Data Programming with R

Isabella Gollini

Lecture 3 - Lists and Data Frames

R graphics

- Some resources
- `plot()`
- Plots in Rstudio
- Adding points and lines; Adding text
- Colours; Legends
- More on customisation
- Graphical parameters
- Saving graphs to files
- Common graph types Multiple panels
- 3D plots
- Maps
- Extended examples

Introduction to R graphics

- R is able to create rich, publication quality graphics very easily
- For some impressive examples, see:
  - http://www.r-graph-gallery.com/
  - https://plot.ly/r/
  - http://shinyapps.stat.ubc.ca/r-graph-catalog/

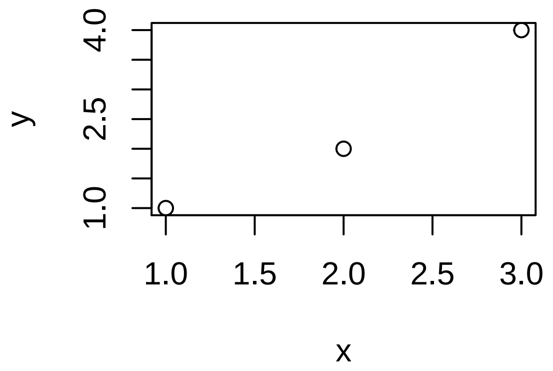
```
demo(graphics)
```

- (Note: some of these are more advanced than we will cover, but don't let that stop you!)
- There are many excellent online guides to creating sophisticated R graphs

The `plot` function

The default way to create a graph in R is to use the `plot` function. The general format is `plot(x, y)` where `x` and `y` are vectors.

```
x <- c(1, 2, 3); y <- c(1, 2, 4)
plot(x, y)
```

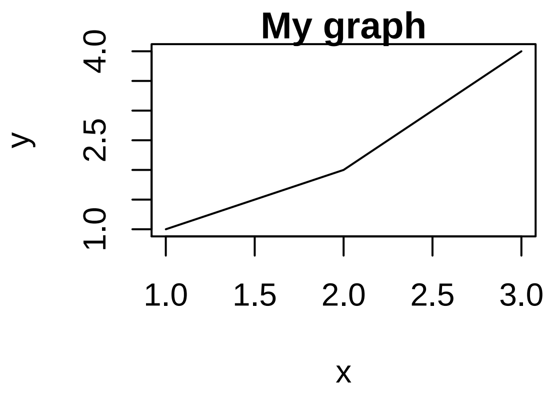


This creates a very plain graph. Note that R has paired these up into (1, 1), (2, 2), (3, 4). By default the plotting character is an empty circle.

The plot function

A slightly fancier version is:

```r
plot(x, y,
  xlab = "x", ylab = "y", # specify the x and y-axis labels
  type = "l", # specify the plot type to be a line
  main = "My graph") # specify the main title
```



Other types include `p` for points (the default), `b` for both points and lines, and `n` for no plotting of points.

Plots in RStudio

SC 1

- RStudio handles plots slightly differently to the default R GUI. It only allows for one graph panel at a time but you can cycle between different plots.
- All plots are shown in the plot tab in the relevant portion of the Rstudio screen. For example:

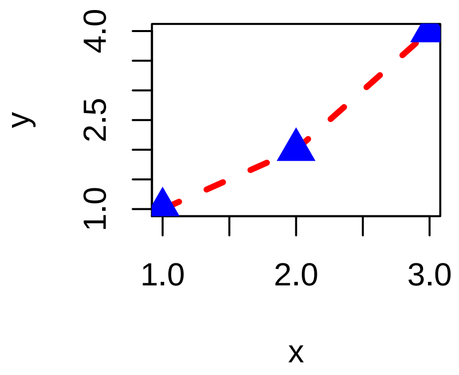
```r
plot(c(1, 2, 3), c(1, 2, 3))
plot(c(3, 2, 1), c(1, 2, 3))
```

- `plot(c(3, 2, 1), c(1, 2, 3))` will overwrite the first plot. However clicking the 'back arrow' button will return to the first plot
- There are also simple buttons to zoom in on a plot or to export it into a pdf file (more on this later).
- See the screencast for a demonstration of how to use the RStudio graphics tools

Adding to graphs

Once we have created a default plot with the plot function we can add points and lines to it with the appropriate functions.

```r
plot(x, y, type = "n") # create blank graph
lines(x, y, col = "red", lty = 2, lwd = 3) # add the red line
points(x, y, col = "blue", pch = 17, cex = 2) # add the blue points
```



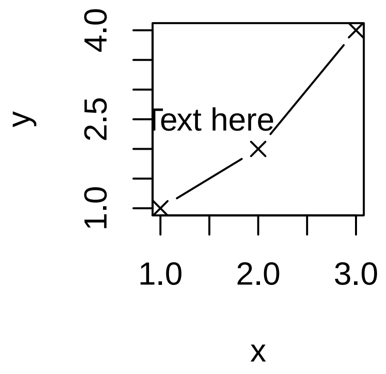
- `lty` specifies the line type. `lwd` specifies the line width. See `help(lines)` for more details.
- `pch` specifies the plotting character. `cex` specifies the point size. See `help(points)` for more options.

Adding text

- We may want to add additional text to the graph.

To add text in the main plotting window use the `text` function.

```r
plot(x, y, type = "b", pch = 4)
text(1.5, 2.5, "Text here")
```

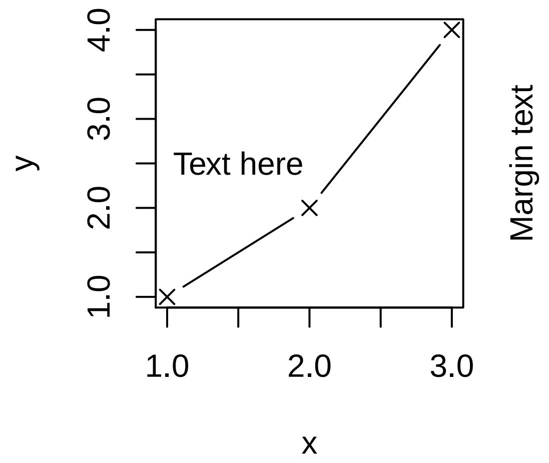


- The first two arguments specify the `x`, `y` locations of the text to be inserted whilst the third argument gives the text itself
- A useful extra argument is `pos` which allows you to put the text below (`pos = 1`), left (`pos = 2`), above (`pos = 3`), right (`pos = 4`) the chosen location

Adding text

If you wish to add text in the margin (i.e. outside the plotting region), use the `mtext` function

```r
mtext("Margin text", side = 4, line = 1)
```



- The `side` argument indicates which margin to write beside (1 = bottom, 2 = left, 3 = top, 4 = right) and `line` tells R how far away from the margin it should.

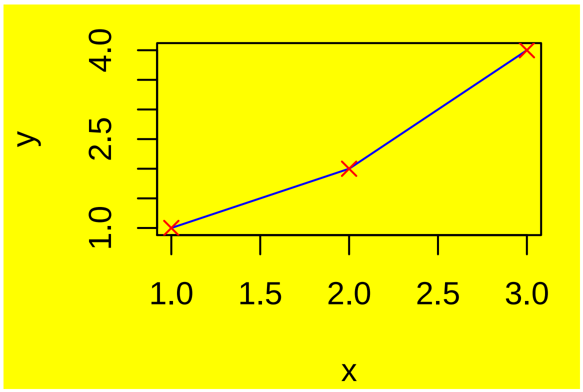
Changing colours

R has a default list of colours. You can see the (very long) list via:

```r
colours()
```

and use them with:

```r
par(bg = "yellow") # yellow background
plot(x, y, type = "n")
lines(x, y, col = "blue") # blue lines
points(x, y, pch = 4, col = "red") # red points
```

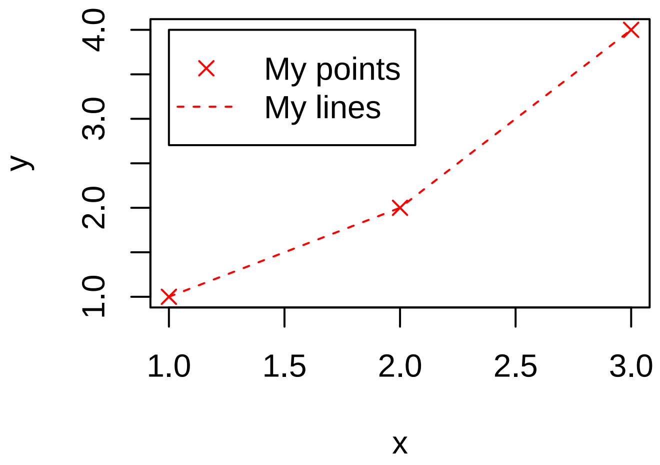


- `par` here controls persistent graphical parameters (more on this later).
- You can also create your own colours via the function `rgb`.

Adding a legend

We can add a legend using the `legend` function. The default usage is `legend(x, y, legend, pch, lty, col)` where `x, y` is the location where you want the legend to be, `legend` is a vector of names, `pch` are the plotting characters, `lty` are the line formats, and `col` are the colours. Example:

```r
plot(x, y, type = "p", pch = 4, col = "red")
lines(x, y, lty = 2, col = "red" )
legend(1, 4, # x, y coordinates for the legend
 legend = c("My points", "My lines"), # vector containing the text to appear in the legend
 pch = c(4, -1), lty = c(-1, 2), col = "red") # graphical param: point type, line type and color
```



- Note here that `-1` produces a blank (i.e. no point or line).
- You can replace `x, y` with `"topleft"` or `"bottomleft"`, etc.
- If you want to get rid of the box use `bty = "n"`.

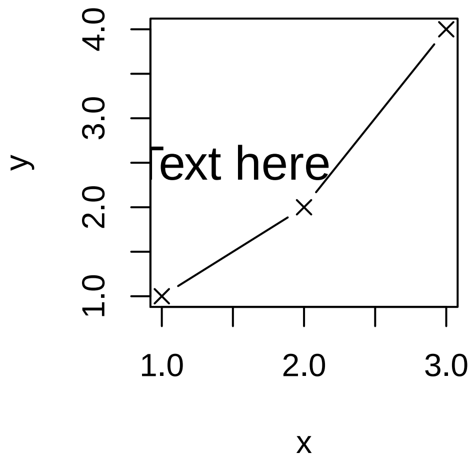
Customisation

More on customisation Graphical parameters Saving graphs to files

Sizes

Many plotting functions have a `cex` argument. This stands for character expansion and is given relative to the standard size. Thus

```r
plot(x, y, type = "b", pch = 4) # same plot of slide 8
text(1.5, 2.5, "Text here", cex = 1.5)
```

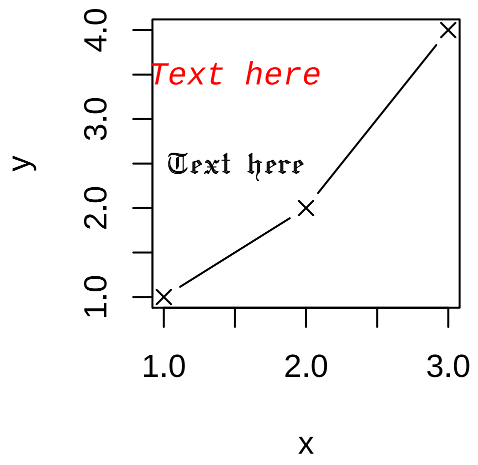


- `cex = 1.5` will give text that is 1.5 times the usual size.
- The same argument works in `plot`, `points`, `main`, `mtext`, etc to change the size of characters (including plotting points)

Fonts

- The `font` and `family` arguments allow you to change from bold/italic (`font = 2` or `3`) whilst `family` allows you to change the font:

```
text(1.5, 2.5, "Text here", font = 2, family = "HersheyGothicEnglish")
text(1.5, 3.5, "Text here", font = 3, family = "mono", col = "red")
```

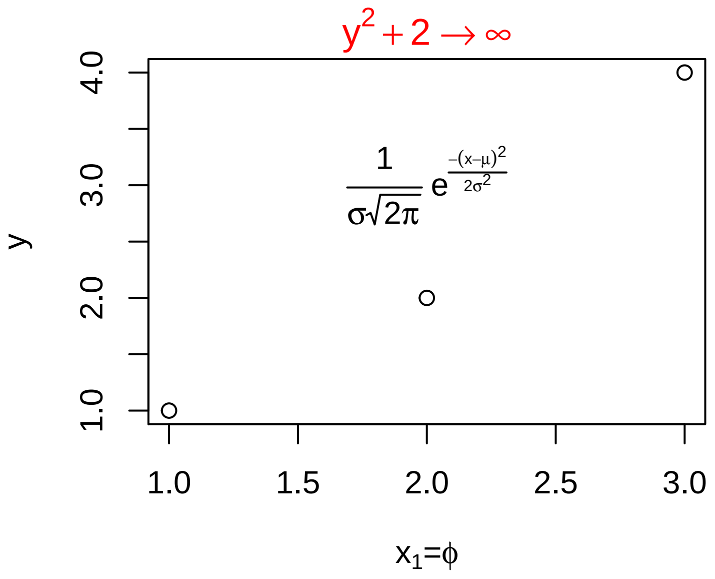


Putting maths in a legend

We often want to use greek characters or other mathematical symbols in a plot, usually in the title or the axis. `help(plotmath)` gives a list of useful commands for inclusion in plots. These use the `expression` and `paste` functions.

Some examples:

```
plot(x, y, xlab = expression(paste(x[1],"=", phi)),
  main = expression(paste(y^2 + 2 %->% infinity)), col.main = "red")
text(2, 3, expression(paste(frac(1, sigma * sqrt(2 * pi)), " ",
                            plain(e)^{frac(-(x - mu)^2, 2 * sigma^2)})))
```

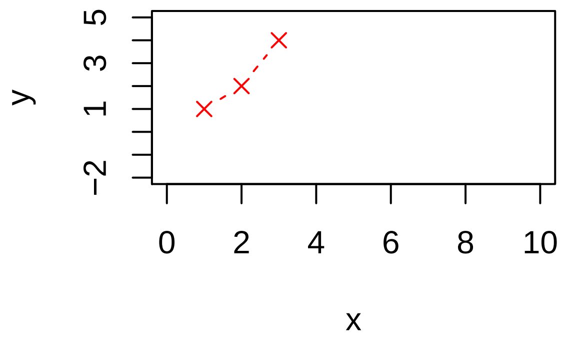


Many of the usual arguments (e.g. `cex`, `col`) can also be applied here

Axes

The `plot` arguments `xlim` and `ylim` will change the axis limits. Example:

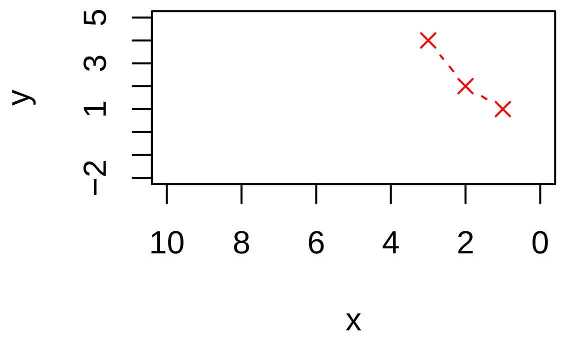
```
plot(x, y, type = "b", pch = 4, col = "red", lty = 2, xlim = c(0, 10), ylim = c(-2, 5))
```



Each requires a vector of length 2 denoting the upper and lower limit.

To reverse, e.g. the `x` axis limit, simply give it in the opposite order:

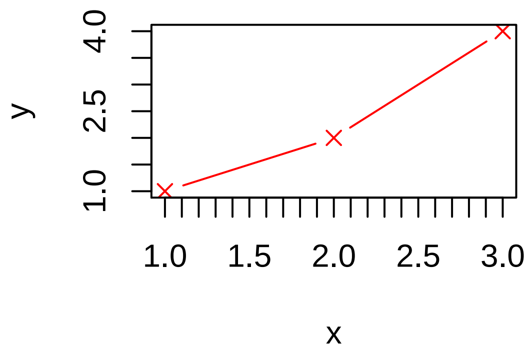
```
plot(x, y, type = "b", pch = 4, col = "red", lty = 2, xlim = c(10, 0), ylim = c(-2, 5))
```



Axes

- To create your own specialist axes, use the `axis` function:

```
plot(x, y, type = "b", pch = 4, col = "red", xaxt = "n")
axis(side = 1, at = seq(1, 3, by = 0.1), labels = TRUE)
```



Setting persistent defaults

- The `par` function creates a default plotting environment although everything you set using it is persistent - be careful.

- `help(par)` is a very long document, but some useful arguments include:
    - `mar` and `mgp` which set the default margins and plotting region.
    - `usr` which gets the current plot limits.
    - `mfrow`/`mfcol` which plots an array of graphs
    - `las` which rotates the axis labels.
    - `bg` which sets the background colour.

A useful default command to run at the start of a function to produce plots is:

```
par(mar = c(3, 3, 2, 1), mgp = c(2, 0.7, 0), las = 1)
```

R graphical parameters

source of the image: http://gastonsanchez.com/r-graphical-parameters-cheatsheet.pdf

Saving graphs to files

- R allows you to save your graph in a variety of formats.
- The RStudio export button tends to produce inconsistent results - it is far better to save your graphs as part of your command script.
- The usual method for doing this is, e.g.

```
pdf("myfilename.pdf", width = 10, height = 8)
# Plotting commands go here
dev.off()
```

- The `dev.off()` function tells R that we have finished our plot and to send it to the file name specified.
- Replacing `pdf` with another function (e.g. `postscript`, `jpeg`, `png`, `tiff`) will save the file in the corresponding format.
- Be careful to choose `width` and `height` carefully as units can change between the different formats (pdf is inches, jpeg pixels).

Fancy graphics

Common graph types Multiple panels 3D plots Maps

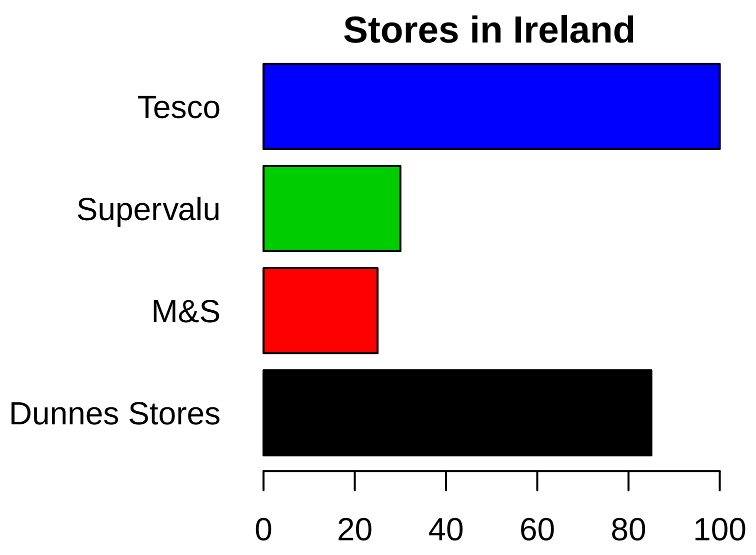
Common graph types

- Aside from the scatter plots which we have focussed on so far, R can create many standard graphical outputs, including:
    - bar charts
    - histograms
    - density plots
    - boxplots
    - pie charts
- Each of these has their own function, though many of the arguments we have covered (axis limits, titles, colours, etc) all work for these plots too.

Bar charts

Bar charts can be used to graph factor-type data

```
stores <- factor(rep(c("Tesco", "Supervalu", "M&S", "Dunnes Stores"), c(100, 30, 25, 85)))
par(mar = c(3, 7, 1, 1)) # Adjust the margins on the four sides of the plot
barplot(height = table(stores), main = "Stores in Ireland", xlab = "Number of stores",
  col = 1:4, horiz = TRUE, las = 1)
```



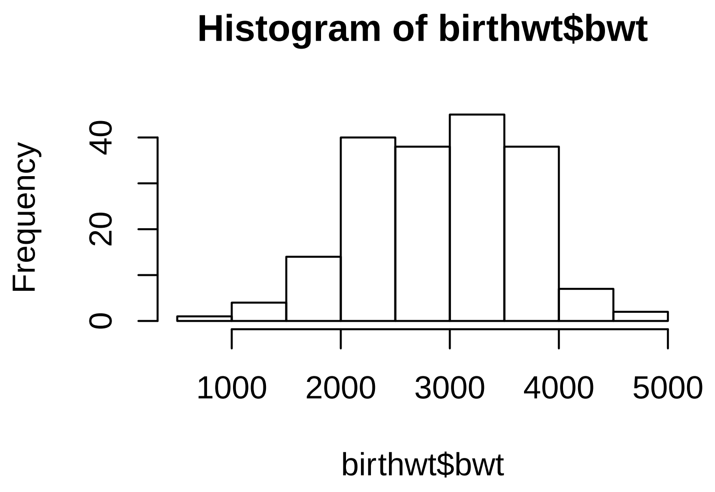
- The important argument is `height` which specifies the height of the bars, here taken from the table of store counts.
- I've also added a title, changed the colour, labelled the x-axis, set the bars to be horizontal, and set the labels the right way up.

Histograms

- Histograms are like bar charts but for numeric data.
- They are creating by putting the observations into bins (or breaks as R calls them) and then counting the number of observations in each bin.

Example:

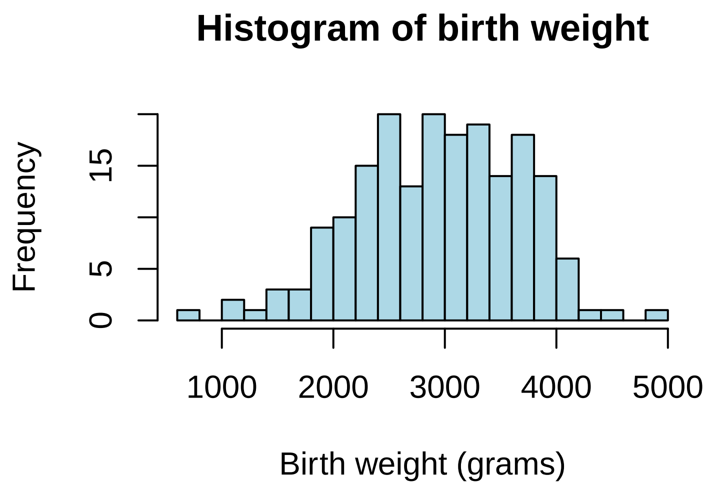
```r
library(MASS)
hist(birthwt$bwt)
```



Histograms

- The default is a bit bland. Try:

```r
hist(birthwt$bwt, breaks = 30, xlab = "Birth weight (grams)",
  main = "Histogram of birth weight", col = "lightblue")
```



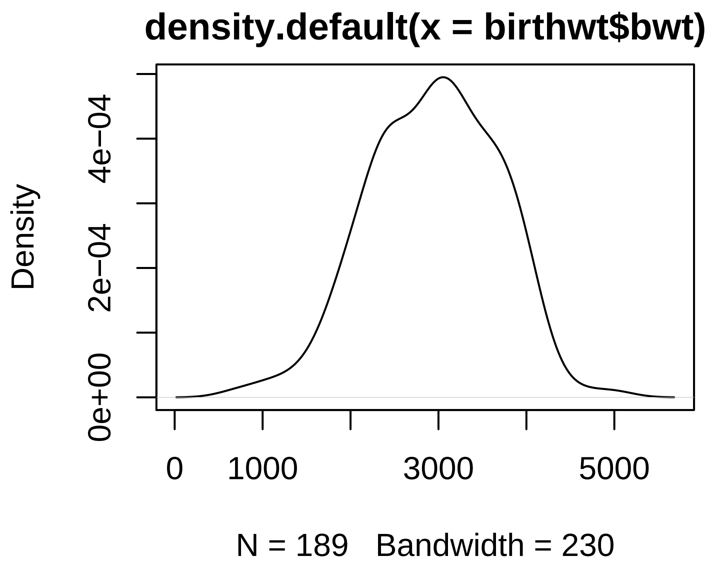
- The `breaks` argument allows you to input either the number of bins or to specify them as, e.g. a sequence.

Density plots

- A density plot (also known as a kernel density plot) is a continuous line version of a histogram. Instead of specifying bins we specify a window size which slides along the x-axis and counts the number of observations in the window.

The function is not a plotting function but can be called directly by `plot`:

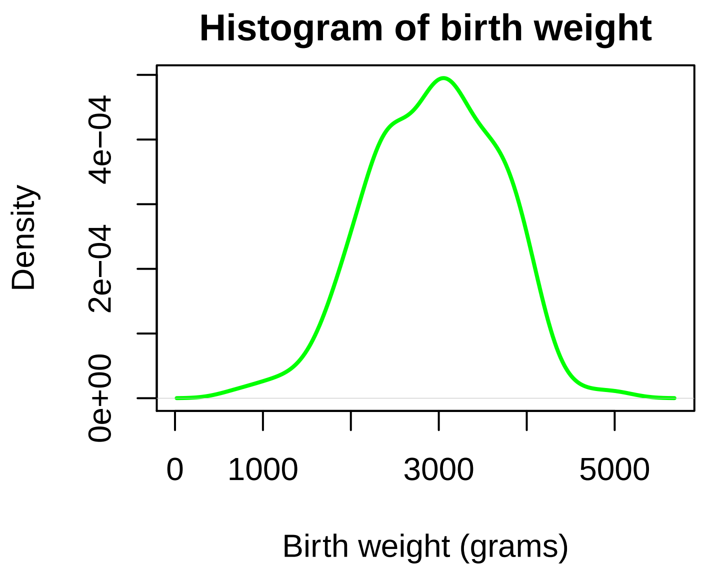
```r
plot(density(birthwt$bwt))
```



Density plots

- Improved version:

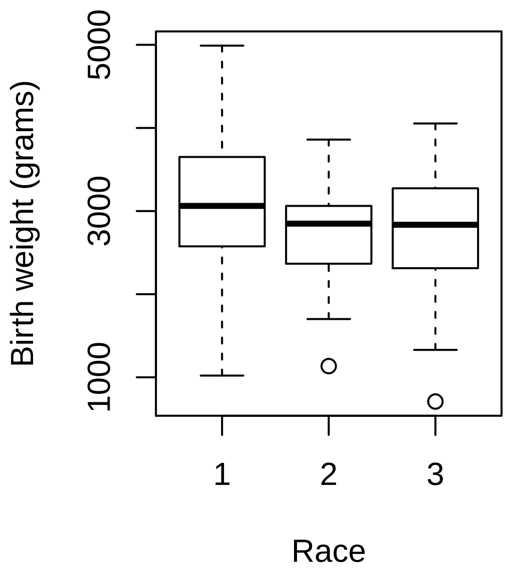
```r
plot(density(birthwt$bwt), xlab = "Birth weight (grams)",
  main = "Histogram of birth weight", col = "green", lwd = 2)
```



Boxplots

Boxplots (also called box and whisker plots) are very useful for comparing numeric measurements across different categories.
A boxplot is made up of 'whiskers' representing the maximum and minimum of the data set, a box representing the quartiles (the 25th and 75th percentiles), and a line representing the median.

```r
boxplot(birthwt$bwt ~ birthwt$race, ylab = "Birth weight (grams)", xlab = "Race")
```

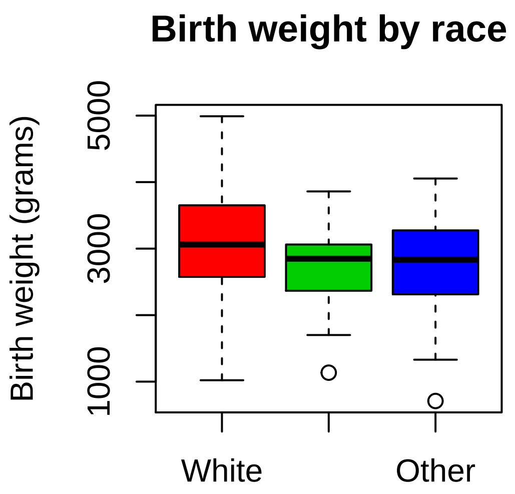


This plot is a bit crude.

Boxplots

A better version:

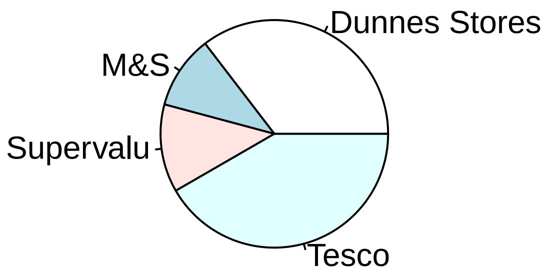
```r
boxplot(birthwt$bwt ~ birthwt$race, xaxt = "n", # xaxt = "n" removes the x axis
  main = "Birth weight by race", col = 2:4, ylab = "Birth weight (grams)")
# Add the x axis with new labels:
axis(1, at = 1:3, labels = c("White", "Black", "Other"))
```



Pie charts

Pie charts are almost always inferior to bar charts and should only be used in very exceptional circumstances.

```r
pie(table(stores))
```

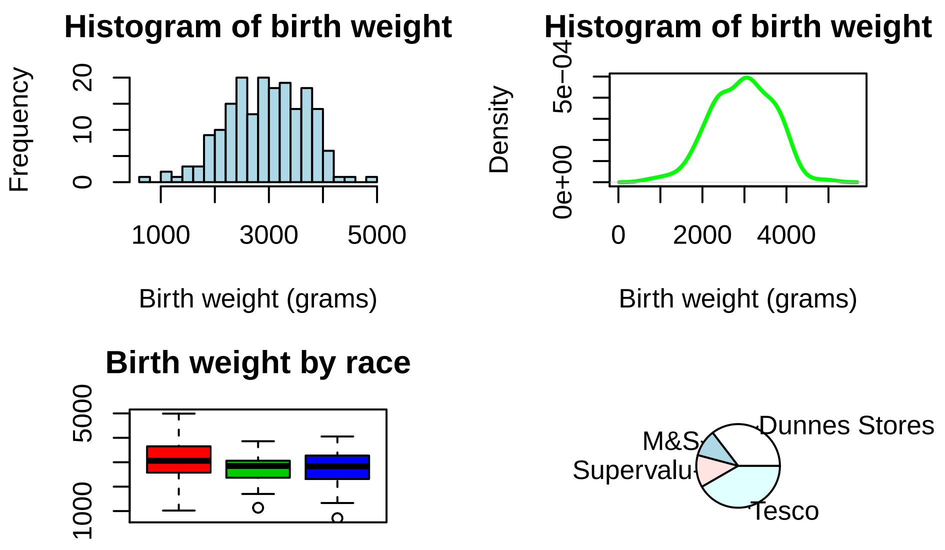


- From `help(pie)`: *Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data..*
- From Wikipedia: *Statisticians generally regard pie charts as a poor method of displaying information, and they are uncommon in scientific literature. One reason is that it is more difficult for comparisons to be made between the size of items in a chart when area is used instead of length and when different items are shown as different shapes.*

Multiple panels

Often we want to plot multiple graphs on a single panel. R has a few different methods for doing this. The simplest is through `par`

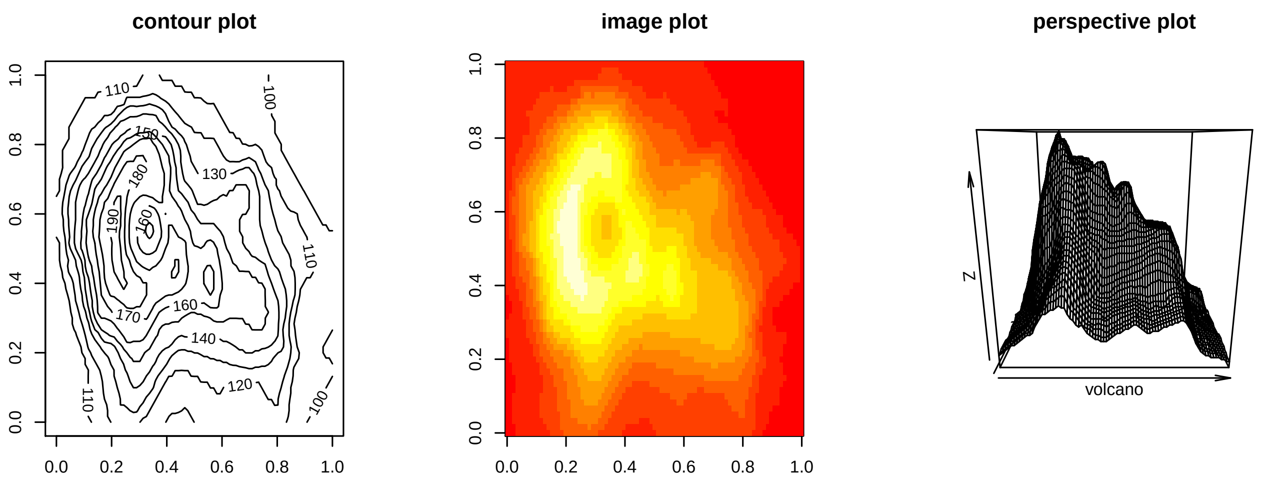
```r
par(mfrow = c(2, 2)) # creates a 2 by 2 plotting matrix which is then filled in by row
hist(birthwt$bwt, breaks = 30, xlab = "Birth weight (grams)",
  main = "Histogram of birth weight", col = "lightblue")
plot(density(birthwt$bwt), xlab = "Birth weight (grams)",
  main = "Histogram of birth weight", col = "green", lwd = 2)
boxplot(birthwt$bwt ~ birthwt$race, xaxt = "n", main = "Birth weight by race", col = 2:4)
pie(table(stores))
```



3D plots

Where data are available in three dimensions, we can create contour plots, image plots, or perspective plots, amongst others. Usually these functions require in put as a matrix or as 3 vectors in the form x, y, z

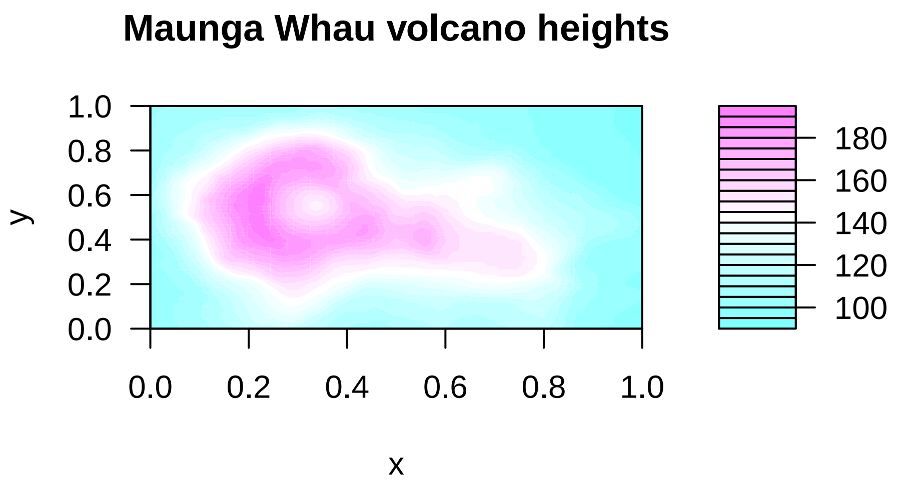
```r
par(mfrow = c(1, 3))
contour(volcano, main = "contour plot")
image(volcano, main = "image plot")
persp(volcano, main = "perspective plot")
```



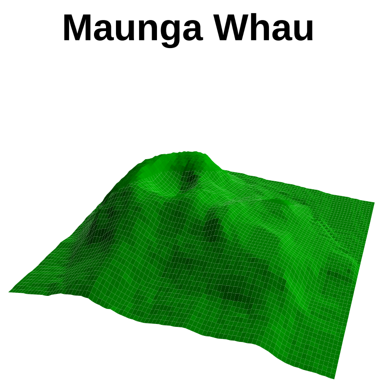
3D plots

Some nicer versions:

```r
filled.contour(volcano,
  main = "Maunga Whau volcano heights",
  xlab = "x",
  ylab = "y")
```



```r
persp(volcano, theta = 20, phi = 30,
  col = "green3", main = "Maunga Whau",
  expand = 0.25, ltheta = 150, shade = 0.75,
  border = NA, box = FALSE)
```



Maps

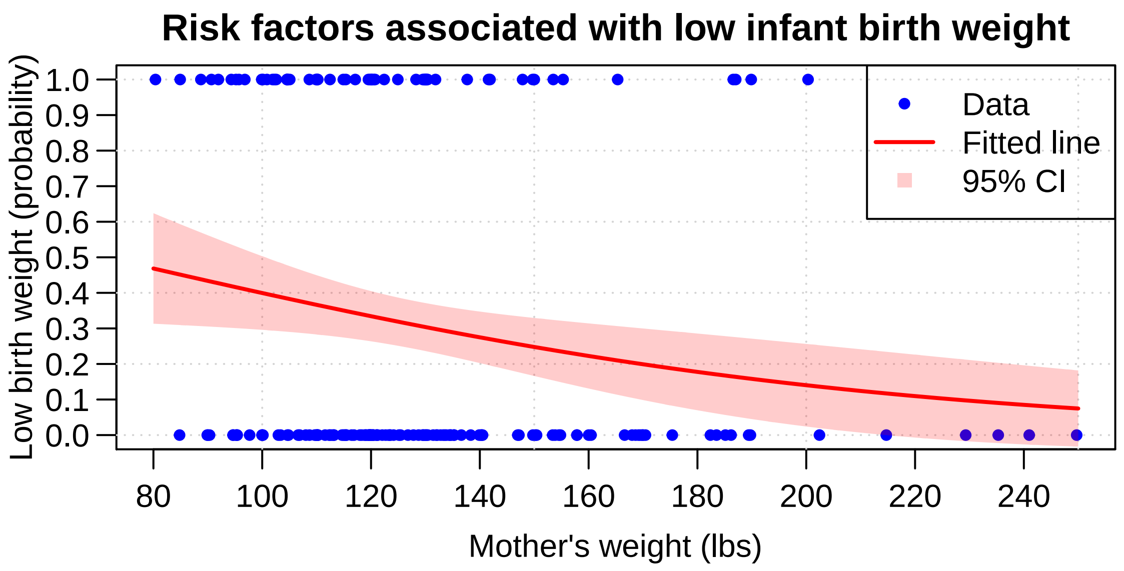
- R has the ability to read in both Google maps and OpenStreetMaps. These essentially load in the map as a background and allow you to plot on top of it.
- We will not go into this in detail as it involves quite a bit of extra coding.
- See these blog posts for more information
  - http://www.flutterbys.com.au/stats/tut/tut5.4.html
  - http://eriqande.github.io/rep-res-web/lectures/making-maps-with-R.html
  - http://blog.fellstat.com/?p=20

Extended example

Creating a publication standard plot

Publication standard graphs

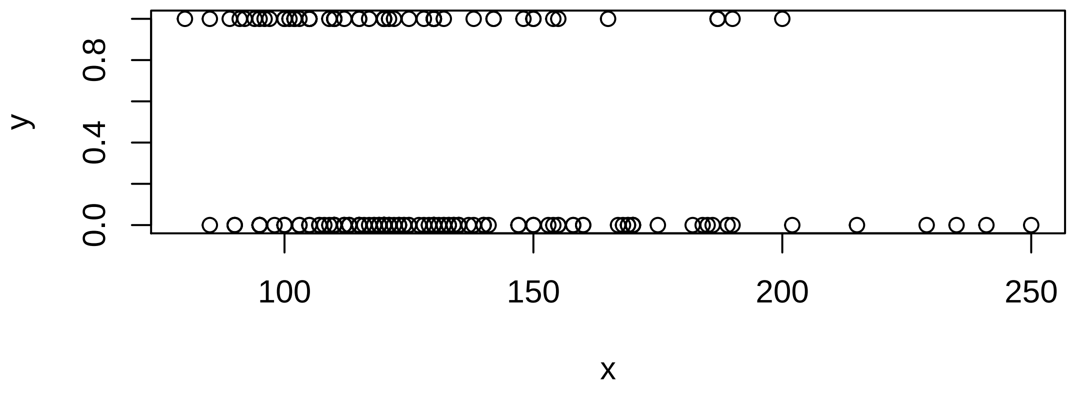
This is the plot we will try to re-create:



First steps

The default command...

```r
x <- birthwt$lwt
y <- birthwt$low
plot(x, y)
```

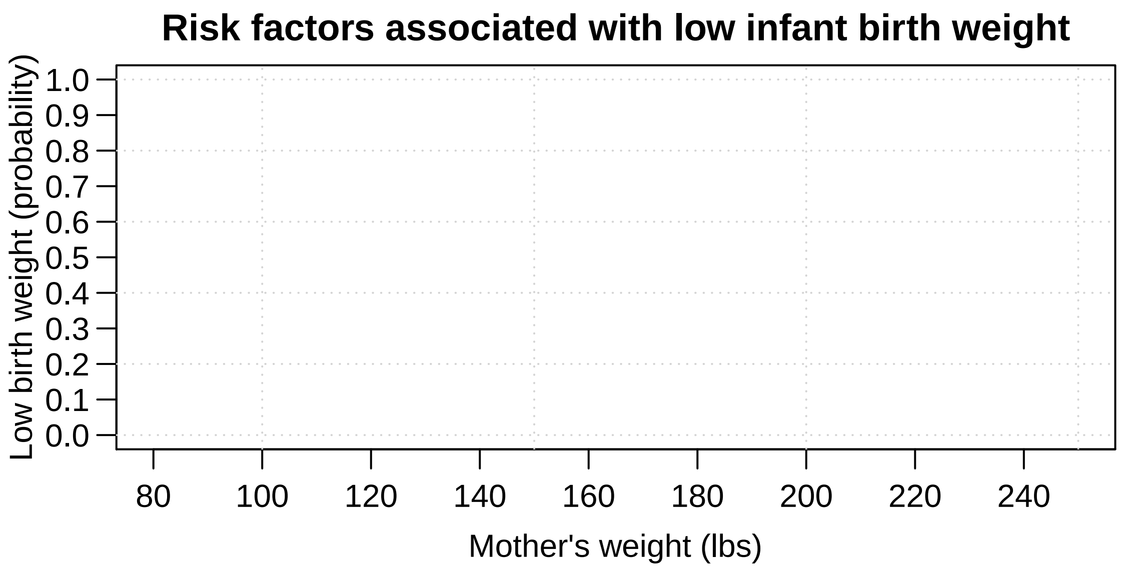


...is pretty ugly.

First steps

Let's now set up the plot frame:

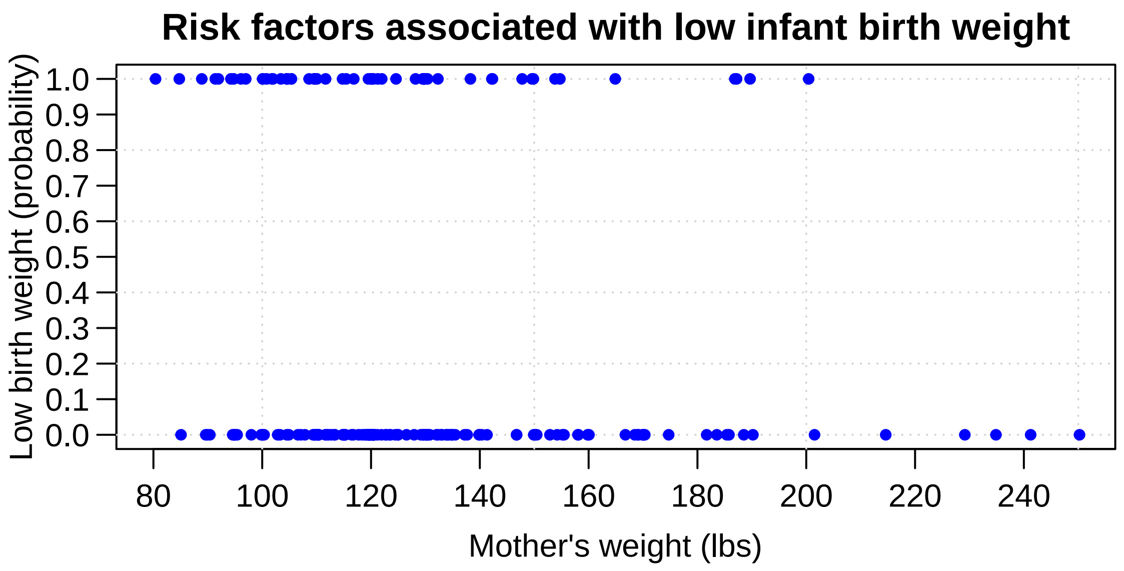
```r
par(mar = c(3, 3, 2, 1), mgp = c(2, 0.7, 0), las = 1)
plot(x, y, type = "n", xaxt = "n", yaxt = "n",
  xlab = "Mother's weight (lbs)", ylab = "Low birth weight (probability)")
title("Risk factors associated with low infant birth weight")
axis(1, at = pretty(x, n = 10))
axis(2, at = pretty(y, n = 10))
grid()
```



Adding points and lines

Let's now add the data points:

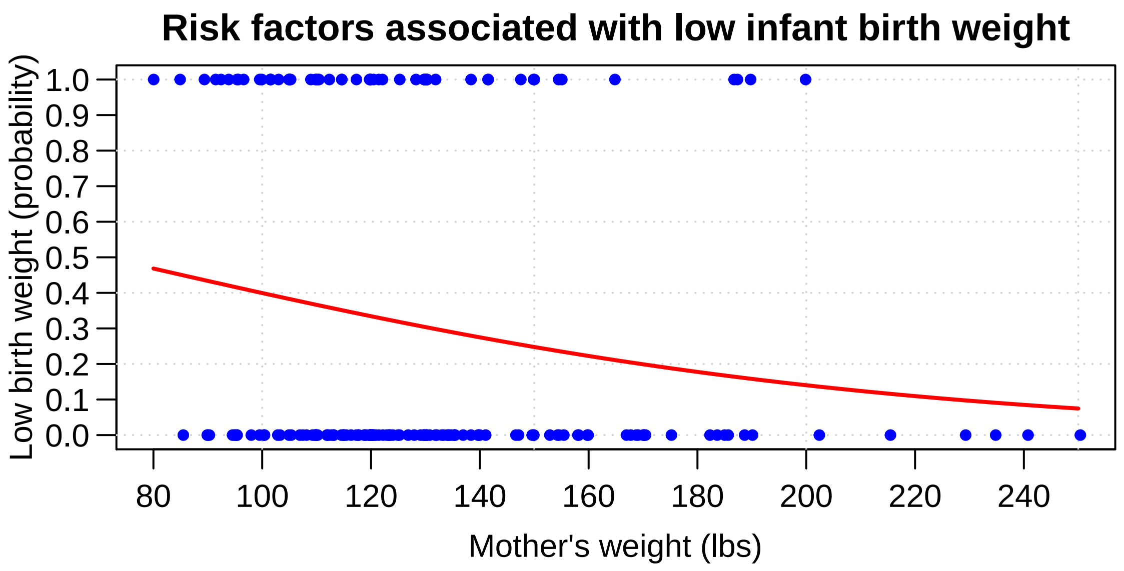
```r
points(jitter(x, amount = 0.5), y, col = "blue", pch = 20)
```



Adding points and lines

and the fitted line from a logistic regression:

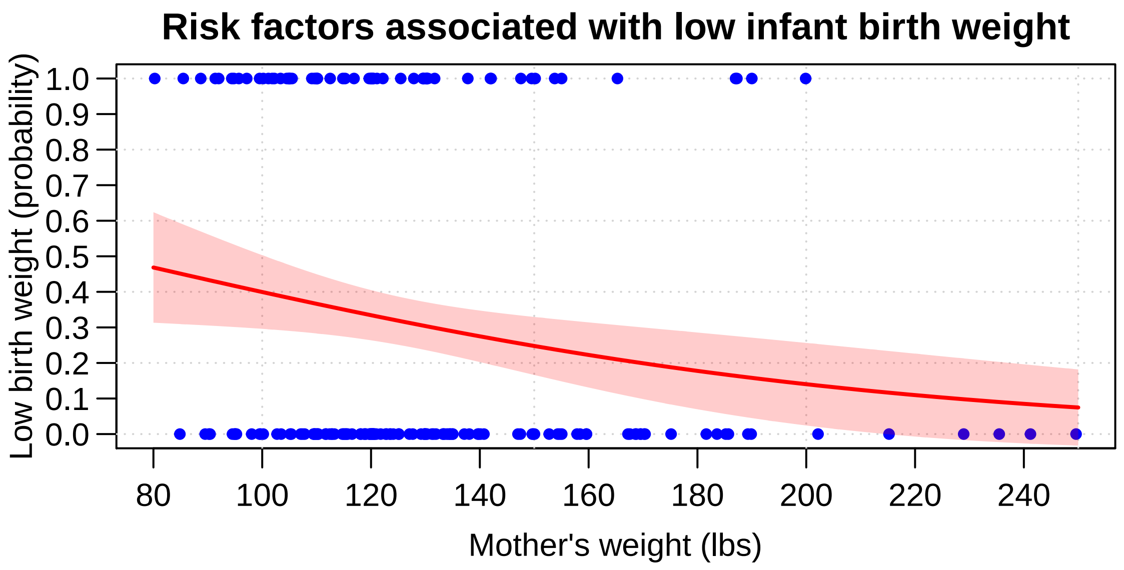
```r
mod <- glm(y ~ x, family = binomial)
xgrid <- seq(min(x), max(x), by = 1)
ypred <- predict(mod, data.frame(x = xgrid), se.fit = TRUE, type = "response")
lines(xgrid, ypred$fit, col = "red", lwd = 2)
```



Uncertainty bands

Add in the uncertainty bands:

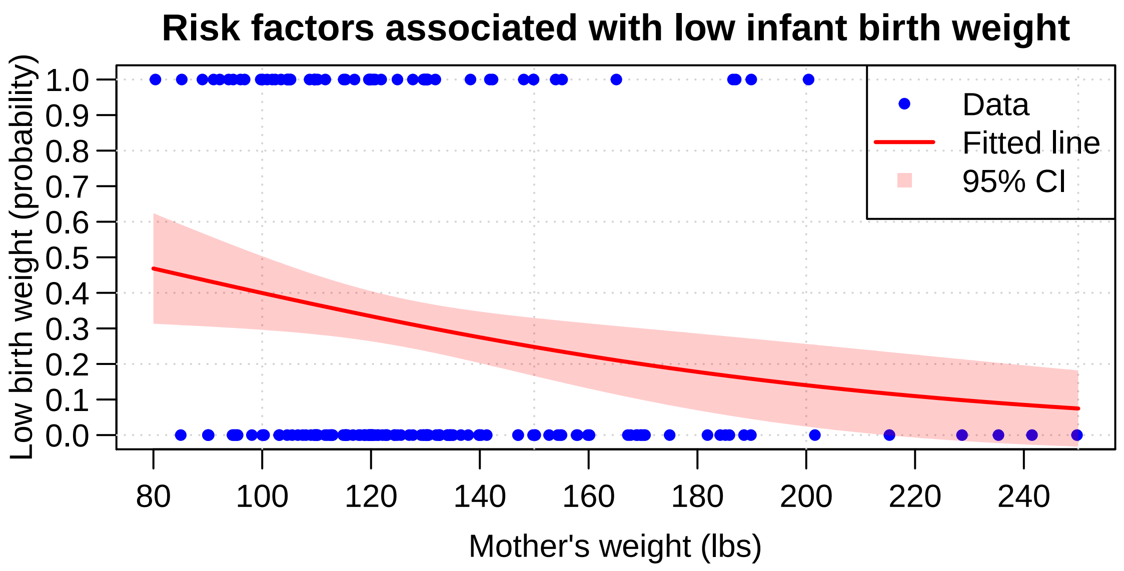
```r
ypred.lower <- ypred$fit - 1.96 * ypred$se.fit
ypred.upper <- ypred$fit + 1.96 * ypred$se.fit
polygon(c(xgrid, rev(xgrid)), c(ypred.lower, rev(ypred.upper)),
  col = newcol, border = NA)
```



Legend

And finally the legend:

```r
legend("topright", legend = c("Data", "Fitted line", "95% CI"),
       pch = c(20, -1, 15), lty = c(-1, 1, -1), lwd = c(-1, 2, -1),
       col = c("blue", "red", newcol))
```



`ggplot2` [not examinable]

- `ggplot2` is an R package created by Hadley Wickham in 2005, based on Leland Wilkinson's *The Grammar of Graphics* book, that allows you to produce professional quality graphs

So, why don't we learn `ggplot2` in this module?

- `ggplot2` uses a different syntax with respect from base plot commands
- Would require an ad hoc module to master it
- Most of the R packages still do their plots using base plot commands
- You will be able to learn `ggplot2` on your own at the end of this module

Useful references:

- `ggplot2` website: http://ggplot2.tidyverse.org/
- Hadley Wickham (2009) **ggplot2: Elegant Graphics for Data Analysis** (2nd ed.). Springer Publishing Company, Incorporated.

Very useful reference for exploratory data analysis with R [do not use this reference for this module!]:

- Hadley Wickham and Garrett Grolemund (2017) **R for Data Science: Import, Tidy, Transform, Visualize, and Model Data** (1st ed.). O'Reilly Media, Inc.
  - Free online version of the book: http://r4ds.had.co.nz/

Lessons from this week

- `plot(x,y)` will give you a basic plot in R, but there are lots of add-ons to create excellent graphics.
- `par` controls many of the defaults. Find a good set of `par` options you like for your plots.
- Separate functions for creation of standard graphics; can also be customised.
- Never ever use pie charts!