R for Health Data Science

Week 03: Graphics in R

Sam Stewart

2021-02-26

1 Introduction

In today's session we're going to cover the basics of producing plots in R. By the end of the session you should know how to produce

- a line plot
- a boxplot
- a histogram

We'll also talk about how to save figures, along with file formats.

We're going to focus on the base graphics functions in this lecture - later in the course we'll learn the `ggplot` family of functions, which can produce produce professional and creative figures, but requires an understanding of a unique grammar that we're not yet prepared for. If you're feeling ambitious the R4DS website has a good intro to ggplot.

I'm going to be following a couple of different online resources - check out the following online tutorials for additional information

- https://sites.harding.edu/fmccown/r/
- https://rstudio-pubs-static.s3.amazonaws.com/7953_4e3efd5b9415444ca065b1167862c349.html

```
library(RColorBrewer)

palette(c("black",brewer.pal(9,'Set1')))

dat = read.csv("data/framinghamFirst.csv",header=TRUE,
               na.strings=".",stringsAsFactors=FALSE)
dat$BMIGroups = cut(dat$BMI,breaks=c(0,18.5,25,30,Inf),
                    labels=c("Underweight","Normal","Overweight","Obese"))
```

This is the first command I include in most of my files - the command `palette()` sets the default color palette in R. When you choose colours you can either pass it a string (from this list of colours), or a number, which corresponds to the base number of colours (I think there are 8 base colours, but it might be 10).

What this command does is change the base palette to the set of 9 colours from the basic palette from `RColorBrewer`, a library of pleasing colour palettes (see all available palettes here).

You can change `'Set1'` in the command to change the default palette, or assign the colours per graph, but this is a good habit to get into.

UPDATE: According to this blog post the default colours in R stopped being terrible at the end of 2019, so setting the palette might not be necessary anymore.

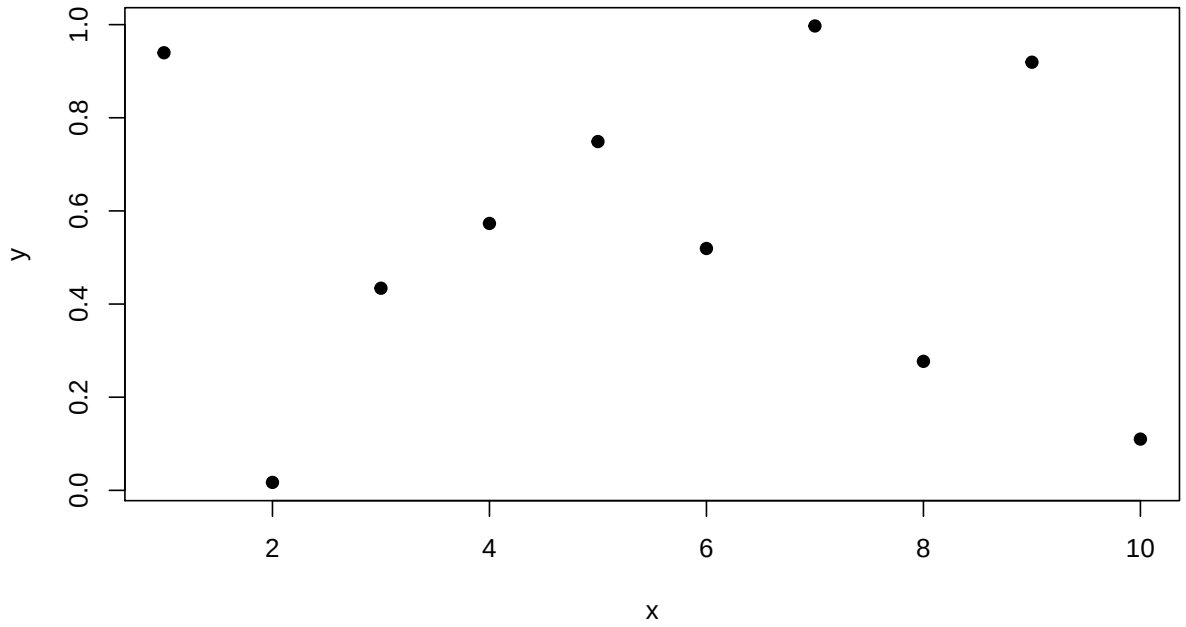
2 Line Plots

```
plot(x, y = NULL, type = "p",  xlim = NULL, ylim = NULL,
     log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
     ann = par("ann"), axes = TRUE, frame.plot = axes,
     panel.first = NULL, panel.last = NULL, asp = NA,
     xgap.axis = NA, ygap.axis = NA,
     ...)
```

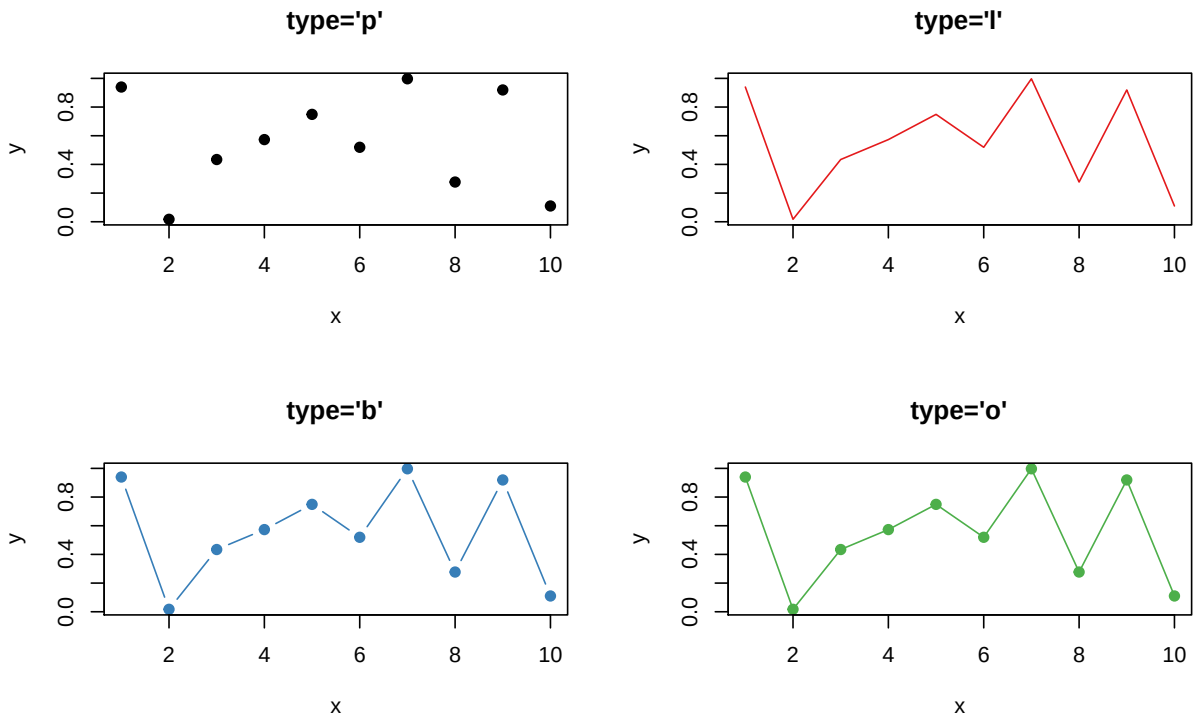
The default scatterplot function in R is simply `plot()`. You will find that other functions will supersede this for specific objects later in the course - this is where object classes and default functions come into play, but for now `plot()` produces a plot of `x` against `y`. There are a couple of arguments that you'll be interested in here

- `type = 'p'` the default is to draw the scatterplot as **p**oints, but other options include 'l', 'b' for both or 'o' for overplotted
  - `pch` is a number that controls the shape of the dot, see `example(points)` for a list
- `col` controls the colour of the plot - either a number or a string describing the colour
- `xlim,ylim` control the ranges of the x-axis and y-axis - the default will be to set them to fit all the data points
- `main,xlab,ylab,sub` are all text options

```
x=1:10
y=runif(10,0,1)
plot(x,y,pch=19)
```




```
par(mfrow=c(2,2))
plot(x,y,pch=19,type='p',col=1,main="type='p'")
plot(x,y,pch=19,type='l',col=2,main="type='l'")
plot(x,y,pch=19,type='b',col=3,main="type='b'")
plot(x,y,pch=19,type='o',col=4,main="type='o'")
```



The first plot demonstrates a simple scatter plot. In the second image we've included the 4 common plot types, in different colours.

2.1 `par()`

To get 4 plots on a single image we used the `par()` command. `par()` lets us set (and query) default graphic parameters. Looking at `help(par)` we can see a number of different parameters that can be set. The most common that I use:

- `mfrow=c(nr,nc)` sets the number of figures to plot as `nr` rows and `nc` columns. Plots are filled across rows, then across columns
- `mar=c(5,3,3,1)` sets the margins sizes, in the order `c(bottom,left,top,right)`. This is useful if you need to expand the x-axis for large variable labels, or shrink a margin because there is no label there (particularly helpful when combined with mfrow)
- `las=0` sets the orientation of the axis labels. 0 means they're parallel to the axis (default), 1 is horizontal, 2 is perpendicular, 3 is always vertical. *NOTE* that this can be passed as a plot argument as well, to change per plot.

There are many others, and may be some useful ones that I'm not aware of.

2.2 Vector arguments

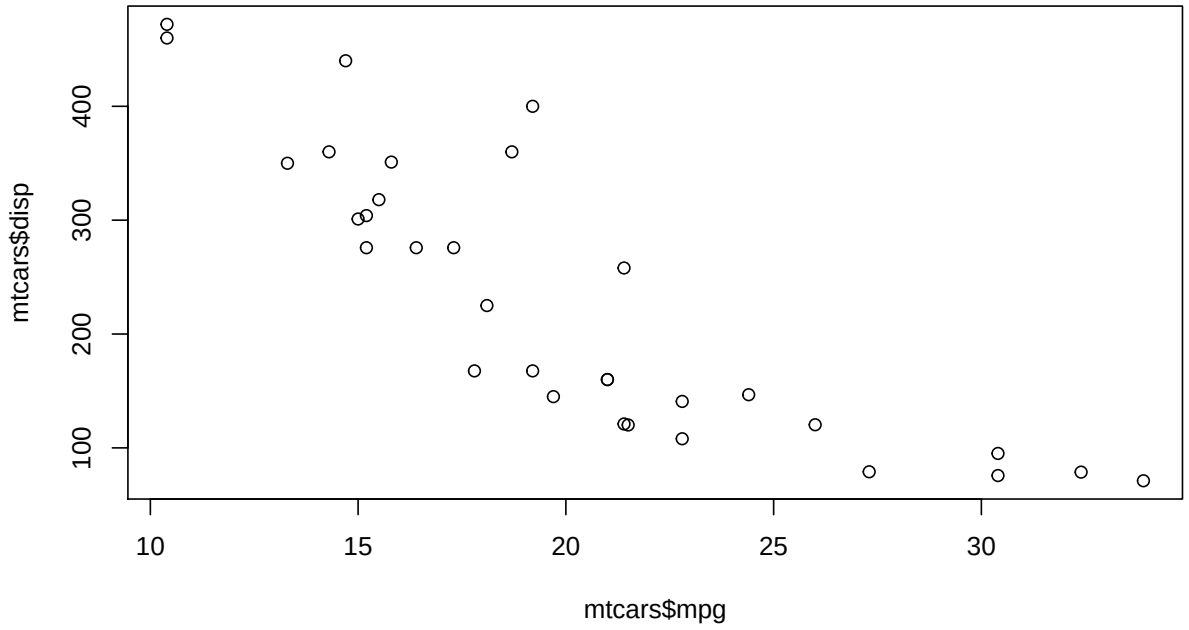
Many of these arguments can be passed as vectors, allowing us to explore the data graphically. I'll use the famous and terrible `mtcars` dataset here, as it's sample size is conducive to plotting.

```
head(mtcars)
```

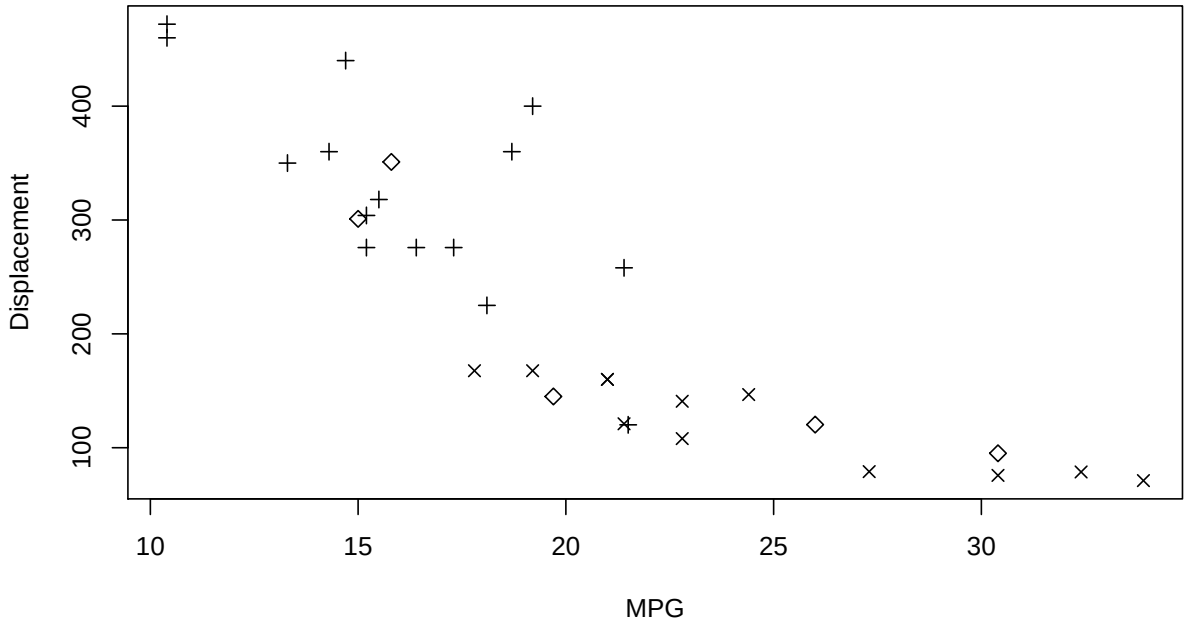
```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4        21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag    21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710       22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

```
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

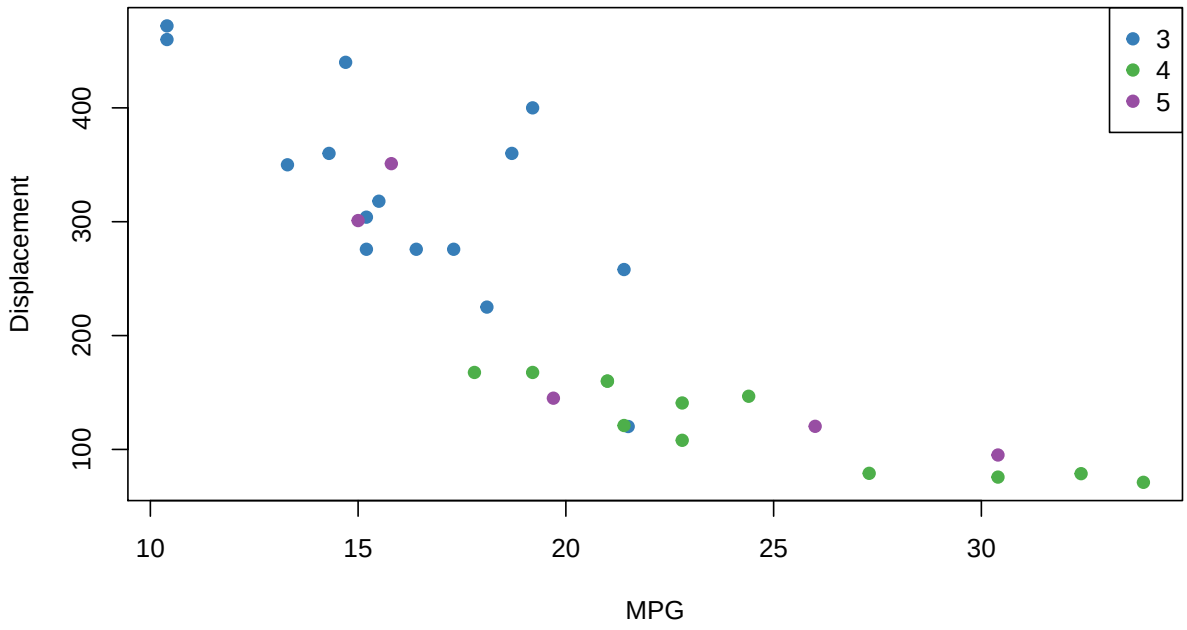
```
plot(mtcars$mpg,mtcars$disp)
```




```
plot(mtcars$mpg,mtcars$disp,pch=mtcars$gear,xlab='MPG',ylab='Displacement')
```




```
plot(mtcars$mpg,mtcars$disp,col=mtcars$gear,xlab='MPG',ylab='Displacement',pch=19)
legend("topright",pc=19,col=3:5,legend=3:5)
```



We can see that `pch` and `col` can both take vectors, allowing us to look for patterns using differences in the

plotting space.

3 Histograms and Barplots

Histograms and barplots are for continuous and categorical variables respectively. For histograms we'll pass the continuous variable, for categorical variables we create the table, then barplot that.

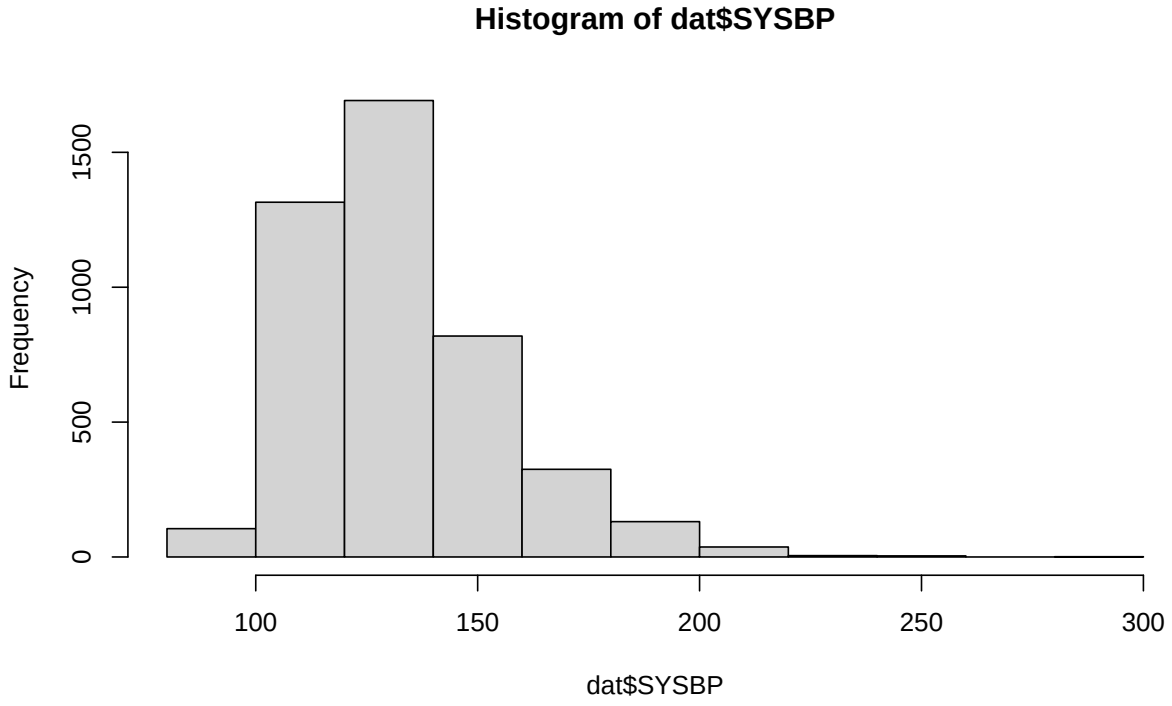
3.1 Histograms

```
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     density = NULL, angle = 45, col = "lightgray", border = NULL,
     main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL,
     xlab = xname, ylab,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, warn.unused = TRUE, ...)
```

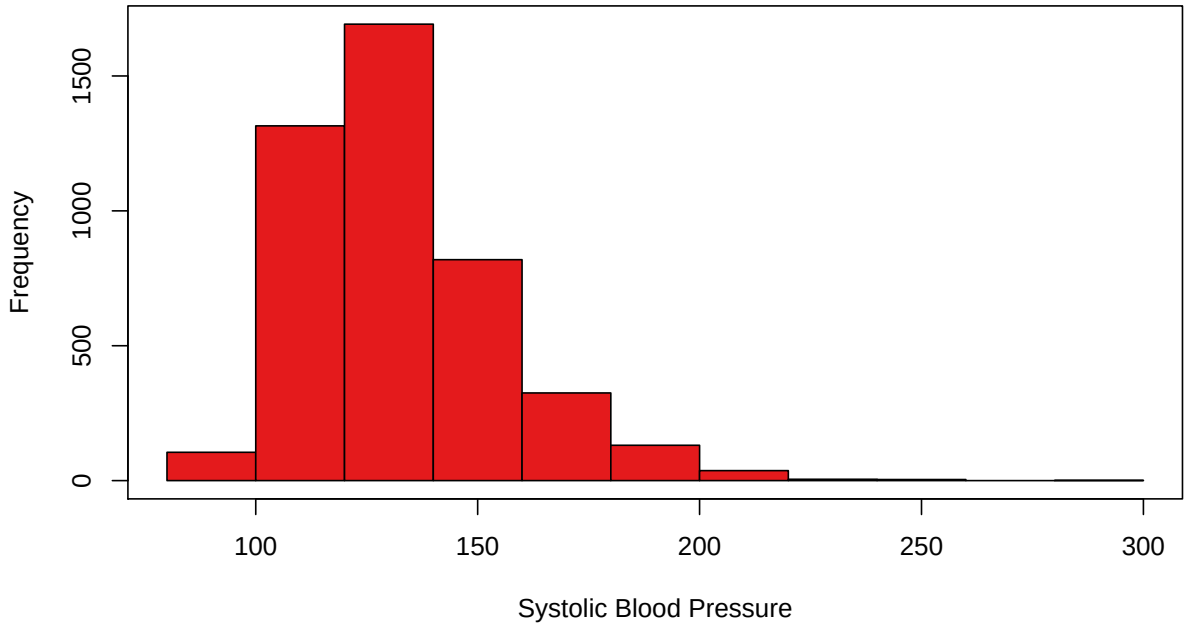
Histograms are run on any continuous variable `x`, and overall there are fewer arguments to be used here, and most that we will use (`col, xlab, main`) are generic in nature. The few that are used

- `freq, prob` control if the x-axis is presented as counts (default) or denisties
- `breaks` takes either than name of an algorithm or the actual points to break the continuous variable at. Note that this is *rarely* adjusted

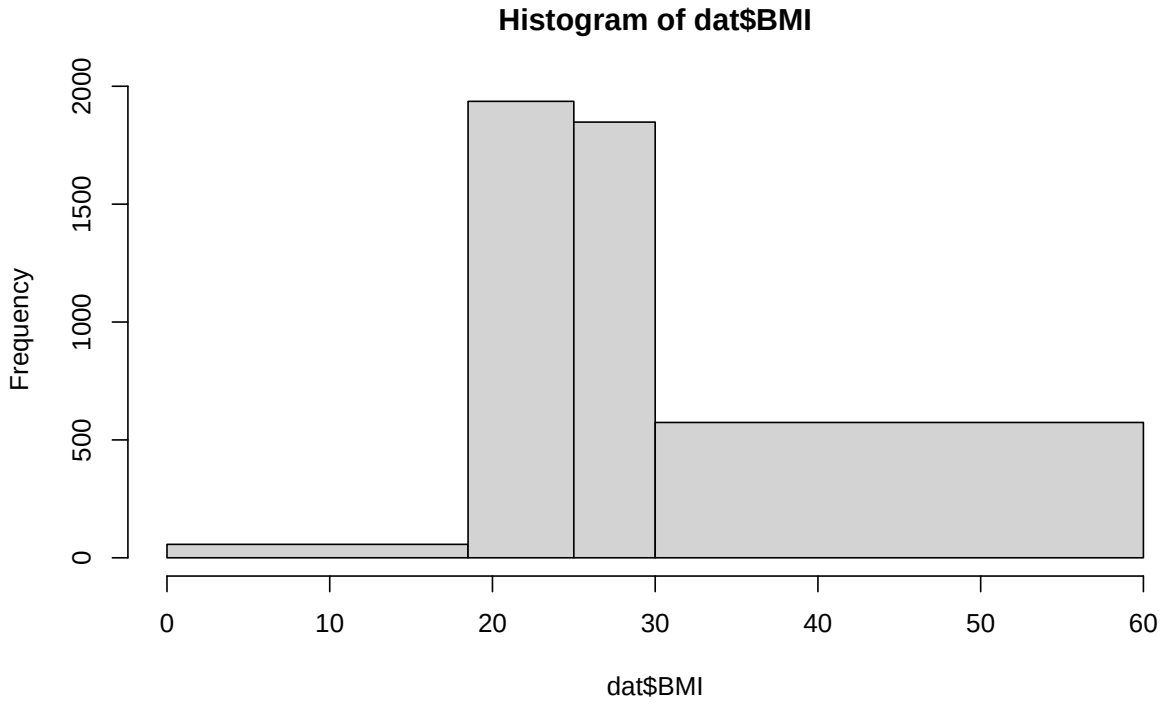
```
hist(dat$SYSBP)
```




```
hist(dat$SYSBP,col=2,xlab='Systolic Blood Pressure',main='')
box()
```



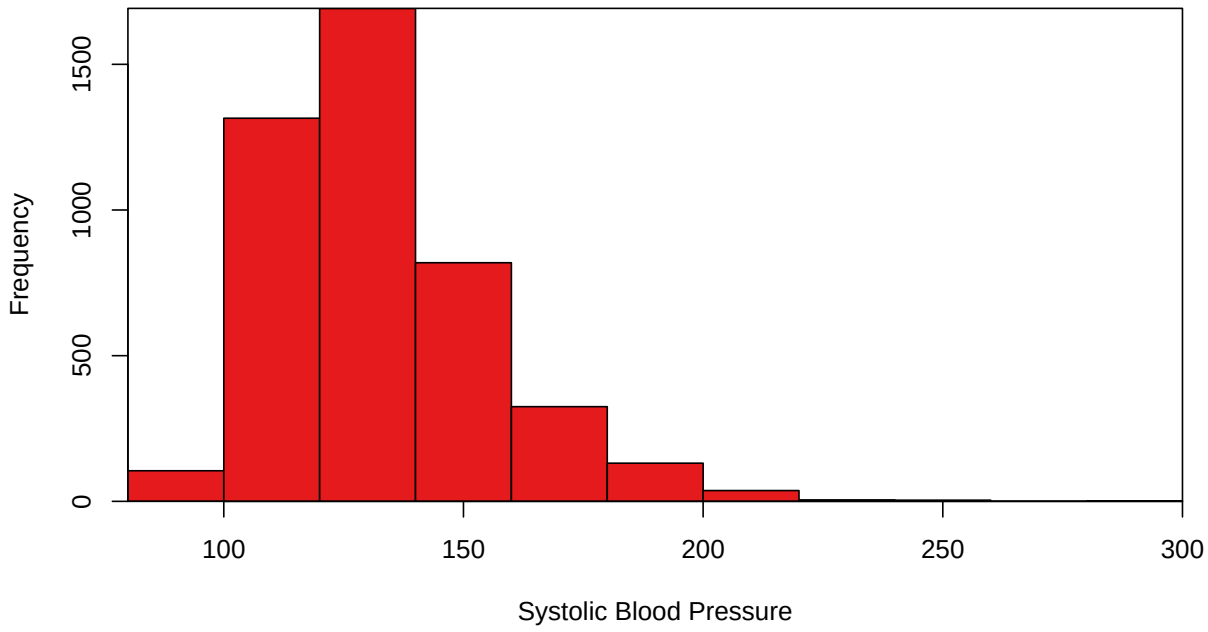

```
#this one looks bad
hist(dat$BMI,breaks=c(0,18.5,25,30,60),freq=TRUE)
```



Note that the `box()` command will complete the drawing around the figure (many plotting functions in R

leave the box open). The default behaviour for the x and y-axes is to extend them by 4% - to use the actual range of the data use `xaxs='i'` and/or `xaxs='i'` (these arguments are described in `par()`).

```
hist(dat$SYSBP,col=2,xlab='Systolic Blood Pressure',main='',xaxs='i',yaxs='i')
box()
```



3.2 Barplots

```
barplot(height, width = 1, space = NULL,
        names.arg = NULL, legend.text = NULL, beside = FALSE,
        horiz = FALSE, density = NULL, angle = 45,
        col = NULL, border = par("fg"),
        main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
        xlim = NULL, ylim = NULL, xpd = TRUE, log = "",
        axes = TRUE, axisnames = TRUE,
        cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
        inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
        add = FALSE, ann = !add && par("ann"), args.legend = NULL, ...)
```

For barplots we need to create the data ourselves in tabular form, then plot it as the `height` argument.

```
tab.sex.smoke  = table(dat$SEX,dat$CURSMOKE)
tab.sex.smoke
```

```
##
##        0    1
##   1  769 1175
##   2 1484 1006
```

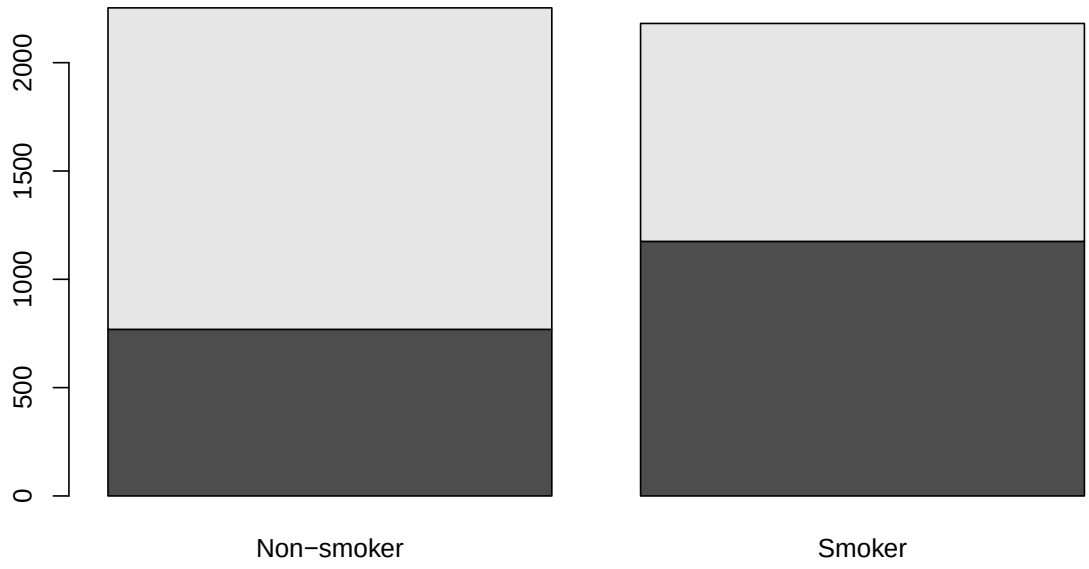
```
#a great example of why variables should be converted to factors with meaningful labels
dat$SEX = factor(dat$SEX,levels=1:2,labels=c("M",'F'))
```

```
dat$CURSMOKE = factor(dat$CURSMOKE,levels=0:1,labels=c("Non-smoker","Smoker"))

tab.sex.smoke  = table(dat$SEX,dat$CURSMOKE)
tab.sex.smoke
```

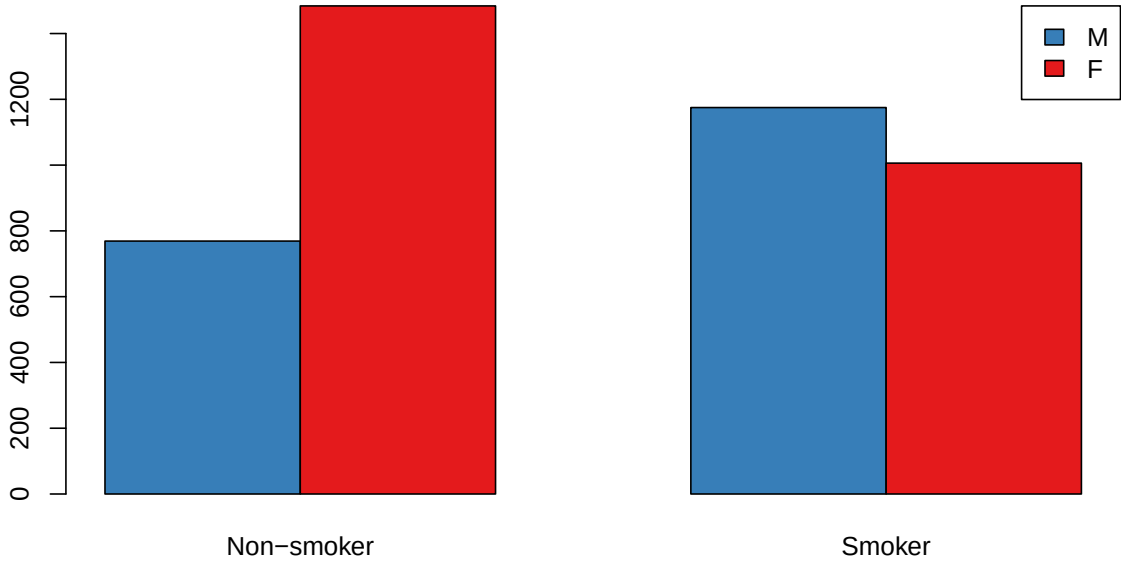
```
##
##     Non-smoker Smoker
##   M        769   1175
##   F       1484   1006
```

```
barplot(tab.sex.smoke)
```

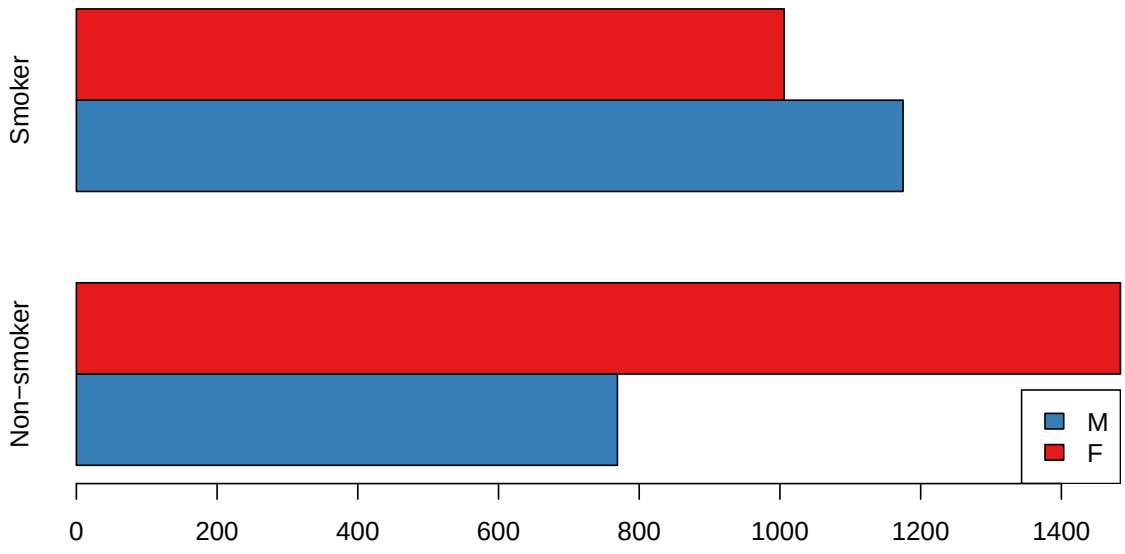


The default behaviour in R is to use stacked barplots - if you want them beside each other use the argument `beside=TRUE`. We can also send them in the other direction using `horiz=TRUE`.

```
barplot(tab.sex.smoke,beside=TRUE,col=3:2)
legend("topright",fill=3:2,legend=c("M","F"))
```



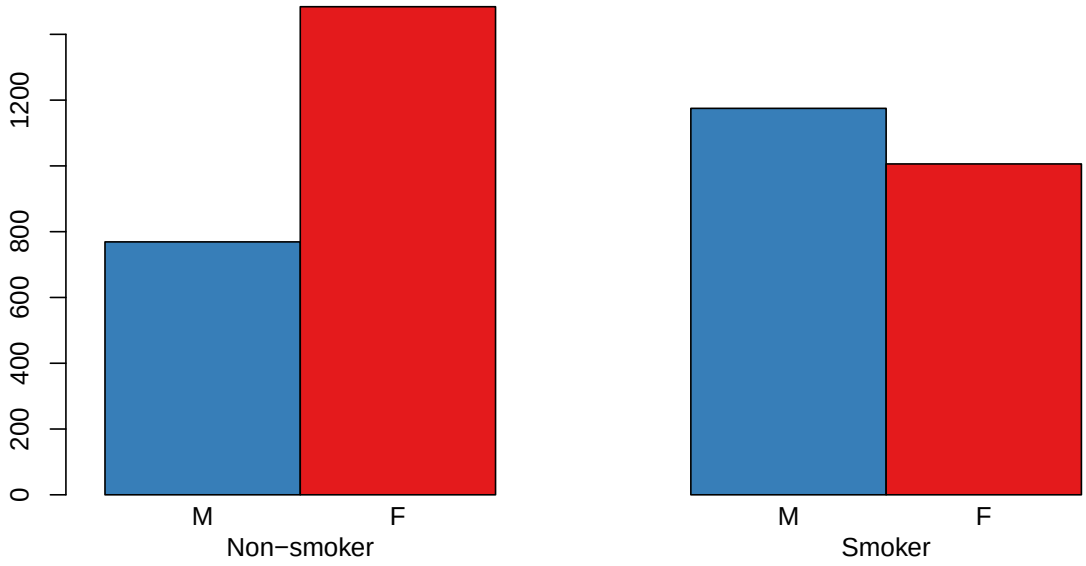

```
barplot(tab.sex.smoke,beside=TRUE,col=3:2,horiz=TRUE)
legend("bottomright",fill=3:2,legend=c("M","F"))
```



Two other tricks - if you assign the plotting function to an object it will return the centre of the blocks on

the x-axis - we can use those numbers and the command `axis()` to add male and female to the x-axis.

```
x=barplot(tab.sex.smoke,beside=TRUE,col=3:2)
axis(side=1,#which side to plot on, in the order bottom, left, right, top
     at=x,#where to draw the axis labels
     labels=c("M","F","M","F"),#what to label
     tick=FALSE,#surpress the tick marks
     line = -1)#I want it 1 line CLOSER to the figure
```



4 Boxplots

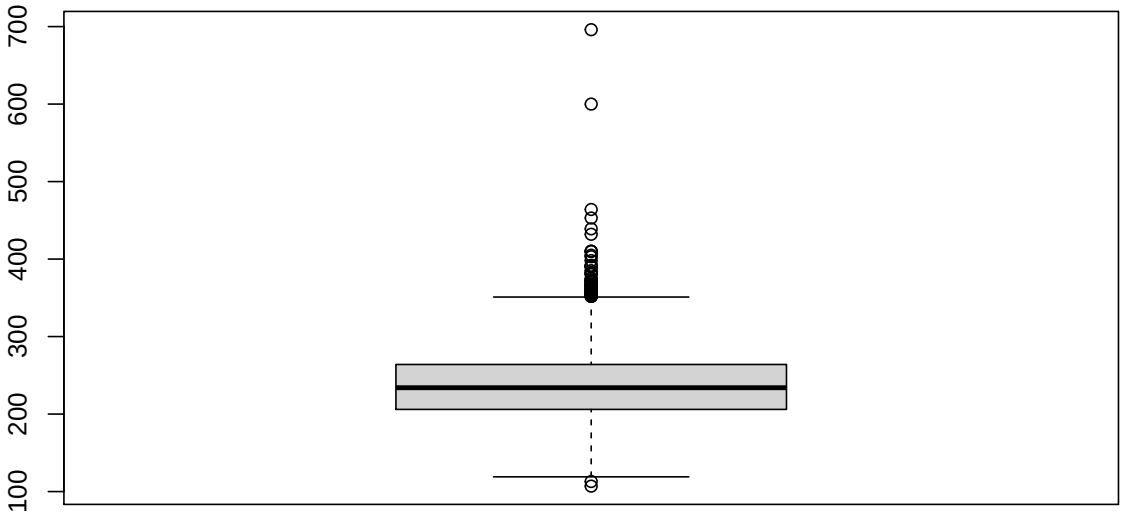
```
## S3 method for class 'formula'
boxplot(formula, data = NULL, ..., subset, na.action = NULL,
        xlab = mklab(y_var = horizontal),
        ylab = mklab(y_var =!horizontal),
        add = FALSE, ann = !add, horizontal = FALSE,
        drop = FALSE, sep = ".", lex.order = FALSE)

## Default S3 method:
boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE,
        notch = FALSE, outline = TRUE, names, plot = TRUE,
        border = par("fg"), col = "lightgray", log = "",
        pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5),
         ann = !add, horizontal = FALSE, add = FALSE, at = NULL)
```

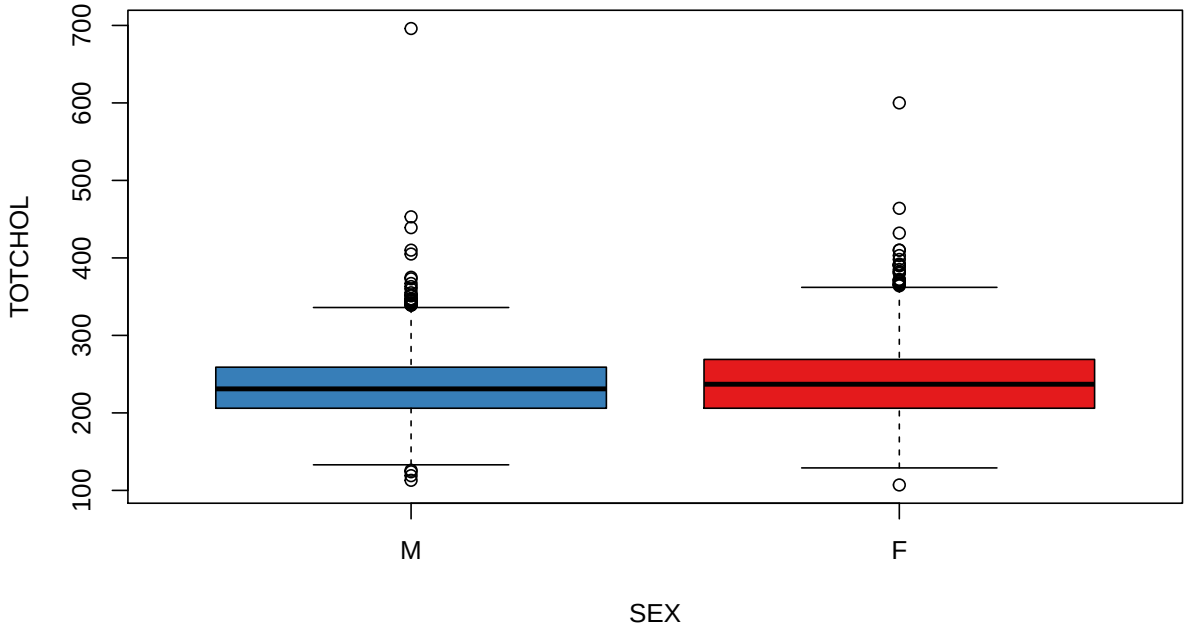
To understand boxplots we have to understand formulas in R. Formulas in R take the form `Y~X1+X2+X3`, where `Y` is the outcome and `X1, X2` and `X3` are the predictors. We'll use this same structure to build regression models in a couple of weeks.

For boxplots we can can use the formula to stratify the boxplots across the levels of `X`.

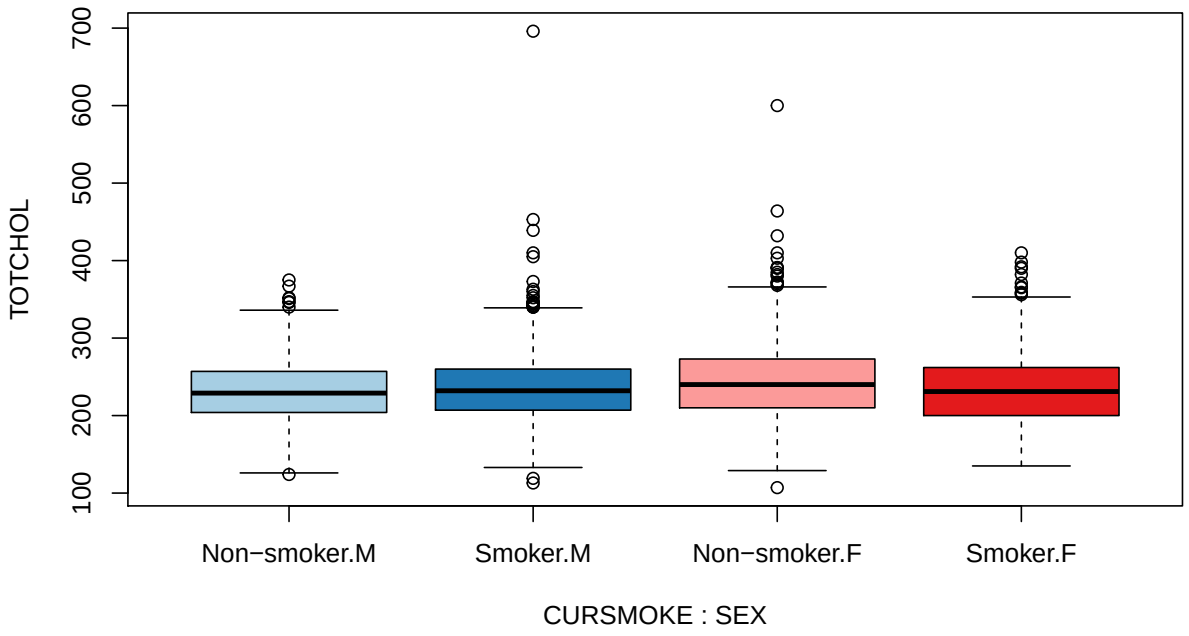
```
boxplot(dat$TOTCHOL)
```



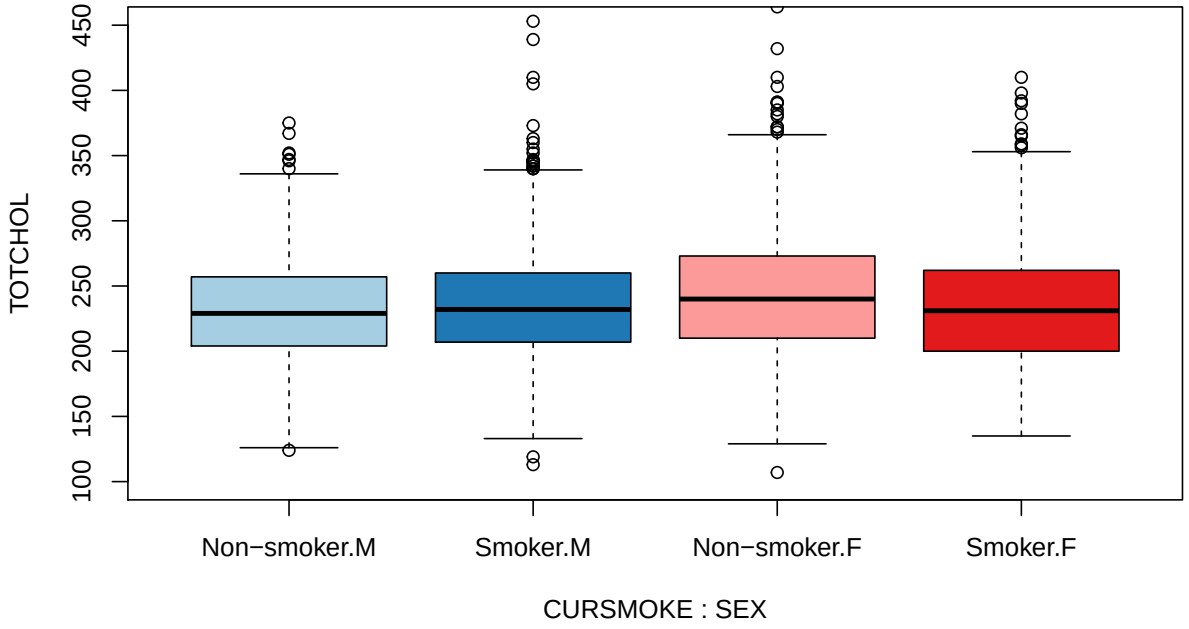

```
boxplot(TOTCHOL~SEX,data=dat,col=3:2)
```



```r
#here's where we can make use of the other colour sets R colour brewer
cols = brewer.pal(4,'Paired')
#I actually want blue and red, so I'll take the 6 colours and drop the middle two
cols = brewer.pal(6,'Paired')[c(1,2,5,6)]
boxplot(TOTCHOL~CURSMOKE+SEX,data=dat,col=cols)
```



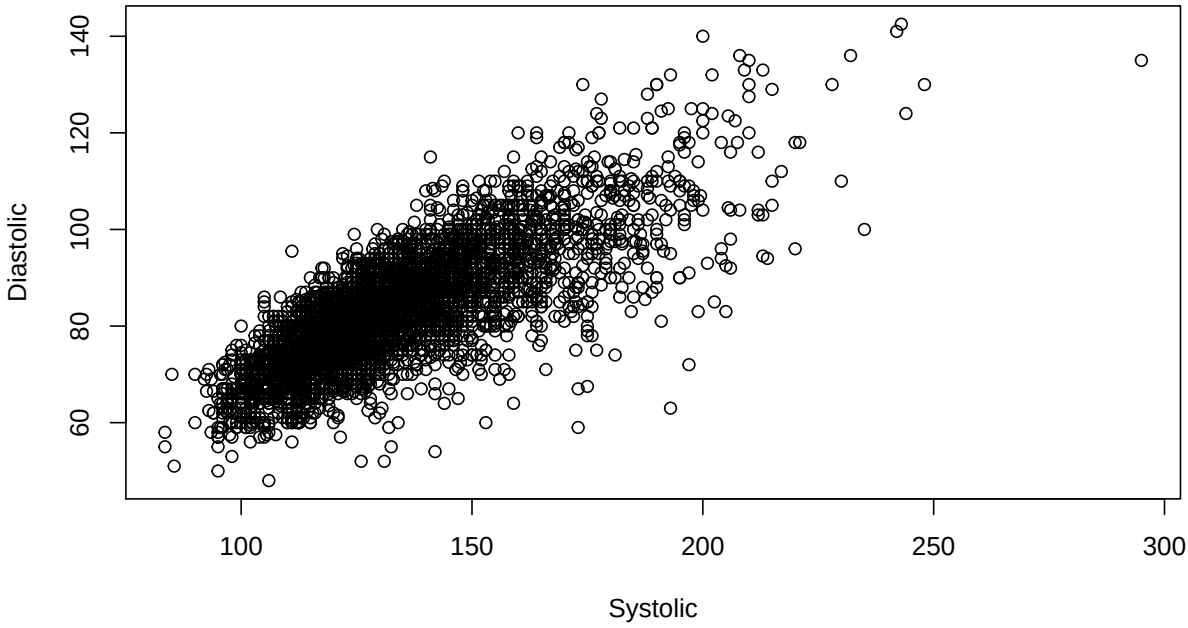

```r
#and I want to shrink the y-axis to get a better sense
boxplot(TOTCHOL~CURSMOKE+SEX,data=dat,col=cols,ylim=c(100,450))
```

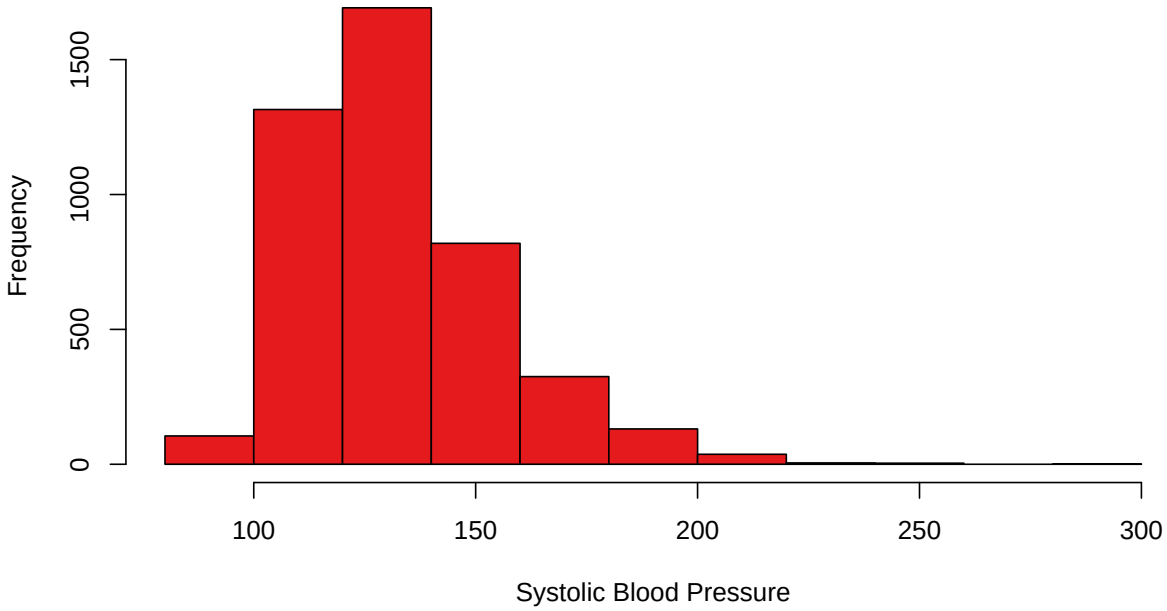


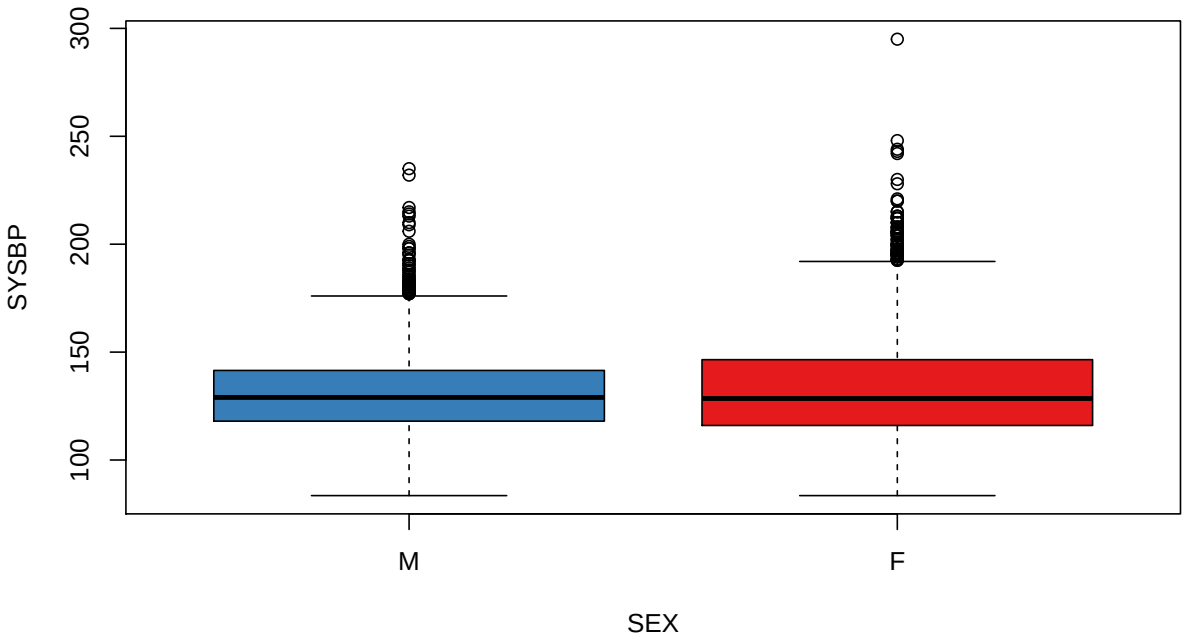
5 Breakout

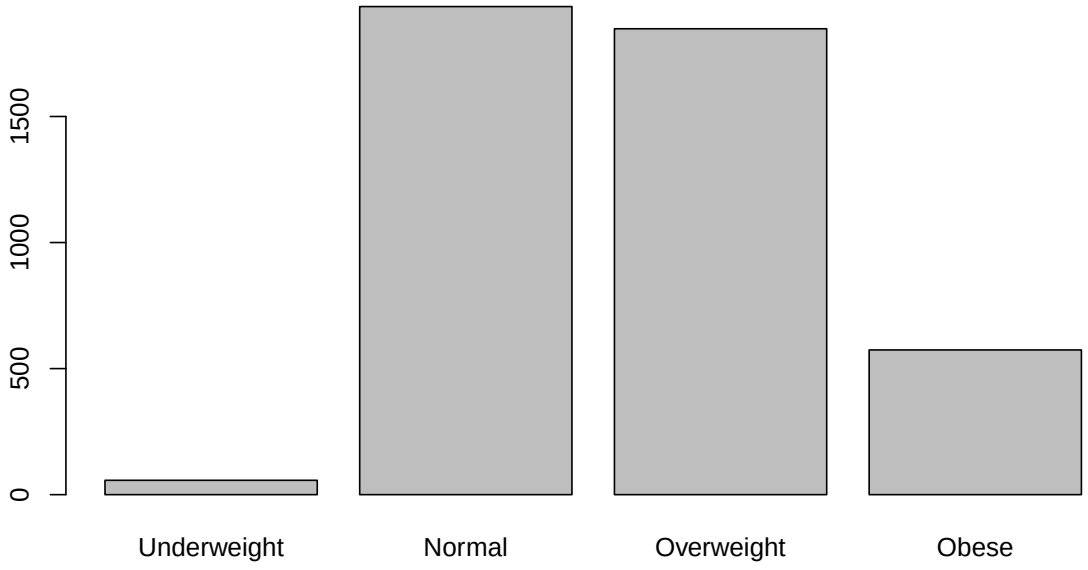
We're going to use the Framingham data for this activity (so no data cleaning this week). Load it (the command at the top of this report should be sufficient) and the try to re-create the 5 plots below. The variables we're using are:

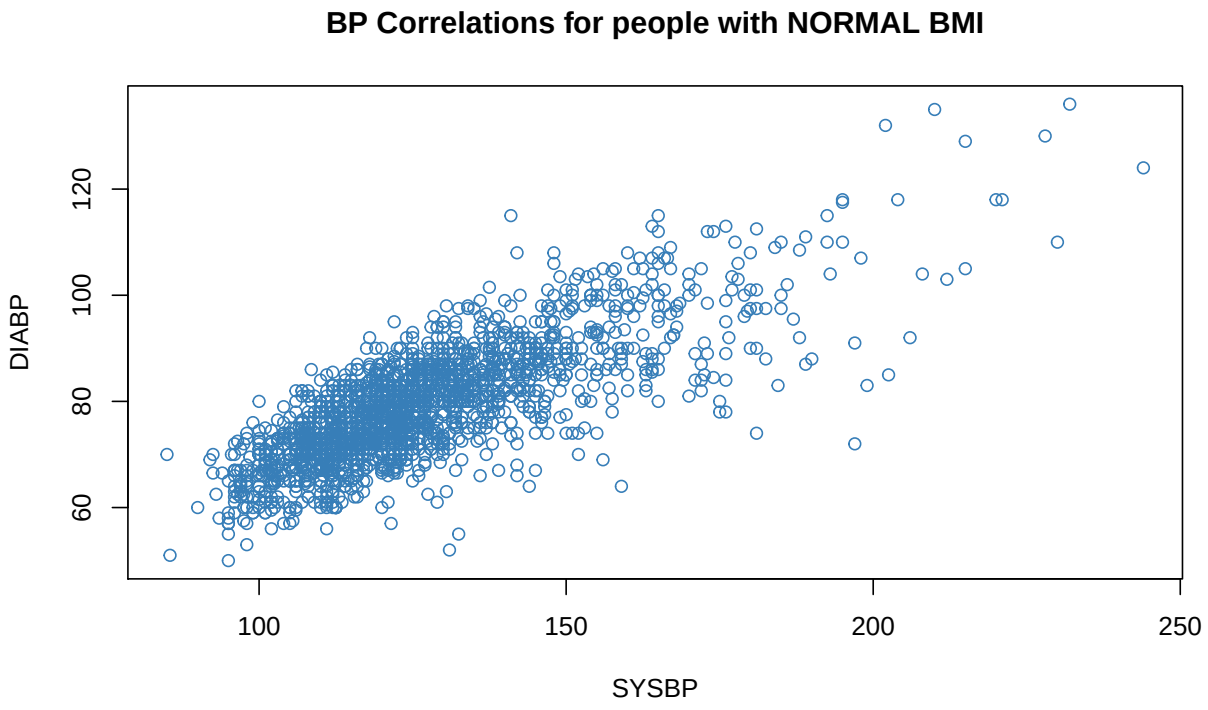
- SYSBP and DIABP
- SEX
- BMIGroups (created above)

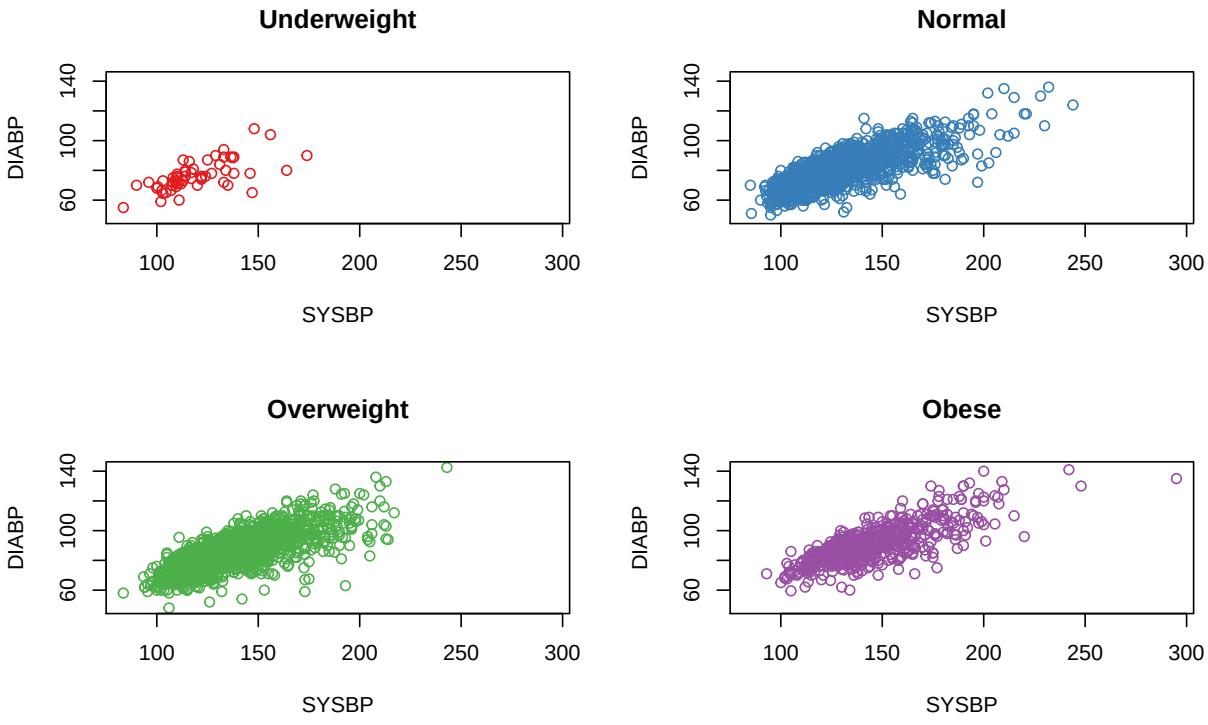

Diastolic
60
80
100
120
140
100
150
200
250
300
Systolic


Frequency
0
500
1000
1500
100
150
200
250
300
Systolic Blood Pressure


SYSBP
100
150
200
250
300
M
F
SEX


0
500
1000
1500
Underweight
Normal
Overweight
Obese





6 Saving Images

There are a couple of ways to save images in R

- Above the figure there is an *Export* button where you can save the image, and control the attributes
- There are functions that change the plotting device to a file - the most common ones I use are `pdf()` and `png()`.

To use the functions you initiate the capture, create the plot(s), and then turn the device off with `dev.off()`. Note that `dev.off()` is also useful if you want to kill your current plotting window - this is useful if you were messing with the arguments in `par()` and want to reset to the default values.

```
pdf("savedImages.pdf",height=5,width=8)
plot(mtcars$mpg,mtcars$disp)
plot(mtcars$mpg,mtcars$disp,pch=mtcars$gear,xlab='MPG',ylab='Displacement')
plot(mtcars$mpg,mtcars$disp,col=mtcars$gear,xlab='MPG',ylab='Displacement',pch=19)
legend("topright",pc=19,col=3:5,legend=3:5)
dev.off()
```

```
## pdf
##   2
```

The nice thing about this approach is that you can control the DPI and image size, and all those other frustrating things some journals care about. When possible submit your images as PDF or EPS - their vector formats are lossless, leading to the crispest images. If you have to use PNG then check the helpfile (and maybe some Googling) to determine how to get your required DPIs.

7 Preview

```
#does that look like an interaction in the SEX*CURSMOKE Boxplot to anyone else?
mod01 = lm(TOTCHOL~CURSMOKE*SEX,data=dat)
car::Anova(mod01)
```

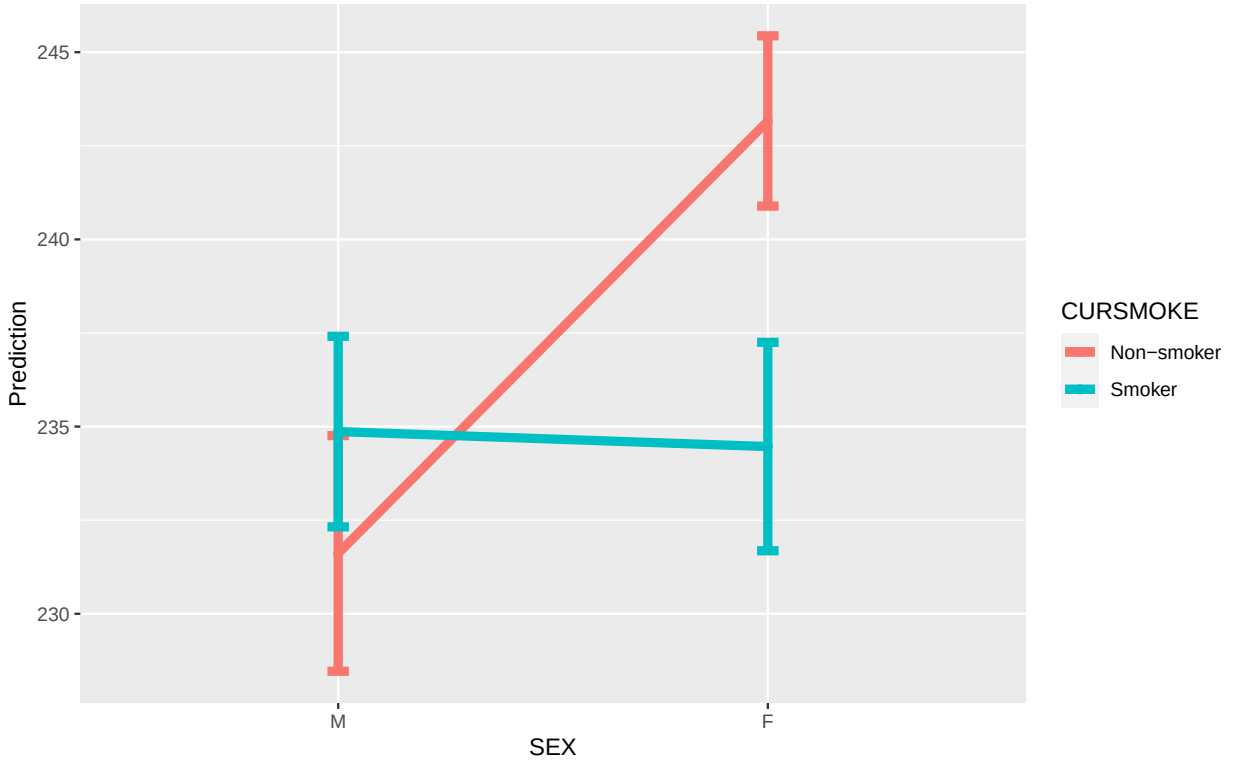
```
## Anova Table (Type II tests)
##
## Response: TOTCHOL
##               Sum Sq   Df F value    Pr(>F)
## CURSMOKE       12328    1  6.2432    0.0125 *
## SEX            30244    1 15.3162 9.232e-05 ***
## CURSMOKE:SEX   36975    1 18.7251 1.544e-05 ***
## Residuals    8644949 4378
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#Hmmm, significant interaction in the Anova, let's dig deeper
library(margins)
pred = prediction(mod01,at=list(SEX=levels(dat$SEX),CURSMOKE=levels(dat$CURSMOKE)),calculate_se = TRUE)
summary(pred)
```

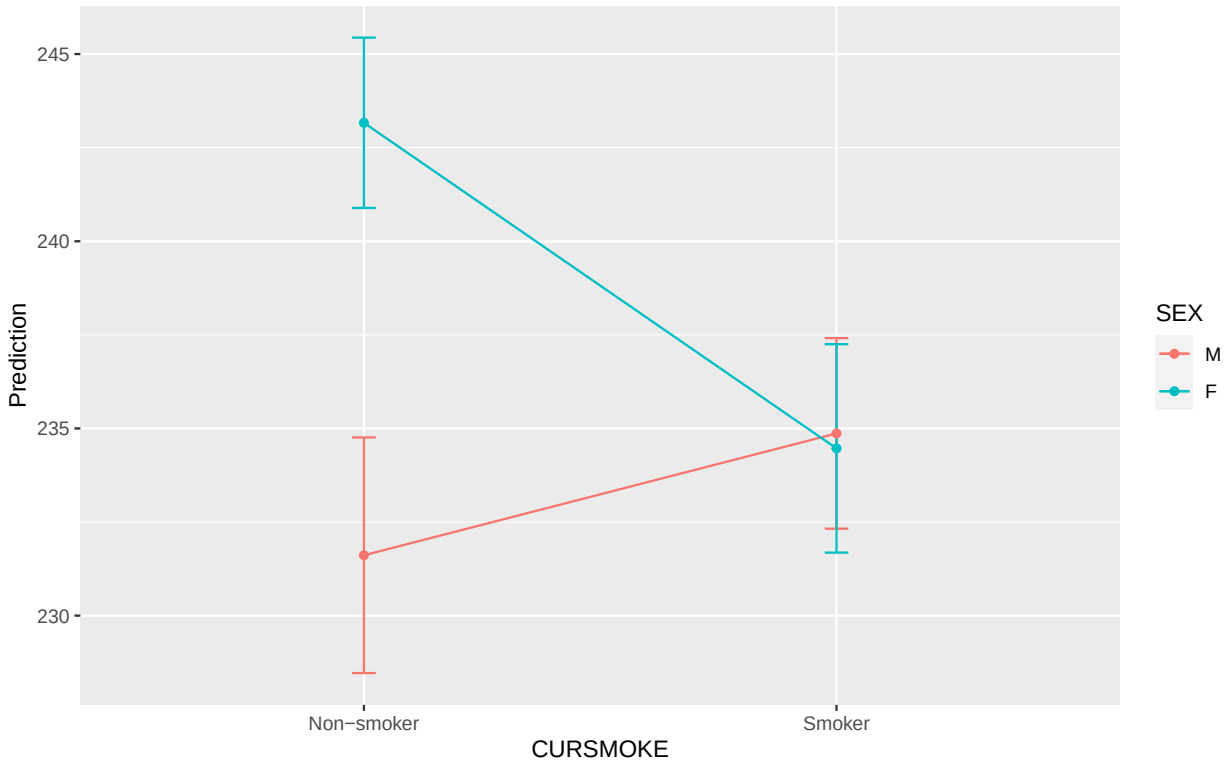
```
##  at(SEX) at(CURSMOKE) Prediction    SE     z p lower upper
##        M   Non-smoker      231.6 1.606 144.3 0 228.5 234.8
##        F   Non-smoker      243.2 1.161 209.5 0 240.9 245.4
##        M       Smoker      234.9 1.299 180.9 0 232.3 237.4
##        F       Smoker      234.5 1.420 165.1 0 231.7 237.3
```

```
#so it looks like the sex difference disappears in smokers, let's visualize
#...and now it gets really weird
d = as.data.frame(summary(pred))
names(d) = gsub("at\\((.*)\\)","\\1",names(d))#ok now you're just showing off
#This is a good ggplot function learn how ggplot is built in parts
```

```
ggplot(d,aes(x=SEX, y=Prediction, colour=CURSMOKE, group=CURSMOKE))+
  geom_errorbar(aes(ymin=lower,ymax=upper),width=0.05,size=2)+
  geom_line(size=2)+
  geom_point()
```




```
#I can't decide if the fat lines are better
ggplot(d,aes(x=CURSMOKE, y=Prediction, colour=SEX, group=SEX))+
  geom_errorbar(aes(ymin=lower,ymax=upper),width=0.05)+
  geom_line()+
  geom_point()
```




```
#we'll end with the marginal differences - this is one of the weaker
#R libraries I have to use consistently, but it still gets the job done
marg01 = margins(mod01,at=list(SEX=levels(dat$SEX)))
summary(marg01)
```

```
##          factor    SEX     AME     SE       z      p    lower   upper
##  CURSMOKESmoker 1.0000  3.2545 2.0650  1.5761 0.1150  -0.7928  7.3018
##  CURSMOKESmoker 2.0000 -8.6969 1.8341 -4.7418 0.0000 -12.2917 -5.1021
##            SEXF 1.0000  5.6876 1.3819  4.1156 0.0000   2.9790  8.3961
##            SEXF 2.0000  5.6876 1.3819  4.1156 0.0000   2.9790  8.3961
```

```
marg02 = margins(mod01,at=list(CURSMOKE=levels(dat$CURSMOKE)))
summary(marg02)
```

```
##          factor CURSMOKE     AME     SE       z      p   lower   upper
##  CURSMOKESmoker   1.0000 -3.4140 1.3713 -2.4896 0.0128 -6.1017 -0.7263
##  CURSMOKESmoker   2.0000 -3.4140 1.3713 -2.4896 0.0128 -6.1017 -0.7263
##            SEXF   1.0000 11.5514 1.9811  5.8308 0.0000  7.6685 15.4343
##            SEXF   2.0000 -0.4000 1.9244 -0.2078 0.8353 -4.1717  3.3718
```