11 Basic Visualizations in R

Producing graphics for data analysis is relatively simple; producing graphics for communications/publication is relatively more complex and typically requires a great deal of tweaking to achieve the desired appearance.

Base R provides a number of functions for visualizing data. In this section, we will provide sufficient guidance so that some of the most desired effects can be achieved, but further investigation of the documentation and experimentation will doubtless be necessary for specific needs.[1]

1: We assume that the reader has had some exposure to R and programming (see [1, ch.1]). Advanced visualization functionality is provided by `ggplot2`, see Chapter 12.

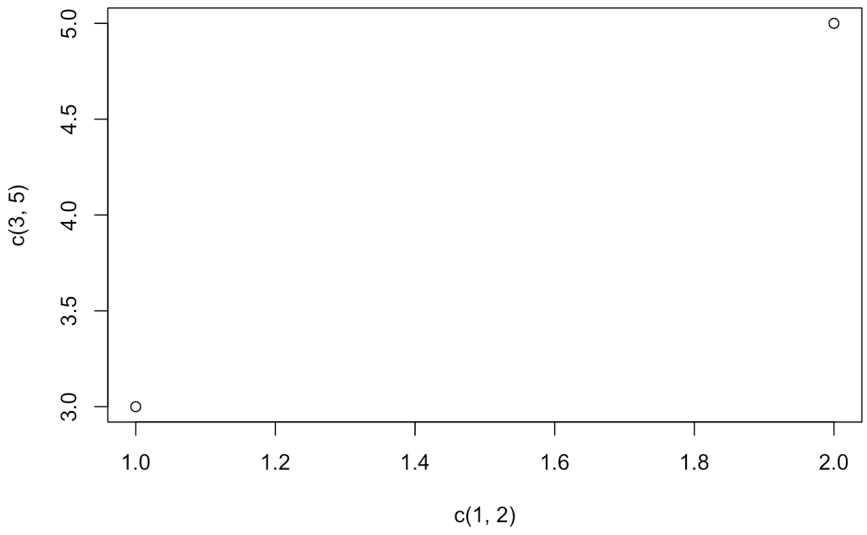
11.1 Scatterplots

The most common plotting function in R is the `plot()` function. It is a generic function – it calls various methods according to the type of the object passed which is passed to it. Generally, we can pass in two vectors and a **scatterplot** of the points is displayed.[2]

2: In the simplest case, we can pass in a vector and we get a scatter plot of magnitude vs index.

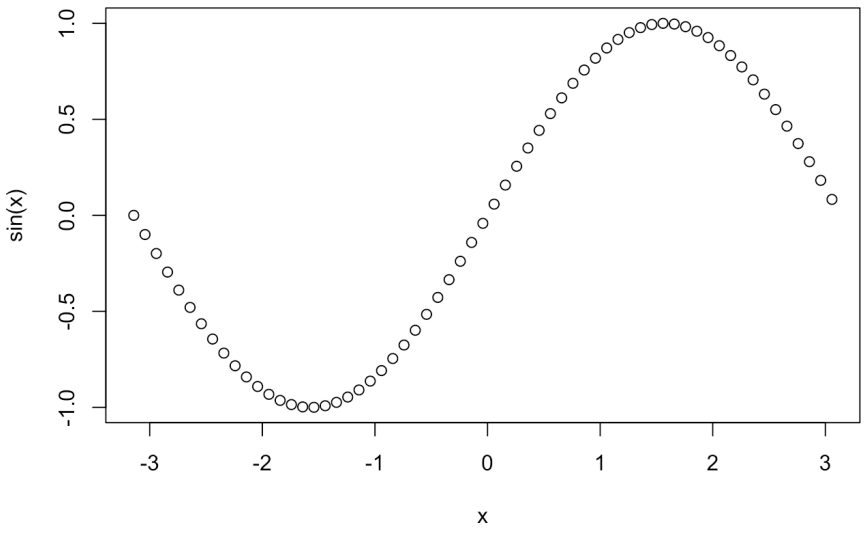
We can display **points** (2D observations) as follows:

```
plot(c(1,2),c(3,5))  # plots the pairs (1,3) and (2,5)
```



Here is a more concrete example showing how to plot the graph of the sine function in the range from $-\pi$ to π.

```
x <- seq(-pi,pi,0.1)
plot(x, sin(x))
```

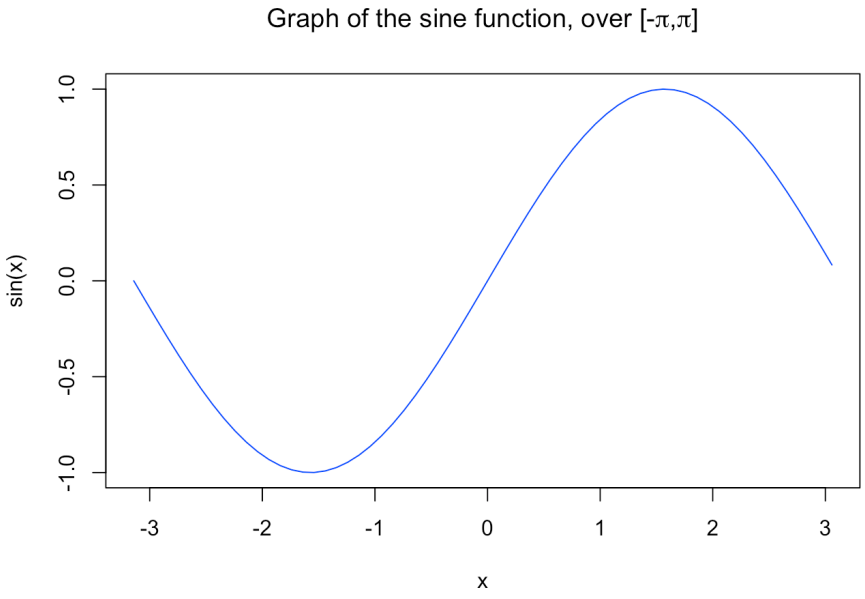


We can add a title to our plot with the parameter `main`. Similarly, `xlab` and `ylab` can be used to label the x-axis and y-axis respectively. The curve is made up of circular black points.[3] We can also specify the colour using the argument `col`.

3: This is the default setting for shape and colour. This can be changed by using the argument `type`. It accepts the following strings (with given effect)

- `p` – points
- `l` – lines
- `b` – both points and lines
- `c` – empty points joined by lines
- `o` – overplotted points and lines
- `s` – stair steps
- `h` – histogram-like vertical lines
- `n` – does not produce points or lines

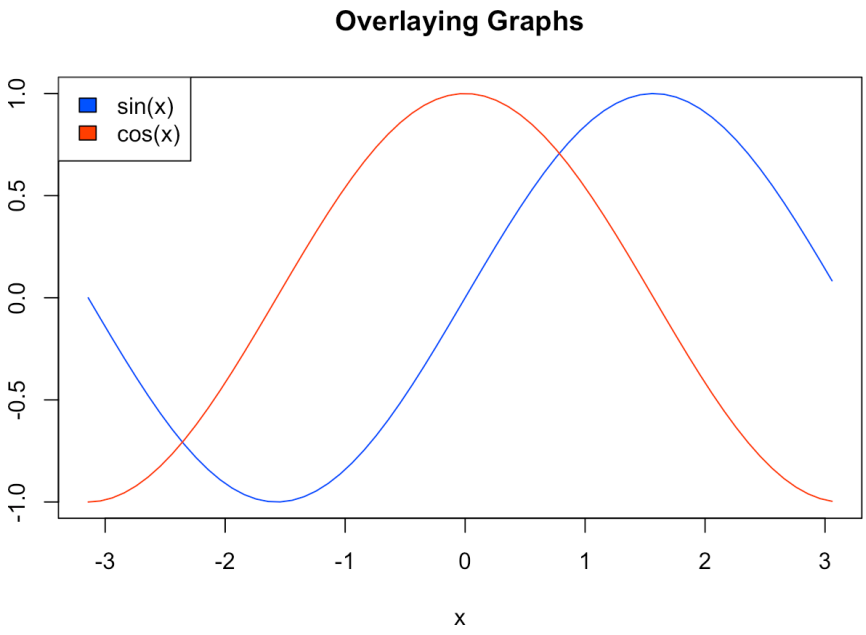
```
plot(x, sin(x), main = expression("Graph of the sine function
                  over [-" * pi * "," * pi * "]"),
 ylab = "sin(x)", type = "l", col = "blue")
```



Calling `plot()` multiple times will have the effect of plotting the current graph on the same window, replacing the previous one.

However, we may sometimes wish to overlay the plots in order to compare the results. This is made possible by the functions `lines()` and `points()`, which add lines and points respectively, to the existing plot.

```
plot(x, sin(x), main="Overlaying Graphs",
     ylab="", type="l", col="blue")
lines(x,cos(x), col="red")
legend("topleft", c("sin(x)","cos(x)"), fill=c("blue","red"))
```



The `legend()` function allows for the appropriate display in the plot.

11.2 Barplots

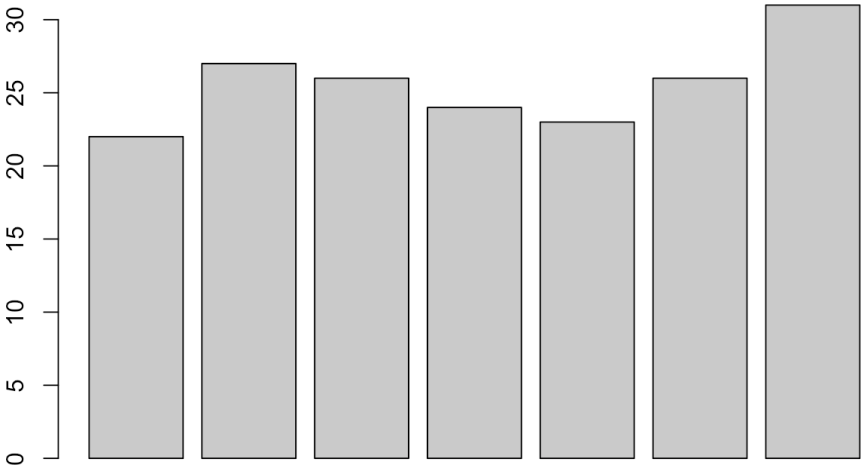
Barplots can be created in `R` using the `barplot()` function. We can supply a vector or matrix to this function, and it will display a bar chart with bar heights equal to the magnitude of the elements in the vector.

Let us suppose that we have a vector of maximum temperatures for seven days, as follows.

```
max.temp <- c(22, 27, 26, 24, 23, 26, 31)
```

We can make a bar chart out of this data using a simple command.

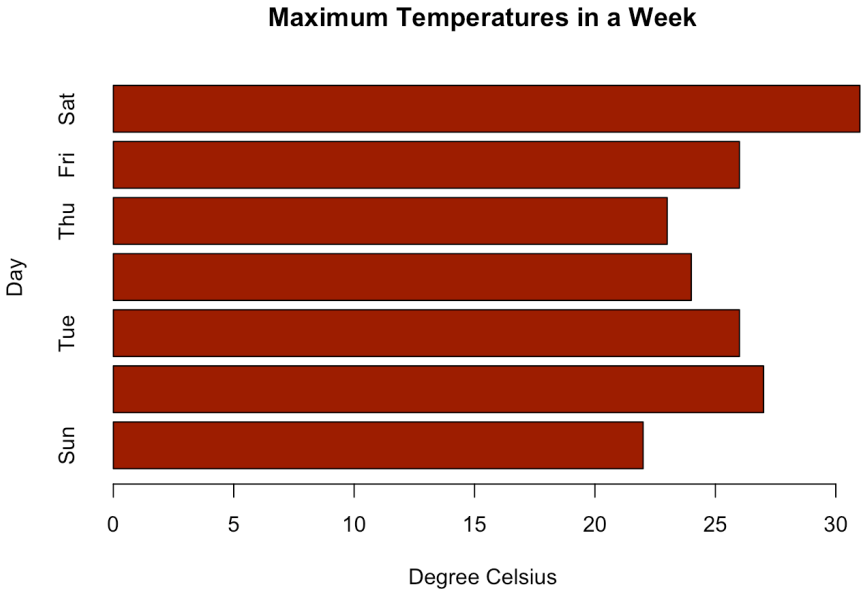
```
barplot(max.temp)
```



This function can take on a number of arguments.[4]

We can **transpose** the plot to have horizontal bars by providing the argument `horiz = TRUE`, such as in the following:

```
barplot(max.temp,
  main = "Maximum Temperatures in a Week",
  xlab = "Degree Celsius", ylab = "Day",
  names.arg = c("Sun","Mon","Tue","Wed","Thu","Fri","Sat"),
  col = "darkred", horiz = TRUE)
```



At times, we may be interested in displaying the count or magnitude for categories in the data. For instance, consider the following vector of age measurements for 21 first-year college students.

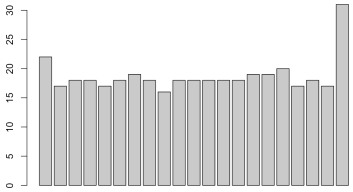
4: Details are available in the help file, which can be queried by entering `?barplot` in the console. Frequently-used arguments include:

- `main` to specify the title
- `xlab` and `ylab` to provide labels for the axes
- `names.arg` to provide a name for each bar
- `col` to define colour, etc.

```
age <- c(22,17,18,18,17,18,19,18,16,18,18,18,18,18,19,19,
  20,17,18,17,31)
```

But calling `barplot(age)` does not display the expected plot.

```
barplot(age)
```



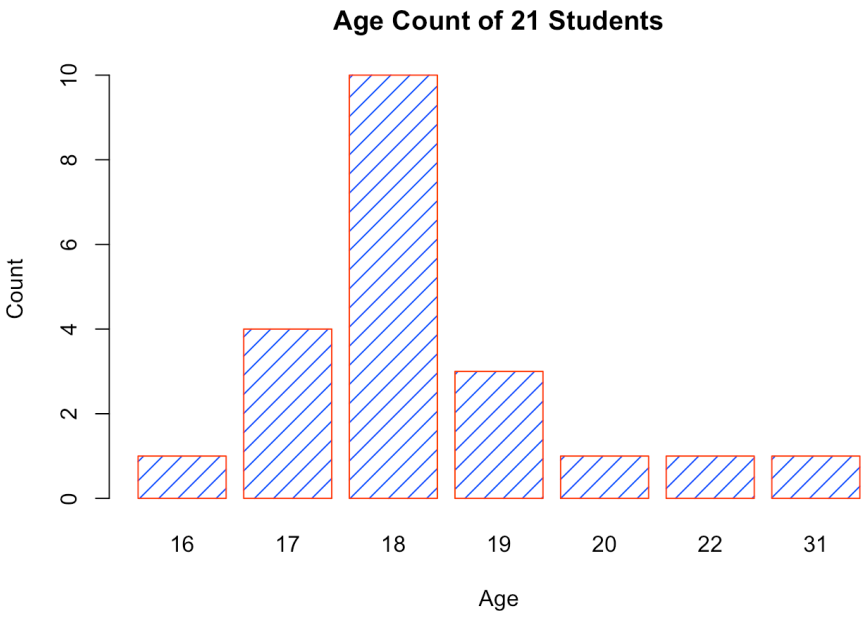
Indeed, the display has 21 bars, with heights corresponding to students' age – the display does not provide the **frequency count** for each age. The required values can be quickly found using the `table()` function.

```
(data = table(age))
```

```
age
16 17 18 19 20 22 31
 1  4 10  3  1  1  1
```

Plotting this data will produce the required barplot (in the code, the argument `density` is used to determine the texture of the fill).

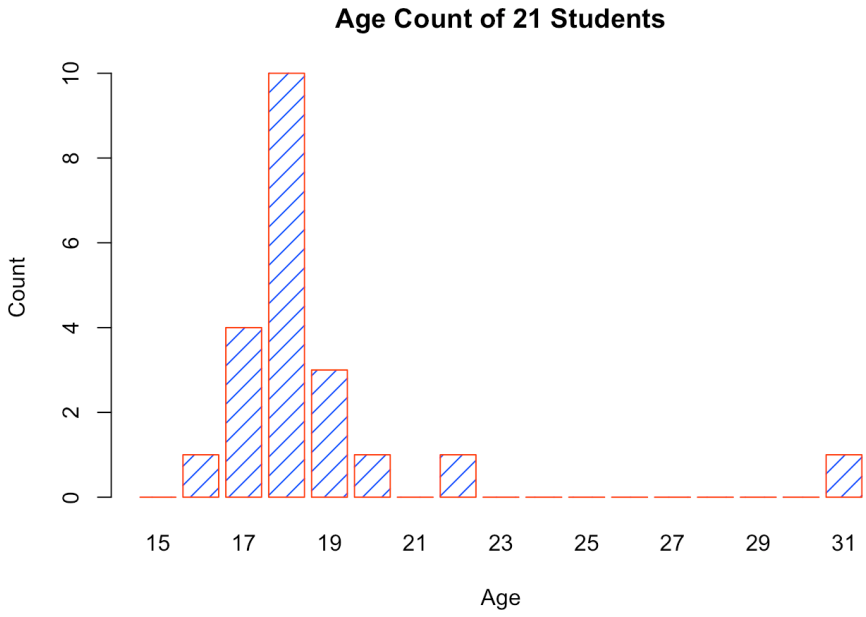
```
barplot(data, main="Age Count of 21 Students",xlab="Age",
  ylab="Count", border="red", col="blue", density=10)
```



The chart respects the relative proportions, but R cannot tell that there should be (empty) categories for `age=21` and between 22 and 31.

This can be remedied by first setting new factor levels for the age measurements, and re-running the code.

```
age=factor(age,levels=c(15:31))
barplot(table(age), main="Age Count of 21 Students",
  xlab="Age", ylab="Count", border="red",
  col="blue", density=10)
```



11.3 Histograms

Histograms display the distribution of a continuous variable by dividing the range of scores into a specified number of bins on the x-axis and displaying the frequency of scores in each bin on the y-axis.

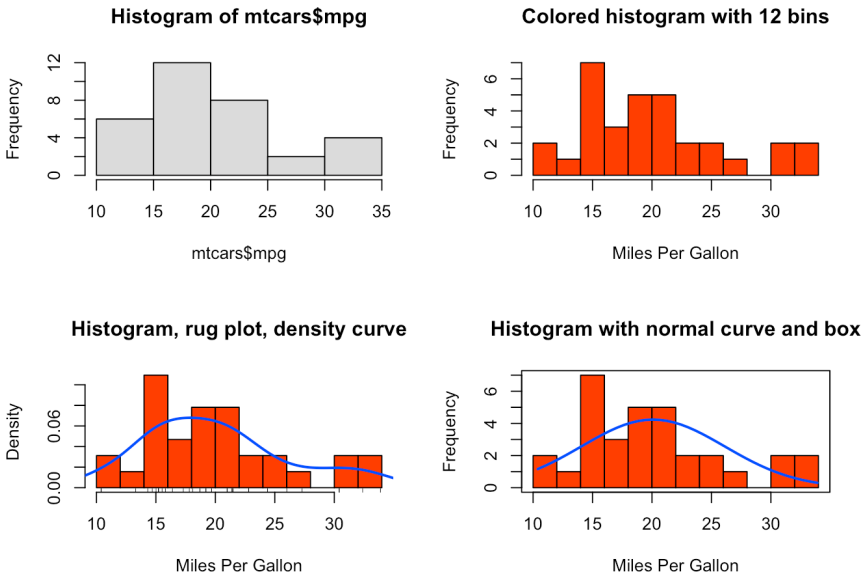
We can create histograms in R with `hist()`:

- the option `freq=FALSE` creates a plot based on **probability densities** rather than frequencies;
- the `breaks` option controls the number of bins: the default produces equally spaced breaks when defining the cells of the histogram.

For illustrative purposes, we will use several of the variables from the *Motor Trend Car Road Tests* (`mtcars`) dataset provided in the base R installation.

Here are four variations on a histogram.

```
par(mfrow=c(2,2)) # to place the charts in the same display
x <- mtcars$mpg

# chart 1 (top left)
hist(x)

# chart 2 (top right)
hist(x, breaks=12, col="red", xlab="Miles Per Gallon",
     main="Colored histogram with 12 bins")

# chart 3 (bottom left)
hist(x, freq=FALSE, breaks=12,  xlab="Miles Per Gallon",
     main="Histogram, rug plot, density curve", col="red")
rug(jitter(x))
lines(density(x), col="blue", lwd=2)

# chart 4 (bottom right)
h <- hist(x, breaks=12, col="red", xlab="Miles Per Gallon",
        main="Histogram with normal curve and box")
xfit <- seq(min(x), max(x), length=40)
yfit <- dnorm(xfit, mean=mean(x), sd=sd(x))
yfit <- yfit*diff(h$mids[1:2])*length(x)
lines(xfit, yfit, col="blue", lwd=2)
box()
```



The top left histogram (chart 1) shows the default plot with **no specified options**: five bins are created, the chart uses the default axis labels and titles. In the top right histogram (chart 2), 12 bins have been specified, as well as a red fill for the bars and **informative** labels and title.

The bottom left histogram (chart 3) uses the same colours, number of bins, labels, and titles as the previous plot but adds a **density curve** and **rug-plot**

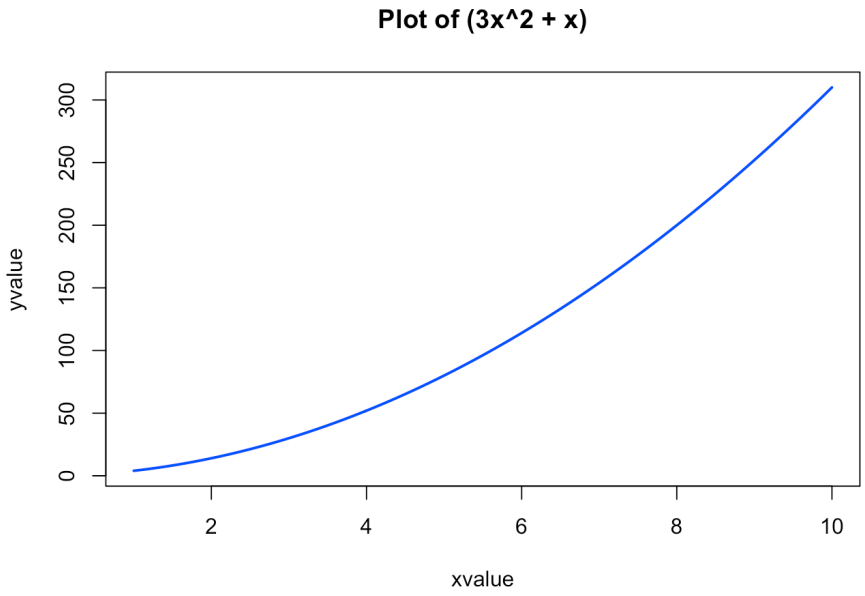
overlay.[5] The bottom right histogram (chart 4) is similar to the second but with a superposed **normal curve** and a box around the figure.[6]

11.4 Curves

Given an expression for a function $y(x)$, we can plot the values of y for various values of x in a given range. This can be accomplished using the R function `curve()`.

For instance, we can plot the simple polynomial function $y = 3x^2 + x$ in the range $x = [1, 10]$ as follows:

```
curve(3*x^2 + x, from=1, to=10, n=300, xlab="xvalue",
      ylab="yvalue", col="blue", lwd=2,
      main="Plot of (3x^2 + x)")
```



The important parameters of the `curve()` function are:

- the **mathematical expression** of the function to plot, written in the format for writing mathematical operations in LaTeX;
- two numeric parameters (`from` and `to`) that represent the **endpoints** of the range of x;
- an integer `n` that represents the number of **equally-spaced values of** x **between** the `from` and `to` points, and
- other parameters (`xlab`, `ylab`, `col`, `lwd`, `main`), with their usual meaning.

5: The density curve is a kernel density estimate and is described in the next section. It provides a smoother description of the distribution of scores. The call to function `lines()` overlays this curve in blue (and a width twice the default line thickness); a rug plot is a one-dimensional representation of the actual data values.

6: The code for superposing the normal curve comes from a suggestion posted to the R-help mailing list by P. Dalgaard.

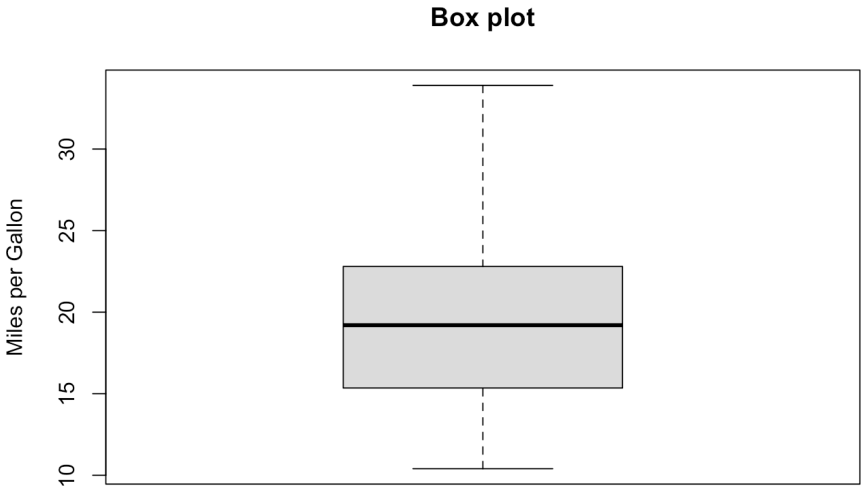
11.5 Boxplots

A box-and-whiskers plot describes the distribution of a continuous variable by plotting its **five-number summary**[7] The **belt** of the boxplot is found at Q_2, its **box boundaries** at Q_1 and Q_3, and its **whiskers** at $\frac{5}{2}Q_1 - \frac{3}{2}Q_3$ and $\frac{5}{2}Q_3 - \frac{3}{2}Q_1$.

7: Namely,

- the **minimum** Q_0;
- the **lower quartile** Q_1 (25th percentile);
- the **median** Q_2 (50th percentile);
- the **upper quartile** Q_3 (75th percentile), and
- the **maximum** Q_4.

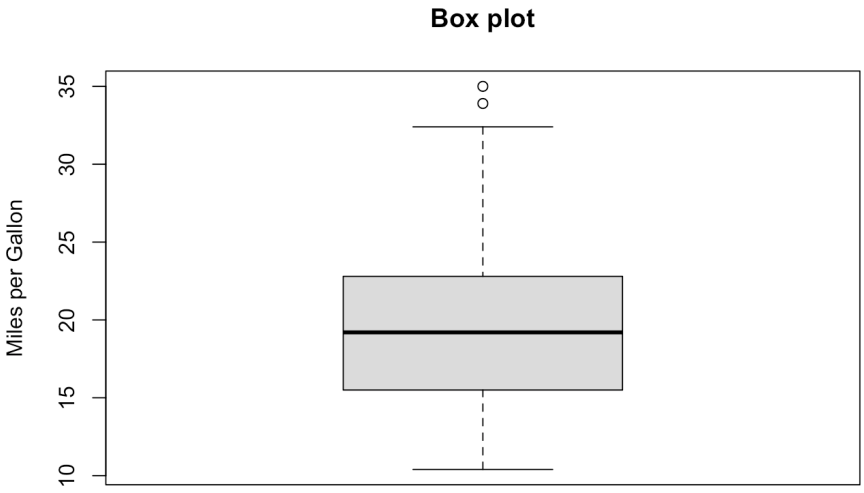
```
boxplot(mtcars$mpg, main="Box plot", ylab="Miles per Gallon")
```



For normally distributed variables, it also displays observations which may be identified as **potential outliers**.[8]

8: Which it to say, values outside the box-and-whiskers range $[\frac{5}{2}Q_1 - \frac{3}{2}Q_3, \frac{5}{2}Q_3 - \frac{3}{2}Q_1]$.

```
boxplot(c(mtcars$mpg,35), # we add an additional value
        main="Box plot", ylab="Miles per Gallon")
```



Not only is the additional value `mpg=35` an outlier, its presence also turns the previous high value of `mpg=34` into an outlier.

11.6 Examples

Let's take a look at some other basic R plots.

Scatterplots On a scatter plot, the features to study depend on the scale of interest:

- for "big" patterns, we look for **form and direction** and **strength**;
- for "small" patterns, we look for **deviations from the pattern** and **outliers**.

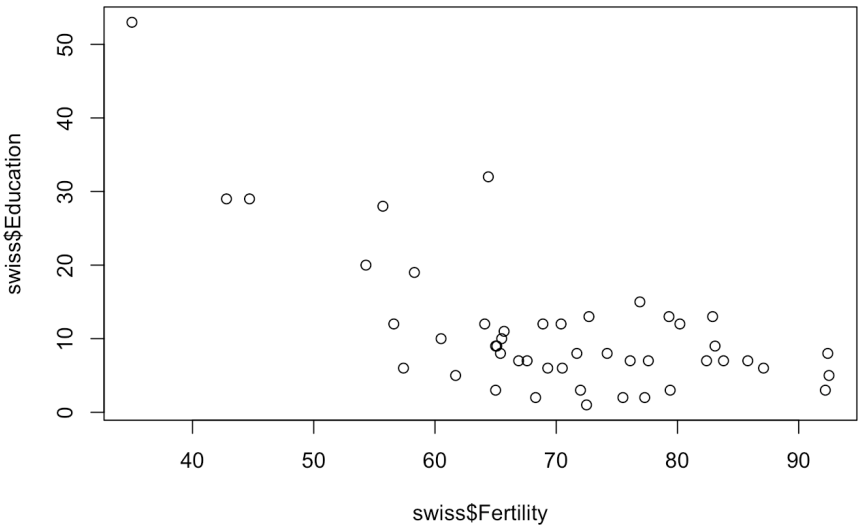
We start with the `swiss` built-in R dataset.

```
str(swiss) # structure of the swiss dataset
```

```
'data.frame':   47 obs. of  6 variables:
 $ Fertility       : num  80.2 83.1 92.5 85.8 ...
 $ Agriculture     : num  17 45.1 39.7 36.5 ...
 $ Examination     : int  15 6 5 12 ...
 $ Education       : int  12 9 5 7 ...
 $ Catholic        : num  9.96 84.84 93.4 33.77 ...
 $ Infant.Mortality: num  22.2 22.2 20.2 20.3 ...
```

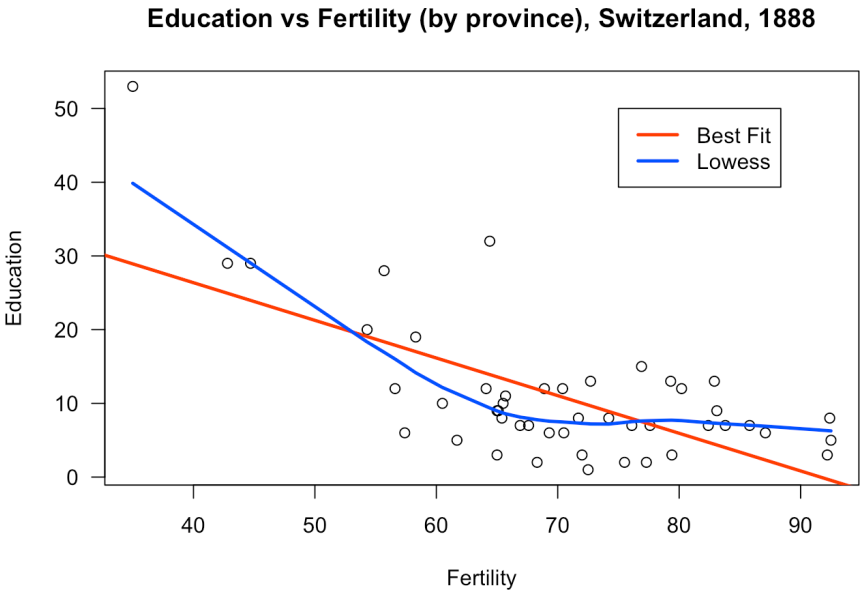
Let's focus on one specific pair: `Fertility` vs. `Education`.

```
# raw plot
plot(swiss$Fertility, swiss$Education)
```



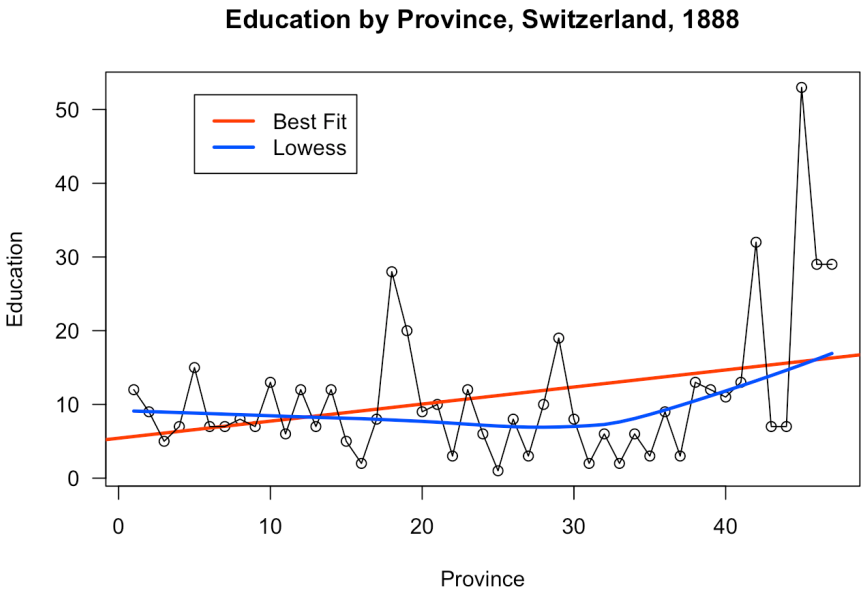
The same plot can be "prettified" and made more informative:

```
# add a title and axis labels
plot(swiss$Fertility, swiss$Education, xlab="Fertility",
     ylab="Education",
     main="Education vs Fertility (by province),
     Switzerland, 1888", las=1)

# add the line of best fit (in red)
abline(lm(swiss$Education~swiss$Fertility), col="red",
       lwd=2.5)

# add the smoothing lowess curve (in blue)
lines(lowess(swiss$Fertility,swiss$Education), col="blue",
      lwd=2.5)

# add a legend
legend(75,50, c("Best Fit","Lowess"), lty=c(1,1),
       lwd=c(2.5,2.5),col=c("red","blue"))
```



Compare that graph with the one produced by:

```
plot(swiss$Education, xlab="Province", ylab="Education",
     main="Education by Province, Switzerland, 1888", las=1)
abline(lm(swiss$Education~row(swiss)[,1]), col="red",lwd=2.5)
lines(swiss$Education)
lines(lowess(row(swiss)[,1],swiss$Education), col="blue",
      lwd=2.5)
legend(5,52, c("Best Fit","Lowess"), lty=c(1,1),
       lwd=c(2.5,2.5), col=c("red","blue"))
```



9: Why is that so, exactly?

Even though R will not balk at producing this graph, it is a nonsense graph.[9] The main take-away of this example is that being able to produce a graph does not guarantee that it will be useful or meaningful in any way.

10: Type in `?iris` at the prompt for information on the dataset.

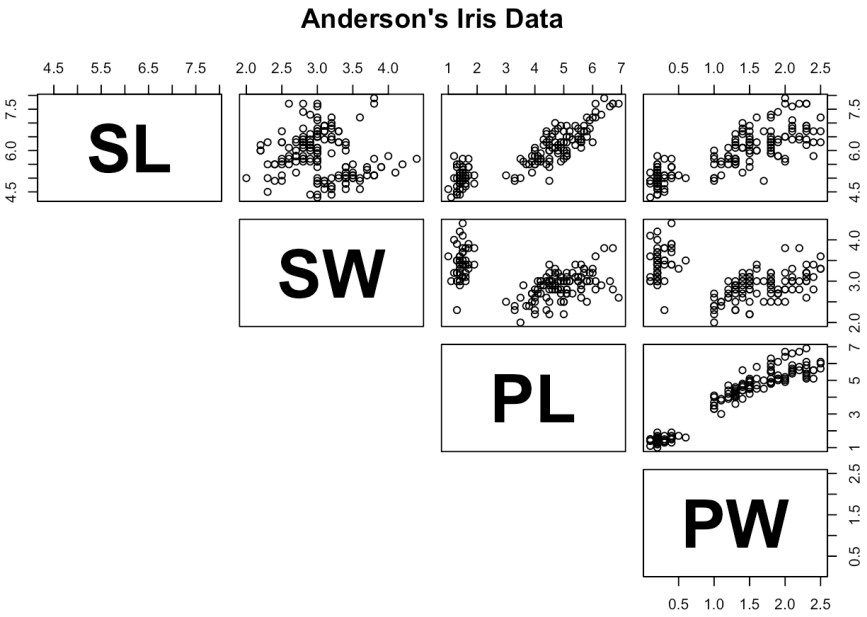
Let's do some thing similar for the infamous built-in `iris` dataset.[10]

```
str(iris) # structure of the dataset
summary(iris) # information on the features
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6  ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 ...

Sepal.Length  Sepal.Width   Petal.Length  Petal.Width   Species
Min.   :4.3   Min.   :2.0   Min.   :1.0   Min.   :0.1   setosa    :50
1st Qu.:5.1   1st Qu.:2.8   1st Qu.:1.6   1st Qu.:0.3   versicolor:50
Median :5.8   Median :3.0   Median :4.3   Median :1.3   virginica :50
Mean   :5.8   Mean   :3.1   Mean   :3.8   Mean   :1.2
3rd Qu.:6.4   3rd Qu.:3.3   3rd Qu.:5.1   3rd Qu.:1.8
Max.   :7.9   Max.   :4.4   Max.   :6.9   Max.   :2.5
```

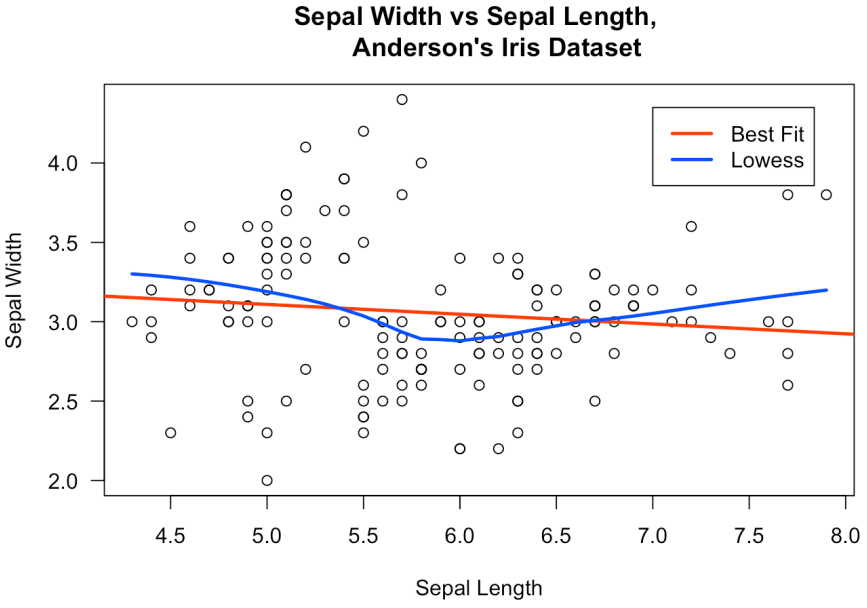
The function `pairs` provides a **scatter plot matrix** of the dataset; the option `lower.panel=NULL` removes the lower panels (due to redundancy).

```
pairs(iris[1:4], main = "Anderson's Iris Data", pch = 21,
      lower.panel=NULL, labels=c("SL","SW","PL","PW"),
      font.labels=2, cex.labels=4.5)
```



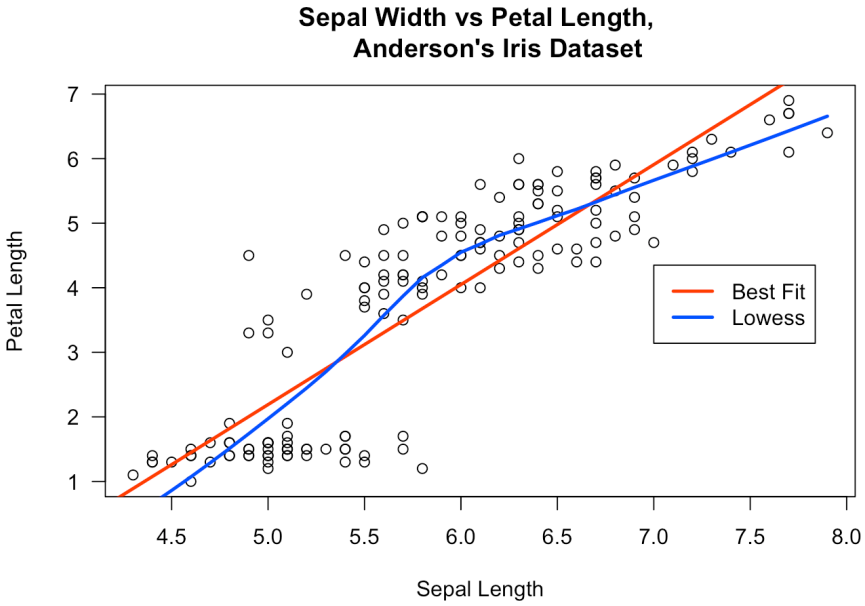
We can compare the sepal width and length variables in a manner similar to what we did with the `swiss` dataset.

```
plot(iris$Sepal.Length, iris$Sepal.Width, xlab="Sepal Length",
     ylab="Sepal Width", main="Sepal Width vs Sepal Length,
     Anderson's Iris Dataset", las=1,
     bg=c("yellow","black","green")[unclass(iris$Species)])
abline(lm(iris$Sepal.Width~iris$Sepal.Length), col="red")
lines(lowess(iris$Sepal.Length,iris$Sepal.Width), col="blue")
legend(7,4.35, c("Best Fit","Lowess"), lty=c(1,1),
       lwd=c(2.5,2.5),col=c("red","blue"))
```



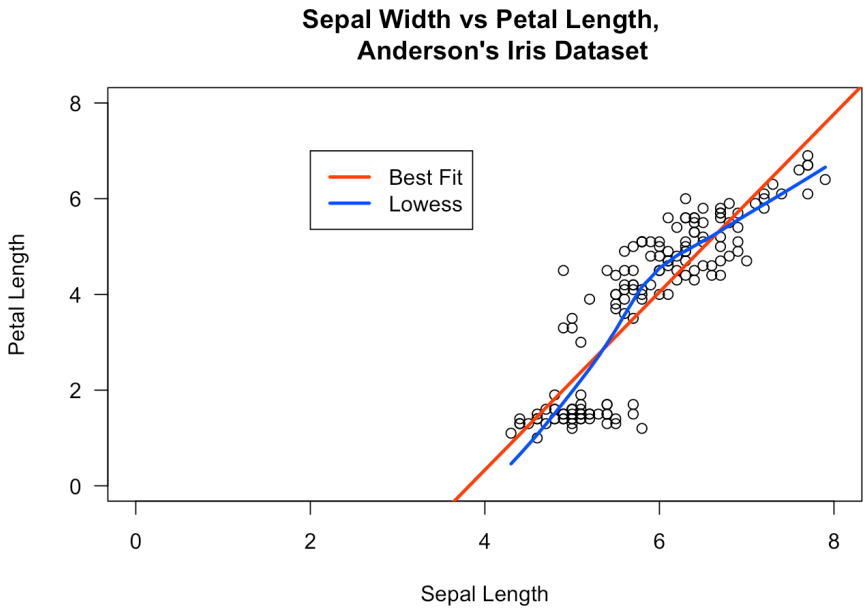
There does not seem to be a very strong relationship between these variables. What about sepal length and petal length?

```
plot(iris$Sepal.Length, iris$Petal.Length, xlab="Sepal Length",
     ylab="Petal Length", main="Sepal Width vs Petal Length,
     Anderson's Iris Dataset", las=1)
abline(lm(iris$Petal.Length~iris$Sepal.Length), col="red")
lines(lowess(iris$Sepal.Length,iris$Petal.Length), col="blue")
legend(7,4.35, c("Best Fit","Lowess"), lty=c(1,1),
       lwd=c(2.5,2.5),col=c("red","blue"))
```



Visually, the relationship is striking: the line seems to have a slope of 1! But notice that the axes are unevenly scaled, and have been cutoff away from the origin. The following graph gives a better idea of the situation.

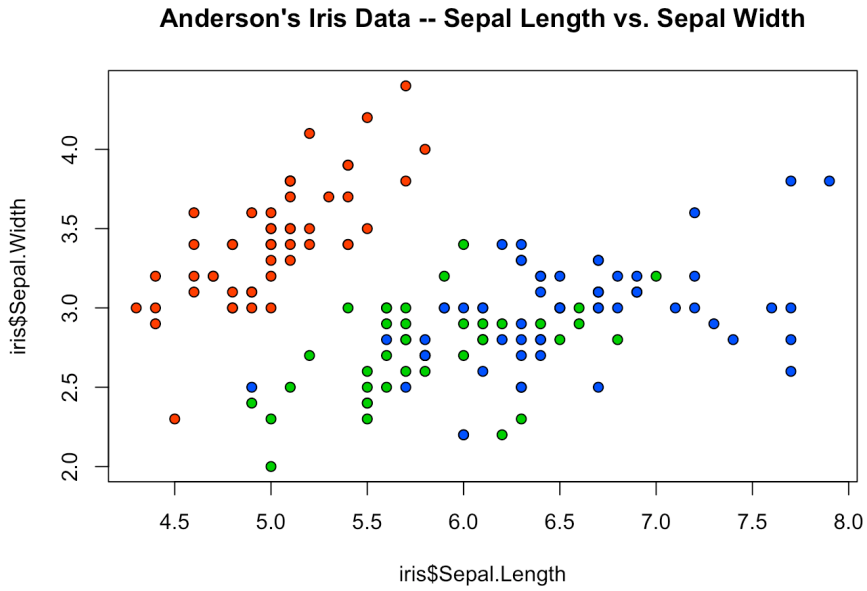
```
plot(iris$Sepal.Length, iris$Petal.Length, xlab="Sepal Length",
     ylab="Petal Length", main="Sepal Width vs Petal Length,
     Anderson's Iris Dataset", xlim=c(0,8), ylim=c(0,8),
     las=1)
abline(lm(iris$Petal.Length~iris$Sepal.Length), col="red",
       lwd=2.5)
lines(lowess(iris$Sepal.Length,iris$Petal.Length), col="blue",
      lwd=2.5)
legend(2,7, c("Best Fit","Lowess"), lty=c(1,1),
       lwd=c(2.5,2.5),col=c("red","blue"))
```



A relationship is still present, but it is **affine**, not linear as could have been guessed by naively looking at the original graph.

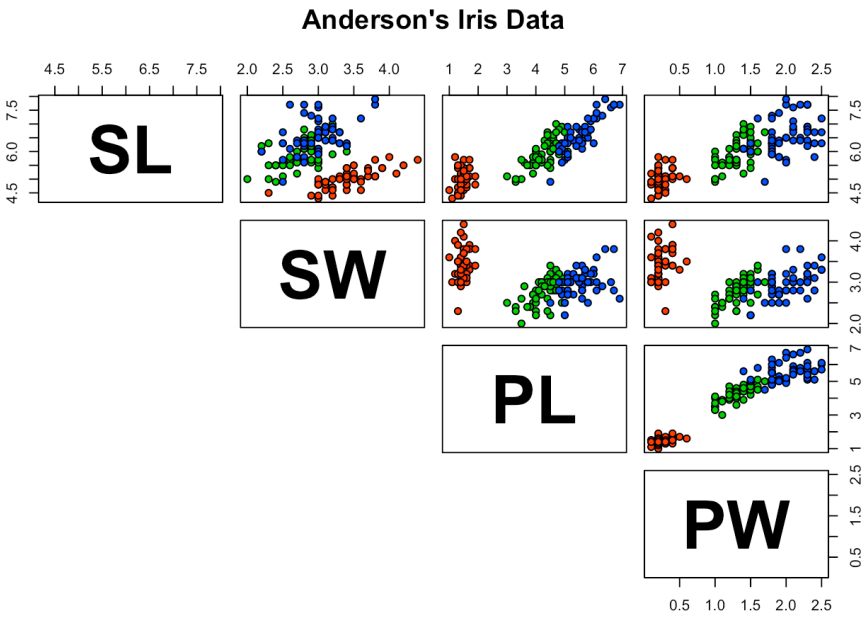
Colour can also be used to highlight various data elements. Here, we colour each observation differently according to its species.

```
plot(iris$Sepal.Length, iris$Sepal.Width, pch=21,
     bg=c("red","green3","blue")[unclass(iris$Species)],
     main="Anderson's Iris Data -- Sepal Length vs.
     Sepal Width", xlim=c(4,8))
```



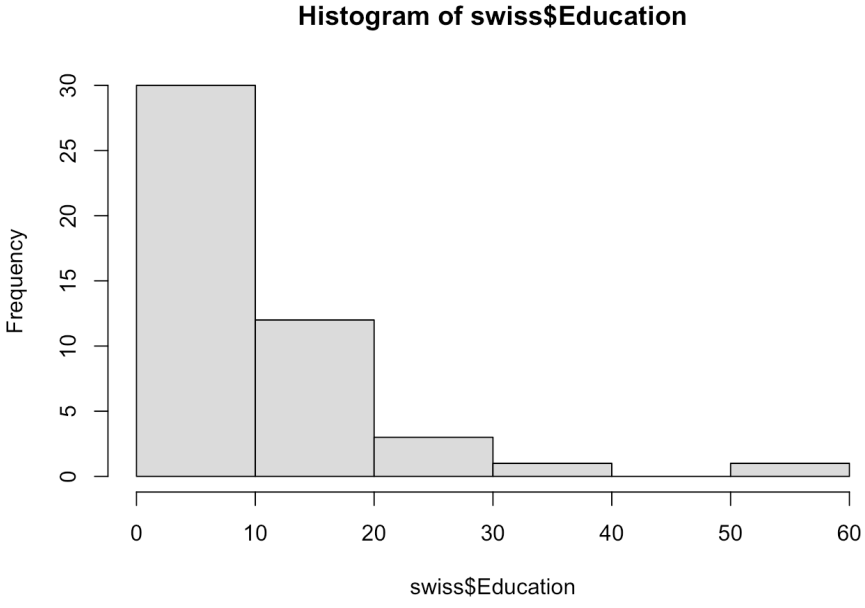
This can be done on all scatterplots concurrently using `pairs`.

```
pairs(iris[1:4], main = "Anderson's Iris Data", pch = 21,
      bg = c("red", "green3", "blue")[unclass(iris$Species)],
      lower.panel=NULL, labels=c("SL","SW","PL","PW"),
      font.labels=2, cex.labels=4.5)
```

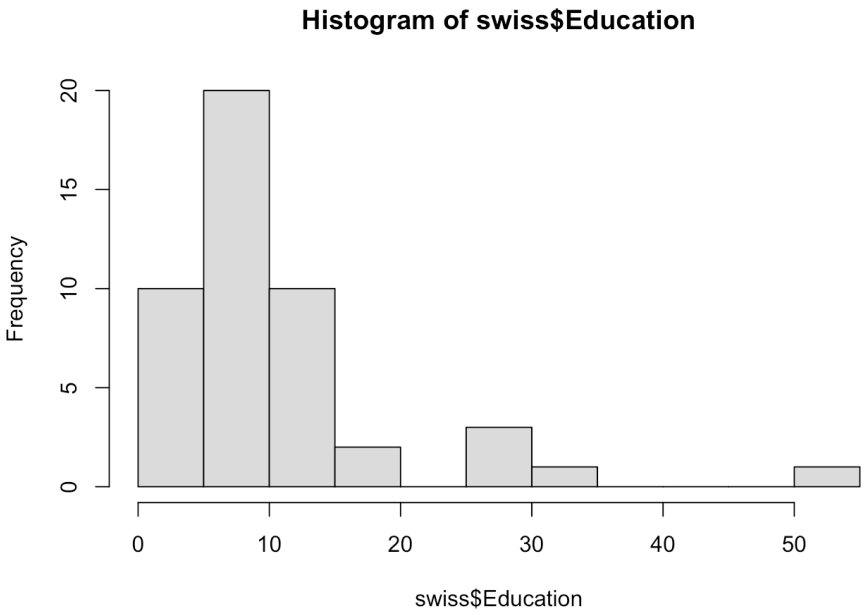


Histograms and Bar Charts In histograms or frequency charts, users should stay on the lookout for **bin size effects**. For instance, what does the distribution of the `Education` variable in the `swiss` dataset look like?

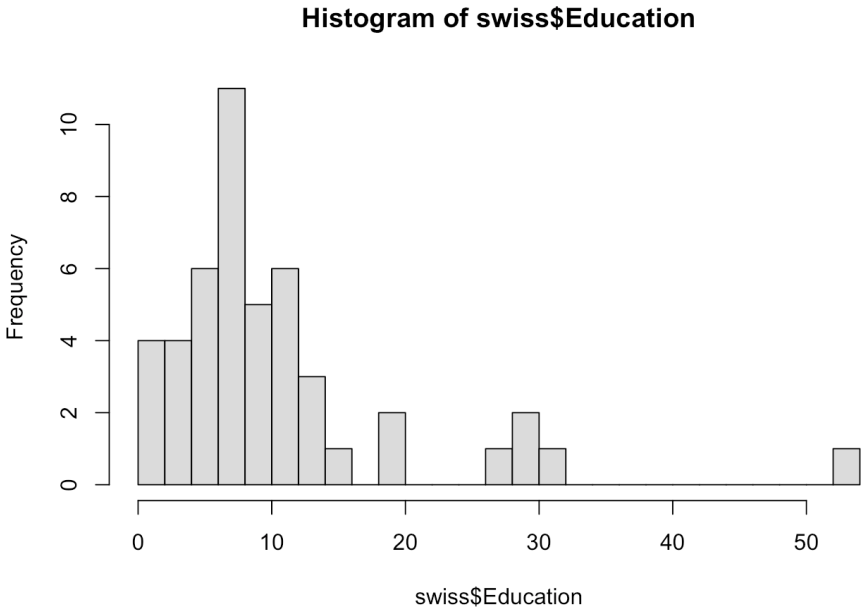
```
hist(swiss$Education)   # default number of bins
```



```
hist(swiss$Education, breaks=10)   # with 10 bins
```



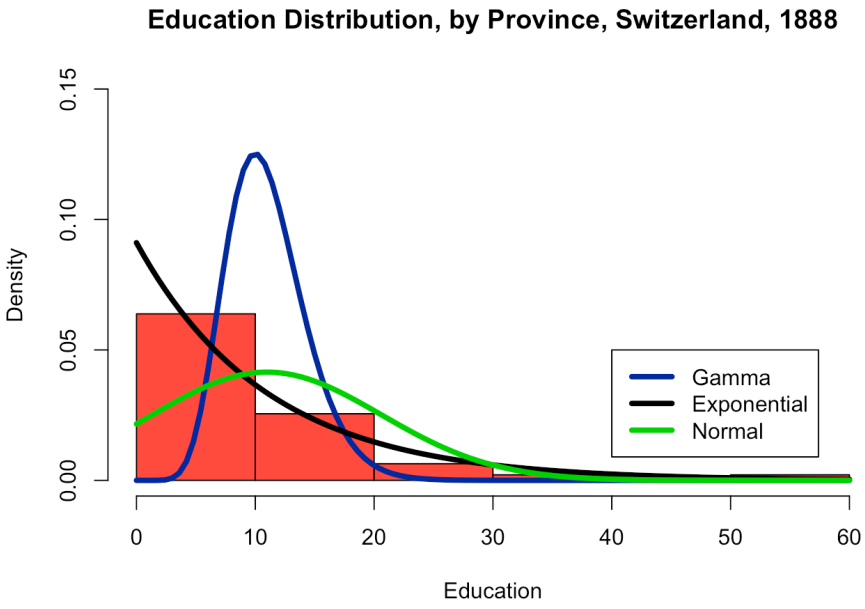

```
hist(swiss$Education, breaks=20)   # with 20 bins
```



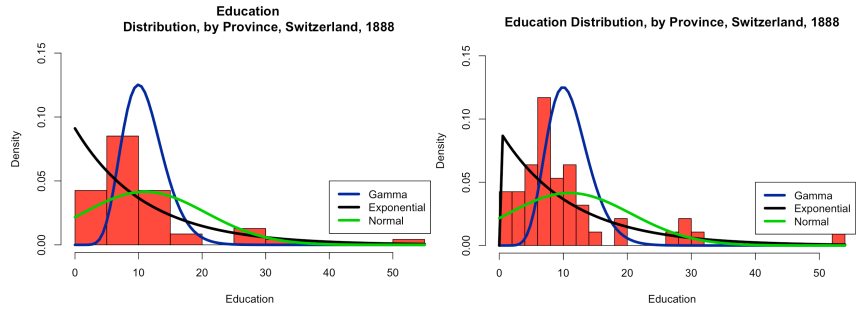
The distribution pattern is distintcly different with 10 and 20 bins. Don't get too carried away with the number of bins, however: too many, and we may end up masking trends if the dataset is not large enough.

We can also look for best fits for various parametric distributions, with the default number of bins.

```
hist(swiss$Education, freq=FALSE, xlab="Education",
     main="Education Distribution, by Province, Switzerland,
     1888", col="firebrick1", ylim=c(0,0.15))
curve(dgamma(x,shape=mean(swiss$Education)),add=TRUE,
      col="darkblue", lwd=4) # fit a gamma dist
curve(dexp(x,rate=1/mean(swiss$Education)),add=TRUE,
      col="black", lwd=4) # fit an exponential dist
curve(dnorm(x,mean=mean(swiss$Education),sd=sd(swiss$Education)),
      add=TRUE, col="green3", lwd=4) # fit a normal dist
legend(40,0.05, c("Gamma","Exponential","Normal"), lty=c(1,1),
       lwd=c(4,4),col=c("darkblue","black", "green3"))
```



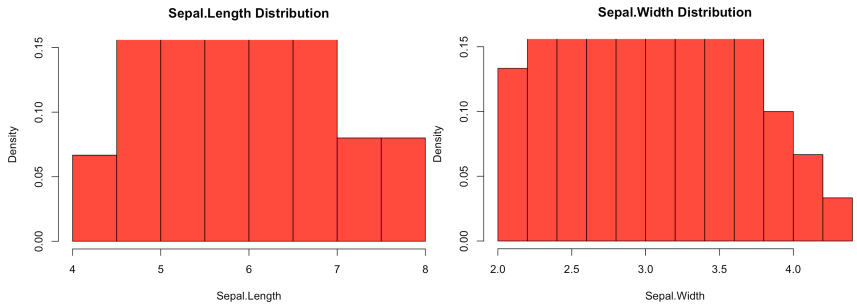
We can also view the histograms with 10 or bins by replacing the value of `breaks` in the code above.



With a small number of bins, the exponential distribution seems like a good fit, visually. With a larger number of bins, neither of the three families seems particularly well-advised.

And now, can you figure out what is happening with these visualizations of the `iris` dataset?

```
hist(iris$Sepal.Length, freq=FALSE, xlab="Sepal.Length",
     main="Sepal.Length Distribution", col="firebrick1",
     ylim=c(0,0.15))
hist(iris$Sepal.Width, freq=FALSE, xlab="Sepal.Width",
     main="Sepal.Width Distribution", col="firebrick1",
     ylim=c(0,0.15))
```

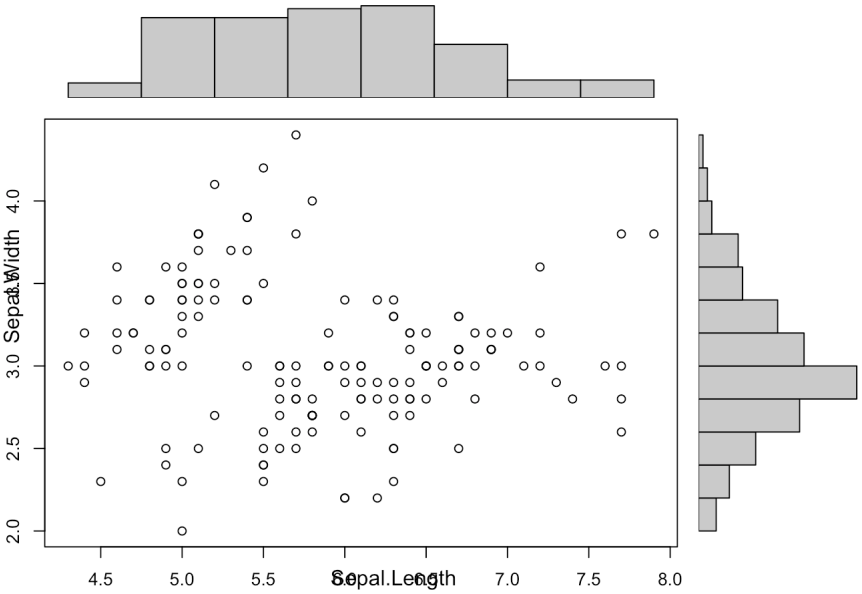


What happens above if we replace `freq=FALSE` with `freq=TRUE`?[11]

11: Try it!

Histograms (1D data representations) can be combined with scatterplots (2D data representations) to provide **marginal** information (this is done through a custom function based on `R-blogger.com`'s `scatterhist()` function).

```
scatterhist = function(x, y, xlab="", ylab=""){
  zones=matrix(c(2,0,1,3), ncol=2, byrow=TRUE)
  layout(zones, widths=c(4/5,1/5), heights=c(1/5,4/5))
  xhist = hist(x, plot=FALSE)
  yhist = hist(y, plot=FALSE)
  top = max(c(xhist$counts, yhist$counts))
  par(mar=c(3,3,1,1))
  plot(x,y)
  par(mar=c(0,3,1,1))
  barplot(xhist$counts, axes=FALSE, ylim=c(0, top), space=0)
  par(mar=c(3,0,1,1))
  barplot(yhist$counts, axes=FALSE, xlim=c(0, top), space=0,
       horiz=TRUE)
  par(oma=c(3,3,0,0))
  mtext(xlab, side=1, line=1, outer=TRUE, adj=0,
    at=.8 * (mean(x) - min(x))/(max(x)-min(x)))
  mtext(ylab, side=2, line=1, outer=TRUE, adj=0,
    at=(.8 * (mean(y) - min(y))/(max(y) - min(y))))
}

ds = iris
with(ds, scatterhist(iris$Sepal.Length, iris$Sepal.Width,
            xlab="Sepal.Length", ylab="Sepal.Width"))
```



Bubble Charts Bubble charts are a neat way to show at least 3 variables on the same 2D display. The **location** of the bubbles' centre takes care of 2 variables: the **size**, **colour**, and **shape** of the bubbles can also be used to represent different data elements.

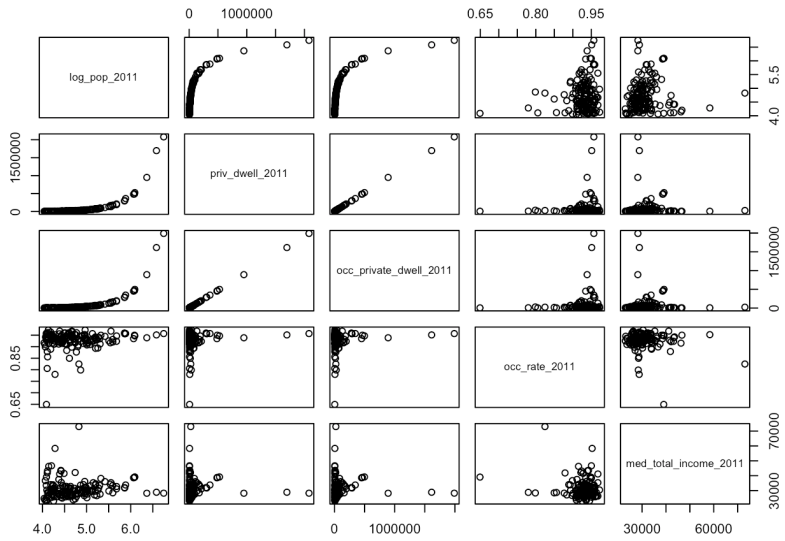
For this first example, we'll take a look at some Statistics Canada 2011 demographic data regarding Canada's census metropolitan areas (CMA) and census agglomerations (CA).

```
can.2011=read.csv("Canada2011.csv", head=TRUE)
str(can.2011)
```

```
'data.frame':   147 obs. of  12 variables:
 $ Geographic.code       : int  1 5 10 15 105 110 205 ...
 $ Geographic.name       : chr  "St. John's" "Bay Roberts" ...
 $ Province              : chr  "NL" "NL" "NL" "NL" ...
 $ Region                : chr  "Atlantic" "Atlantic" ...
 $ Type                  : chr  "CMA" "CA" "CA" "CA" ...
 $ pop_2011              : int  196966 10871 13725 27202 ...
 $ log_pop_2011          : num  5.29 4.04 4.14 4.43 4.81 ...
 $ pop_rank_2011         : int  20 147 128 94 52 120 13 97 ...
 $ priv_dwell_2011       : int  84542 4601 6134 11697 ...
 $ occ_private_dwell_2011: int  78960 4218 5723 11110 26192 ...
 $ occ_rate_2011         : num  0.934 0.917 0.933 0.95 0.907 ...
 $ med_total_income_2011 : int  33420 24700 26920 27430 ...
```

Before jumping aboard the bubble chart train, let's see what the dataset looks like in the scatterplot framework for 5 of the variables.

```
pairs(can.2011[,c(7,9,10,11,12)])
```



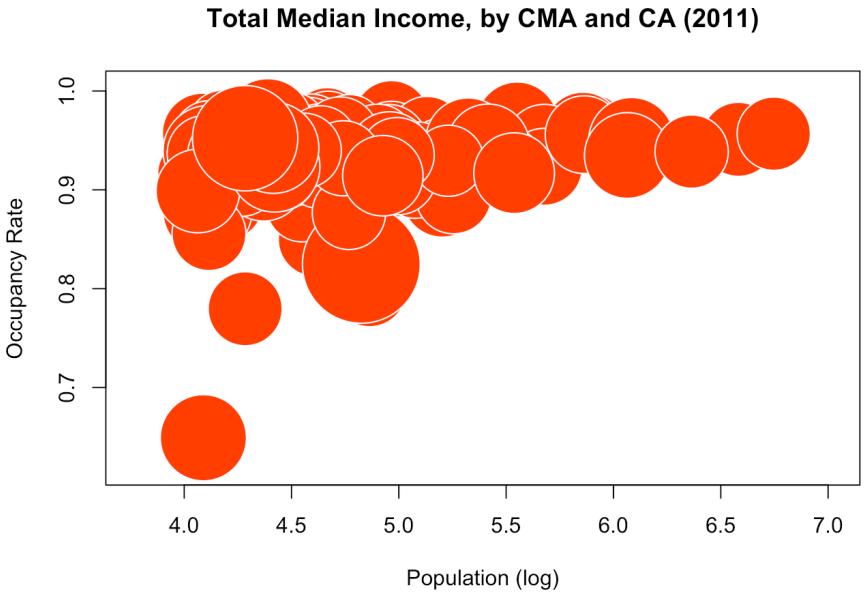
It's ... underwhelming, if that's a word.[12] There are some interesting tidbits, but nothing jumps as being meaningful beyond a first pass.

Can anything else be derived using bubble charts?[13] We use median income as the radius for the bubbles, and focus on occupancy rates and population.

12: We know it's not 'cause we looked it up. It's one of the skills we learned in grade school.

13: Implemented in the `R` function `symbol`.

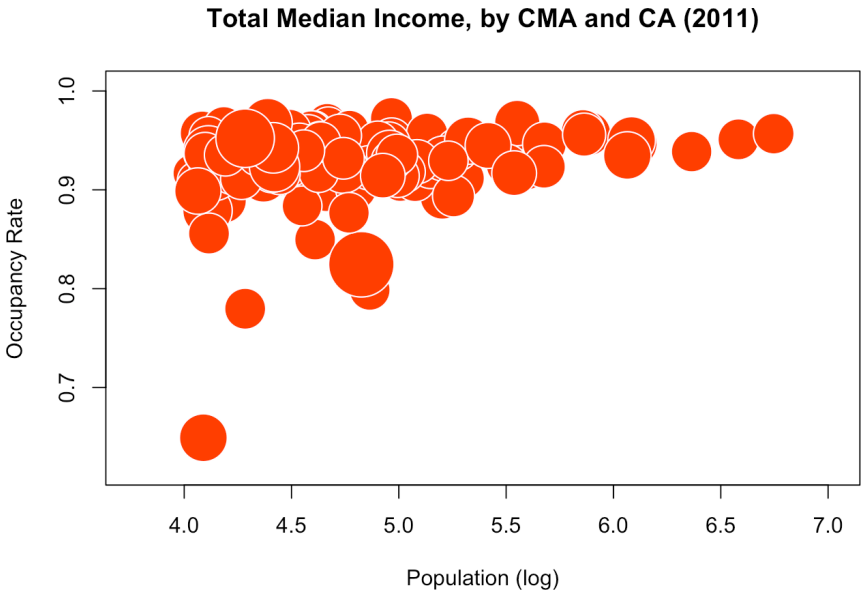
```
radius.med.income.2011<-sqrt(can.2011$med_total_income_2011/pi)
symbols(can.2011$log_pop_2011, can.2011$occ_rate_2011,
        circles=radius.med.income.2011, inches=0.45,
        fg="white", bg="red", xlab="Population (log)",
        ylab="Occupancy Rate")
title("Total Median Income, by CMA and CA (2011)")
```



Clearly, an increase in population seems to be associated with (and not necessarily a cause of) a rise in occupancy rates. But the median income seems to have very little correlation with either of the other two variables. Perhaps such a correlation is hidden by the default unit used to draw the bubbles?

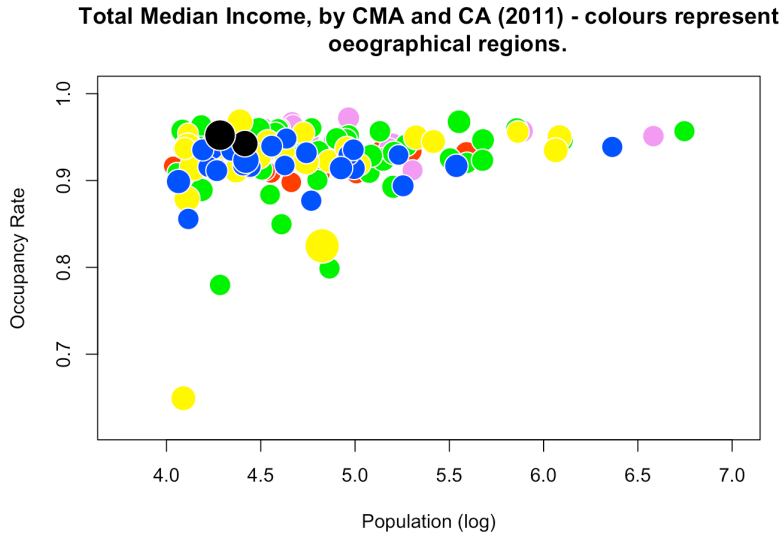
Let's shrink it from 0.45 to 0.25 and see if anything pops out.

```
symbols(can.2011$log_pop_2011, can.2011$occ_rate_2011,
        circles=radius.med.income.2011,
        inches=0.25, fg="white", bg="red",
        xlab="Population (log)", ylab="Occupancy Rate")
title("Total Median Income, by CMA and CA (2011)")
```



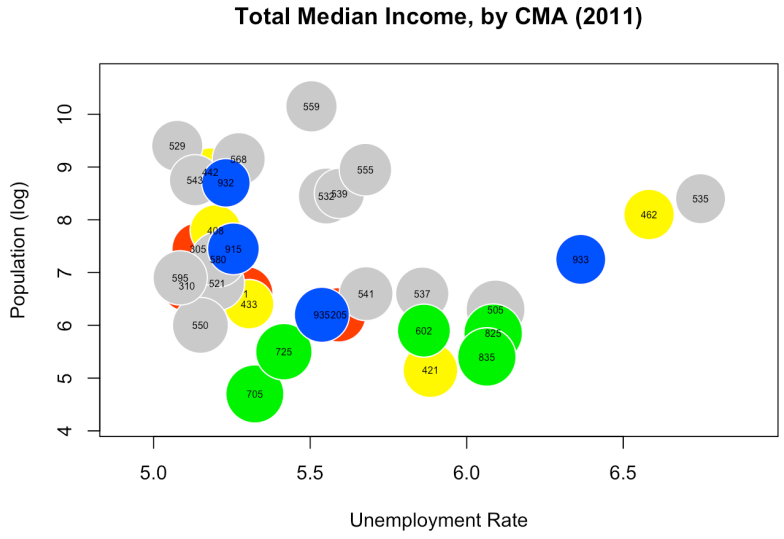
Not much to see. But surely there would be a relationship in these quantities if we included the CMA/CA's region?

```
symbols(can.2011$log_pop_2011, can.2011$occ_rate_2011,
        circles=radius.med.income.2011,
        inches=0.15, fg="white",
        bg=c("red","blue","black","green",
           "yellow","violet")[factor(can.2011$Region)],
        xlab="Population (log)", ylab="Occupancy Rate")
title("Total Median Income, by CMA and CA (2011) -
        colours represent goeographical regions.")
```



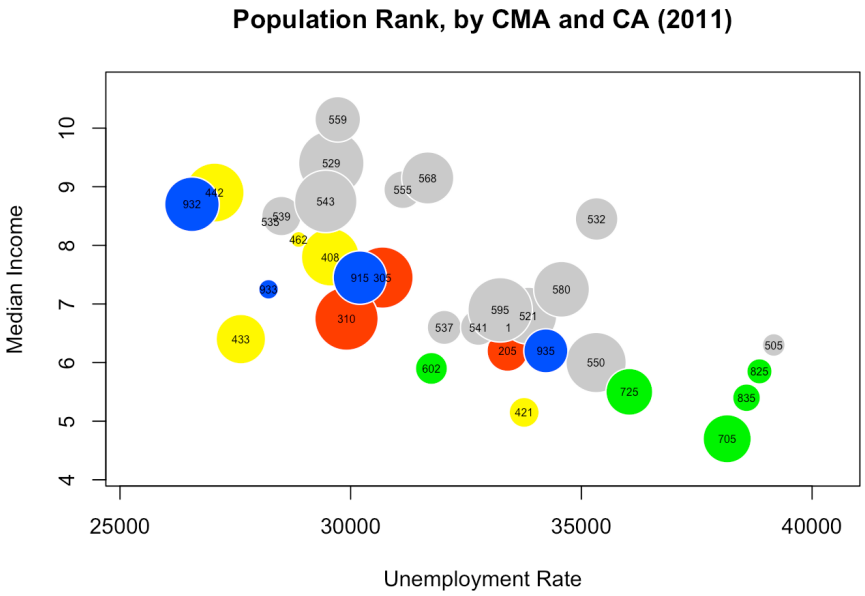
What do you think? Perhaps the CA distort the full picture (given that they are small and more numerous). Let's analyze the subset of CMAs instead, for population and unemployment.

```
can.2011.CMA = read.csv("Canada2011_CMA.csv", head=TRUE)
radius.med.income.2011.CMA <-
        sqrt(can.2011.CMA$med_total_income_2011/pi)
symbols(can.2011.CMA$log_pop_2011,
        can.2011.CMA$med_unemployment_2011, fg="white",
        circles=radius.med.income.2011.CMA, inches=0.25,
        bg=c("red","blue","gray","green",
           "yellow")[factor(can.2011.CMA$Region)],
        ylab="Population (log)", xlab="Unemployment Rate")
title("Total Median Income, by CMA (2011)")
text(can.2011.CMA$log_pop_2011, can.2011.CMA$Geographic.code,
     can.2011.CMA$med_unemployment_2011, cex=0.5)
```



Part of the problem is that median income seems to be roughly uniform among CMAs. What if we used rank statistics instead? Or switch the radius to population rank, say?

```
radius.pop.rank.2011.CMA<-sqrt(can.2011.CMA$pop_rank_2011/pi)
symbols(can.2011.CMA$med_total_income_2011,
        can.2011.CMA$med_unemployment_2011,
        circles=radius.pop.rank.2011.CMA, inches=0.25,
        fg="white", ylab="Median Income",
        xlab="Unemployment Rate",
        bg=c("red","blue","gray","green",
           "yellow")[factor(can.2011.CMA$Region)])
title("Population Rank, by CMA and CA (2011)")
text(can.2011.CMA$med_total_income_2011,
     can.2011.CMA$Geographic.code,
     can.2011.CMA$med_unemployment_2011, cex=0.5)
```



There's a bit more structure in there, isn't there? Can you figure out to what regions the colours correspond?[14]

14: **Hint:** look at the first digit of the bubble labels, and the frequency of the colours.

So we can make data charts fairly easily with R, but let's be honest – they are not exactly the best looking charts anyone has ever seen.[15]

15: We can actually use the base R graphing capabilities, but it does require a fair amount of work to produce something decent.

`ggplot2` is the R implementation of Wilkinson and Wickham's *Layered Grammar of Graphics* (see Section 5.4 and [54, 55]); it extends R's graphing functionality to "easily" produce compelling (and pretty) data pictures.