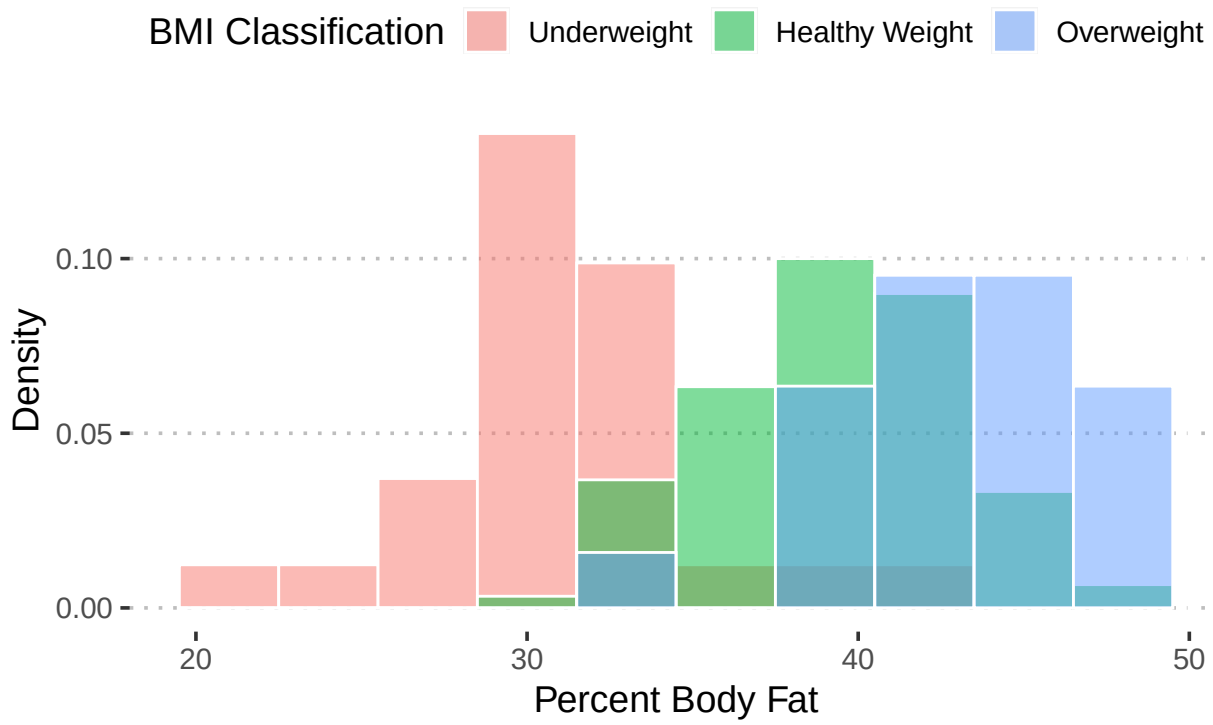


Exercise: Describe the percent body fat for each of the BMI classes individually and in contrast to each other.

Enter text here

Instead of displaying the frequency or count of observations falling into each interval or bin, we can display the densities (or probabilities) instead.

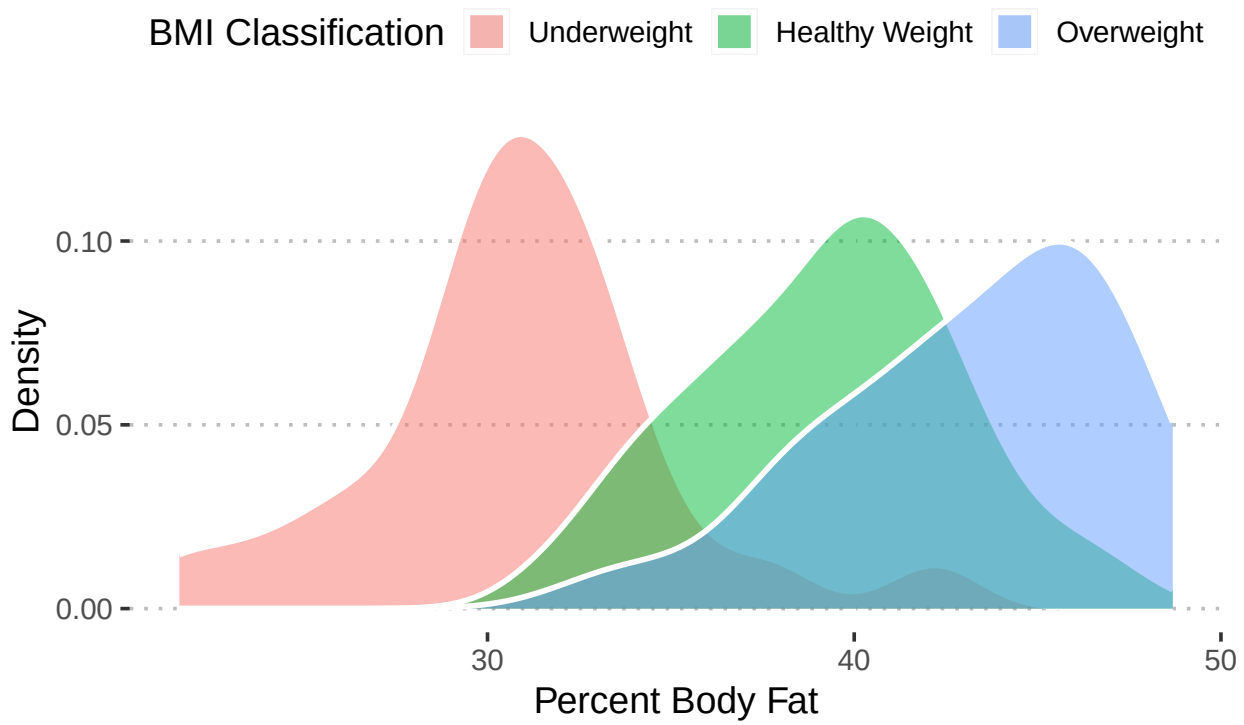
```
g9 <- ggplot(ElderlyPopWA,aes(x=Pc_Body_Fat,fill=BMI.class)) +
        geom_histogram(aes(y=..density..),  #Show densities instead of frequencies
                       colour="white", #border colour of the bars
                       alpha=0.5,  #Transparency of bars
                       binwidth=3,  #width of the bins
                       position="identity")  #default is stacked. Can be "identity" or "dodge"

#Clean up the plot
g9.clean <- g9 +
             theme_pubclean(base_size=14) +
             labs(x="Percent Body Fat", y="Density",fill="BMI Classification"); g9.clean
```



Notice how this did not affect the shape of the distribution, but the heights of the distributions are now similar. This is because they have been scaled in such a way that the areas of the bars sum to 1. Note that density is different to relative frequency, which is calculated by dividing each frequency by total number of observations for the corresponding group.

A histogram can be an inadequate visual tool for determining the shape of the distribution as it is heavily influenced by the bin width or the number of bins. A more effective way to viewing distributions is by using density curves.

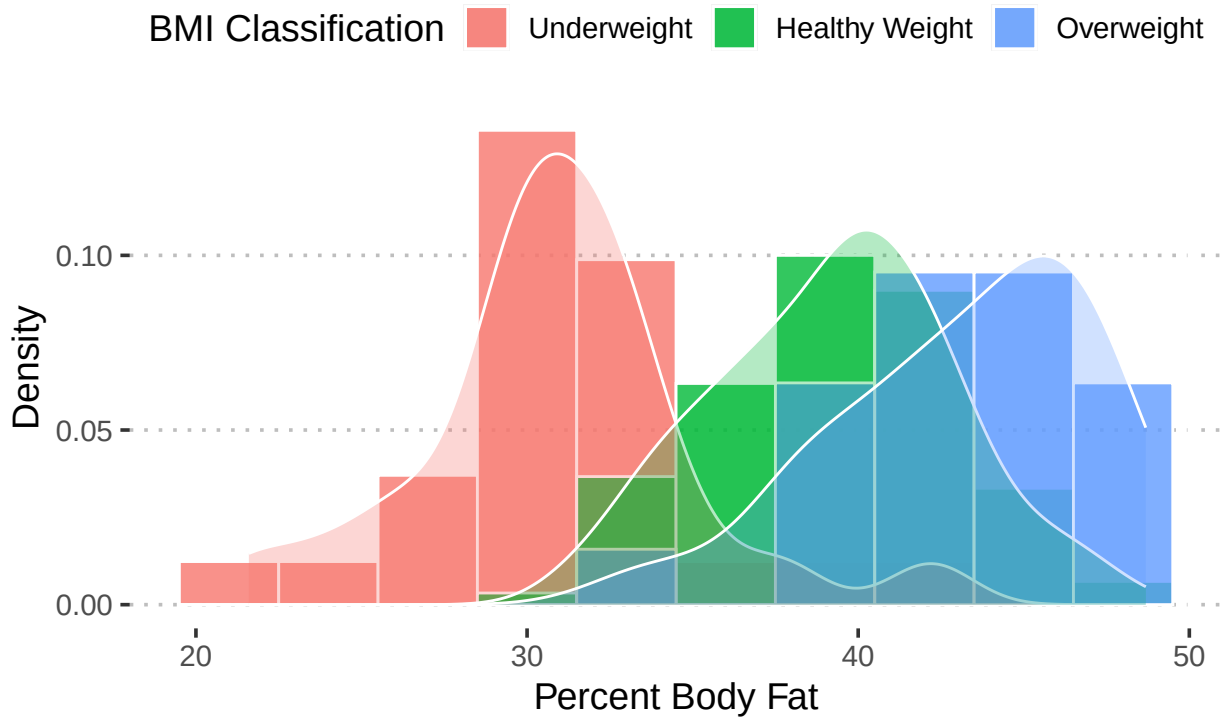
```
g10 <- ggplot(ElderlyPopWA,aes(x=Pc_Body_Fat,fill=BMI.class)) +
        geom_density(colour="white",size=1,alpha=0.5)

#Clean up the plot
g10.clean <- g10 +
             theme_pubclean(base_size=14) +
             labs(x="Percent Body Fat", y="Density",fill="BMI Classification"); g10.clean
```



Alternatively, we can plot both the histograms and density curves on the same axes. If the two match up quite well, then this gives credence to our selection of the bin width.

```
g11 <- ggplot(ElderlyPopWA,aes(x=Pc_Body_Fat,fill=BMI.class)) +
        geom_histogram(aes(y=..density..),
                       colour="white", #border colour of the bars
                       alpha=0.8,  #Transparency of bars
                       binwidth=3,  #width of the bins
```

```
                        position="identity") +  #default is stacked. Can also be "dodge"
        geom_density(colour="white",  #Colour of the outline of the curves
                     size=0.5,  #Width of the outline
                     alpha=0.3) #Transparency of the curves

#Clean up the plot
g11.clean <- g11 +
              theme_pubclean(base_size=14) +
              labs(x="Percent Body Fat", y="Density",fill="BMI Classification"); g11.clean
```



4 Boxplot

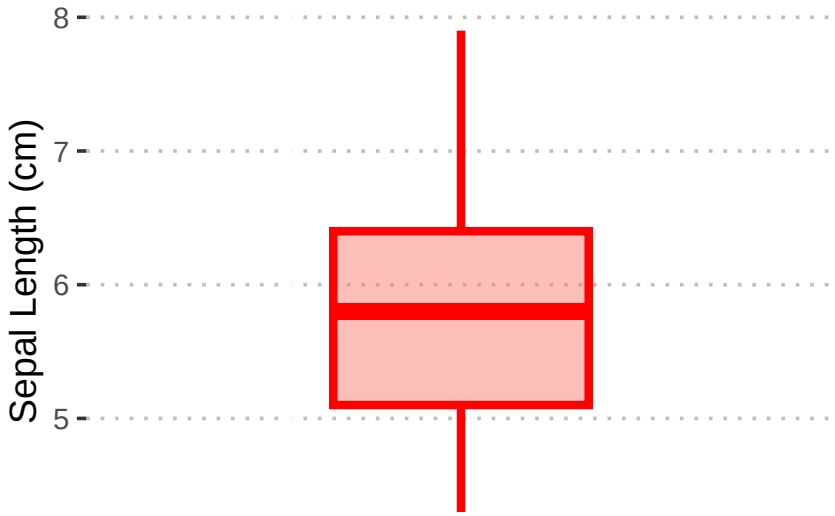
A boxplot (or box and whiskers plot) is a visualisation of the 5-number summary. A great feature of a boxplot in R (and other software) is that outliers are flagged. By default, a point x is an outlier if it falls outside of the range given by $Q_1 - 1.5 \times IQR \leq x \leq Q_3 + 1.5 \times IQR$. The value of 1.5 can be altered to say, 3, by specifying the argument **oulier.size** in the *geom_boxplot(.)* command.

We will use the iris data for this demonstration. Type **?iris** for more information regarding this particular dataset.

```
data(iris)  #Load the dataset, which is built
View(iris)  #View the iris dataset
```

Firstly, let us examine the iris' sepal lengths, and without regard to the species for the time being.

```
g12 <- ggplot(iris,aes(y=Sepal.Length)) +
         geom_boxplot(size=1.5,  #Thickness of the outline of the boxplot
                      colour="red",
                      fill="salmon",
                      alpha=0.5)

#Clean up the plot
g12.clean <- g12 +
               theme_pubclean(base_size=14) +
               ylab("Sepal Length (cm)") +
               xlim(-1,1) +  #Widen the limits on x-axis, thus reducing the boxplot width
               theme(axis.ticks.x=element_blank(),  #Remove the tick marks
                     axis.text.x=element_blank())  #Remove tick labels
g12.clean
```

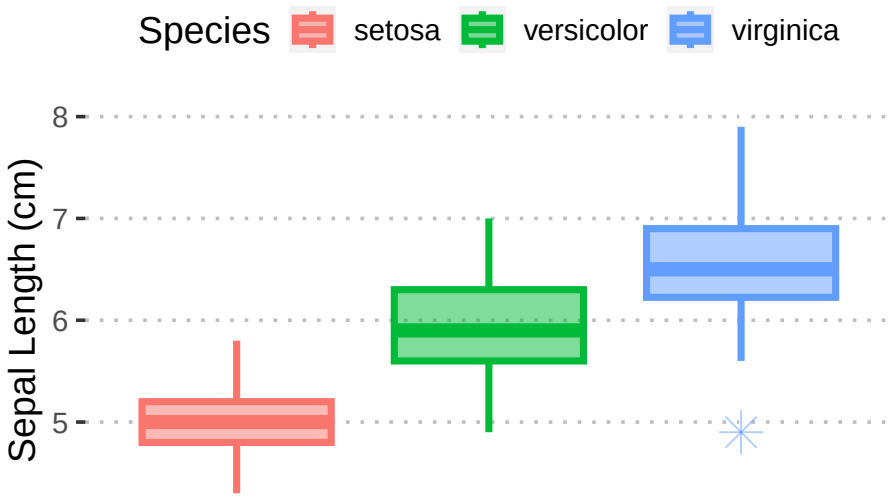


Exercise: Make the necessary adjustment to the above code so that the boxplot is presented horizontally.

Now, let us plot the sepal lengths by their species.

```
g13 <- ggplot(iris,aes(x=Species,y=Sepal.Length)) +
         geom_boxplot(aes(colour=Species,fill=Species),
                      size=1.25,
                      alpha=0.5,
                      outlier.size=5,  #Size of the outlier(s), if any
                      outlier.shape=8)  #Shape of the outlier(s), if any

#Clean up the plot
g13.clean <- g13 +
               theme_pubclean(base_size=14) +
               labs(x="",y="Sepal Length (cm)") +
               theme(axis.ticks.x=element_blank(),
                     axis.text.x=element_blank());  #No need for label since there is a legend
```

```
g13.clean
```

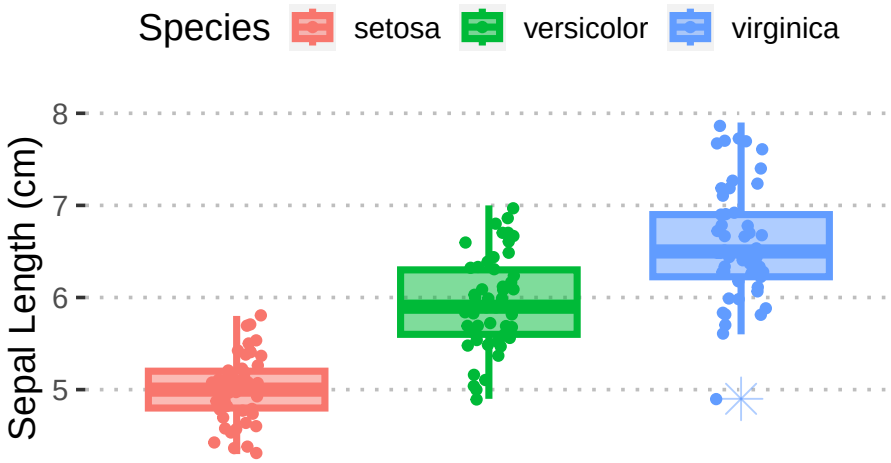


Exercise: Describe the sepal length distribution for each species individually and comparatively.

Enter text here

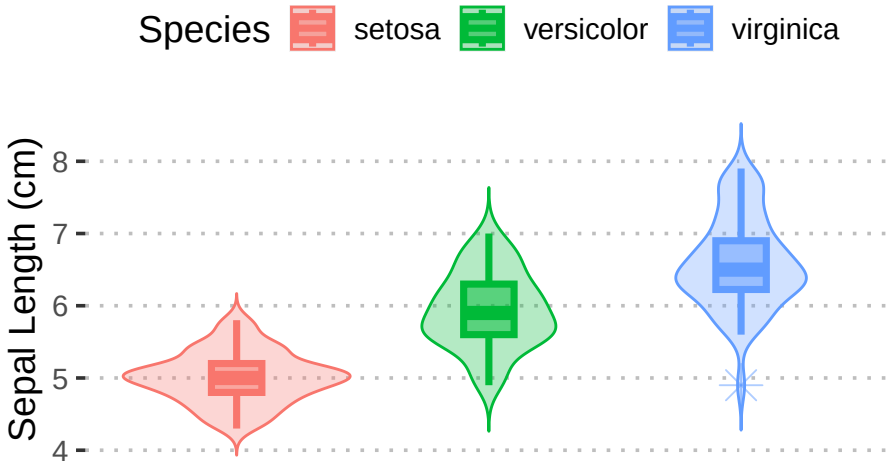
Boxplots can be dangerous in the sense that the exact distribution and the density of the observations are hidden behind the boxes. For example, the length of the whiskers could be due to one or two observations, and we would not know any better. To enhance the visualisation of boxplots, we can include the actual observation points, but in a jittery manner. Notice in the code below that we need to specify **Species** along the x-axis in *aes(.)* within the *ggplot(.)* function so that the jitter points can be correctly mapped to the boxplots.

```
g14 <- ggplot(iris,aes(x=Species,y=Sepal.Length,colour=Species,fill=Species)) +
         geom_boxplot(width=0.7,size=1.25,alpha=0.5,outlier.size=5,outlier.shape=8) +
         geom_jitter(size=1.5,position=position_jitter(0.1))

#Clean up the plot
g14.clean <- g14 +
              theme_pubclean(base_size=14) +
              labs(x="",y="Sepal Length (cm)") +
              theme(axis.ticks.x=element_blank(),
                    axis.text.x=element_blank()); g14.clean
```



Another way the enhance a boxplot is by including a violin plot. A violin plot is a rotated density plot on each side. To fit a boxplot within a violin plot, we can introduce the **width** argument to the *geom_boxplot(.)* function.

```
g15 <- ggplot(iris,aes(x=Species,y=Sepal.Length,colour=Species,fill=Species)) +
         geom_boxplot(width=0.2,size=1.25,alpha=0.5,outlier.size=5,outlier.shape=8) +
         geom_violin(alpha=0.3,trim=FALSE)

#Clean up the plot
g15.clean <- g15 +
              theme_pubclean(base_size=14) +
              labs(x="",y="Sepal Length (cm)") +
              theme(axis.ticks.x=element_blank(),
                    axis.text.x=element_blank()); g15.clean
```



Exercise: Plot the sepal **width** data for the three iris species on the same axes using boxplots, and overlay them with jitter points and violin plots.

5 Scatter Plot

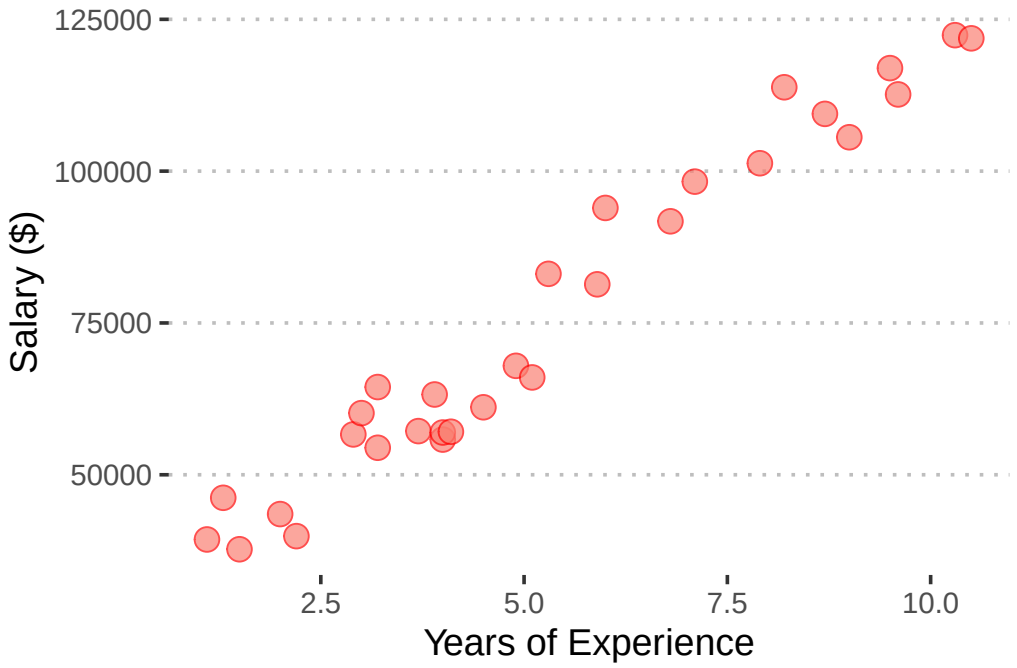
A scatter plot utilises Cartesian coordinates to display the values of two (or three) continuous variables. Scatter plots are a great way to visualise the relationship between the variables. Quite often with scatter plots, we able to ascertain the strength and the nature (e.g. linear or non-linear) of the relationships, and even outliers. This is extremely helpful in deciding the appropriate measures and models to use to describe the relationship.

Firstly, load the salary data set as follows.

```
##Import the salary dataset. Change the path if required.
Salary_Data <- read.csv("~/Desktop/Salary_Data.csv")
View(Salary_Data)
```

Next, we will use a scatter plot to illustrate the relationship between the two variable. Note that the predictor/feature is typically plotted along the x-axis and the outcome/response along the y-axis. In this instance, it makes sense that years of experience is a predictor of salary, and not the other way around.

```
g16 <- ggplot(Salary_Data,aes(x=YearsExperience,y=Salary)) +
         geom_point(colour="red",fill="salmon",shape=21,size=4,alpha=0.7)

#Clean up the plot
g16.clean <- g16 +
               theme_pubclean(base_size=14) +
               xlab("Years of Experience") +
               ylab("Salary ($)"); g16.clean
```



Exercise: Describe the relationship between years of experience and Salary. Is there relationship at all? If so, what is the nature of the relationship? Is it a strong relationship?

Enter text here

Exercise: Import the data from the *Friends.csv* file into R. The dataset has two variables, minutes/day spent online and number of friends. Visualise the two variables (on separate axes) using histograms. How may "outlier(s)" do you suspect there are here? Now plot the two variables against each other using a scatter plot. How many outlier(s) can you see now?

Enter text here

We will now look at another version of a scatter plot, and that is the bubble chart A bubble chart allows us

to incorporate a third numeric variable into a scatter plot, without having to resort to a 3-D scatter plot, which can be difficult to identity trends unless the viewing angle is adequate.

To illustrate this, we will use the New York air quality data, which were collected from May to September in 1973. You will see that there are missing values in the dataset.

```
data(airquality) #Load the air quality data
View(airquality)  #View the dataset
```

Exercise: For each of the variables in the dataset, determine the number and % of missing observations.

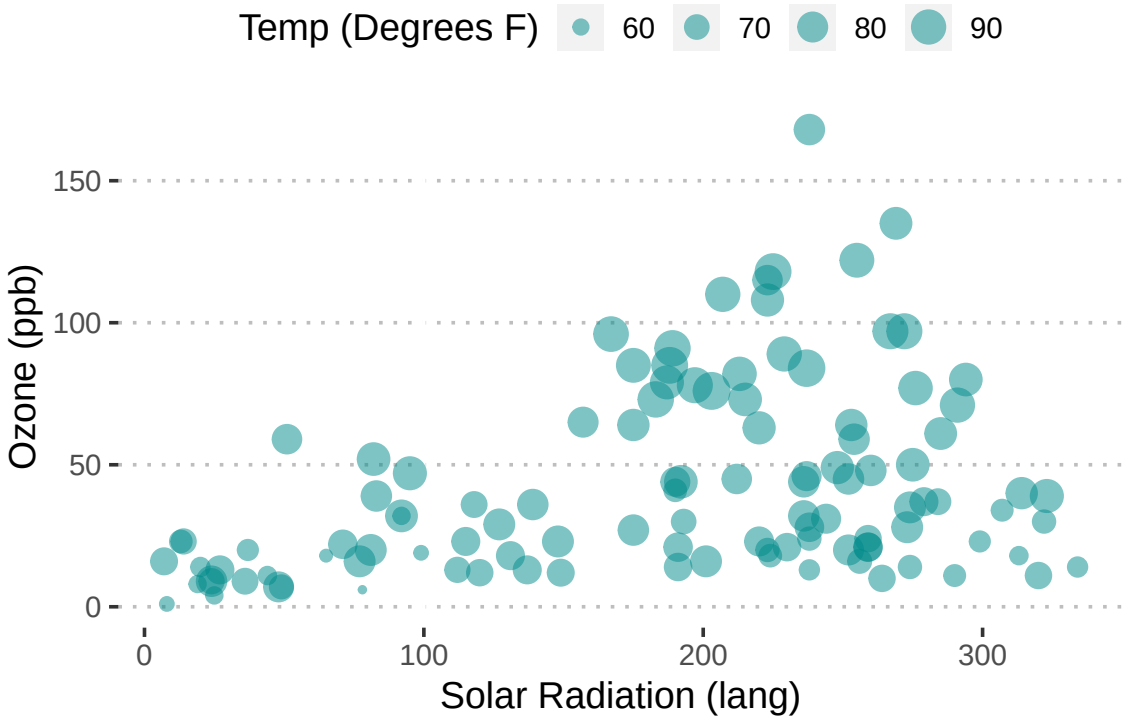
Enter text here

We will now proceed with complete cases only, and eliminate any row with at least one missing value.

```
aq <- na.omit(airquality)  #Remove all missing cases
```

Let us now create a bubble chart with Ozone (y) vs Solar Radiation(x), with Temperature represented by the bubble size.

```
g17 <- ggplot(aq,aes(x=Solar.R,y=Ozone)) +
         geom_point(aes(size=Temp),colour="cyan4",alpha=0.5)

g17.clean <- g17 +
               theme_pubclean(base_size=14) +
               xlab("Solar Radiation (lang)") +
               ylab("Ozone (ppb)") +
               labs(size="Temp (Degrees F)"); g17.clean
```



Exercise: Describe the relationship between Ozone with Solar R and Temp.

Enter text here

6 Line Chart (Optional)

Note: This section is optional for this unit

A line chart displays the progression of one or more numeric variables. It's similar to a scatter plot, except that the points are connected by line segments, and the measurement points along the x-axis are (typically) ordered. For example, time-series data are best presented with a line chart, where time is plotted along the x-axis.

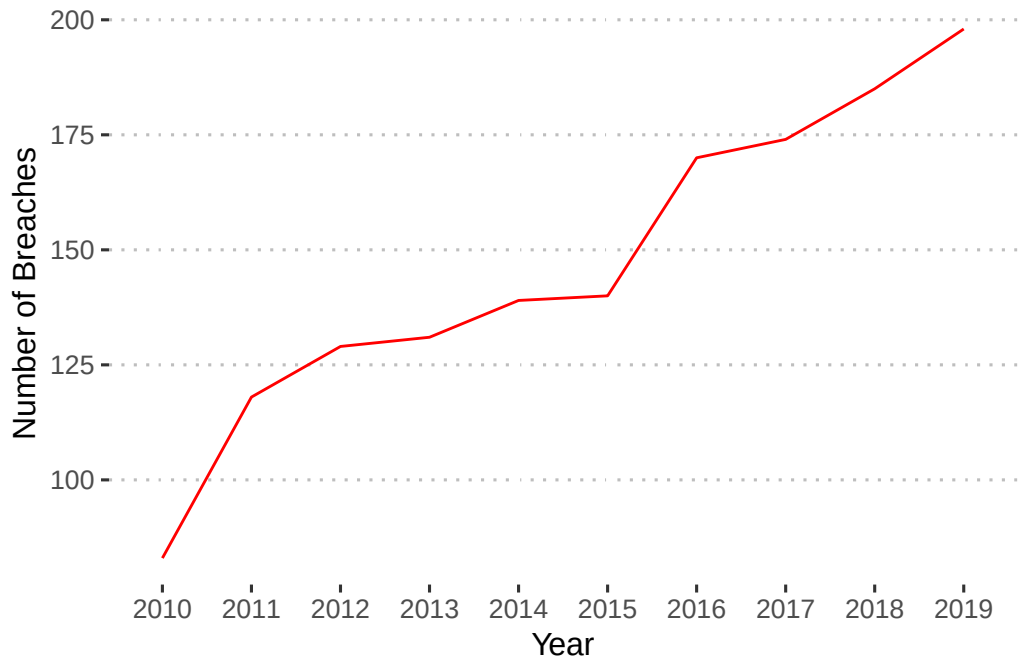
Let us create some fictitious data relating to the number of data breaches across all businesses in Australia in the past 10 years.

```
set.seed(12)
Year <- c(2010:2019);
Australia <- sample(c(50:200),size=10,replace=FALSE) %>%
               sort(.,decreasing=FALSE)
```

```
df.breaches <- data.frame(Year,Australia); View(df.breaches)
```

Now, let us generate the line chart for this dataset. Note that we need to convert **Year** to a factor, otherwise the x-axis will be labelled like a numeric scale. By doing so, we will need to specify the **group** argument in *aes(.)* and set it to 1.

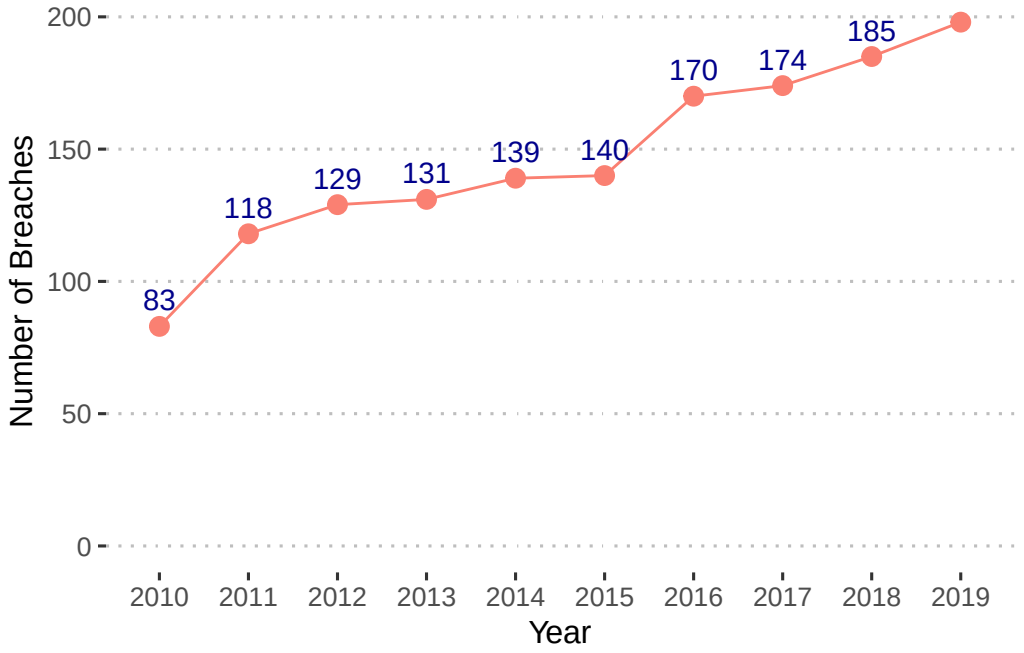
```
g18 <- ggplot(df.breaches,aes(x=as.factor(Year),y=Australia,group=1)) +
         geom_line(colour="red")

g18.clean <- g18 +
               theme_pubclean() +
               labs(x="Year",y="Number of Breaches"); g18.clean
```



We will now overlay the line chart with a scatter plot and add some annotations.

```
g19 <- ggplot(df.breaches,aes(x=as.factor(Year),y=Australia,group=1)) +
         geom_line(colour="salmon") +
         geom_point(colour="salmon",size=3) +
         annotate("text",  #Annotate plot with texts
                  x=as.factor(Year),   #x-position of text
                  y=Australia+10,  #y-position of text
                  label=Australia, #text labels
                  colour="blue4") #Colour of text labels

g19.clean <- g19 +
               theme_pubclean() +
```

```
            labs(x="Year",y="Number of Breaches") +
            ylim(0,200); g19.clean
```

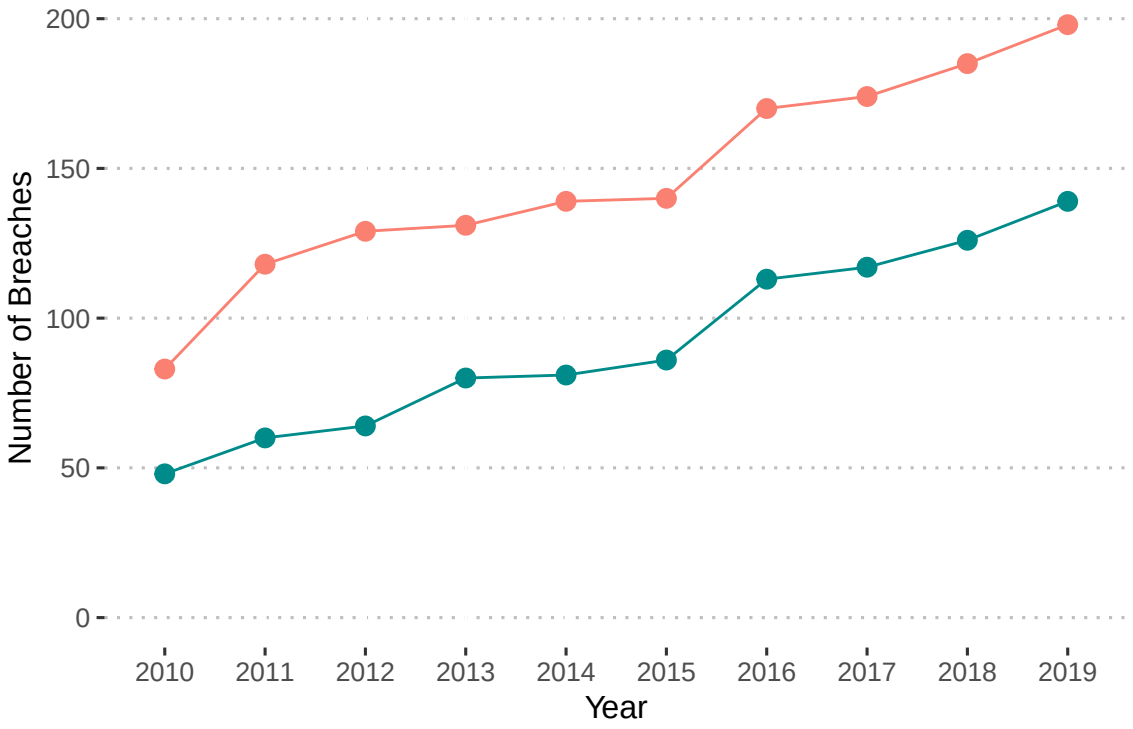


Suppose we wish to compare the Australia data to another country, say Canada.

```
set.seed(20)
df.breaches$Canada <- sample(c(20:150),size=10,replace=FALSE) %>%
                      sort(.,decreasing=FALSE)  #Fictitious data for Canada
View(df.breaches)
```

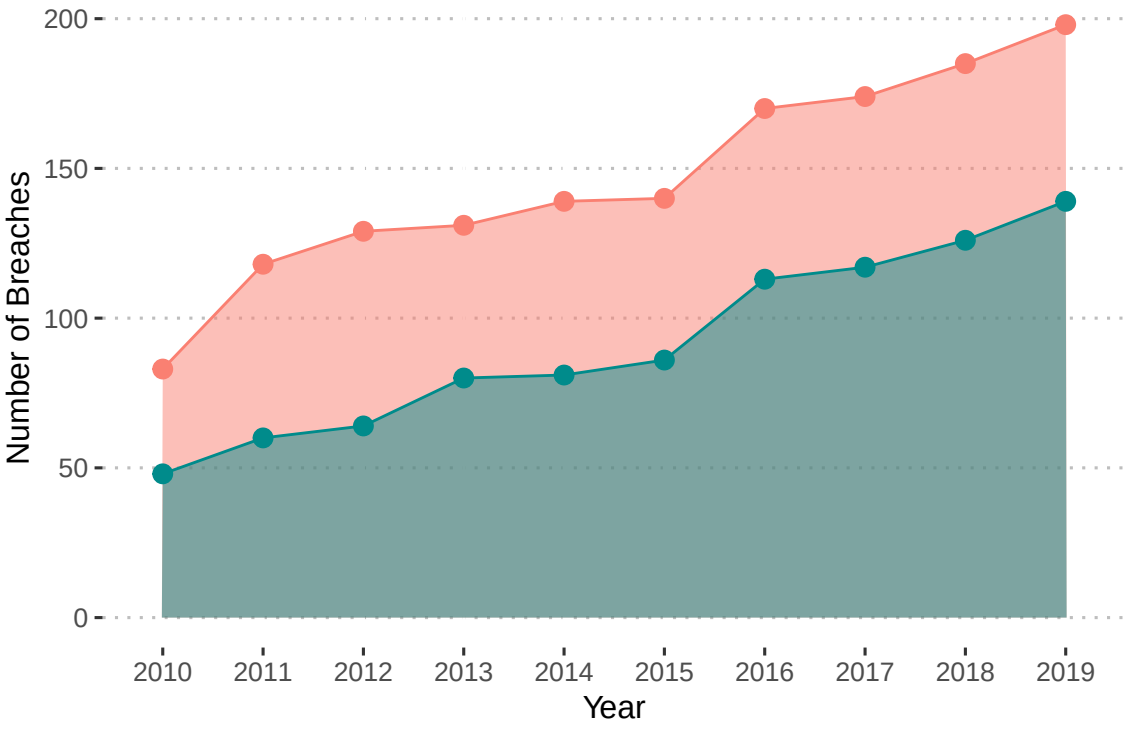
Now we can plot the Australian and Canadian data on the same axes.

```
g20 <- ggplot(df.breaches,aes(x=as.factor(Year),group=1)) +
         geom_line(aes(y=Australia),colour="salmon") +
         geom_point(aes(y=Australia),colour="salmon",size=3) +
         geom_line(aes(y=Canada),colour="cyan4") +
         geom_point(aes(y=Canada),colour="cyan4",size=3)

g20.clean <- g20 +
               theme_pubclean() +
               labs(x="Year",y="Number of Breaches") +
               ylim(0,200); g20.clean
```



We can further enhance the line chart by overlaying it with an area plot.

```
g21 <- ggplot(df.breaches,aes(x=as.factor(Year),group=1)) +
         geom_line(aes(y=Australia),colour="salmon") +
         geom_point(aes(y=Australia),colour="salmon",size=3) +
         geom_area(aes(y=Australia),fill="salmon",alpha=0.5) +
         geom_line(aes(y=Canada),colour="cyan4") +
         geom_point(aes(y=Canada),colour="cyan4",size=3) +
         geom_area(aes(y=Canada),fill="cyan4",alpha=0.5,density=50)

g21.clean <- g21 +
               theme_pubclean() +
               labs(x="Year",y="Number of Breaches") +
               ylim(0,200); g21.clean
```



7 Faceting in R (Optional)

Note: This section is optional for this unit

ggplot2 has a special function called faceting that allows the user to split one plot into multi-panel plots based on a factor include in the dataset.

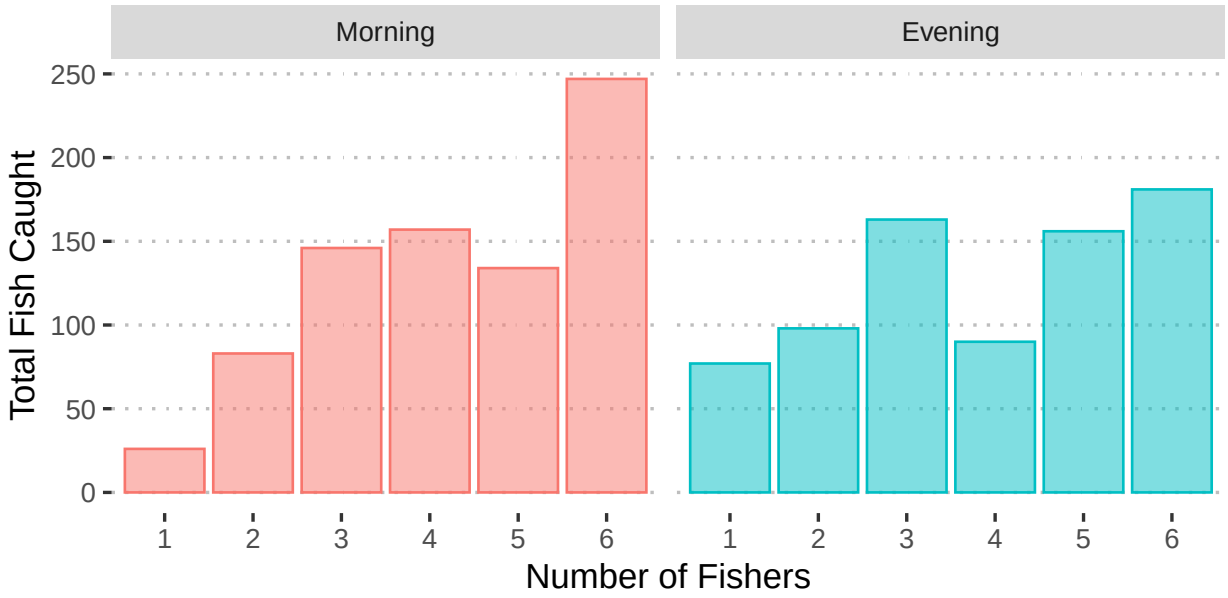
There are two types of facet functions in **ggplot2**:

- facet_wrap(): arranges a one-dimensional sequence of panels to allow them to cleanly fit on one page.
- facet_grid(): allows you to form a matrix of rows and columns of panels.

Both geometries allow to to specify faceting variables specified within *vars(.)*. For example, *facet_wrap(facets = vars(facet_variable))* or *facet_grid(rows = vars(row_variable), cols = vars(col_variable))*.

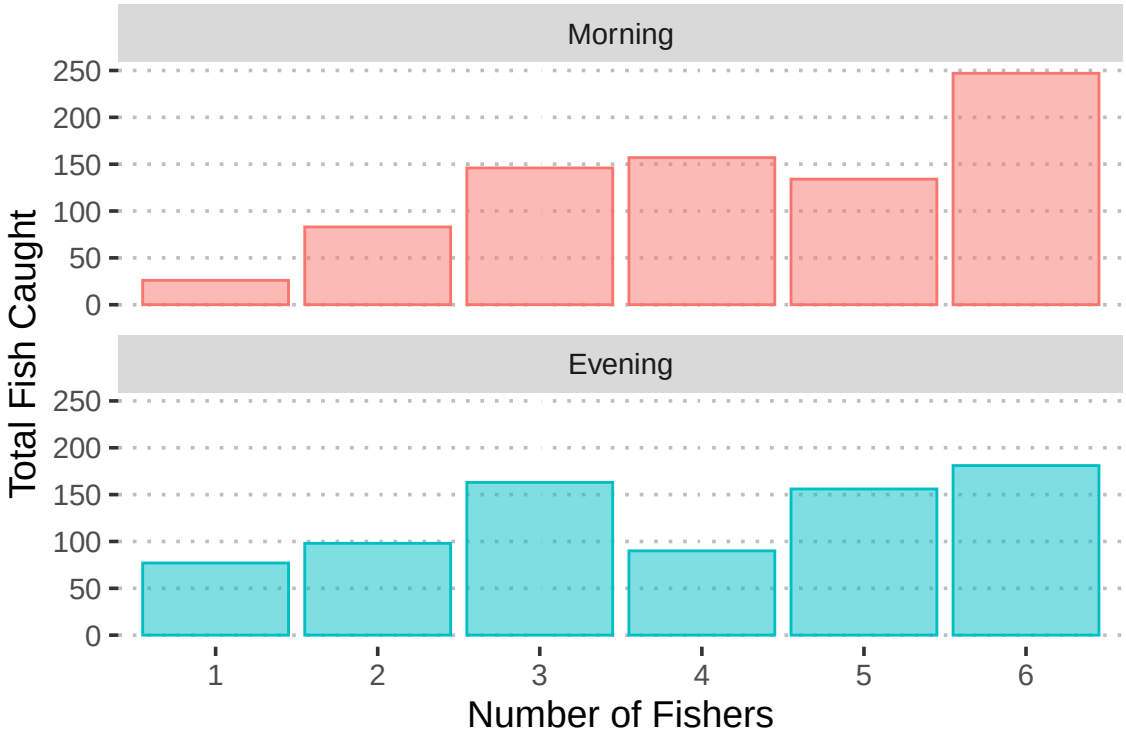
Let us revisit the fish data, and specifically the *g4.clean*. Suppose we want to present the data for **Morning** and **Afternoon** on separate axes. We can do this easily by using one of the faceting functions.

```
g4.clean +
  facet_wrap(vars(ToD)) +
  theme(legend.position="")  #Remove legend as it is now redundant
```



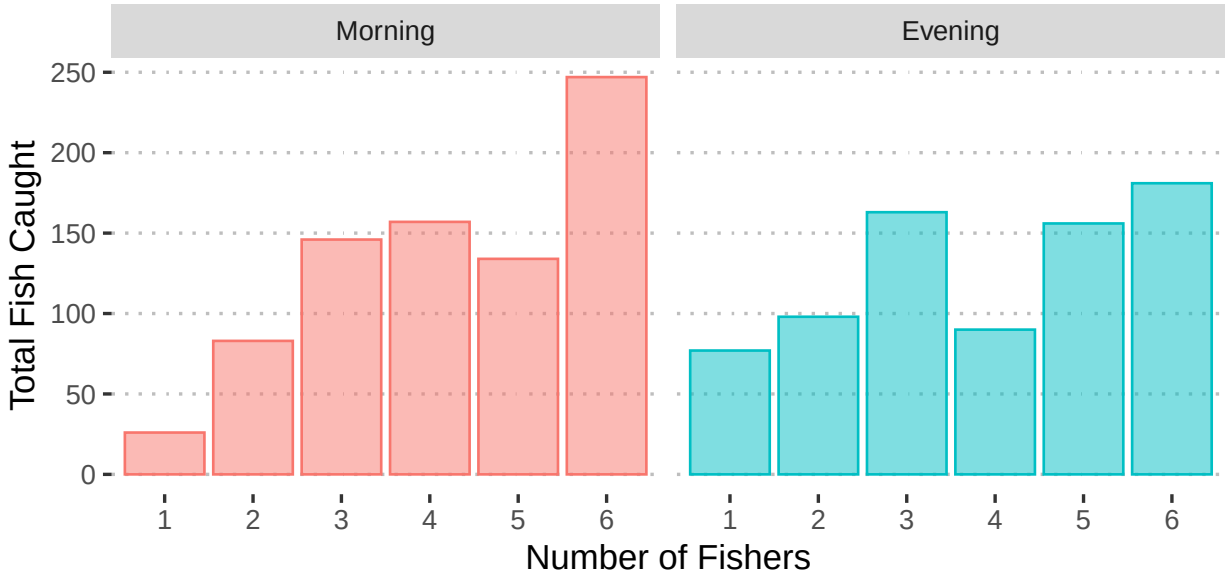
Notice how the panels are arranged horizontally to fit the screen. If you want to arrange them vertically, then we just need to specify the **nrow** argument.

```
g4.clean +
  facet_wrap(vars(ToD),nrow=2) +
  theme(legend.position="")  #Remove legend as it is now redundant
```

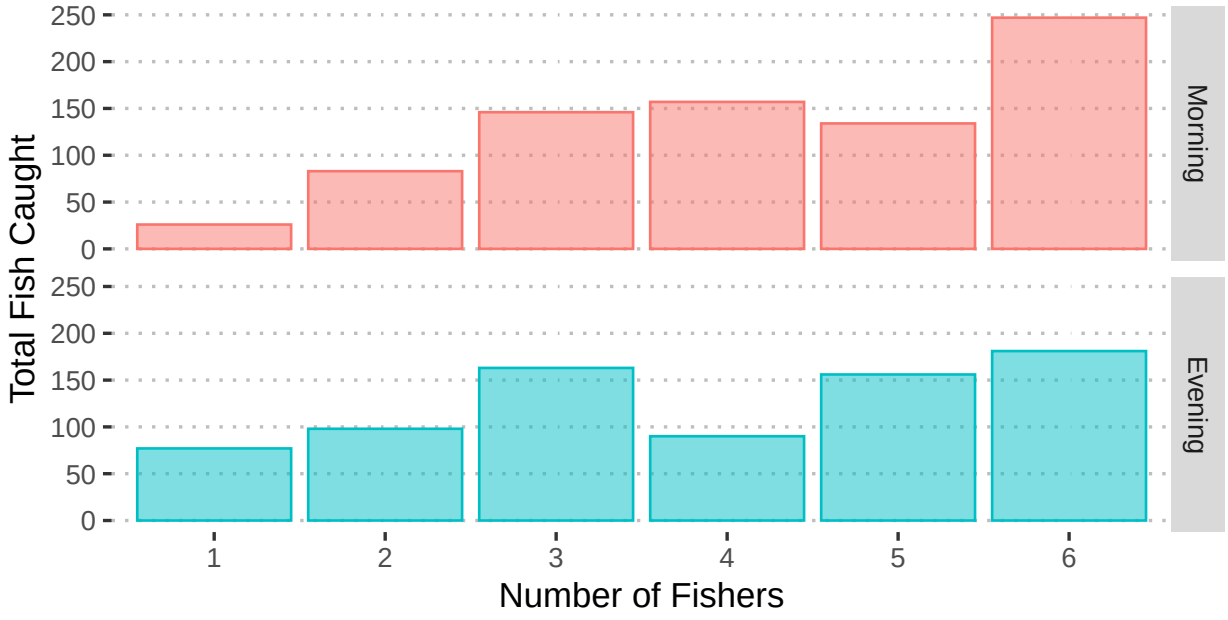


We can get the same result with the *face_grid(.)* function, although when arranged vertically, the panel strips are positioned differently.

```
g4.clean +
  facet_grid(col=vars(ToD)) +
  theme(legend.position="")  #Remove legend as it is now redundant
```

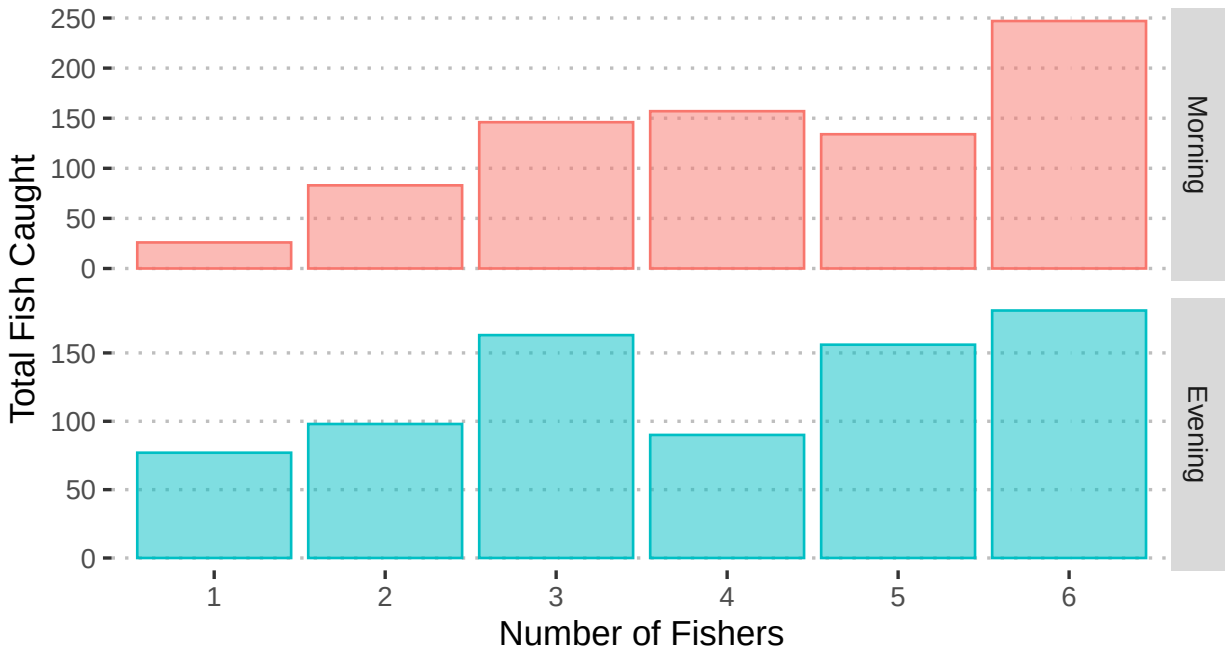



```
g4.clean +
  facet_grid(row=vars(ToD)) +
  theme(legend.position="")  #Remove legend as it is now redundant
```



By default, all the panels have the same scales (i.e. **scales="fixed"**). They can be made independent, by setting scales to **free**, **free_x**, or **free_y**.

```
g4.clean +
  facet_grid(row=vars(ToD),scales="free") +
  theme(legend.position="")  #Remove legend as it is now redundant
```



Now the range on y-axis are different for **Morning** and **Evening**.

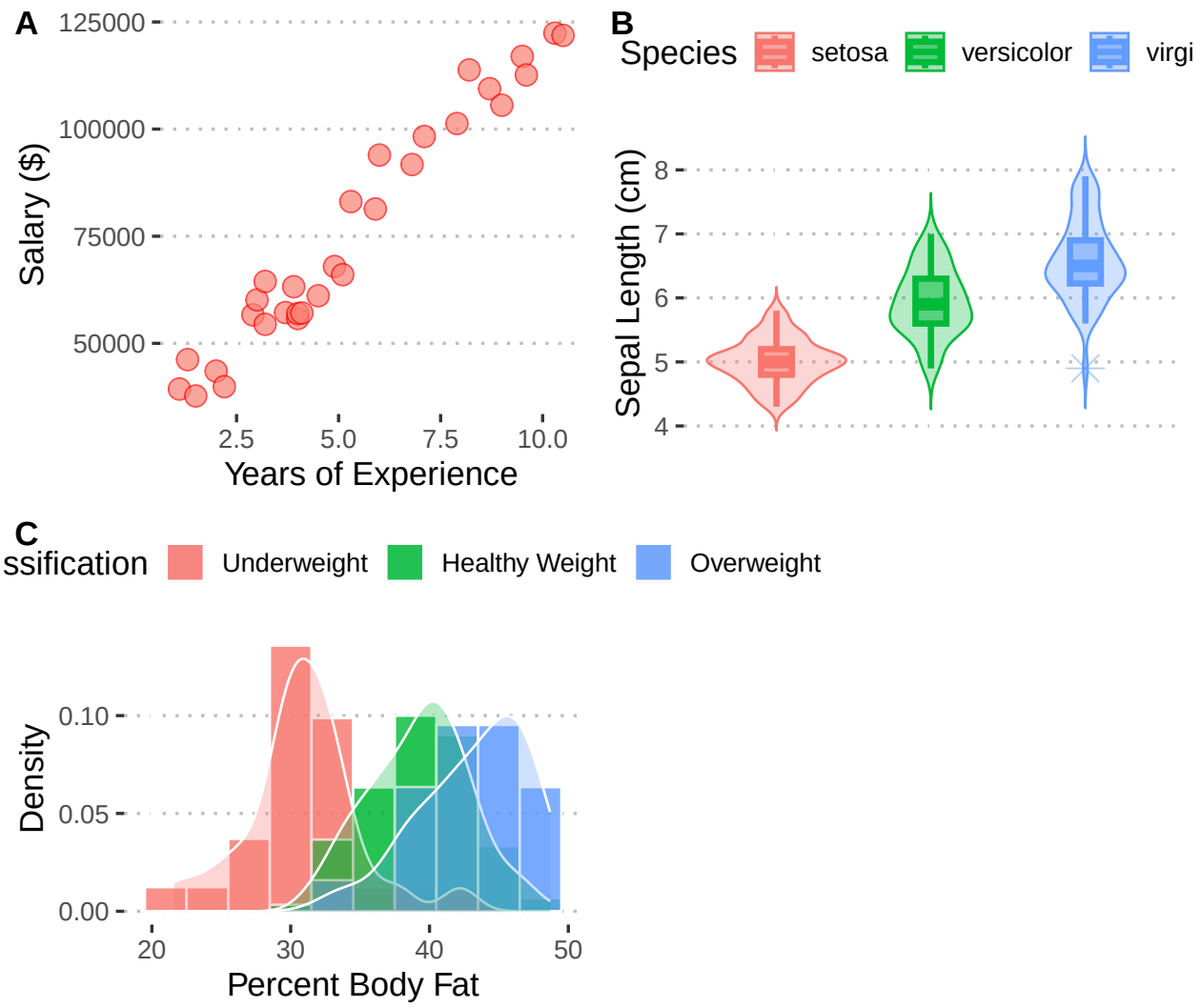
8 Arranging and Exporting GG-Plots

In many instances, you may want to combine several different plots together to form a single plot, i.e. for publication purpose. This can be achieved by using the *ggarrange(.)* function. We can later export the plot and save it as, say a *pdf* file, using the *ggexport(.)* function. Note that both of these functions are from the **ggpubr** package.

8.1 Arranging and Combining Multiple GG-Plots

We will now arrange three of the previously generated plots into one figure, namely **g11.clean**, **g15.clean** and **g16.clean**.

```
ggarrange(g16.clean,g15.clean,g11.clean,
          labels = c("A", "B", "C"),
          ncol = 2, nrow = 2);
```

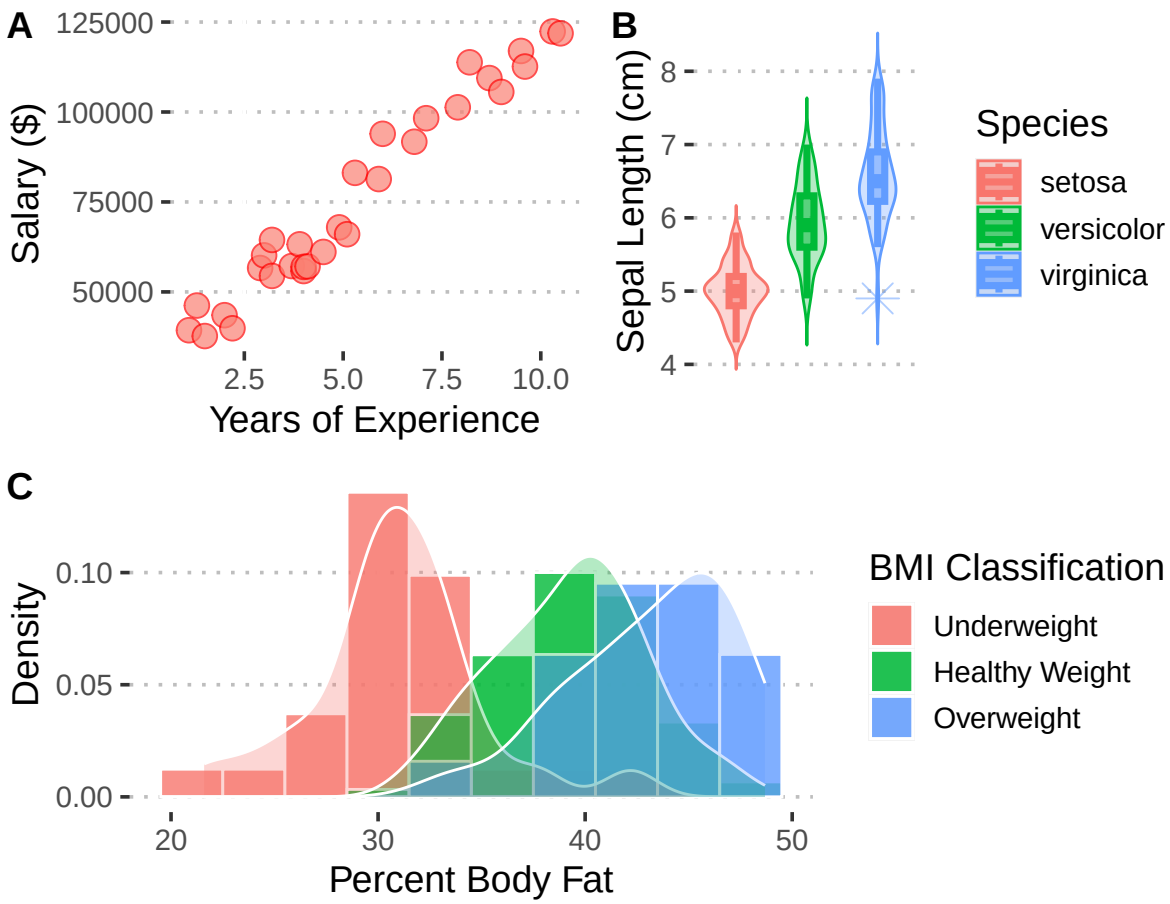


Not too bad, but the above plot can be improved. We will change the following.

1. Set the legend to the right of the Plots B and C;
2. Have Plot C span across the bottom row panel.

To achieve this, we will firstly combine Plots A and B together, then combined them with Plot C.

```
multiplot1 <- ggarrange(g16.clean,
                        g15.clean + theme(legend.position="right"),
                        labels = c("A", "B"),
                        ncol = 2, nrow = 1);

multiplot2 <- ggarrange(multiplot1,
                        g11.clean + theme(legend.position="right"),
                        labels=c("","C"),nrow=2)
multiplot2
```



If the plots share a common legend then you can specifying the **common.legend** argument to **TRUE**. You can also set the position of this common legend by additionally specifying the **legend=DIRECTION** argument. The **DIRECTION** can be either **"top"**,**"bottom"**, **"left"** or **"right"**.

Exercise: Combine the plots **g8.clean** and **g10.clean** (side-by-side) into a single figure, and display the common legend at the top of the plot.

8.2 Exporting GG-Plots

Using the *ggexport(.)* command, you can export a single plot onto one page or multiple plots across multiple pages. The commands below will export the files into your working directory by default.

```
#Single plot
ggexport(multiplot2,filename="Multiplots in one.pdf")

#Multiple plots across two pages, with 2 plots per page.
ggexport(g1.clean,g2.clean,g3.clean,g4.clean,  #4 plots
         nrow=2,ncol=1,  #Arrange them vertically with 2 per page
         filename="Multiplots.pdf")
```