Lab 3 (Part 1) - Graphing and Plots in R

Contents

Lab Objectives

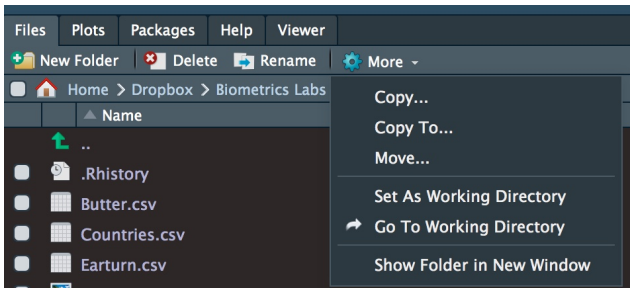
- Create a variety of graphs in R
- Customise aspects of your plots
- Exporting and saving plots and data

Set your working directory

Before getting started, set your working directory to where the course files are located on your computer. The code below is commented out, since you will have to provide your own working directory.

```
# setwd('Path to your files') # i.e.
# C:/Users/YourName/Documents/Biometrics Labs/ note: this
# depends on your computer's OS
```

In RStudio, you can also change your working directory using the graphical interface:



Exercise 1: Graphing in R

R has the capability to create many types of charts, although producing these graphs requires a modicum of coding. The base R installation has sufficient capacity for graphing, although additional and more powerful graphing packages can be installed to produce sophisticated plots suitable for publication and presentation. In this exercise, we will focus on using base R plotting capacities. We'll begin with frequency distributions.

Recall that a frequency distribution plots the number of occurrences or counts for each value of a single variable. For categorical variables a frequency distribution is displayed in a bar graph, while for numerical variables a frequency distribution is displayed in a histogram.

Bar graphs

Let's start by looking at the frequency of social problems from the data in the file Appendix D. This file contains a variety of information about a group of individuals, including their score on an ADD test, an IQ test, their GPA in grade 9, whether they were observed to have social problems, and whether they went on to drop out of high school.

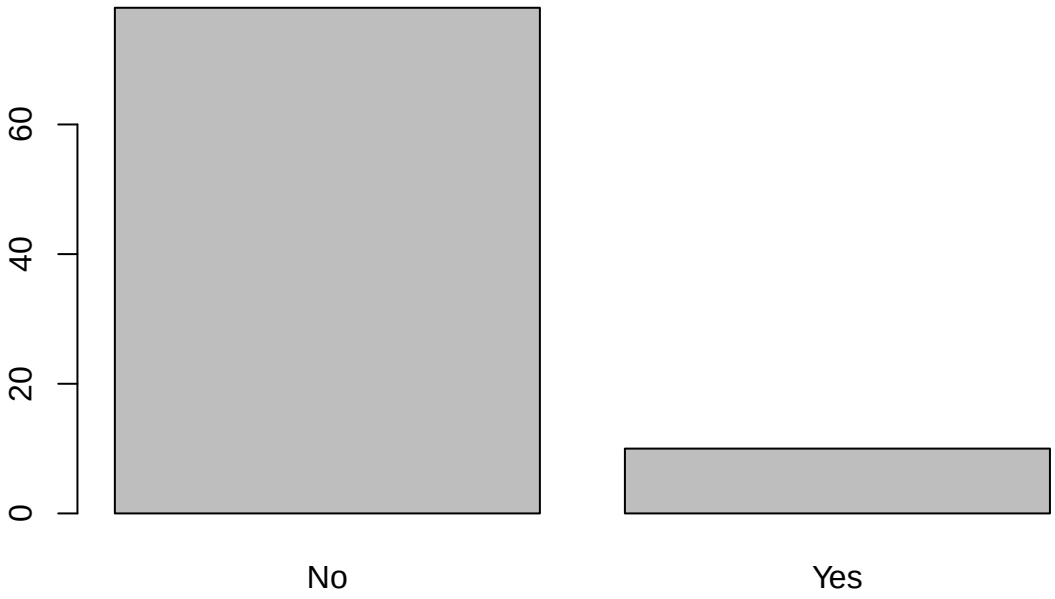
Open appendixd.csv:

```
d <- read.csv("appendixd.csv")
str(d)
```

```
## 'data.frame':    88 obs. of  10 variables:
##  $ ID     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ addsc  : int  45 50 49 55 39 68 69 56 58 48 ...
##  $ gender : Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 ...
##  $ repeat.: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 2 2 1 1 1 ...
##  $ iq     : int  111 102 108 109 118 79 88 102 105 92 ...
##  $ engl   : int  2 2 2 2 2 2 2 2 3 2 ...
##  $ engg   : int  3 3 4 2 3 2 2 4 1 4 ...
##  $ gpa    : num  2.6 2.75 4 2.25 3 1.67 2.25 3.4 1.33 3.5 ...
##  $ socprob: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 2 1 1 1 ...
##  $ dropout: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 2 2 1 1 1 ...
```

Focus on social problems data, which appears as a series of "No" and "Yes" values. With this kind of categorical data, R readily counts and interprets these when you call a simple plot:

```
plot(d$socprob)
```

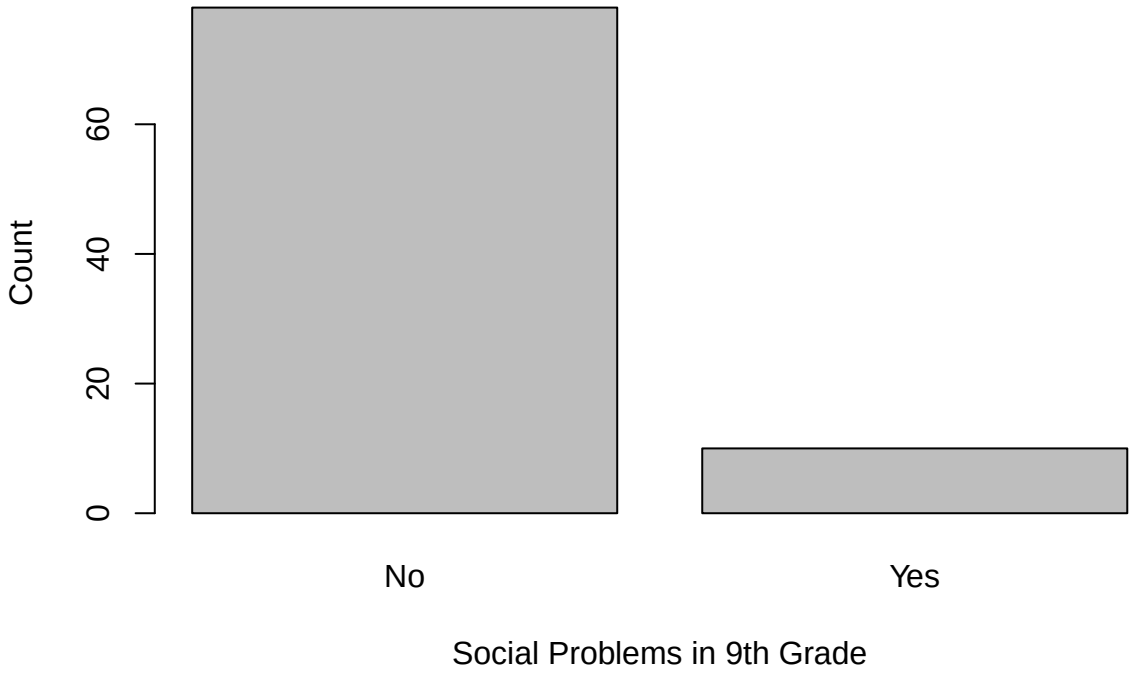


By default, the graph is poorly labelled, other than the two categories, Yes and No. On the y axis is the count of students belonging to each category.

The **plot()** function has many potential settings you can adjust. You can try typing ?plot in the R console to see what types of settings are available. The list is long and far too detailedd for us to cover here. But we will explore some basic settings.

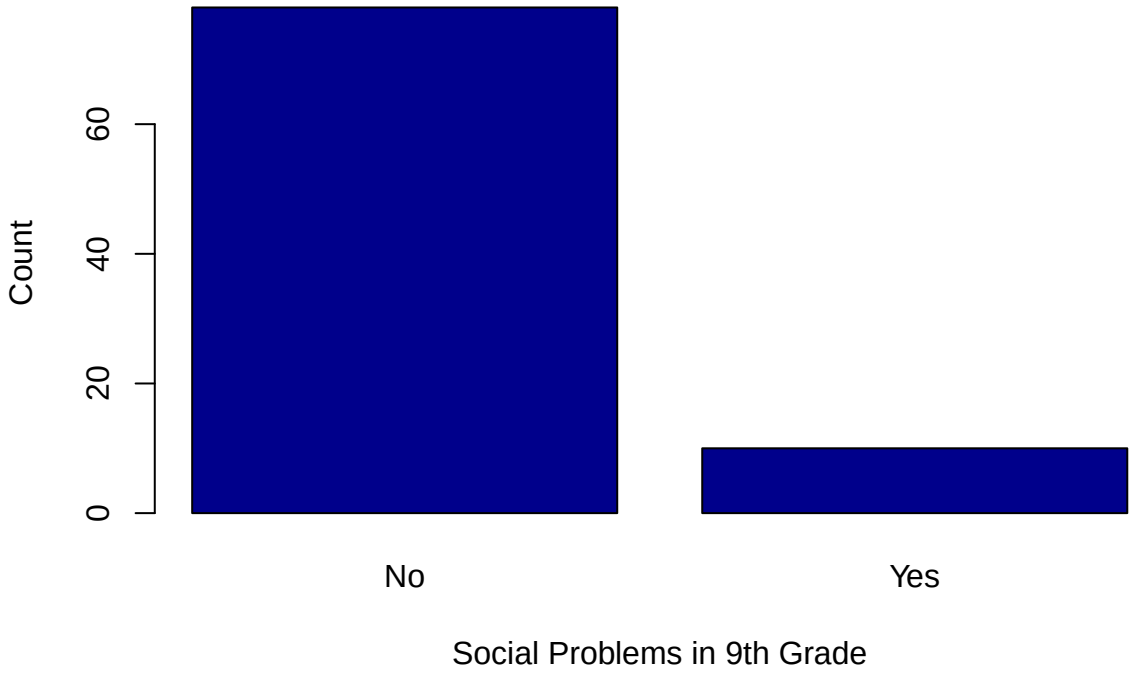
The Y- axis should indicate "Count". Re-plot it but this time provide a label for the y-axis and x-axis by specifying values for **ylab** and **xlab**:

```
plot(d$socprob, ylab = "Count", xlab = "Social Problems in 9th Grade")
```



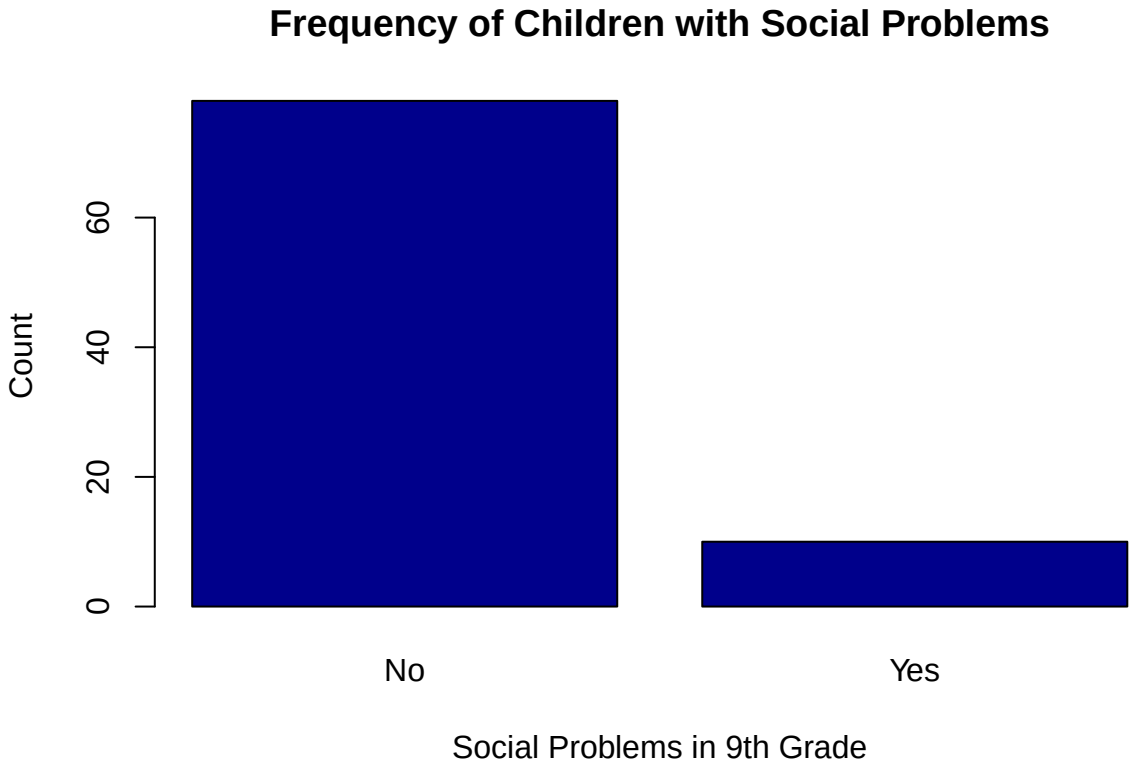
To change the colour of the bars, use the **col** option:

```
plot(d$socprob, ylab = "Count", xlab = "Social Problems in 9th Grade",
    col = "dark blue")
```



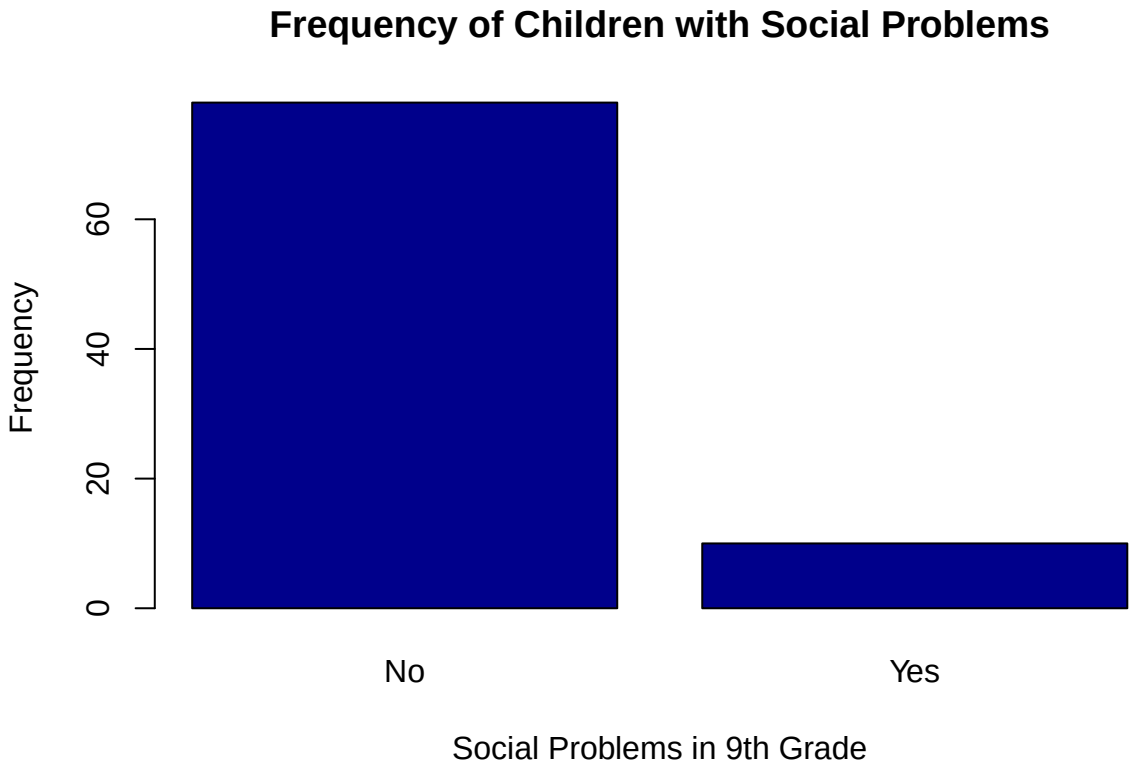
To add a title, use the **title** function:

```
plot(d$socprob, ylab = "Count", xlab = "Social Problems in 9th Grade",
    col = "dark blue")
title("Frequency of Children with Social Problems")
```



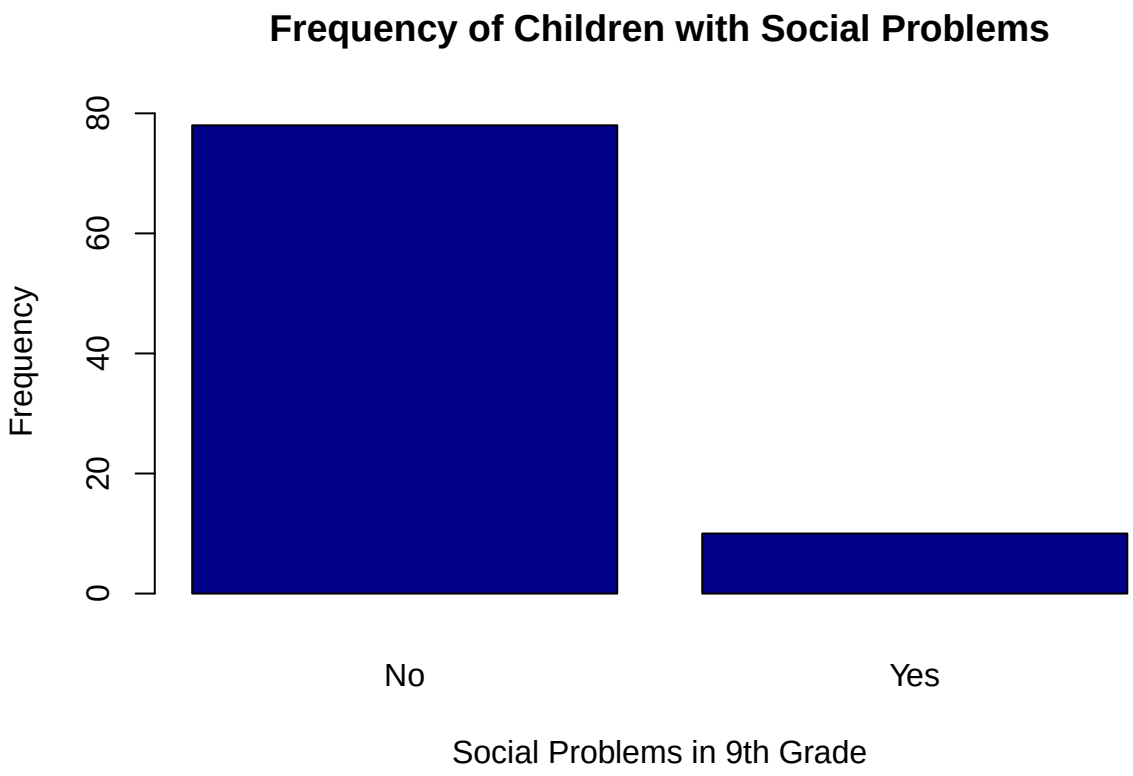
Now, change the title of the Y-axis from Count to Frequency, again by changing the **ylab** setting.

```
plot(d$socprob, ylab = "Frequency", xlab = "Social Problems in 9th Grade",
    col = "dark blue")
title("Frequency of Children with Social Problems")
```



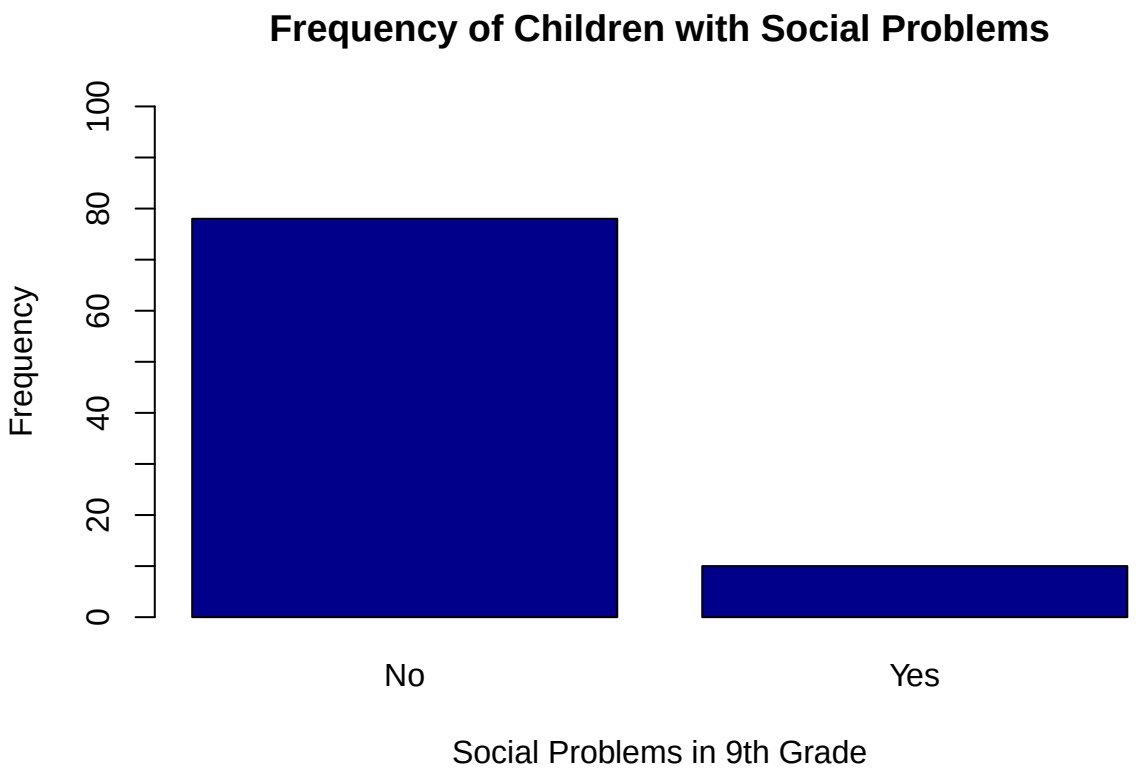
Sometimes, the default axis style in R produces an axis too small relative to the range of the data, producing unusual graphs. This can be changed by altering the **yaxs** = "r", where "r"" refers to a regular axis style:

```
plot(d$socprob, ylab = "Frequency", xlab = "Social Problems in 9th Grade",
    col = "dark blue", yaxs = "r")
title("Frequency of Children with Social Problems")
```



Now, adjust the axis so it goes from 0 to 100 and change the major increment to be 10, using the **yaxp** setting:

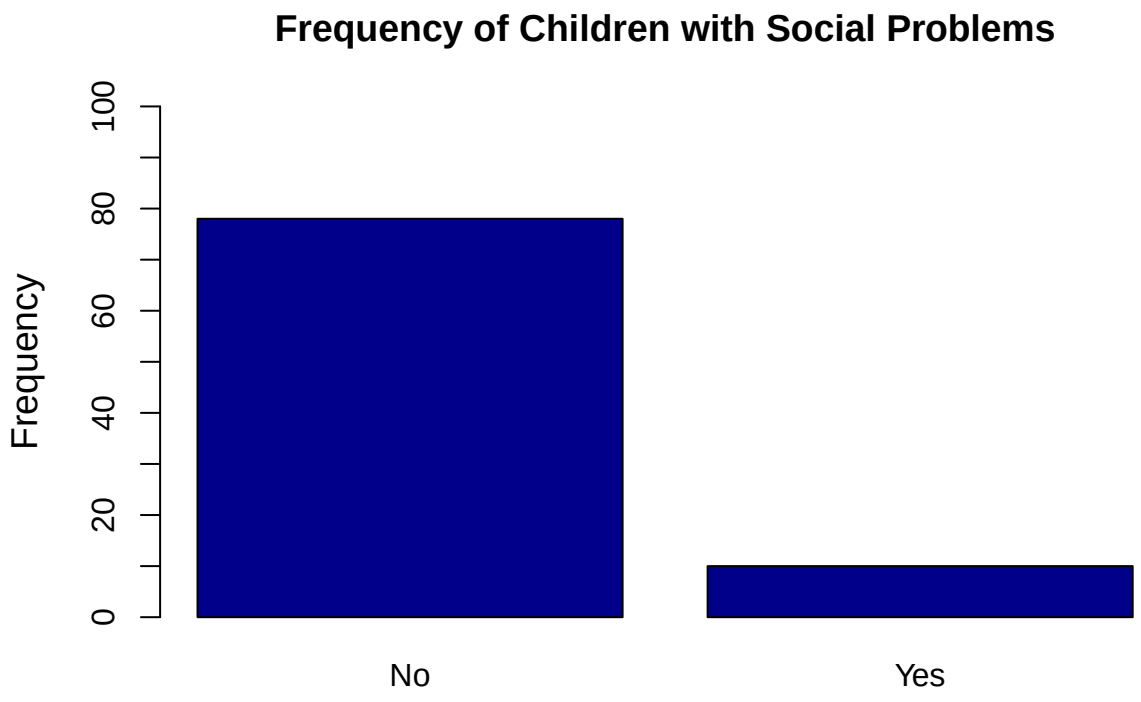
```
plot(d$socprob, ylab = "Frequency", xlab = "Social Problems in 9th Grade",
    col = "dark blue", yaxt = "n", ylim = c(0, 100), yaxp = c(0,
        100, 10))
axis(2, at = seq(0, 100, 10), labels = seq(0, 100, 10))
title("Frequency of Children with Social Problems")
```



Adjusting font size in R plots is tricky, because you are working with plots that are sized relative to the graphics window on your particular computer. The size if not "fixed" until the plot is saved to an external file.

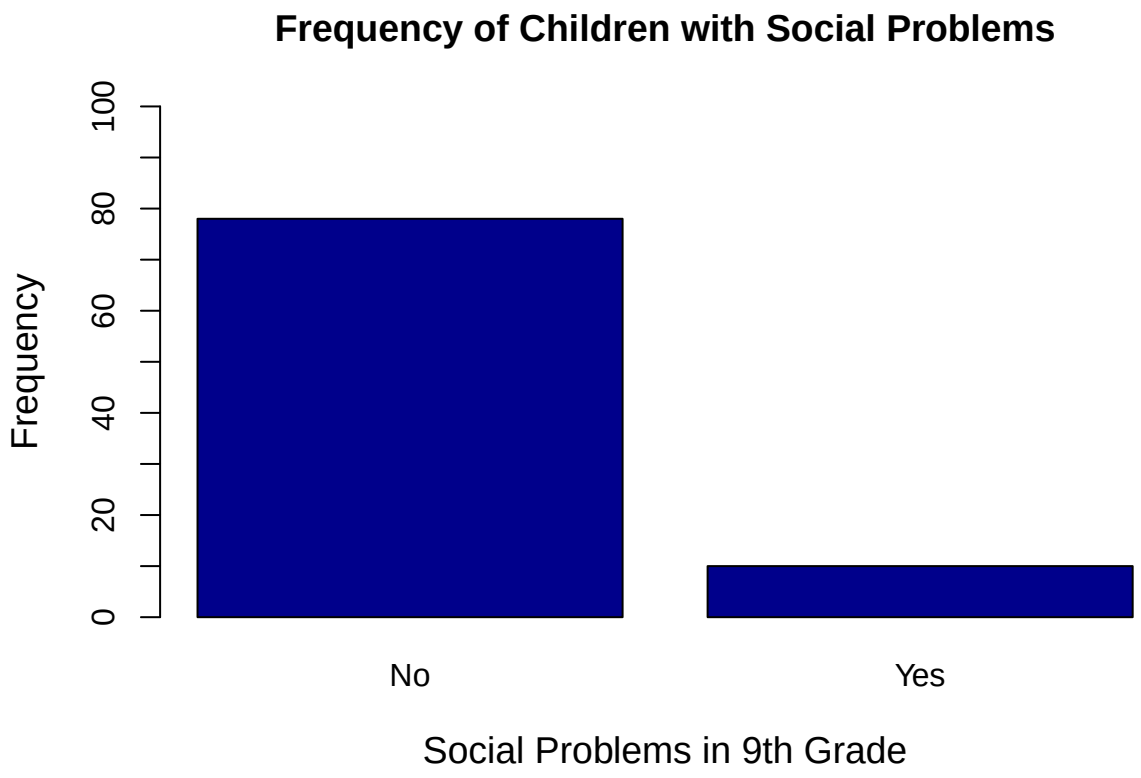
Base R plots allow you to set font sizes in a relative fashion using settings like **cex.lab**, for example, to change the relative size of the axis title fonts. Change the font size of your two axis titles to be 120% of the default font size by setting to 1.2 and the axis labels to 80% of default by changing the **cex.axis** setting:

```
plot(d$socprob, ylab = "Frequency", xlab = "Social Problems in 9th Grade",
    col = "dark blue", yaxt = "n", ylim = c(0, 100), yaxp = c(0,
        100, 10), cex.lab = 1.2, cex.axis = 0.8)
axis(2, at = seq(0, 100, 10), labels = seq(0, 100, 10))
title("Frequency of Children with Social Problems")
```



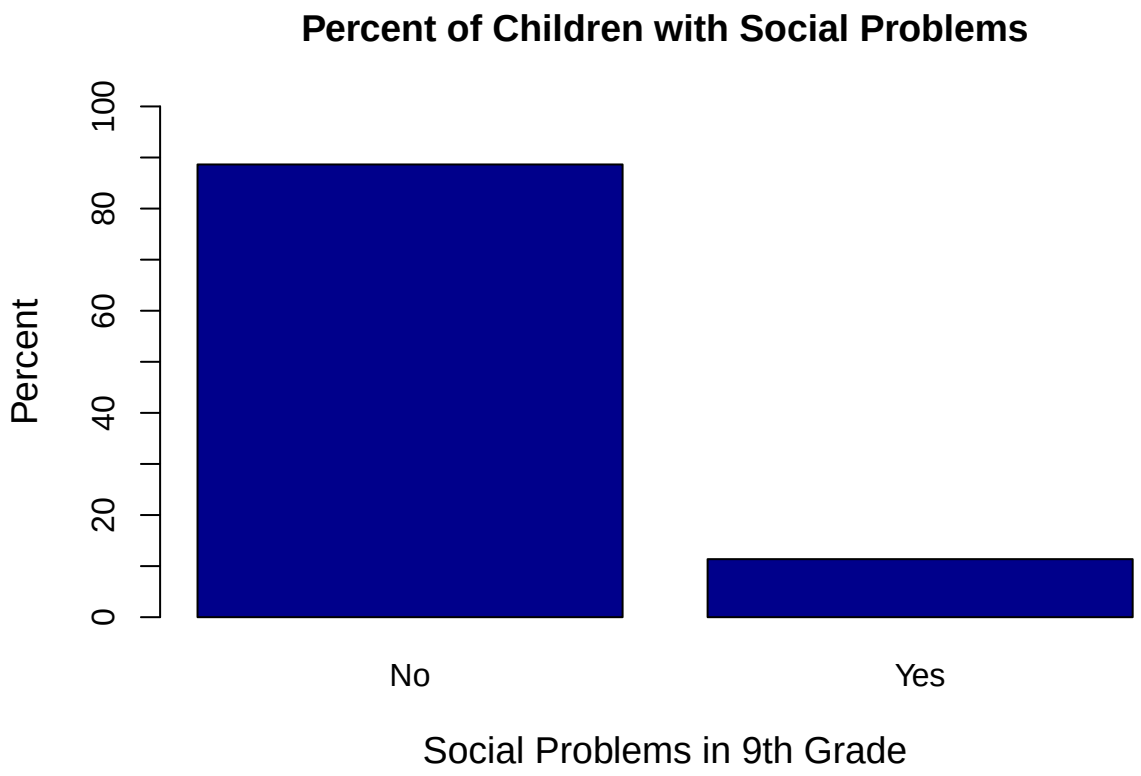
So far, we did this plot with data stored in our data.frame. But what if you already have the counts stored in a variable and you don't want to have to put them in a file? You could simply define your own data variable, and use the **barplot()** function, which specifically plots bar charts from the numbers provided:

```
dat <- cbind(No = 78, Yes = 10)

barplot(dat, ylab = "Frequency", xlab = "Social Problems in 9th Grade",
    col = "dark blue", yaxt = "n", ylim = c(0, 100), yaxp = c(0,
        100, 10), cex.lab = 1.2, cex.axis = 0.8)
axis(2, at = seq(0, 100, 10), labels = seq(0, 100, 10))
title("Frequency of Children with Social Problems")
```



The advantage of this is that you can re-plot your values as percentages by simply creating a new variable that performs simple mathematical calculations by dividing your counts by the totals and multiplying by 100:

```
dat100 <- 100 * dat/(78 + 10)

barplot(dat100, ylab = "Percent", xlab = "Social Problems in 9th Grade",
    col = "dark blue", yaxt = "n", ylim = c(0, 100), yaxp = c(0,
        100, 10), cex.lab = 1.2, cex.axis = 0.8)
axis(2, at = seq(0, 100, 10), labels = seq(0, 100, 10))
title("Percent of Children with Social Problems")
```

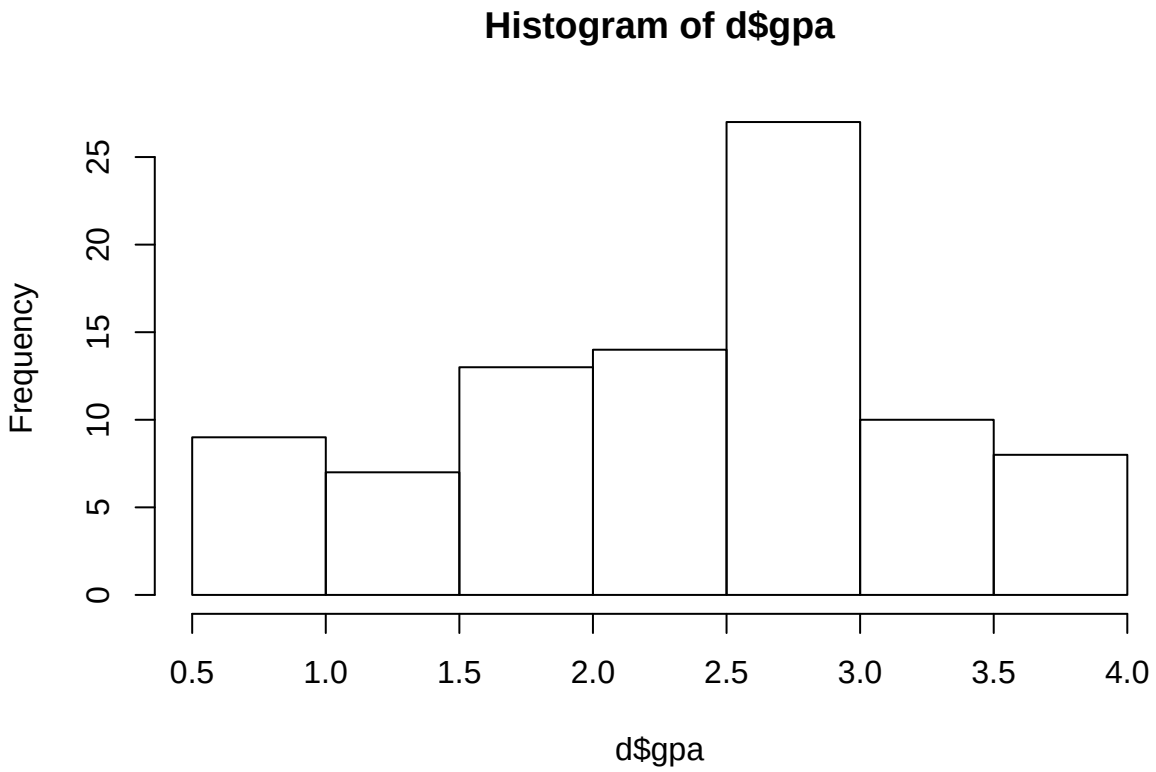


Check that your final graph looks like the one above before proceeding.

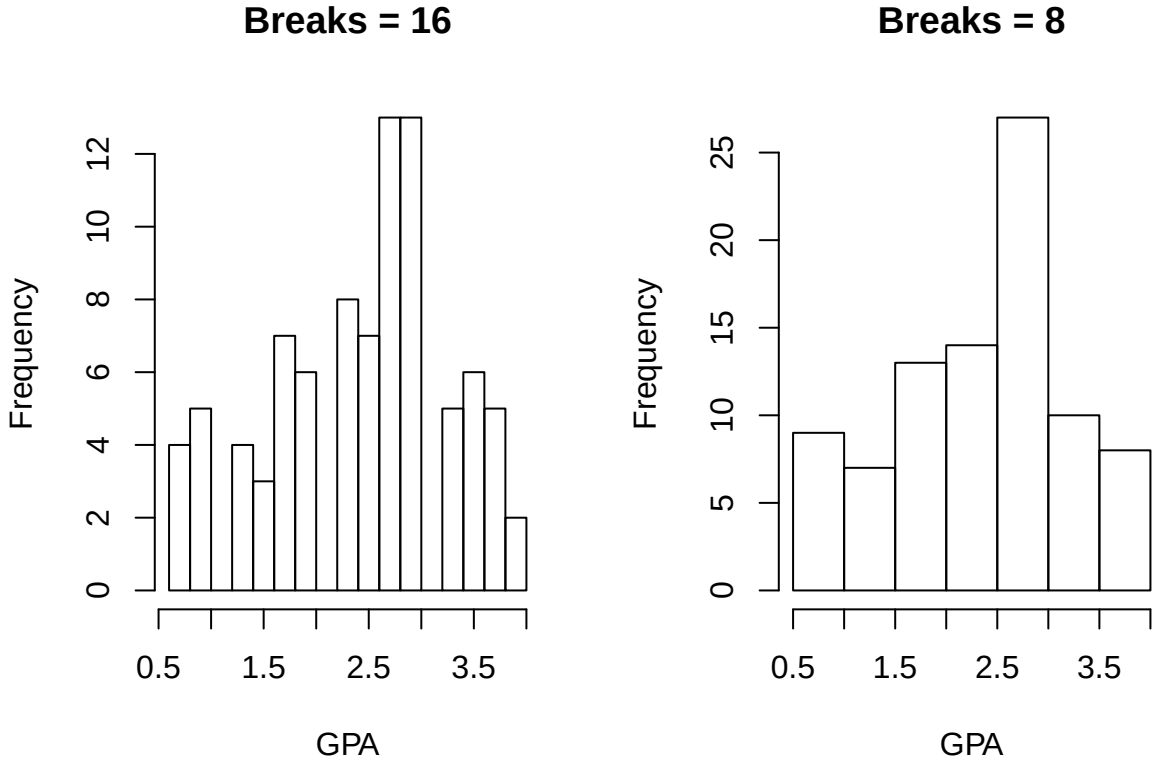
Histograms

Histograms are essentially a frequency distribution for ranges of values rather than individual values. A histogram is a perfect way to display the GPA data from the Appendix D file, loaded earlier.

```
hist(d$gpa)
```

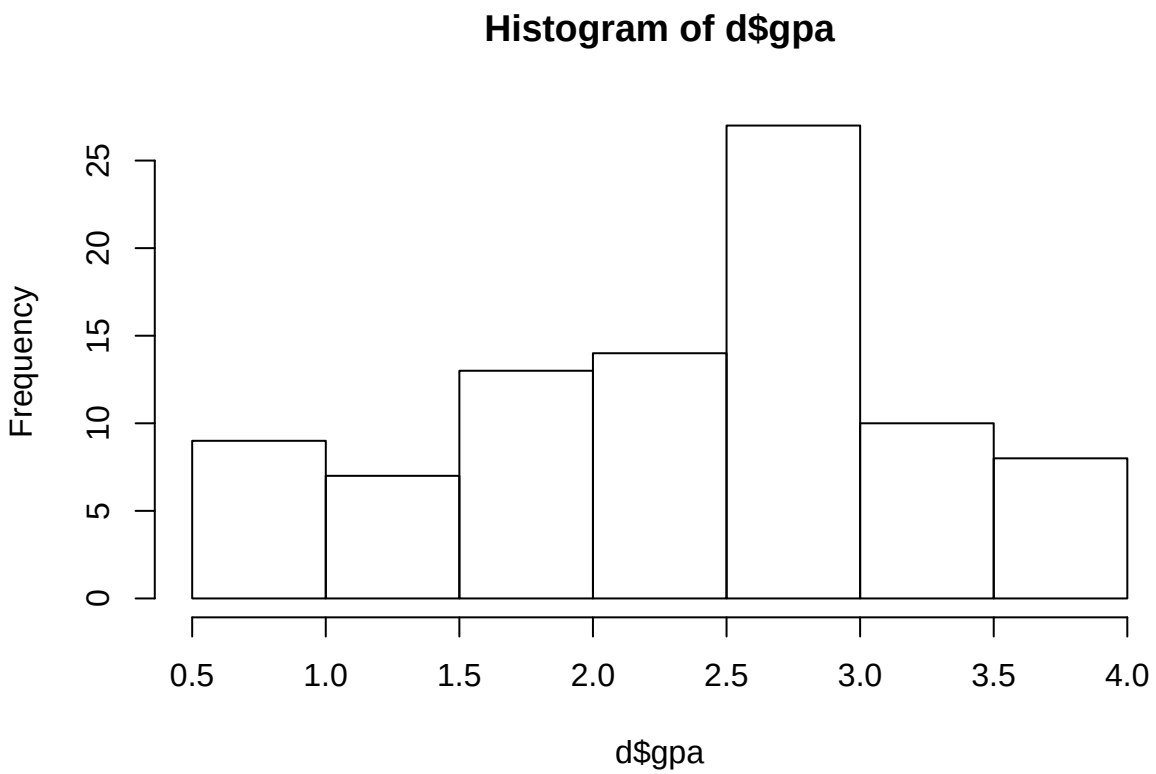


Note that you can change the number of intervals, or the interval width within the **hist()** funciton by changing the **breaks** setting. A higher number breaks the data into smaller intervals, a lower number breaks them into wider intervals.



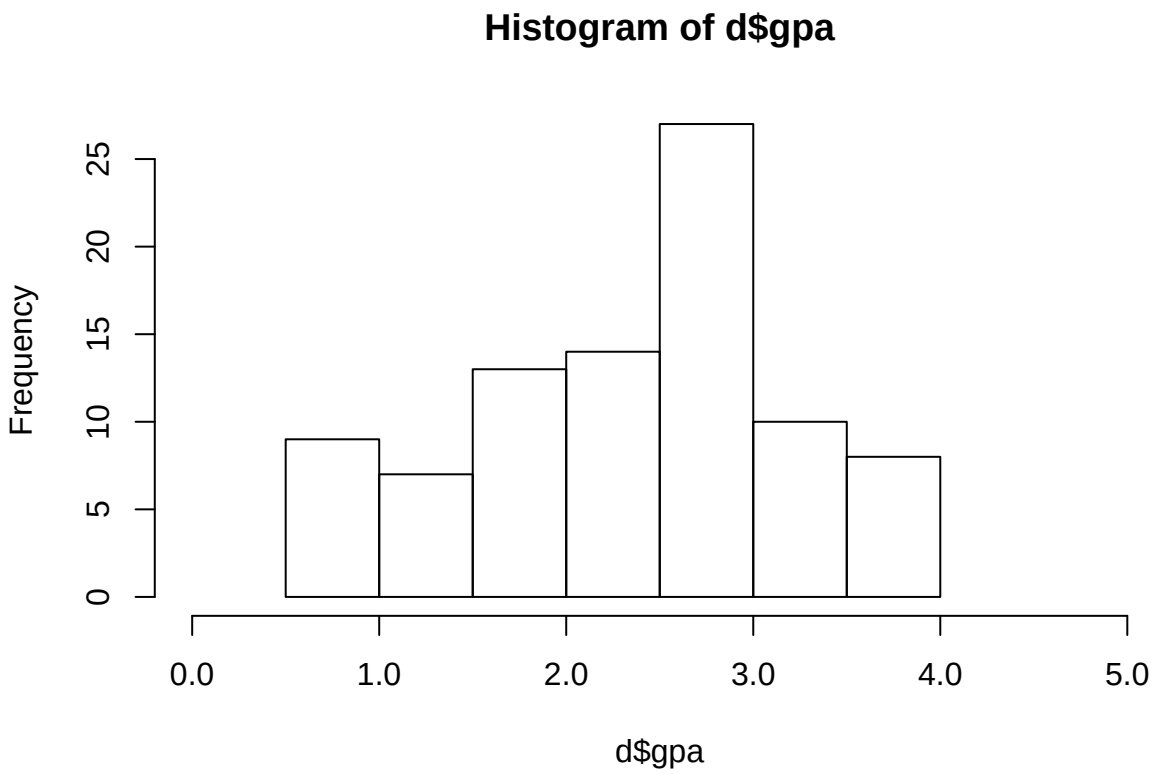
We will divide these data into 8 even intervals (from 0 to 4, or every 0.5), spanning the range of values:

```r
hist(d$gpa, breaks = 8)
```



As in the **plot()** function, you can also change the axis scales, but perhaps you prefer to have decimal formating for the ticks. This is tricky but requires you format the labels explicity with the number of digits set to 1 (this corresponds to the number of decimal places):

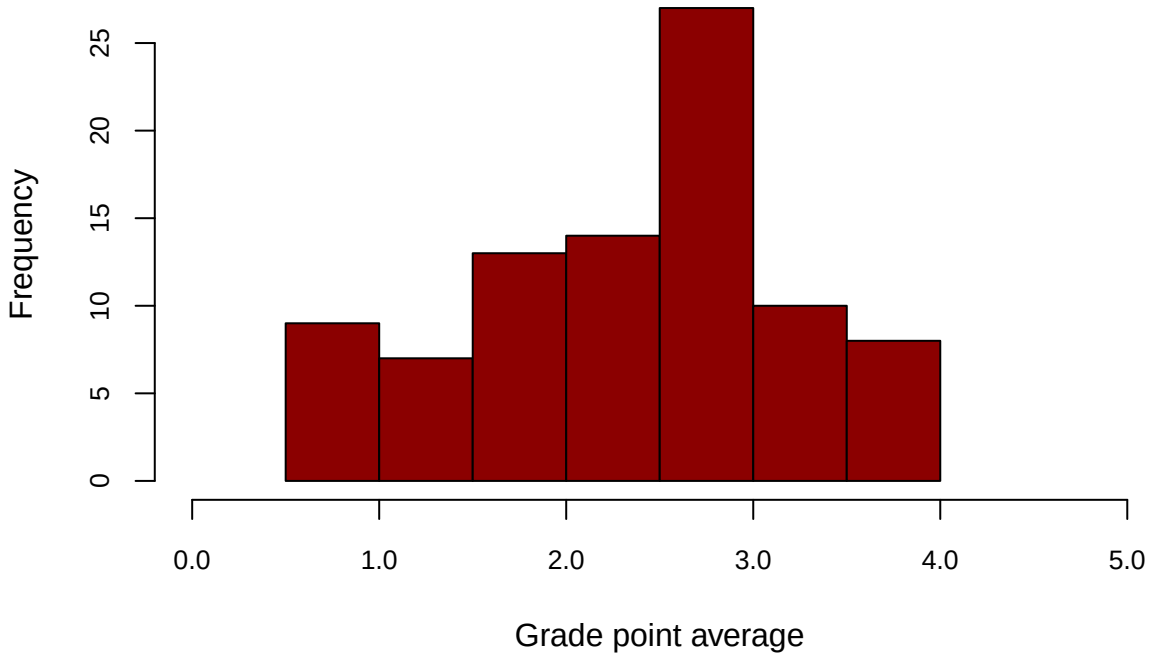
```r
hist(d$gpa, breaks = 8, xlim = c(0, 5), xaxt = "n")
axis(1, at = 0:5, labels = formatC(0:5, format = "f", digits = 1))
```



In the example above, we first call the **hist()** function to build the plot, but we initially suppress the x-axis, since the second function **axis()** has more features. We set xaxt="n", to turn the axis off. Note that we need to define the range of x values using the **xlim** setting first, since the plot is drawn first and not redrawn when making the **axis()** call.

Let's change the bars to dark red, add an x-axis title, remove the graph title (set main=NA), and change the font sizes:

```
hist(d$gpa, breaks = 8, xlim = c(0, 5), xaxt = "n", col = "dark red",
    xlab = "Grade point average", main = NA, cex.axis = 0.8)
axis(1, at = 0:5, labels = formatC(0:5, format = "f", digits = 1),
    cex.axis = 0.8)
```



Review your histogram and make any other changes you feel would improve its presentation. This is an opportunity to learn, so ask your TA, search for help online, or use R's built-in help.

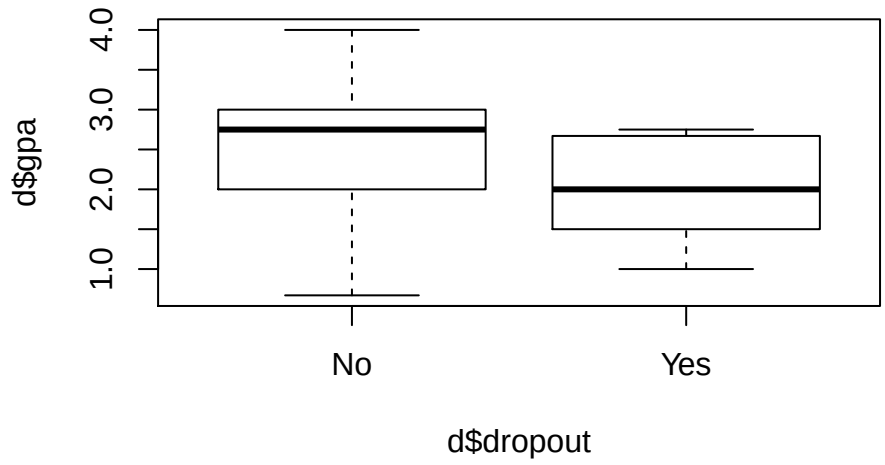
Your final graph should look similar to the one above.

Boxplots

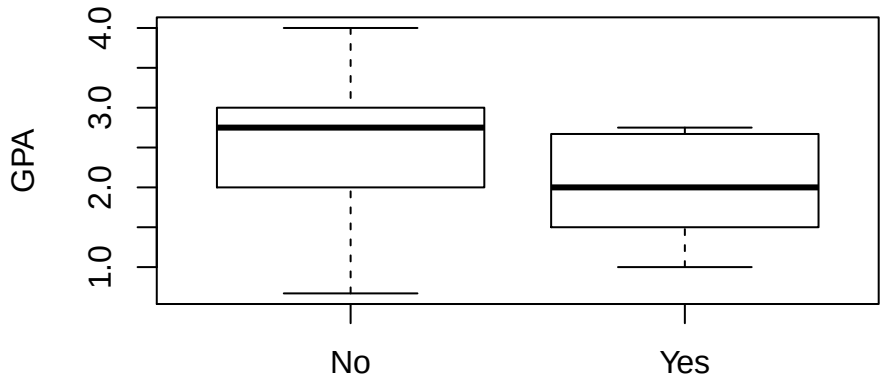
Boxplots are useful to illustrate the dispersion of data. They are frequently used with data that compare a numerical variable and a categorical variable. Let's create a boxplot using the same data file.

In base R this can be acheived with the **boxplot()** function, or simply with the **plot()** function, depending on how your data are categorised. There are some minor differences in how you code these functions, although both allow you to plot a variable using the formula (~) command, with "numeric dependent variable" ~ "categorical independent variable" as the typical format:

```
plot(d$gpa ~ d$dropout)
```



```
boxplot(gpa ~ dropout, data = d, ylab = "GPA")
```



So, now we have a boxplot of the GPA in the ninth grade as a function of whether students ultimately dropped out of high school.

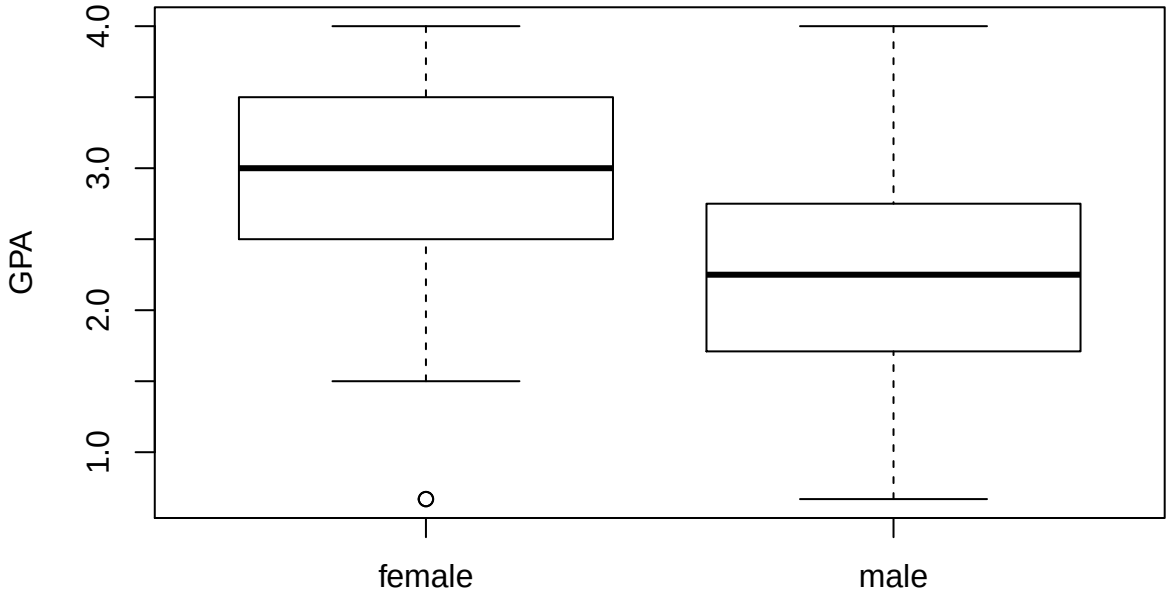
Review the box plot, and remind yourself of what the different components of the box plot indicate. There are 5 values calculated for each plot to produce the box and the whiskers.

If in doubt, type ?boxplot.stats and examine the help information for the default calculations:

```
`?`(boxplot.stats)
```

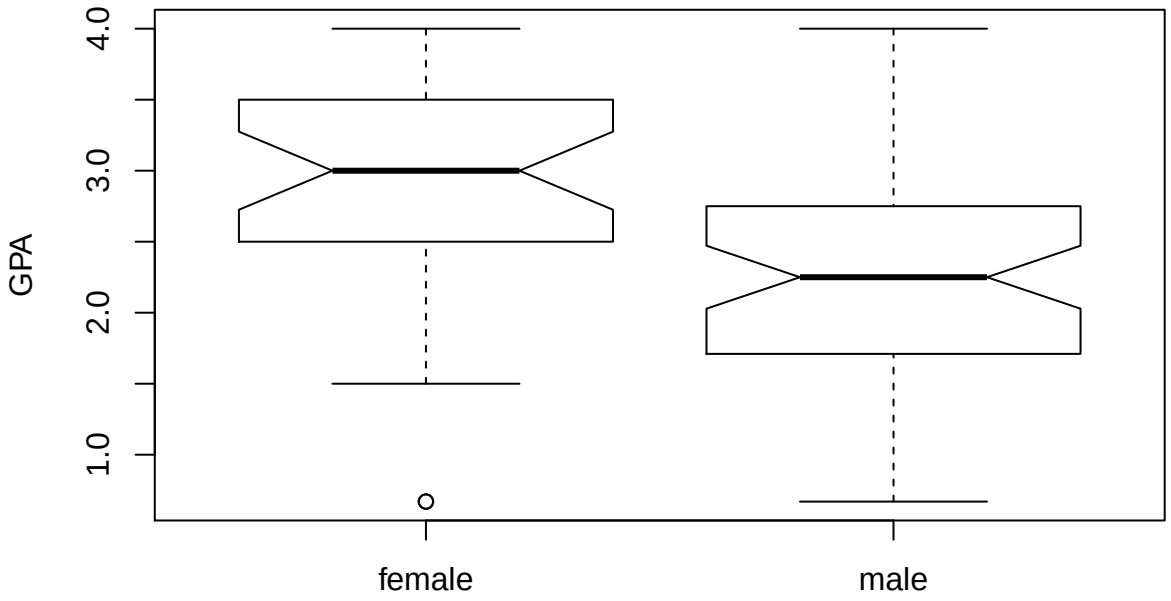
Repeat the previous step, except this time compare the GPA in ninth grade between males and females by selecting the gender variable.

```
boxplot(gpa ~ gender, data = d, ylab = "GPA")
```



An interesting feature of the boxplot is the **notch** setting, where a notch is drawn in each side of the boxes. If the notches of two plots do not overlap this is 'strong evidence' that the two medians differ. See ?boxplot.stats for the calculations used:

```
boxplot(gpa ~ gender, notch = T, data = d, ylab = "GPA")
```



Using what you have learned in the previous examples, edit the code for the box plot to improve its appearance (axis labels, colour, title, etc.)

Bar Graphs

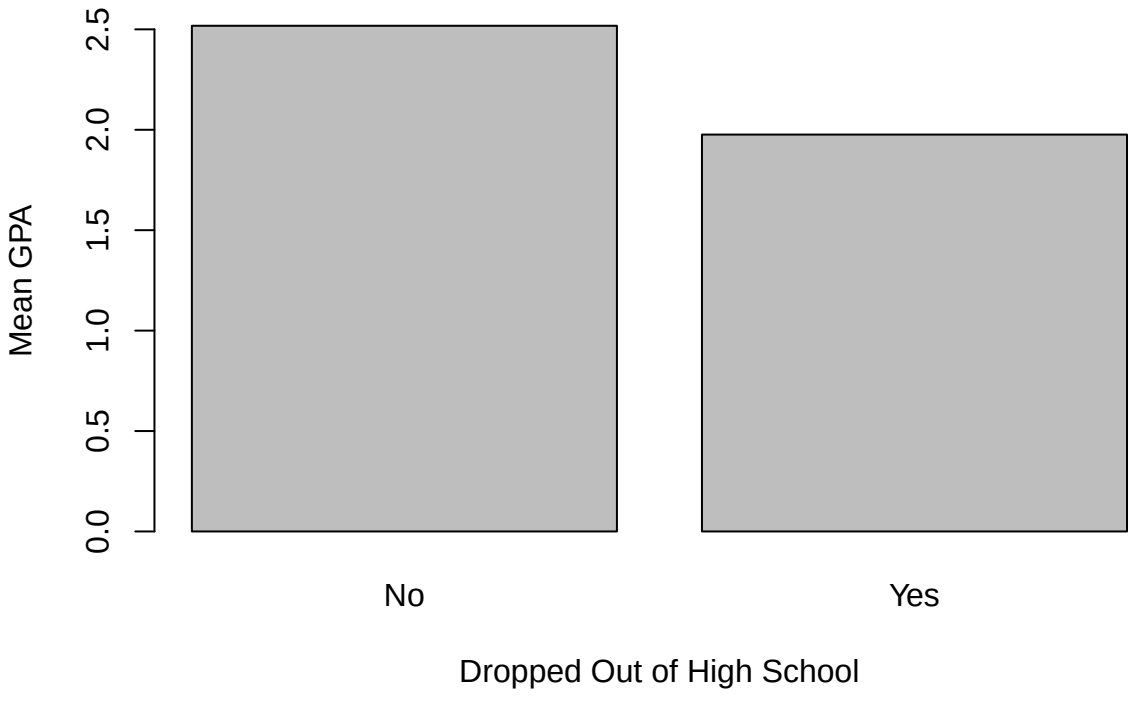
Another way to visually compare the data from different groups is with a bar graph. When comparing one numerical variable and one categorical variable, bar graphs do not show the data as well as other methods (strip chart, box plot, multiple histogram), but they are frequently used in scientific publications. Let's create one from the same example as above so we can compare them.

First, however, we need to summarise the data by taking the mean of the gpa for each level within the dropout category. This requires use of the **tapply()** function.

```
meanDropout <- tapply(X = d$gpa, INDEX = d$dropout, FUN = mean)
meanDropout
```

```
##       No      Yes
## 2.517821 1.976000
```

```
barplot(meanDropout, ylab = "Mean GPA", xlab = "Dropped Out of High School")
```

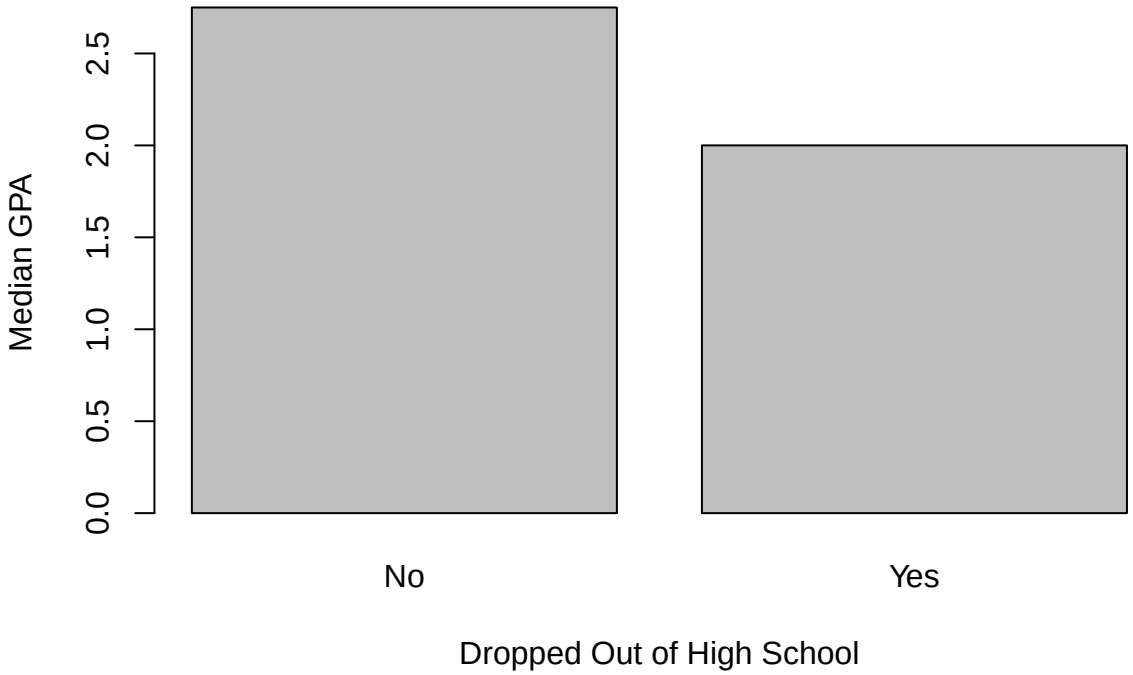


If you would prefer to plot the median, you can simply recalculate the summaries in the **tapply()** function, setting FUN=median:

```
meanDropout <- tapply(X = d$gpa, INDEX = d$dropout, FUN = median)
meanDropout
```

```
##   No  Yes
## 2.75 2.00
```

```
barplot(meanDropout, ylab = "Median GPA", xlab = "Dropped Out of High School")
```



Notice the biggest difference between this and the boxplot on the previous page is that the boxplot gives you

a sense of variability and the shape of the data (i.e. skew and potential heterogeneity of variances).

Usually for data this simple, a table would suffice, rather than use so much graphical space to show so little information!

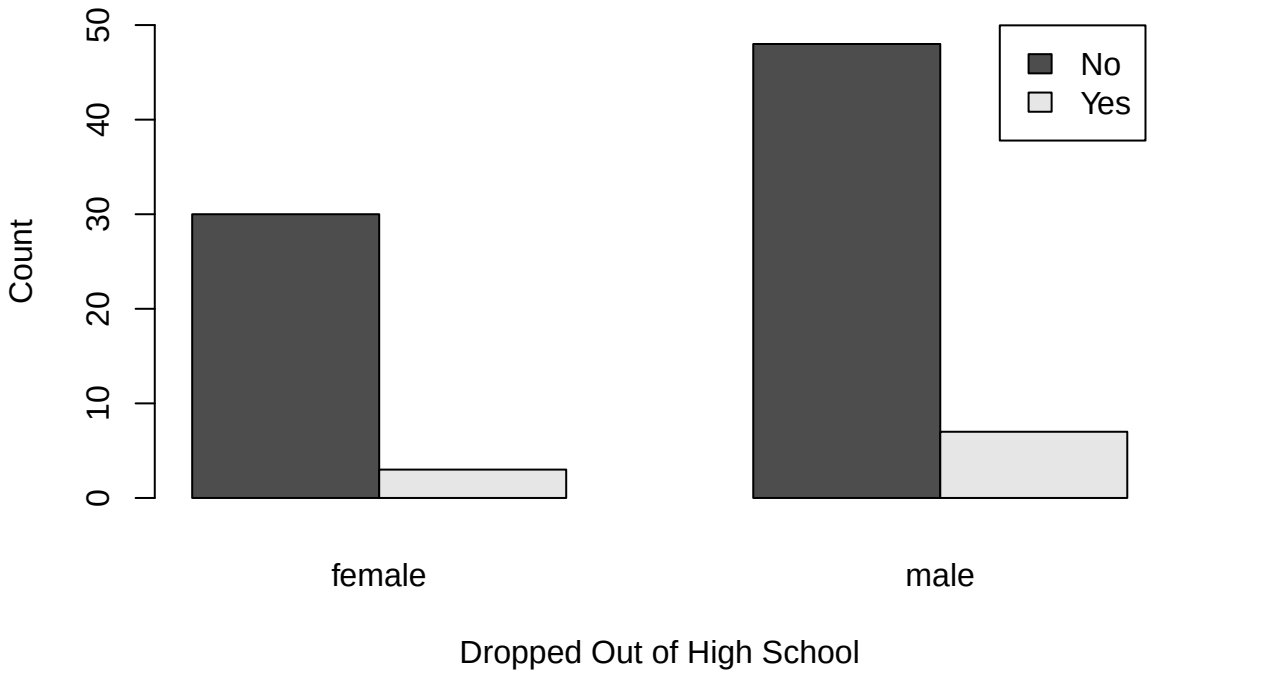
A grouped bar graph is used frequently to compare two categorical variables. Let's use a grouped bar graph to compare the frequency counts of males and females who dropped out of high school.

But to do this, we first need to count up the data from our data.frame, d into their respective categories. This can be done with the **xtabs()** function or the the **table()** function (either produces the same output, but one looks easier to read). We will store these frequencies in a variable called "groups":

```
groups <- xtabs(~d$dropout + d$gender)
groups <- table(d$dropout, d$gender)
groups
```

```
##
##       female male
##   No      30   48
##   Yes      3    7
```

```
barplot(groups, beside = TRUE, legend = T, yaxs = "r", ylim = c(0,
    50), ylab = "Count", xlab = "Dropped Out of High School")
```



Changing bar widths is possible by changing the width setting in the **barplot()** function call, but this particular feature is not very powerful in base R graphics. Alternative plotting packages, like ggplot2 offer much better control of graphing features.

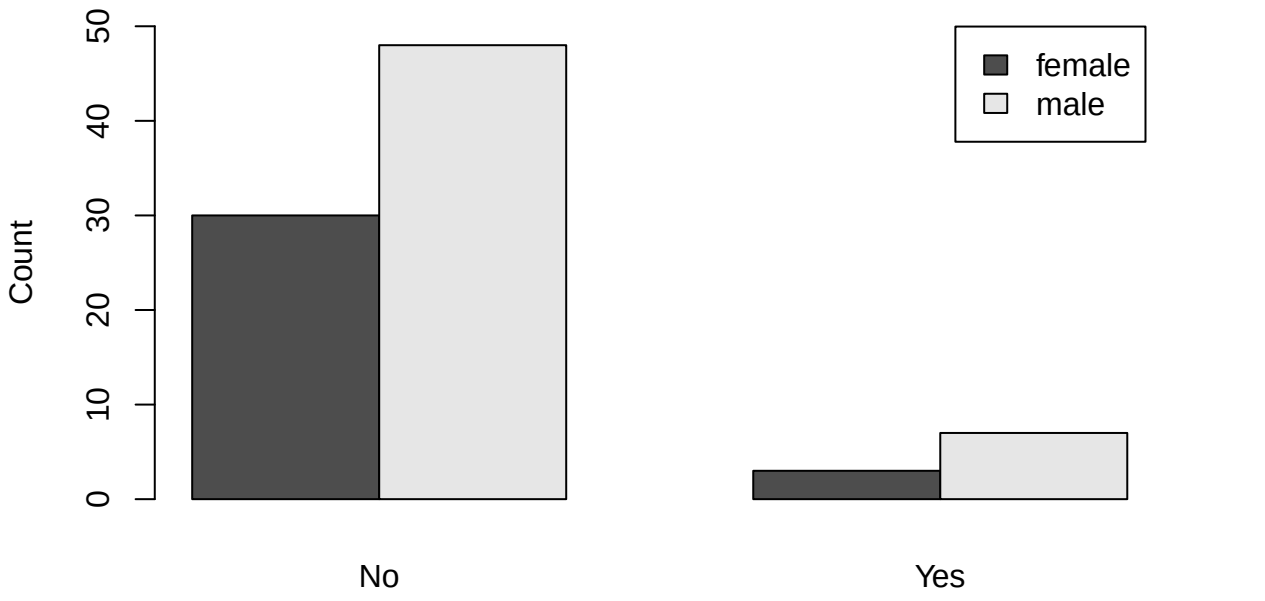
When there are multiple independent variables, you have to decide which one should be displayed on the X axis and which one should be used as the categories. Let's review the alternate way these data could be graphed.

Change the order of the independent clusters in the **table()** function call:

```
groups <- table(d$gender, d$dropout)
groups
```

```
##
##          No Yes
##   female 30   3
##   male   48   7
```

```
barplot(groups, beside = TRUE, legend = T, yaxs = "r", ylim = c(0,
    50), ylab = "Count", xlab = "Dropped Out of High School")
```

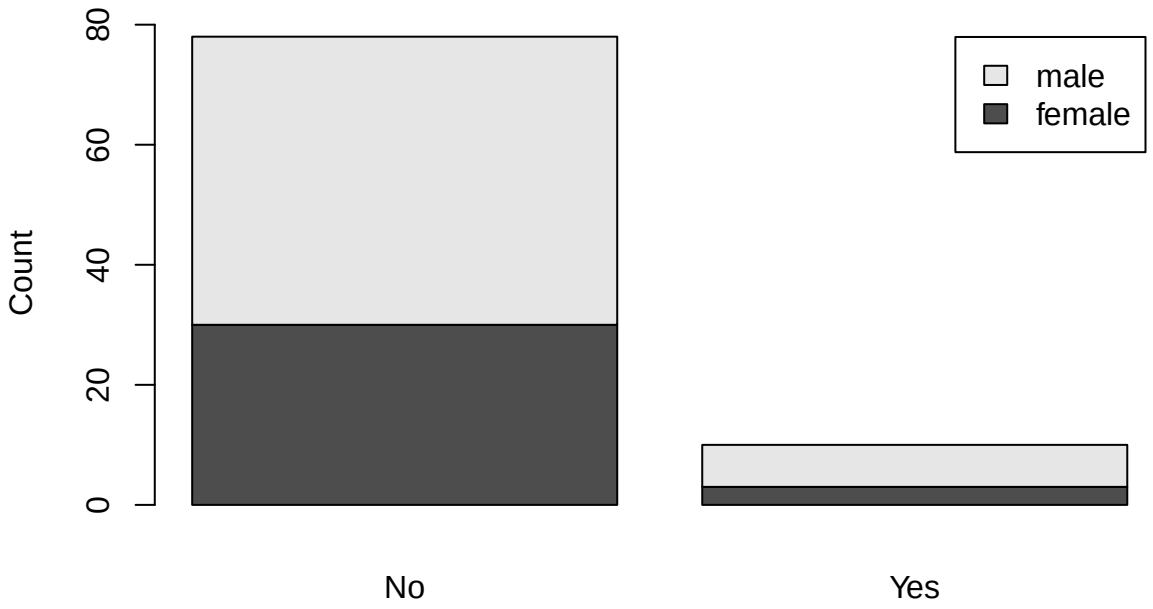


Review the new layout illustrated in the figure.

Remember from class that grouped bar graphs are typically graphed with bars grouped by categories of the explanatory variable, and different categories of the response variable indicated by different colours or shades. Which graph do you think is best?

A stacked bar graph is a variation on the clustered bar graphs above, and in R can be performed by changing just one setting, changing **beside** to **FALSE**:

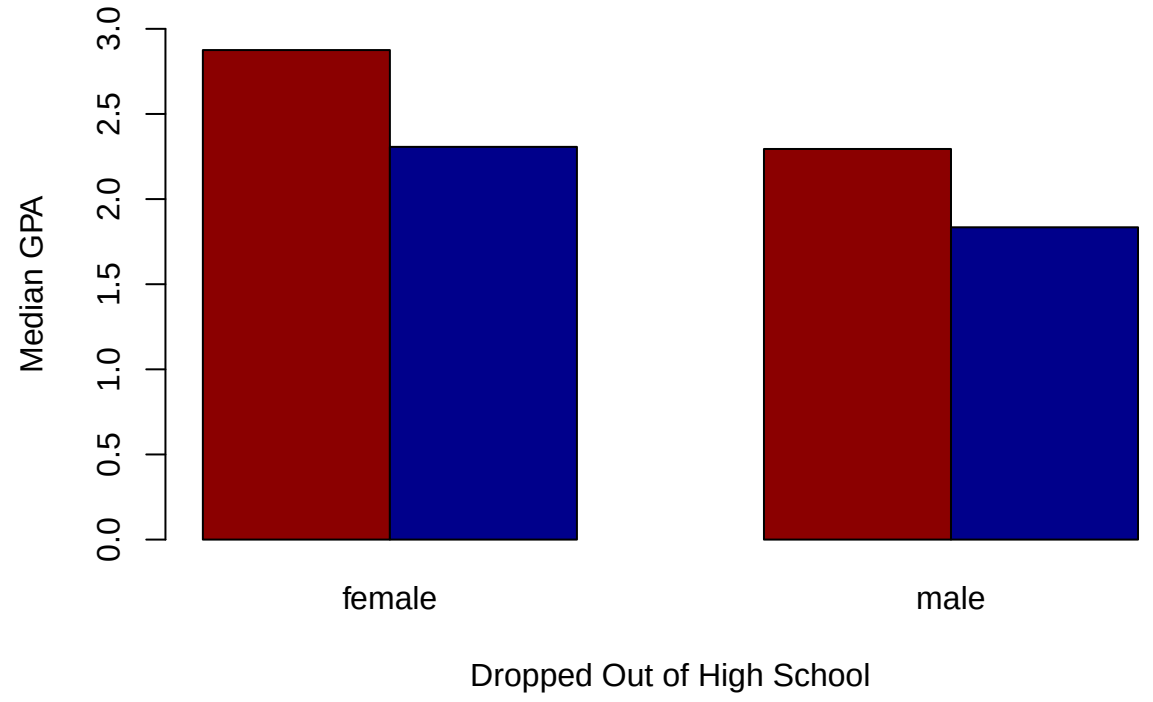
```
barplot(groups, beside = FALSE, legend = T, yaxs = "r", ylab = "Count",
    xlab = "Dropped Out of High School")
```



A grouped bar graph can also be used to compare two categorical variables, for the value of a numerical variable. This time, we'll graph the mean grade point average based on both gender and whether or not the student ultimately dropped out.

We will make use of the **tapply()** to calculate the mean within each category:

```
meanDropout <- tapply(X = d$gpa, INDEX = list(d$dropout, d$gender),
    FUN = mean)
barplot(meanDropout, beside = T, ylim = c(0, 3), ylab = "Median GPA",
    xlab = "Dropped Out of High School", col = c("dark red",
        "dark blue"))
```



Use what you learned above to edit the graph until you are happy with it.

Try changing the colour of one series only and adding a legend.

Line Graphs

Line graphs are most commonly used to display differences over time, or that can be justifiably connected. Connecting points with lines implies that the values in between those points fall along the line, Let's make a line graph to display the number of new measles cases over time.

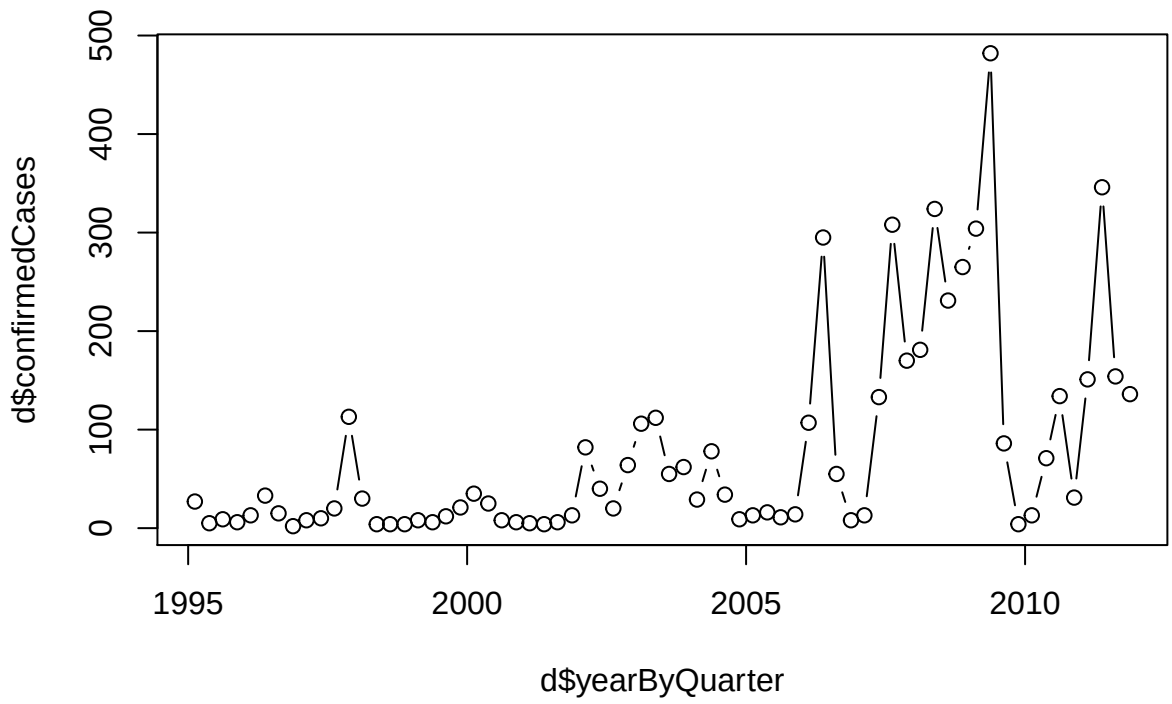
Open the file measles outbreak.csv:

```
d <- read.csv("Measles Outbreak.csv")
str(d)
```

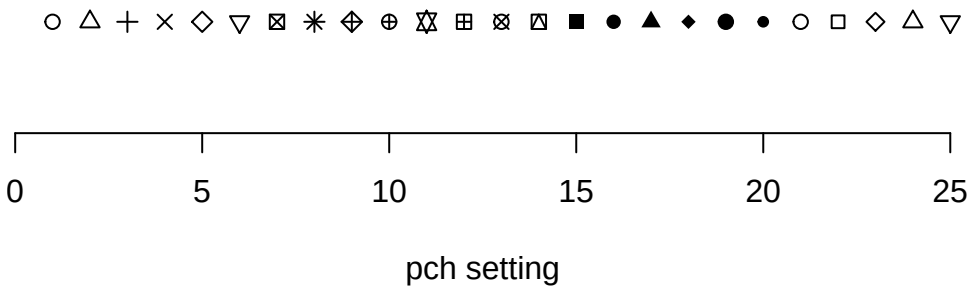
```
## 'data.frame':    68 obs. of  4 variables:
##  $ year          : int  2011 2011 2011 2011 2010 2010 2010 2010 2009 2009 ...
##  $ quarter       : Factor w/ 4 levels "1st","2nd","3rd",..: 4 3 2 1 4 3 2 1 4 3 ...
##  $ confirmedCases: int  136 154 346 151 31 134 71 13 4 86 ...
##  $ yearByQuarter : num  2012 2012 2011 2011 2011 ...
```

We want to plot the confirmed cases as a function of the year, by every quarter. Since technically this is a time-series plot, it would help to have lines connecting the points. We can do this by setting the **type** setting to "l" for a line graph, to "p" to a point graph, or to "b" to depict both line and points:

```
plot(d$confirmedCases ~ d$yearByQuarter, type = "b")
```

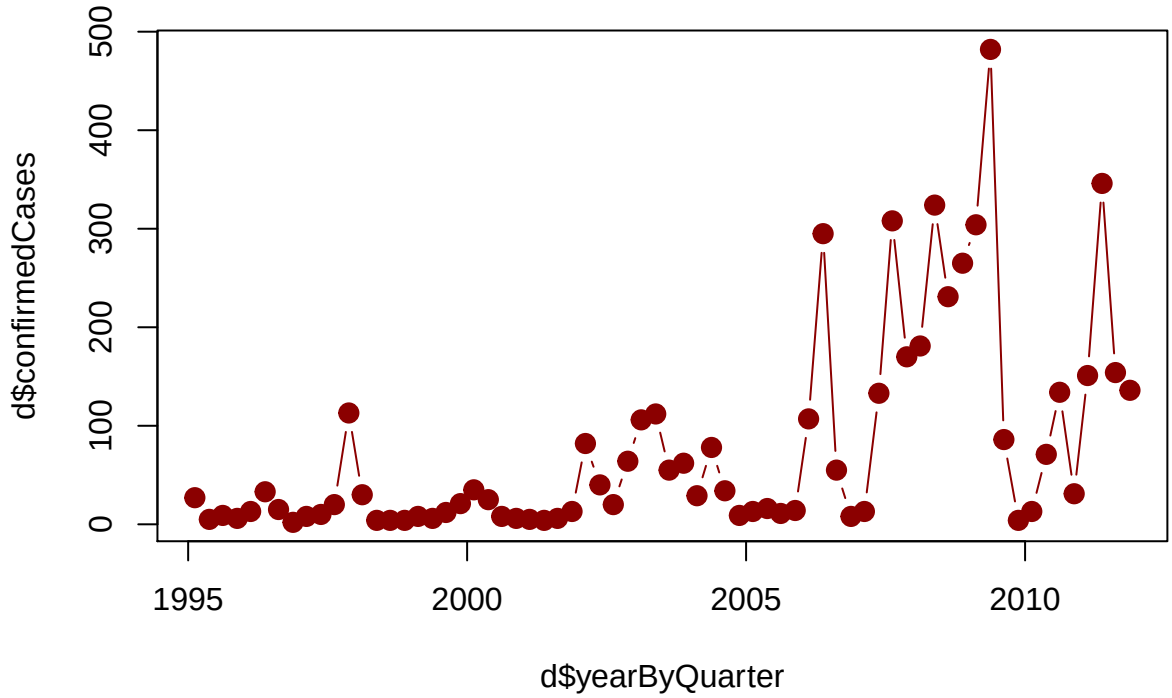


To change the point symbol you first need to decide what symbol you want. Commonly used symbols are shown in the graph below, and are set with the **pch** setting in the plot function.



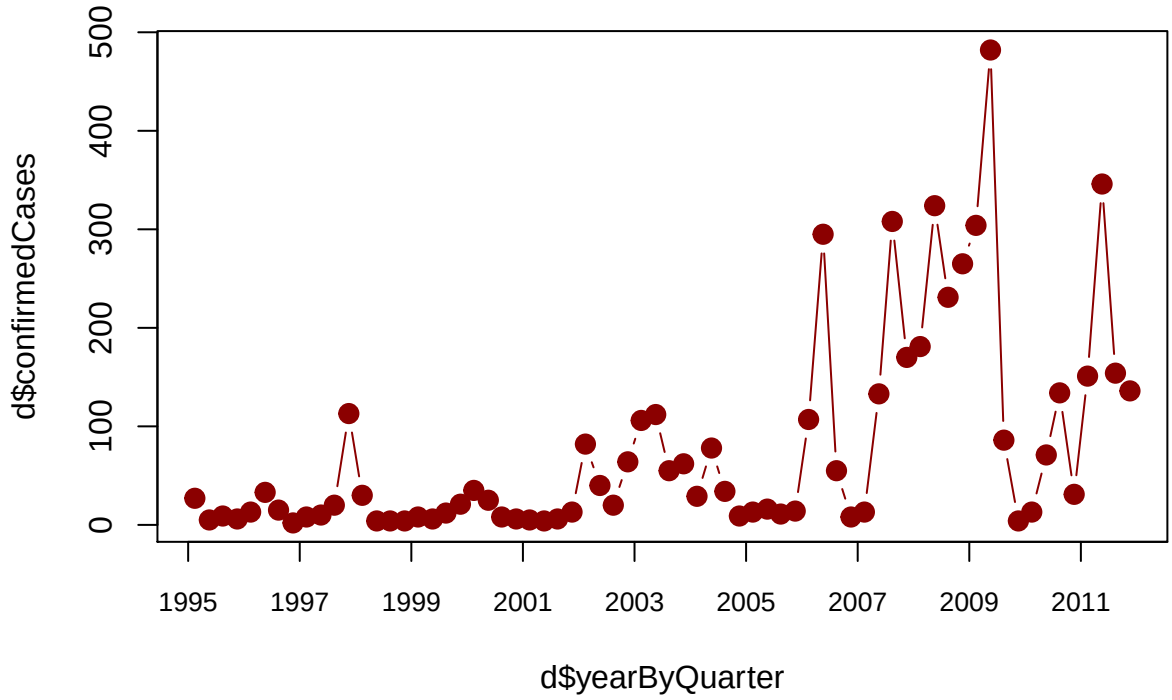
Let's choose pch=20, which will produce a solid circle, set the size of the point, using cex=2, and set the colour to dark red using col="dark red":

```
plot(d$confirmedCases ~ d$yearByQuarter, pch = 20, cex = 2, col = "dark red",
    type = "b")
```



As done earlier, we can set the x axis decimal places, but that requires suppressing the axis (xaxt="n") in the primary **plot()** function and then applying the separate **axis()** function:

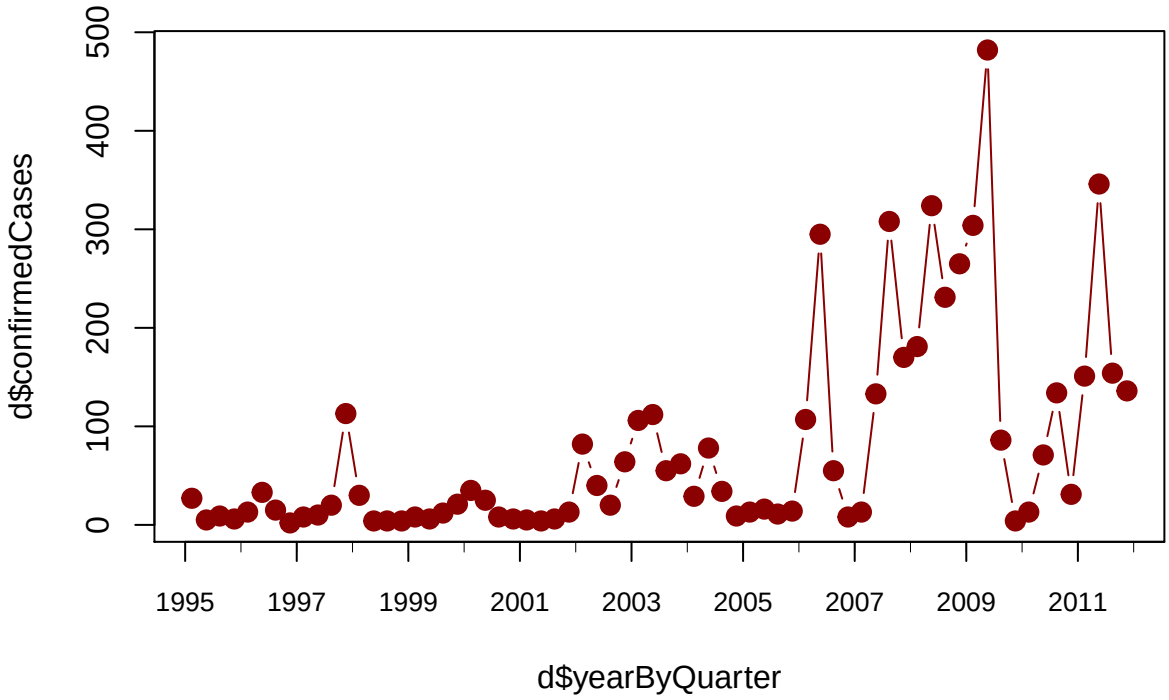
```
plot(d$confirmedCases ~ d$yearByQuarter, pch = 20, cex = 2, col = "dark red",
    type = "b", xaxt = "n")
axis(1, at = seq(1995, 2013, 2), labels = formatC(seq(1995, 2013,
    2), format = "f", digits = 0), cex.axis = 0.8)
```



Above, we set the decimal places to zero (usually this is not necessary, but is here to illustrative purposes), and constrained the x-axis from 1995 to 2013 with major increments of 2.

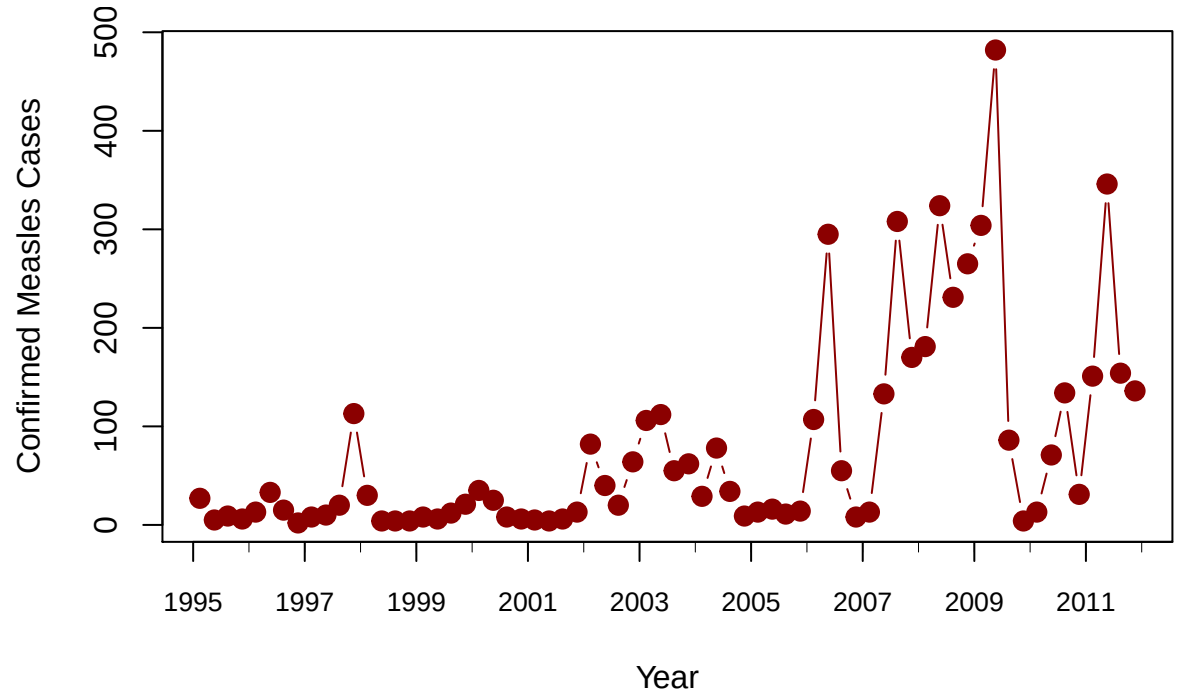
Sometimes you might want to have minor ticks that are not labelled. Base R does not make this easy while there are other plotting packages that do. But we will illustrate that here, making use of the **rug()** function (rug is a literal term for this function. . . the lines are meant to look like a rug or carpet but by making the rug negative in size, they dip below the axis, not above it):

```
plot(d$confirmedCases ~ d$yearByQuarter, pch = 20, cex = 2, col = "dark red",
    type = "b", xaxt = "n")
axis(1, at = seq(1995, 2013, 2), labels = formatC(seq(1995, 2013,
    2), format = "f", digits = 0), cex.axis = 0.8)
rug(x = seq(1996, 2012, 1), ticksize = -0.02, side = 1)
```



Above, we simply set the rug to be every other value from the major tick. You might want more minor ticks. If so, you would modify the seq values above to smaller intervals.

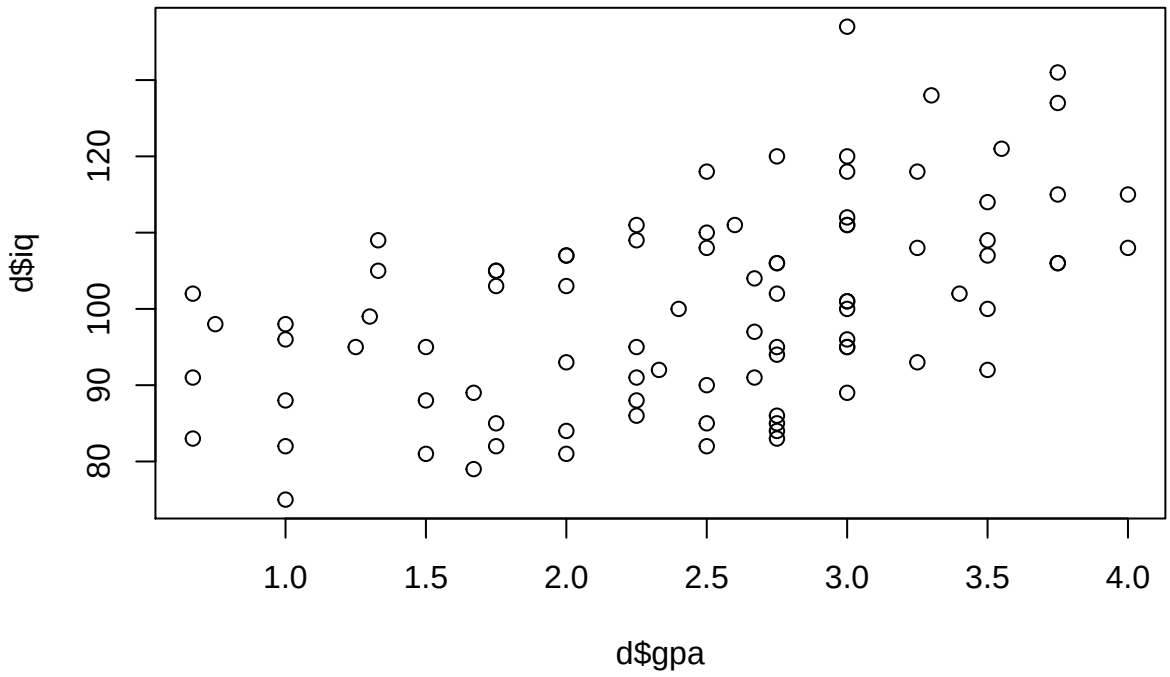
Fix the y and x axis titles to they are accurate and depict the appropriate date, and verify that your graph looks similar to this one:



Scatterplots

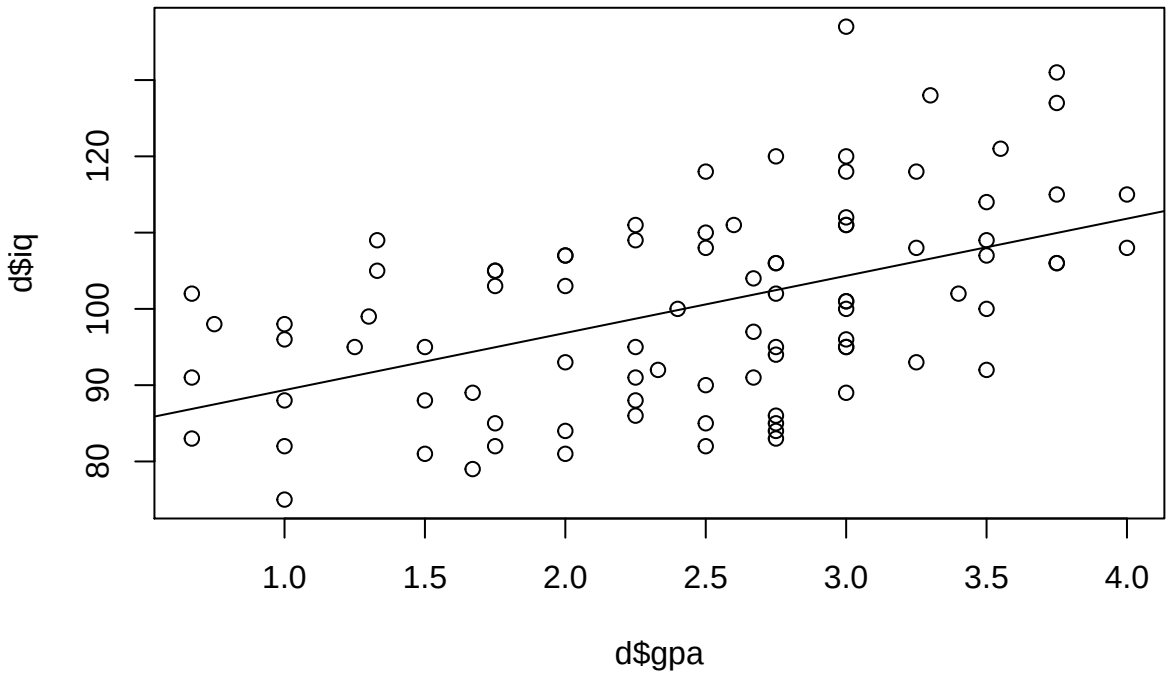
Scatterplots are typically used to visualize relationships between two numerical variables. In Appendix D, we may hypothesize that there is a positive association between IQ and GPA. Let's create this scatterplot to check.

```
d <- read.csv("appendixd.csv")
plot(d$iq ~ d$gpa)
```

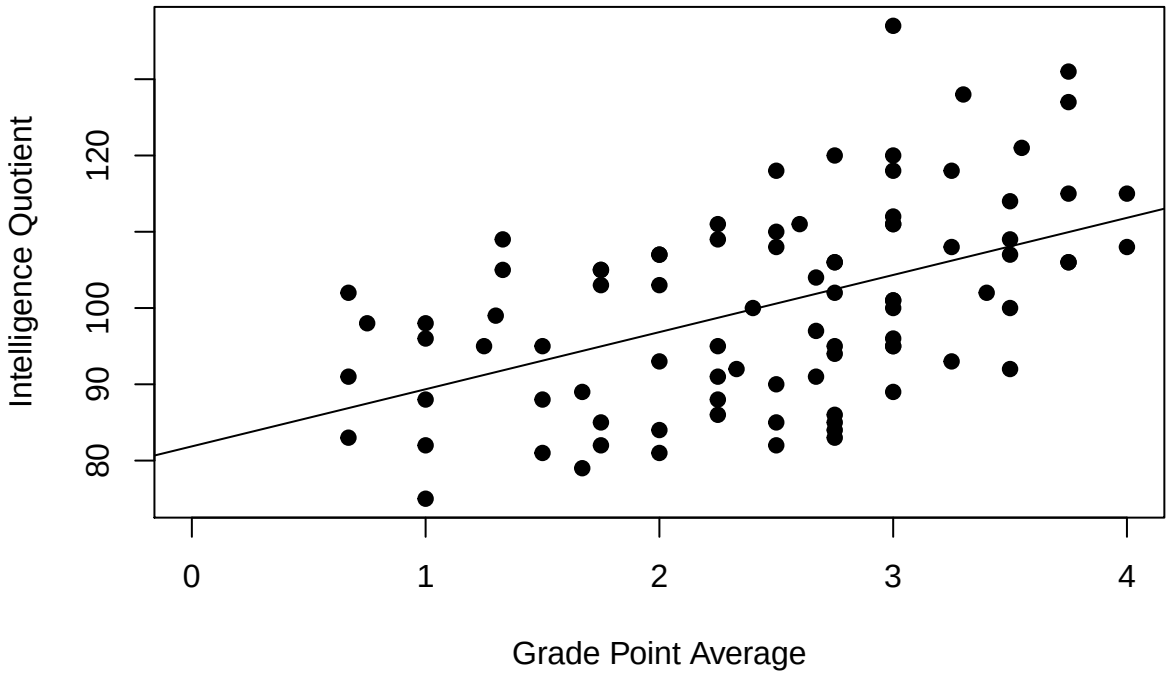


R is first a statistical language, and secondarily a plotting and graphing tool. If you want to add a regression equation to the above plot, at least in base R graphics, you first must fit a regression to the data. We will do that here in code, but you will come back to this in a later lab. What we will use is the **abline()** function that plots a line on top of your scatterplot. The a and b in the function name refer to the intercept (a) and slope (b) that would be returned from the linear regression. If you know a and b already you can specify them manually in the **abline()** function:

```
fit <- lm(iq ~ gpa, data = d)
plot(d$iq ~ d$gpa)
abline(fit)
```



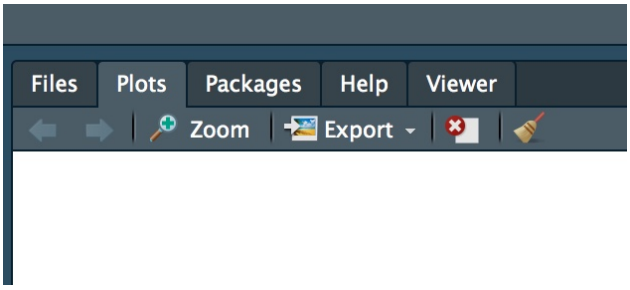
Change your points to a solid fill point and edit any other elements of the graph to improve its appearance using code learned from earlier examples. You final output should look similar to this:



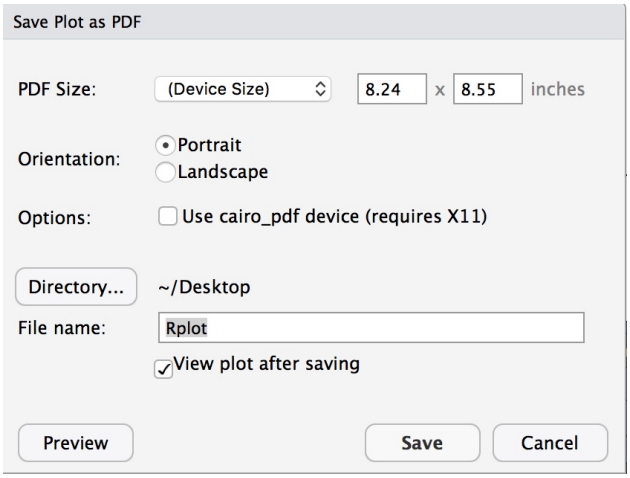
Go to Sakai and answer the questions about your graphs. All of your graphs should still be visible in your output window. You can return to them and use the back and forward arrows in the RStudio plot window if needed, or simply re-run your code.

Exercise 2: Exporting figures

Return to your working RStudio environment. Assuming you have an active plot showing in the plot window, select the Export pull down option and choose 'pdf':



A new window will pop up requesting settings on the size of the plot to export, file name, and folder location:



The default units may be in inches (RStudio authors might be US).

Chooosing pdfs to export is easier, since the plots tend to be exported as vectorised images rather than rasterised, making them clearer at whatever zoom level you view the plots at, however selecting 'save as image' provides you with multiple alternative image types.

One disadvantage of exporting plots in R is the plot window size has no relationship with the size of the plot when you export it. As a result, your point size and font sizes might be quite different than you expect when you export.

As you can see, R is capable of creating many different chart types, and each type has many options. It is important that you are comfortable using R for graphing for the rest of this term. Don't be afraid to play around with graphing the data. Try out several different options along the way, even those that we have not covered.

Always remember to save and organise your code. You will come back to it many times.

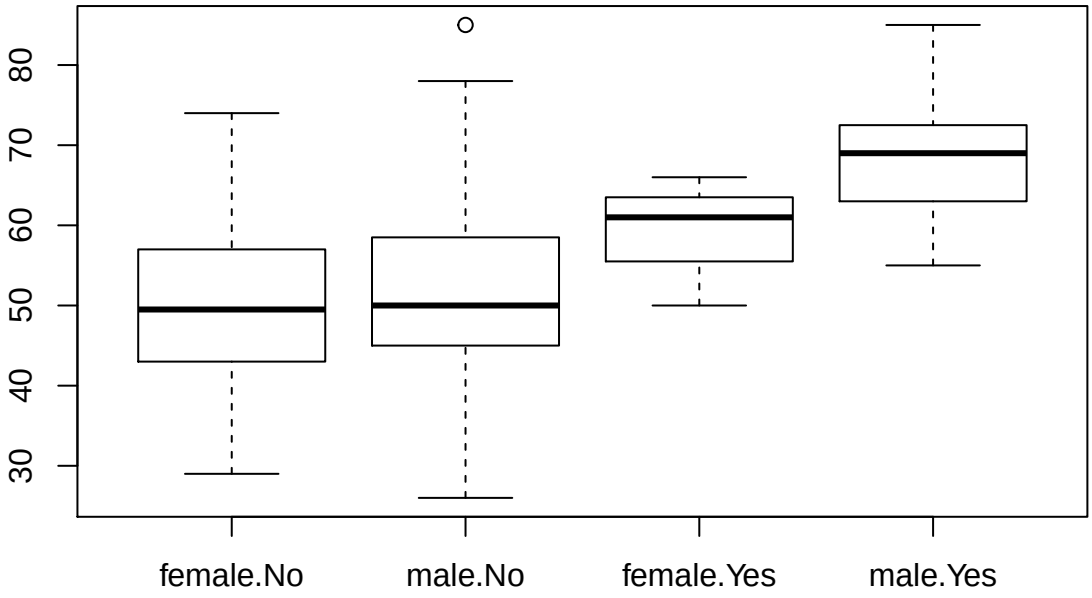
Additional Practice

ADD Score:

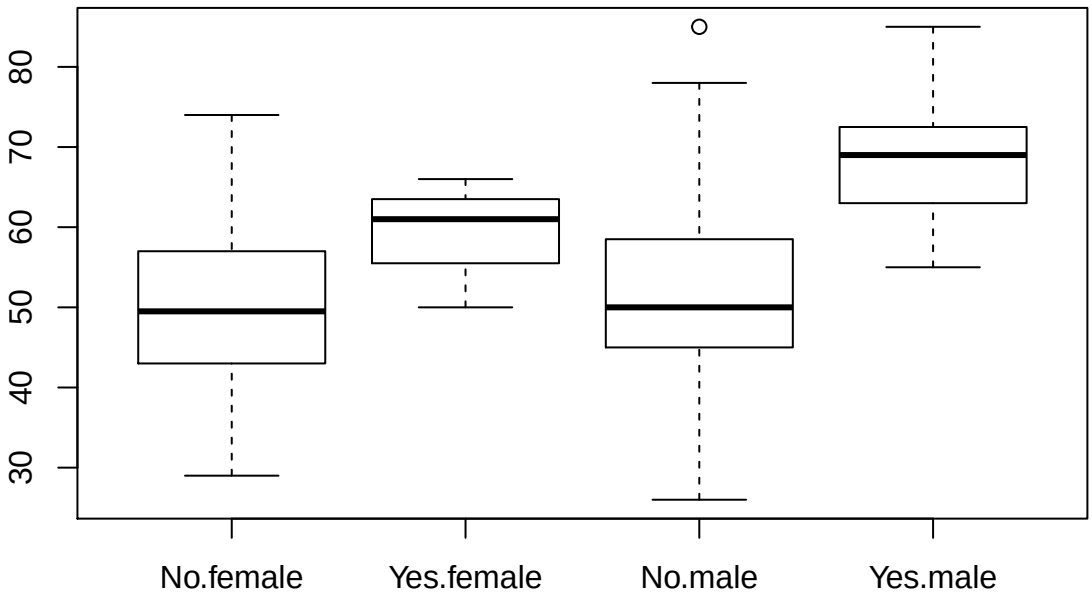
Using the appendix D file, create a clustered boxplot to compare the ADD score of males and females who did and did not drop out of high school.

With two factors on which to derive the boxplot, a simple rule in R is to note that if you want to plot something as a function of two interacting factors, you indicate this interaction with the * symbol:

```
d <- read.csv("appendixd.csv")
boxplot(addsc ~ gender * dropout, data = d)
```




```
boxplot(addsc ~ dropout * gender, data = d)
```



Answer the question on Sakai about this data.

Diet breadth:

The data in the file "diet breadth.csv" are from an ecological study of the rainforest community in El Verde in Puerto Rico. Diet breadth is the number of types of food eaten by an animal species. Import this data into R and visualize the frequency distribution using the appropriate type of graph. Adjust the X axis so it ranges from 0 to 70 with a major increment of 10, and use a binning interval width of 5.

Answer the two questions on Sakai about this data.