Chapter 7: Data and the plug-in principle

S520

These notes are written to accompany Trosset chapter 7.

Reading data into R

Data as a vector

You can enter data manually using the `c()` function.

```
die = c(1, 2, 3, 4, 5, 6)
```

However, with more than a few data points, you'll want to scan data in from other files you have lying around.

Let's say you want to read in the file `nerve.txt` that's posted on Canvas. The file just consists of one set of numbers – waiting times (in seconds) between successive pulses along a nerve fiber. We'd like to read this into R as one object. To do this, first save the file on to your own computer, somewhere where you can easily find it. For example, I keeping all the data files I use for teaching in a folder unimaginatively called "Teaching". I have a copy of `nerve.txt` in that folder. To read in the data as a vector called `nerve`, I firstly make sure that my working directory is `Teaching`. You can set your working directory in RStudio by going to the Session menu and selecting "Set Working Directory" (easiest), or by using the `setwd()` command. Then I run:

```
nerve = scan("nerve.txt")
head(nerve)
```

```
## [1] 0.21 0.03 0.05 0.11 0.59 0.06
```

Note that you need plain quotes, not smart quotes. The `head()` function lists the first few elements of the data set. We check that these are what we expected.

If you ever get lost, the `getwd()` function will tell your current working directory.

```
getwd()
```

```
## [1] "/Users/jianyu/Box Sync/SU17 Online S520/Lecture Notes"
```

Finally, the `file.choose()` command can be used instead of specifying a filename. The line `nerve = scan(file.choose())` brings up a GUI window that lets you select the file location. However, this is bad practice because once you have more than a few data files, it's very easy to forget where they are, and if you have several version of the same file, it's impractical to keep track of which is which.

Data frames

Most real data doesn't just contain one variable – it may contain a few, or hundreds. If we have several variables measured for the same individuals, we can read in the data as a **data frame**. The file `singer.txt` contains the heights of 235 singers in the New York Choral Society. After downloading the file to your working directory, run:

```
singer = read.table("singer.txt", header=TRUE)
head(singer)
```

```
##   height voice.part
## 1     64  Soprano 1
## 2     62  Soprano 1
## 3     66  Soprano 1
## 4     65  Soprano 1
## 5     60  Soprano 1
## 6     61  Soprano 1
```

The data frame `singer` contains two variables: a numeric variable, `height`, and a non-numeric “factor” variable, `voice.part`. Suppose we only deal with the heights of the singers. We can isolate this variable:

```
singer$height
```

```
##   [1] 64 62 66 65 60 61 65 66 65 63 67 65 62 65 68 65 63 65 62 65 66 62 65
##  [24] 63 65 66 65 62 65 66 65 61 65 66 65 62 63 67 60 67 66 62 65 62 61 62
##  [47] 66 60 65 65 61 64 68 64 63 62 64 62 64 65 60 65 70 63 67 66 65 62 68
##  [70] 67 67 63 67 66 63 72 62 61 66 64 60 61 66 66 66 62 70 65 64 63 65 69
##  [93] 61 66 65 61 63 64 67 66 68 70 65 65 65 64 66 64 70 63 70 64 63 67 65
## [116] 63 66 66 64 64 70 70 66 66 66 69 67 65 69 72 71 66 76 74 71 66 68 67
## [139] 70 65 72 70 68 64 73 66 68 67 64 68 73 69 71 69 76 71 69 71 66 69 71
## [162] 71 71 69 70 69 68 70 68 69 72 70 72 69 73 71 72 68 68 71 66 68 71 73
## [185] 73 70 68 70 75 68 71 70 74 70 75 75 69 72 71 70 71 68 70 75 72 66 72
## [208] 70 69 72 75 67 75 74 72 72 74 72 72 74 70 66 68 75 68 70 72 67 70 70
## [231] 69 72 71 74 75
```

If we only wanted the heights of the first ten singers in the data set:

```
singer$height[1:10]
```

```
##  [1] 64 62 66 65 60 61 65 66 65 63
```

Finally, instead of a file location on our computer, we can also load in data directly from the Internet, if it’s in a friendly form. One friendly data set is the catalog of 2014 earthquakes from the Southern California Earthquake Center:

```
earthquake.data = read.table("https://service.scedc.caltech.edu/ftp/catalogs/SCEC_DC/2014.catalog")
head(earthquake.data)
```

```
##           V1          V2 V3 V4   V5 V6     V7       V8   V9 V10      V11
## 1 2014/01/01 00:53:59.13 eq  l 0.10  l 33.462 -116.473 12.6   A 11408434
## 2 2014/01/01 01:04:28.33 eq  l 0.58  l 33.388 -116.390 11.6   A 11408442
## 3 2014/01/01 02:00:42.00 eq  l 2.40  l 33.225 -116.081  9.1   A 11408450
## 4 2014/01/01 03:09:23.82 eq  l 1.42  l 34.301 -118.576  8.2   A 11408458
## 5 2014/01/01 03:28:37.26 eq  l 0.39  l 33.485 -116.493 11.3   A 11408466
## 6 2014/01/01 03:35:53.43 eq  l 1.01  l 34.206 -117.588 12.2   A 11408474
##   V12 V13
## 1  24   0
## 2  25   0
## 3 129   0
## 4  38   0
## 5  33   0
## 6  38   0
```

Looking at the documentation, the magnitude of the earthquake is given in the fifth column. Let’s pull this out and give it a name. To select a column of a data frame:

```
magnitude = earthquake.data[,5]
```

How many earthquakes are there in the catalog? Here are a couple of ways to find out:

```
length(magnitude)
```

```
## [1] 14429
```

```r
nrow(earthquake.data)
```

```
## [1] 14429
```

If you have data stored in some proprietary format (e.g. Excel, SPSS, SAS) then you'll either need to convert to an easy-to-read format first (e.g. save an Excel spreadsheet as a .csv file) or install a package that lets R read data from those formats (at the time of writing, the `rio` package seems to be the one that works most generally.) To install a package that you haven't previously downloaded, make sure you're connected to the internet and then use the `install.packages()` function:

```r
install.packages("rio")
```

Then to actually use the package you downloaded, run the `library()` function:

```r
library(rio)
```

For just about any package you care about, there's a ton of information on how to use it floating around the web. Google is your friend.

The plug-in principle

Trosset 7.1

Can we connect data to all that probability we learned, or was the first month of the course just a waste of time? Fortunately, there's an extremely simple solution: Assign a probability of $1/n$ to each observation in the data set, where n is the number of observations in the data set. This results in a discrete distribution called the **empirical distribution**.

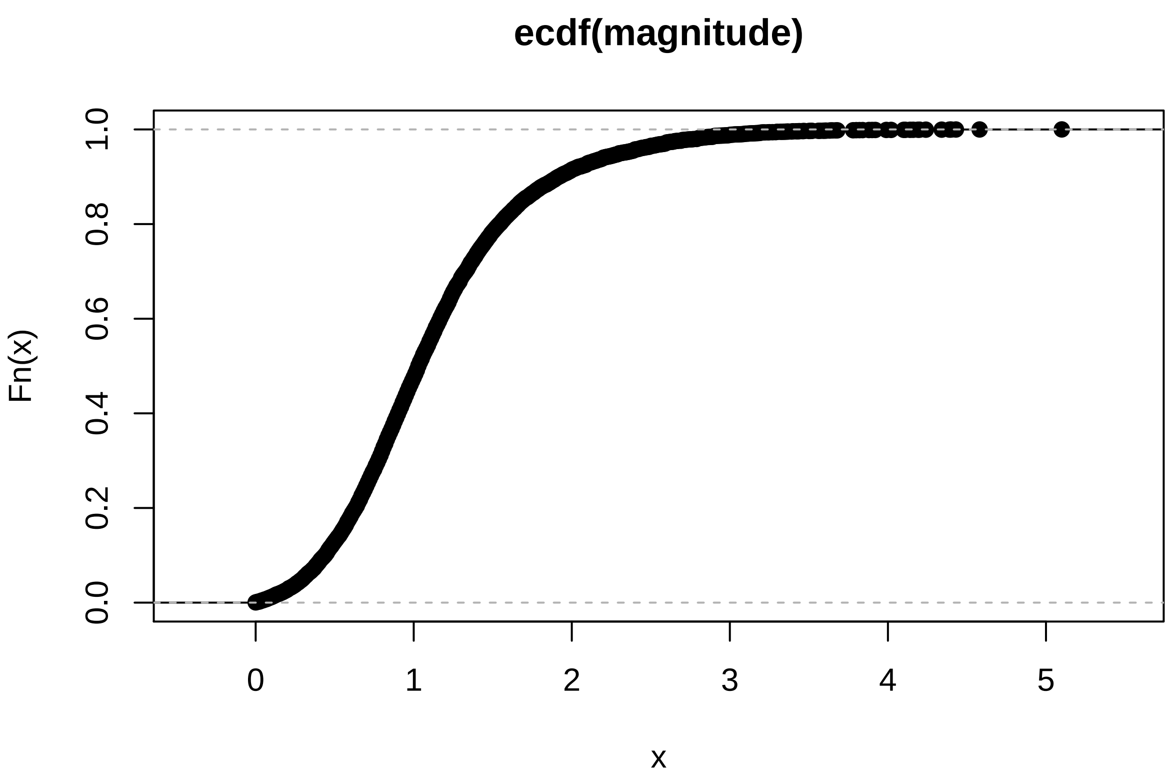
Just like any other probability distribution, the empirical distribution can be describe by its CDF. Let's say we want to find the CDF of the earthquake magnitudes at $y = 1$. Since each earthquake has a probability of $1/n$ attached to it, the CDF at 1 is the number of earthquakes that have a magnitude of 1 or less, multiplied by $1/n$.

```r
sum(magnitude <= 1) / length(magnitude)
```

```
## [1] 0.4760552
```

Well, we don't just want to know the CDF of the empirical distribution at magnitude 1, we want to know the CDF of the empirical distribution for *all* magnitudes. It's easiest to study the **empirical CDF**, or **ECDF**, as a graph:

```r
plot(ecdf(magnitude))
```



The empirical distribution is discrete, so it doesn't have a PDF. So to estimate the PDF, we need more than the plug-in principle. We'll come back to this later.

A quick aside on graphs

If you're using R Markdown, graphs are easy, but then you have to learn R Markdown. Otherwise, you have a couple of options. In RStudio, you can just go to the graph window, click "Export", and save the displayed graph as a .png or .pdf file. For reproducibility reasons, it's better to use code to save your graph, as follows:

```
plot(ecdf(magnitude))
dev.print(pdf, "myGreatGraph.pdf")
```

This saves the graph as a PDF file called `myGreatGraph.pdf` in your working directory. You can then insert it into whatever documents you wish.

Plug-in mean and variance

Trosset 7.2

The **plug-in mean**, also called the **sample mean**, is the expected value of the empirical distribution. Since the empirical distribution is discrete and all data points have probability $1/n$, this just means we add up the numbers and divide by n. So it's the same as the mean you learned in grade school.

```
n = length(magnitude)
sum(magnitude) / n
```

```
## [1] 1.127678
```

```
mean(magnitude)
```

```
## [1] 1.127678
```

Similarly, the **plug-in variance** is the variance of the empirical distribution. Remember our shortcut for finding variance. Find the expected value of X^2 (this would be `mean(x^2)`), find the square of the expected value (this would be `mean(x)^2`), then take the difference.

```
mean(magnitude^2) - mean(magnitude)^2
```

```
## [1] 0.3766893
```

Note that the `var()` function does *not* give you the plug-in variance – it gives you something slightly different:

```
var(magnitude)
```

```
## [1] 0.3767154
```

The fifth significant figure of variance really isn't a big deal, so we ignore this issue for now. (But if this nags at you, skip ahead to pp. 201-202 of Trosset for the spoiler.)

If you estimated it, then you shoulda put a hat on it

The usual way to notate something that's estimated from data is to put a hat on it. For the plug-in mean of the magnitudes, we write

$$\hat{\mu}_n = 1.128$$

For the plug-in variance, we write

$$\hat{\sigma^2_n} = 0.38$$

The subscript n is a good reminder that we are dealig with something estimated from a finite amount of data. However I often omit it because I am lazy.

Plug-in quantiles

Trosset 7.3

We previously defined quantiles for continuous random variables, but the empirical distribution is discrete. We can amend our definition:

> Let X be a random variable and let $0 < \alpha < 1$. If $P(X < q) \leq \alpha$ and $P(X > q) \leq 1 - \alpha$, then q is an α-quantile of X.

Let's focus on the 0.5-quantile. Suppose our data is $1, 2, 3, 4, 5$. The plug-in principle says to assign a probability of 1/5 to each of these. Is 3 the plug-in median? By the empirical distribution, $P(X < 3) = 0.4$ and $P(X > 3) = 0.4$, so yes, it is.

What if our data is $1, 2, 3, 4, 5, 6$? Now note that all real numbers between 3 and 4 (inclusive) satisfy the definition. So technically, any $3 \leq q \leq 4$ is a 0.5-quantile. Well, we want a unique value, so we'll declare the plug-in median to be the midpoint of all the 0.5-quantiles – that's 3.5. Does R agree?

```
x1 = c(1, 2, 3, 4, 5)
quantile(x1, 0.5)
```

```
## 50%
##   3
```

```
median(x1)
```

```
## [1] 3
```

```r
x2 = c(1, 2, 3, 4, 5, 6)
quantile(x2, 0.5)
```

```
## 50%
## 3.5
```

```r
median(x2)
```

```
## [1] 3.5
```

R agrees. Pretty much every statistician agrees.

Unfortunately, statisticians stop agreeing when it comes to quartiles. For example, for the data set $(1, 2, 3, 4, 5, 6)$, our plug-in method gives 2 as the first quartile and 5 as the third quartile. What does R say?

```r
quantile(x2, c(0.25, 0.75))
```

```
##  25%  75%
## 2.25 4.75
```

To get R to agree with our plug-in estimates, we need to tell R to use "type 2" quantiles:

```r
quantile(x2, c(0.25, 0.75), type=2)
```

```
## 25% 75%
##   2   5
```

Whatever. The type of quantile you use is profoundly unimportant. If using different methods for finding sample quantiles gives you vastly different results, you should take a bigger sample. ("Take a bigger sample" will be a mantra for this course.)

The five-number summary and the boxplot

We can get a good idea about what a set of data looks like by finding:

- The minimum value
- The first quartile
- The median
- The third quartile
- The maximum

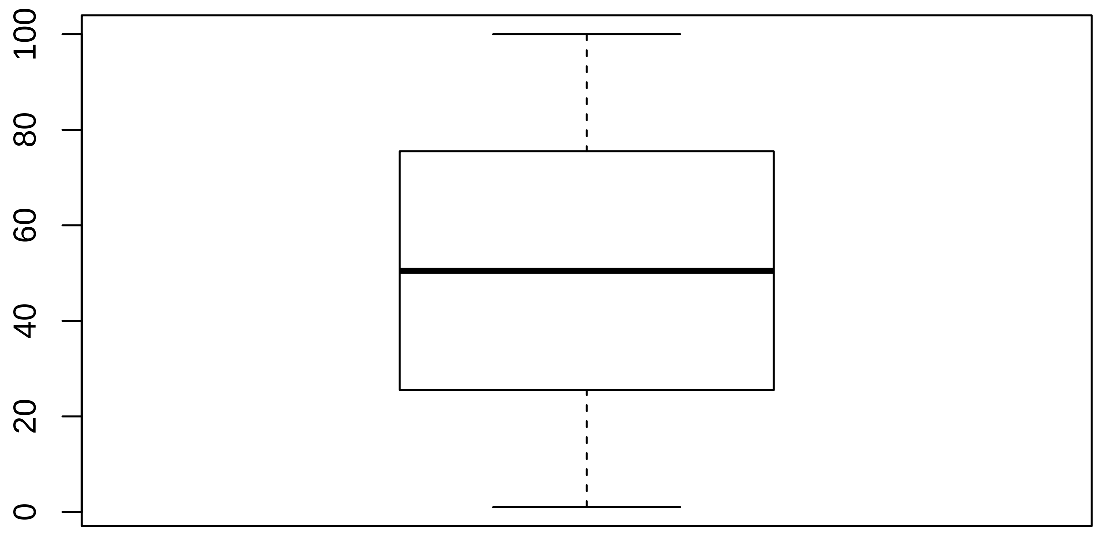
The `summary()` function gives you this five-number summary, and throws in the sample mean as a bonus.

```r
summary(magnitude)
```

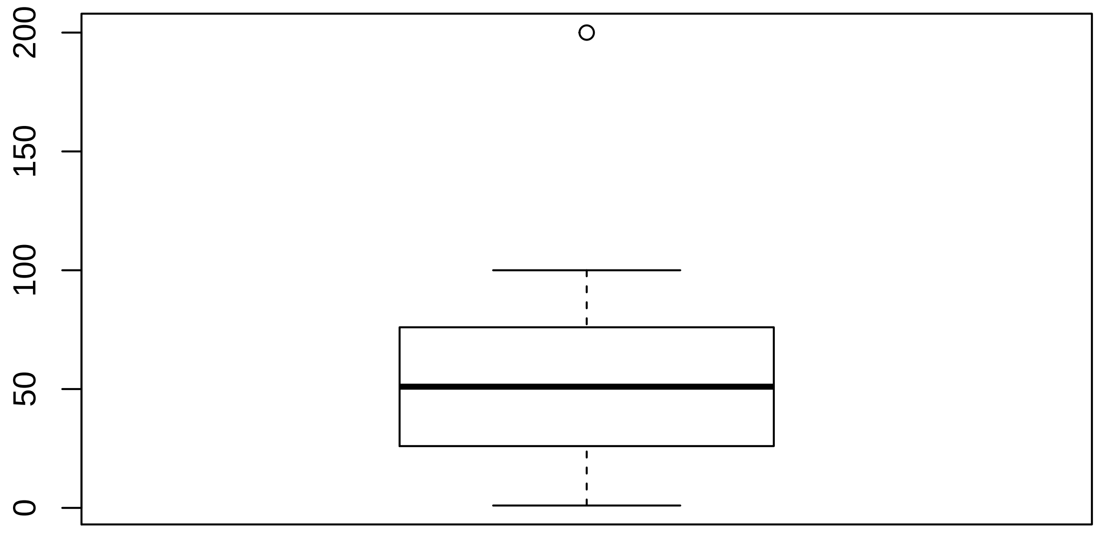
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.710   1.030   1.128   1.430   5.100
```

Instead of a list of numbers, it's easier to interpret the five-number summary by displaying it as a **boxplot**. We draw a "box" that starts from the first quartile and ends at the third quartile. Inside the box, draw a line to show the median. Then add "whiskers" from q_1 to the minimum, and from q_3 to the maximum. Or, if there are **outliers** that don't fit the pattern of the data, draw the whiskers to the furthest non-outliers non-outliers and mark the outliers with dots.

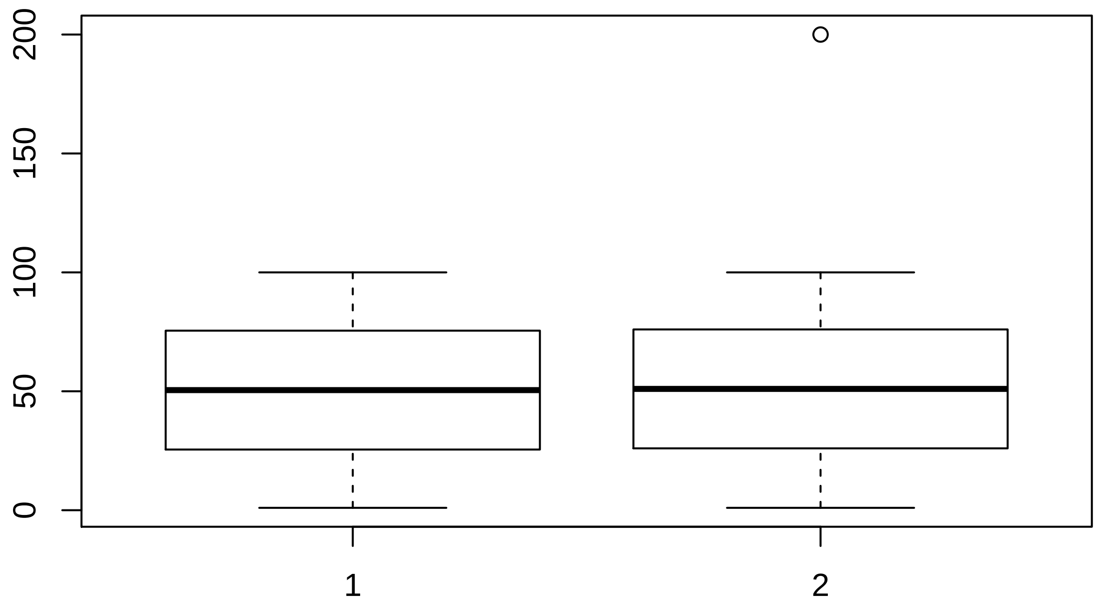
```r
x100 = 1:100 # Sequence of counting numbers
boxplot(x100)
```



```
x100a = c(1:100, 200)
boxplot(x100a)
```



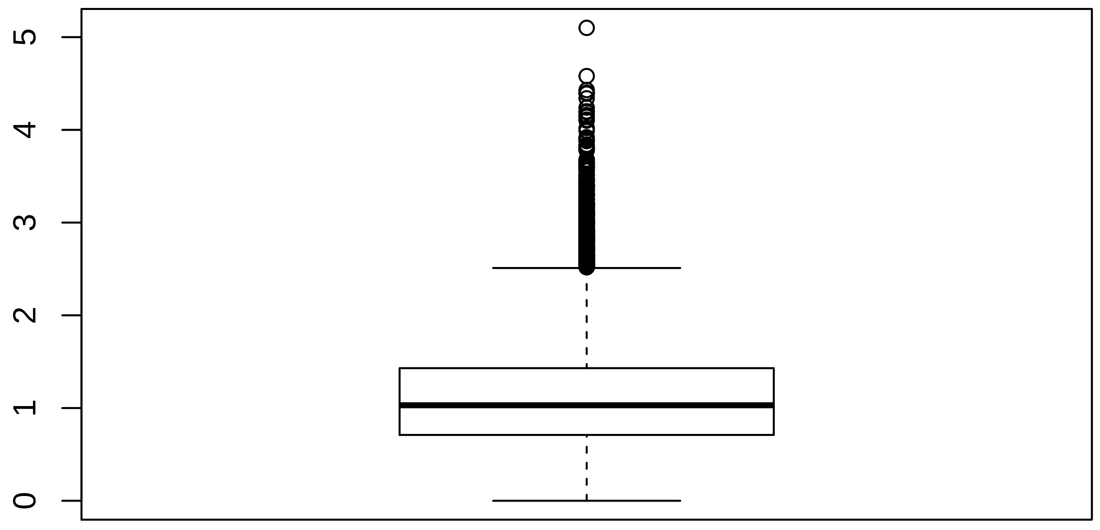
```
boxplot(x100, x100a)
```



One boxplot by itself is kind of boring, but drawing boxplots side-by-side on the same scale lets us make comparisons easily.

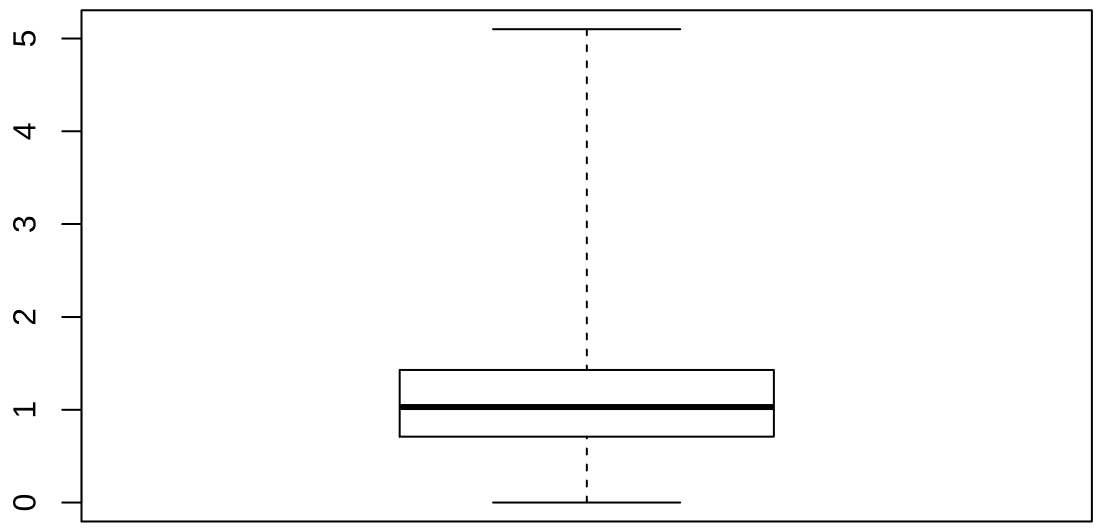
Let's try the boxplot out on the magnitude data.

```
boxplot(magnitude)
```



This is pretty ugly – if you're identifying hundreds of outliers, then maybe they're not really outliers. We can turn outlier detection off:

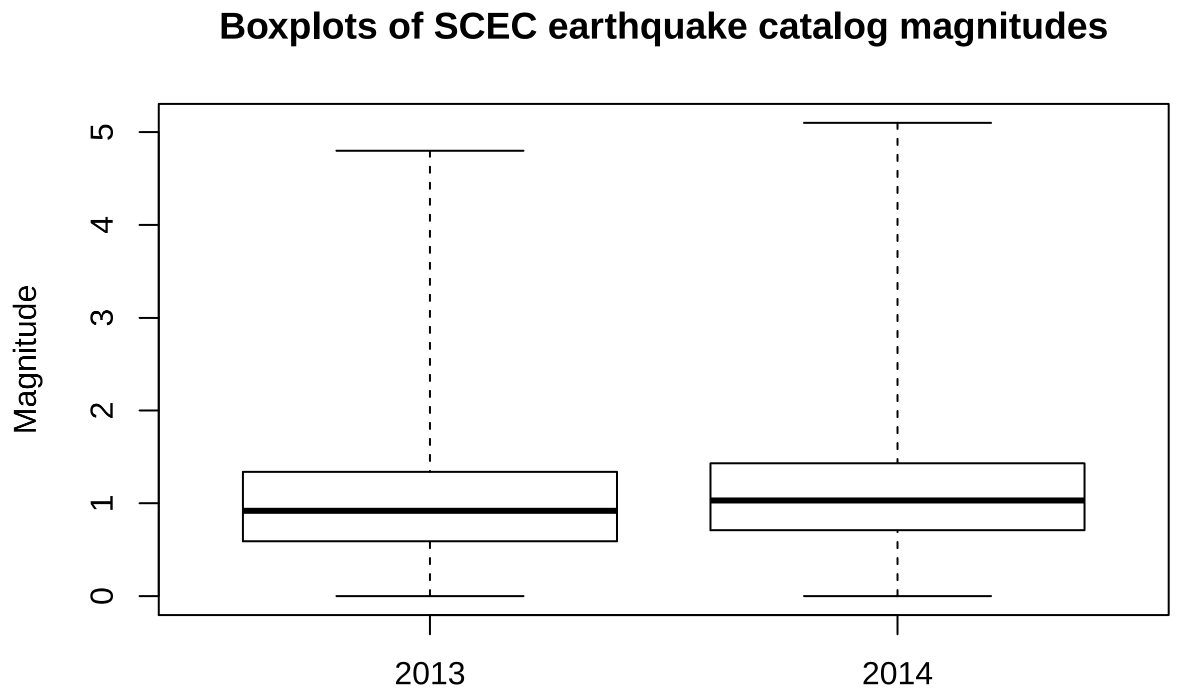
```
boxplot(magnitude, range=0)
```



Prettier graphs

Most of R's plots have reasonable defaults for things like axis labels, but some don't, and most of the time you'll want to specify the axis labels yourself anyway. Within any plotting command, there are **arguments** that let you control how your plot looks – just like the `range` argument we specified above. Here's an example:

```
earthquake2013 = read.table("https://service.scedc.caltech.edu/ftp/catalogs/SCEC_DC/2013.catalog")
magnitude2013 = earthquake2013[,5]
boxplot(magnitude2013, magnitude, range=0,
  main="Boxplots of SCEC earthquake catalog magnitudes",
  ylab="Magnitude",
  names=c(2013, 2014))
```



We can see that typical, earthquakes in the 2014 catalog were a little bigger than earthquakes in the 2013 catalog.

Try it yourself

The file `richlist.txt` on Canvas contains the wealth (in billions of U.S. dollars) of the 100 richest people in the world. Load the data into R and:

1. Graph the ECDF.
2. Find the five-number summary.
3. Draw a boxplot of the data.

What is a QQ plot?

Let's simulate 50 observations from a normal distribution.

```
x = rnorm(50, 0, 1)
sort(x)
```

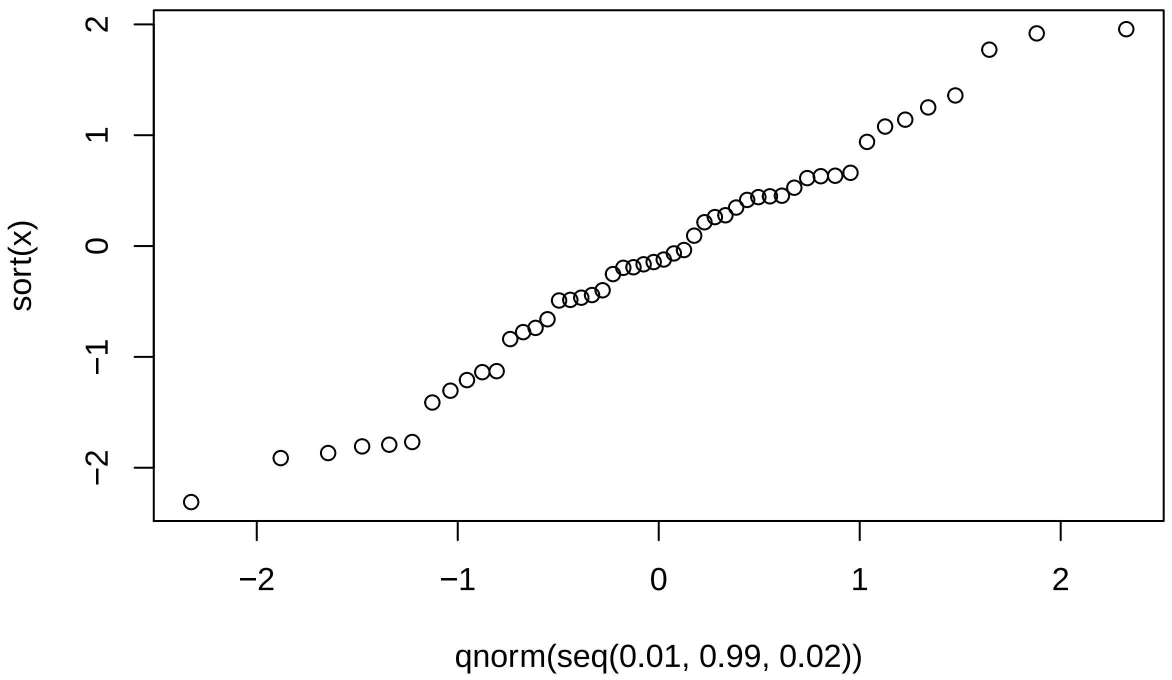
```
##  [1] -2.30979167 -1.91290690 -1.86754078 -1.80764001 -1.79214413
##  [6] -1.76802447 -1.41164836 -1.30493057 -1.20924948 -1.13781901
## [11] -1.12850180 -0.83922039 -0.77654826 -0.73837283 -0.66038802
## [16] -0.49106540 -0.48560608 -0.46579791 -0.44210679 -0.39882622
## [21] -0.25277850 -0.19628646 -0.19110262 -0.16458233 -0.14444784
## [26] -0.12150220 -0.06633999 -0.03590670  0.09370725  0.21477896
## [31]  0.26130205  0.27773127  0.34742626  0.41627620  0.44135958
## [36]  0.44874474  0.45429273  0.52629865  0.61296100  0.63015607
## [41]  0.63428523  0.66085745  0.93974911  1.07803395  1.14021069
## [46]  1.25084920  1.35857348  1.77172267  1.91850534  1.95653526
```

If we repeat this, we'll get different values. So what *should* the values be? One answer to this is to take fifty evenly-spaced quantiles from the standard normal distribution.

```
qnorm(seq(0.01, 0.99, 0.02))
```

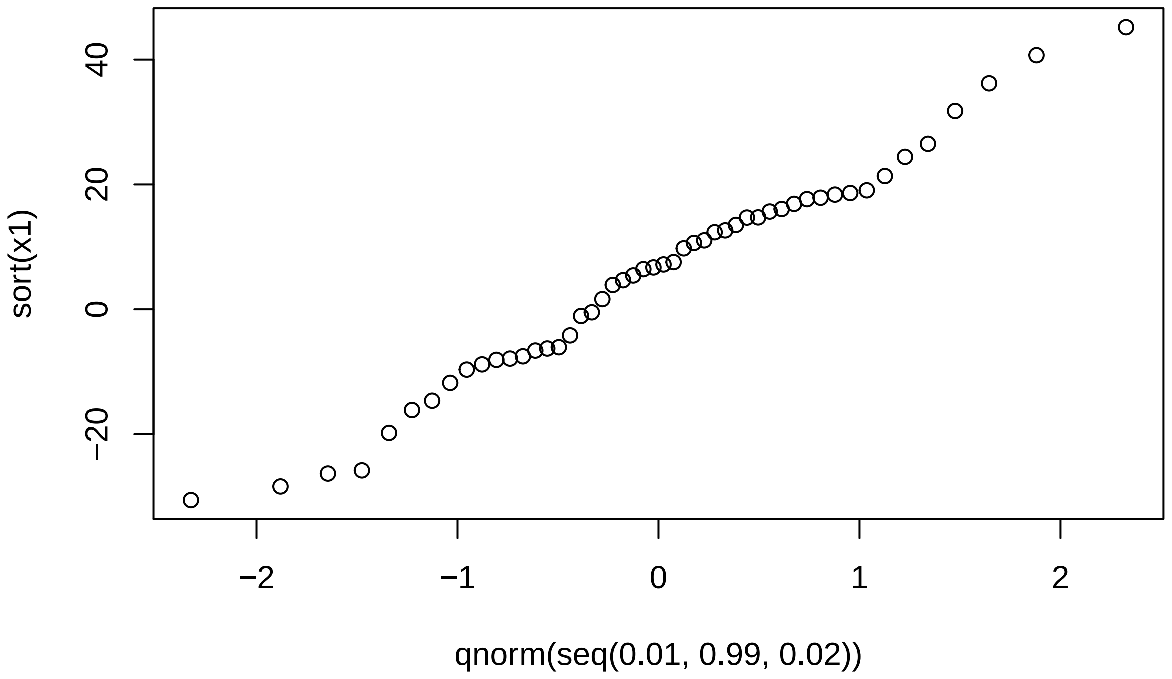
```
##  [1] -2.32634787 -1.88079361 -1.64485363 -1.47579103 -1.34075503
##  [6] -1.22652812 -1.12639113 -1.03643339 -0.95416525 -0.87789630
## [11] -0.80642125 -0.73884685 -0.67448975 -0.61281299 -0.55338472
## [16] -0.49585035 -0.43991317 -0.38532047 -0.33185335 -0.27931903
## [21] -0.22754498 -0.17637416 -0.12566135 -0.07526986 -0.02506891
## [26]  0.02506891  0.07526986  0.12566135  0.17637416  0.22754498
## [31]  0.27931903  0.33185335  0.38532047  0.43991317  0.49585035
## [36]  0.55338472  0.61281299  0.67448975  0.73884685  0.80642125
## [41]  0.87789630  0.95416525  1.03643339  1.12639113  1.22652812
## [46]  1.34075503  1.47579103  1.64485363  1.88079361  2.32634787
```

```
plot(qnorm(seq(0.01, 0.99, 0.02)),
     sort(x))
```



The scatterplot is well-approximated by the line $y = x$. What if we tried this with data simulated from a non-standard normal distribution?

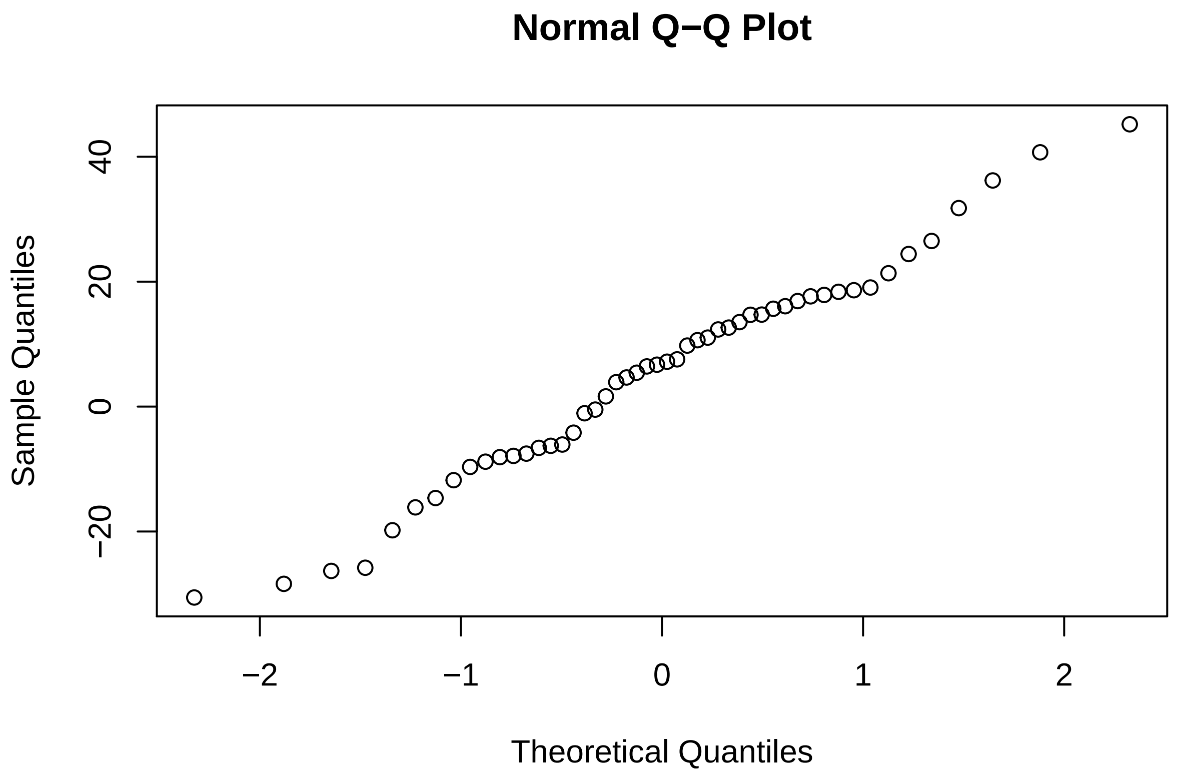
```r
x1 = rnorm(50, mean=10, sd=20)
plot(qnorm(seq(0.01, 0.99, 0.02)),
     sort(x1))
```



While it's no longer the line $y = x$, it's still pretty close to a straight line.

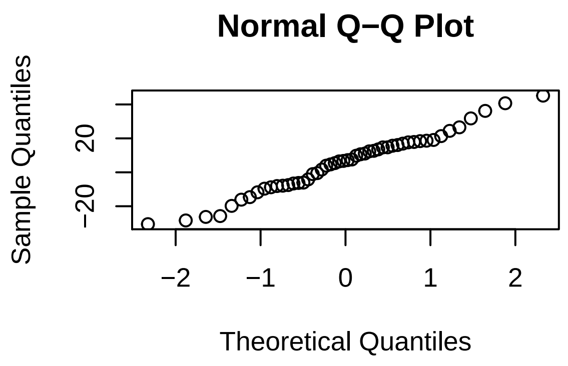
This kind of plot, which plots quantiles of the data against quantiles of the normal, is called a **normal probability plot** or a **normal quantile-quantile plot** or a **normal QQ plot**. Here's an easier way to generate it:

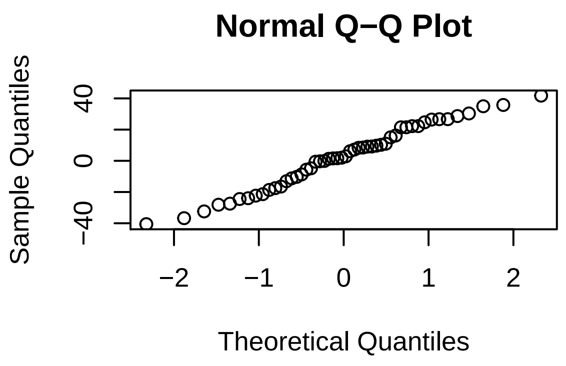
```r
qqnorm(x1)
```

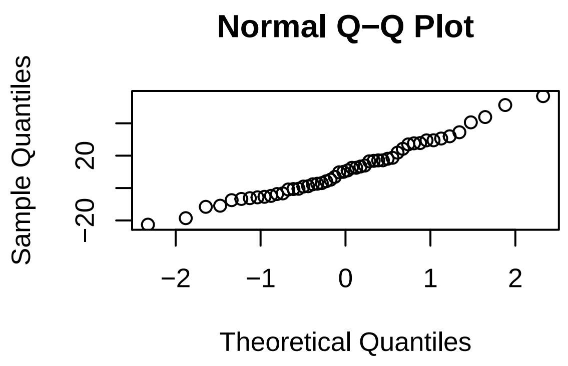


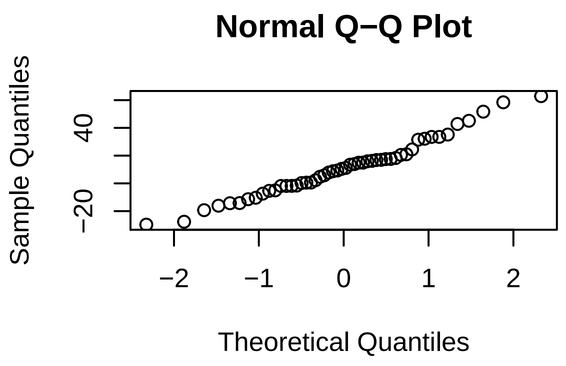
Of course, this is just one set of random data. Let's repeat this a few times and see how similar the plots are. We'll use the `par()` function to set up a two-by-two grid of graphs.

```r
par(mfrow=c(2,2))

qqnorm(x1)

x2 = rnorm(50, mean=10, sd=20)
qqnorm(x2)

x3 = rnorm(50, mean=10, sd=20)
qqnorm(x3)

x4 = rnorm(50, mean=10, sd=20)
qqnorm(x4)
```



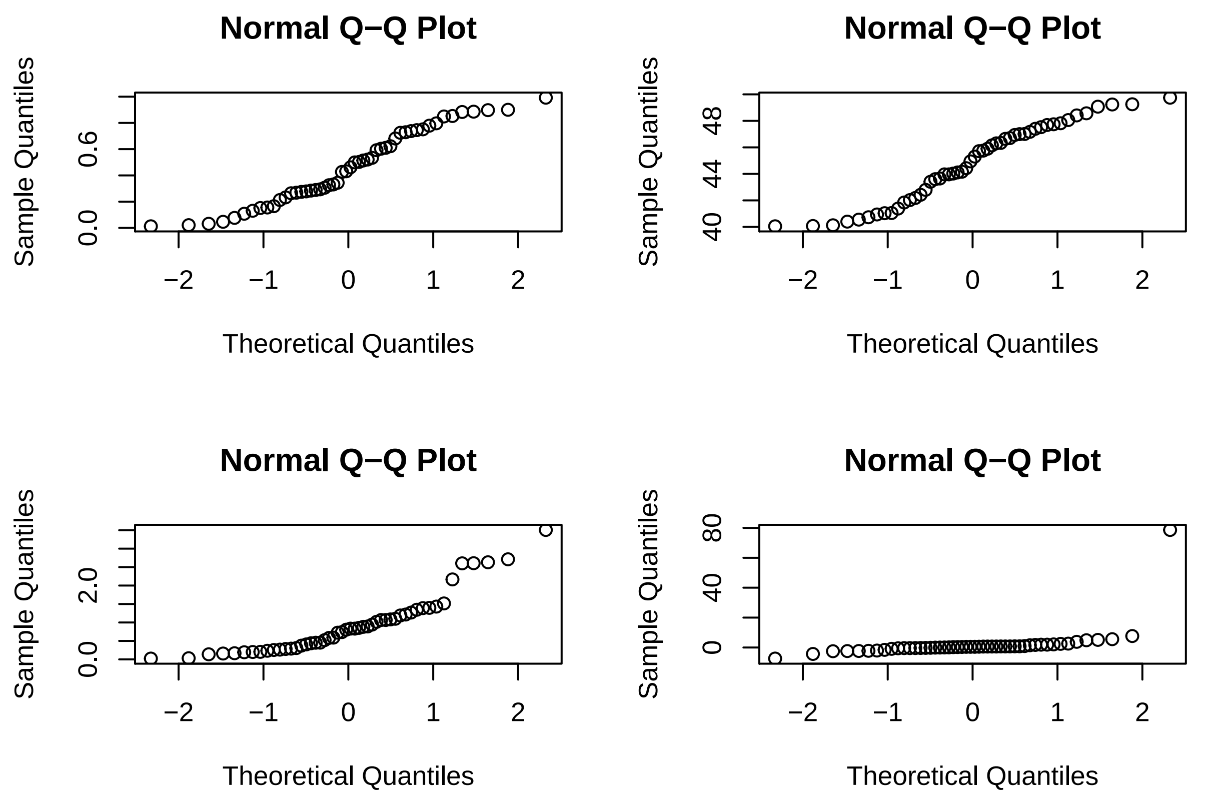






In each case, we get pretty close to a straight line, allowing for a little wiggliness at the extremes. What if we try this for non-normal distributions?

```r
par(mfrow=c(2,2))

y1 = runif(50)
qqnorm(y1)

y2 = 10 * runif(50) + 40
qqnorm(y2)

y3 = rexp(50)
qqnorm(y3)

y4 = rnorm(50) / rnorm(50)
qqnorm(y4)
```



```
par(mfrow=c(1,1)) # reset to single graph
```

In all of these QQ plots we see not just a little wiggliness but a systematic bend in the graphs. They're not normal.

Heights of the New York Choral Society

Let's return to the choral singer heights.

```
singer = read.table("singer.txt",
  header=TRUE)
head(singer)
```

```
##   height voice.part
## 1     64  Soprano 1
## 2     62  Soprano 1
## 3     66  Soprano 1
## 4     65  Soprano 1
## 5     60  Soprano 1
## 6     61  Soprano 1
```

```
length(singer$height)
```

```
## [1] 235
```

```
summary(singer$height)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    60.0    65.0    67.0    67.3    70.0    76.0
```

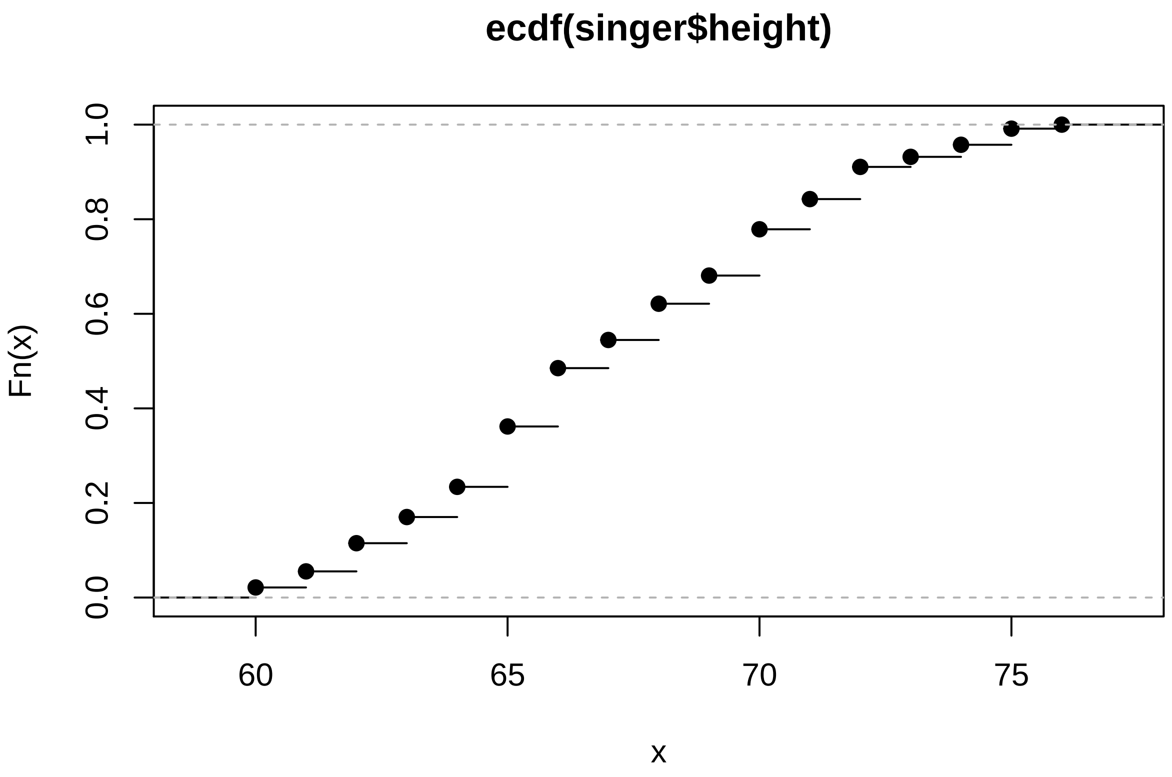
```r
var(singer$height)
```

```
## [1] 14.62884
```

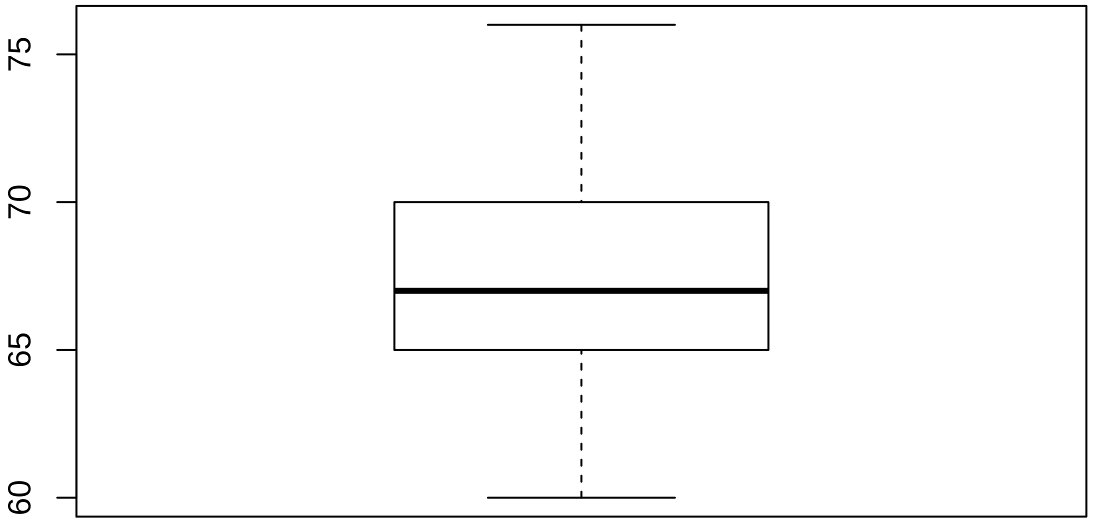
```r
sd(singer$height)
```

```
## [1] 3.824767
```

```r
plot(ecdf(singer$height))
```

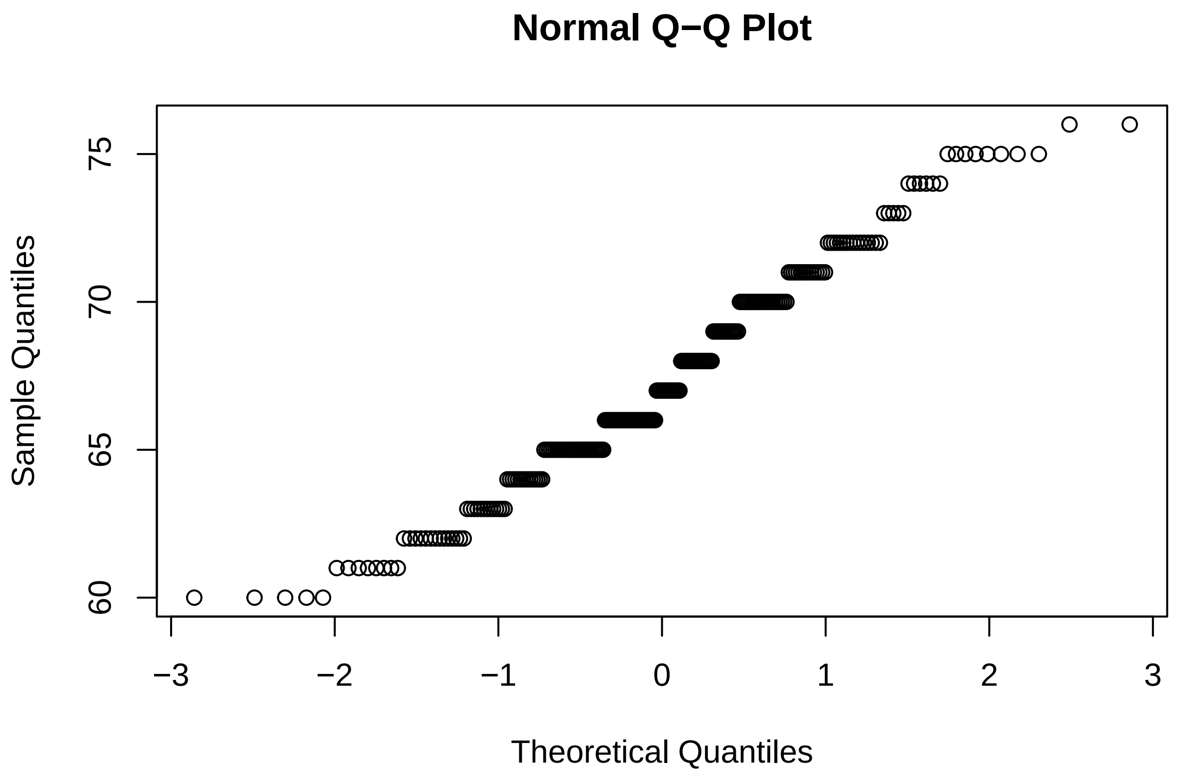


```r
boxplot(singer$height)
```



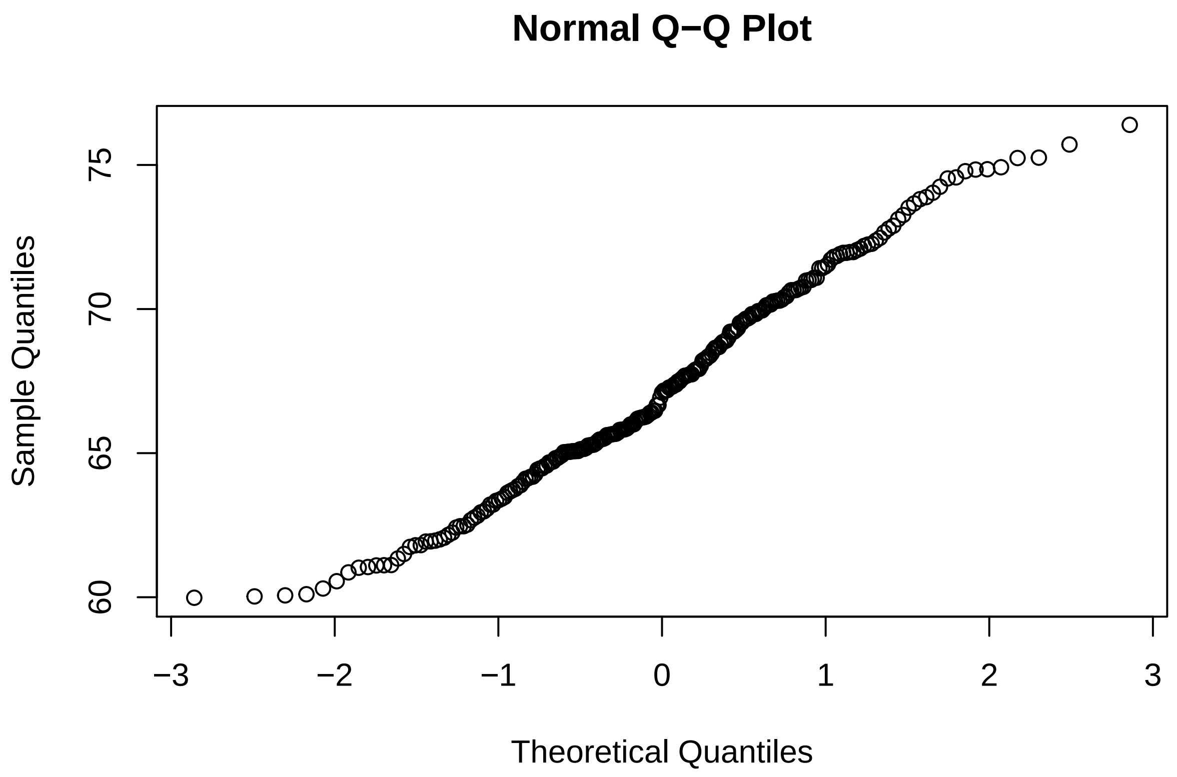
We quickly see the singers' heights have a reasonably symmetric distribution. But is it normal?

```r
qqnorm(singer$height)
```



Here we run into a problem. The data is rounded to the nearest inch, so the distribution is in practice discrete. This means the our normal QQ plot will look lumpy. We can get around this by "unrounding" the data: add some random noise to approximate what the data would have looked like without rounding.

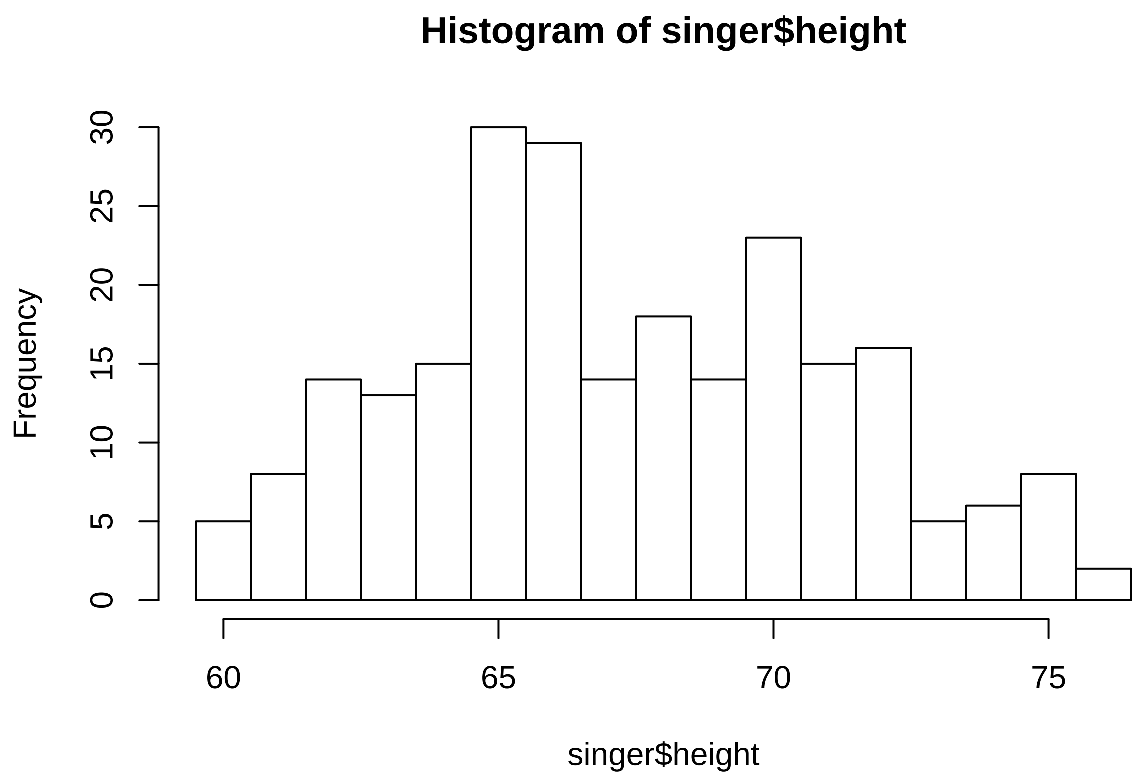
```r
fakeheights = singer$height +
  runif(235, min=-0.5, max=0.5)
qqnorm(fakeheights)
```



The data doesn't quite look normal – there's a bit of an "S" shape in the QQ plot.

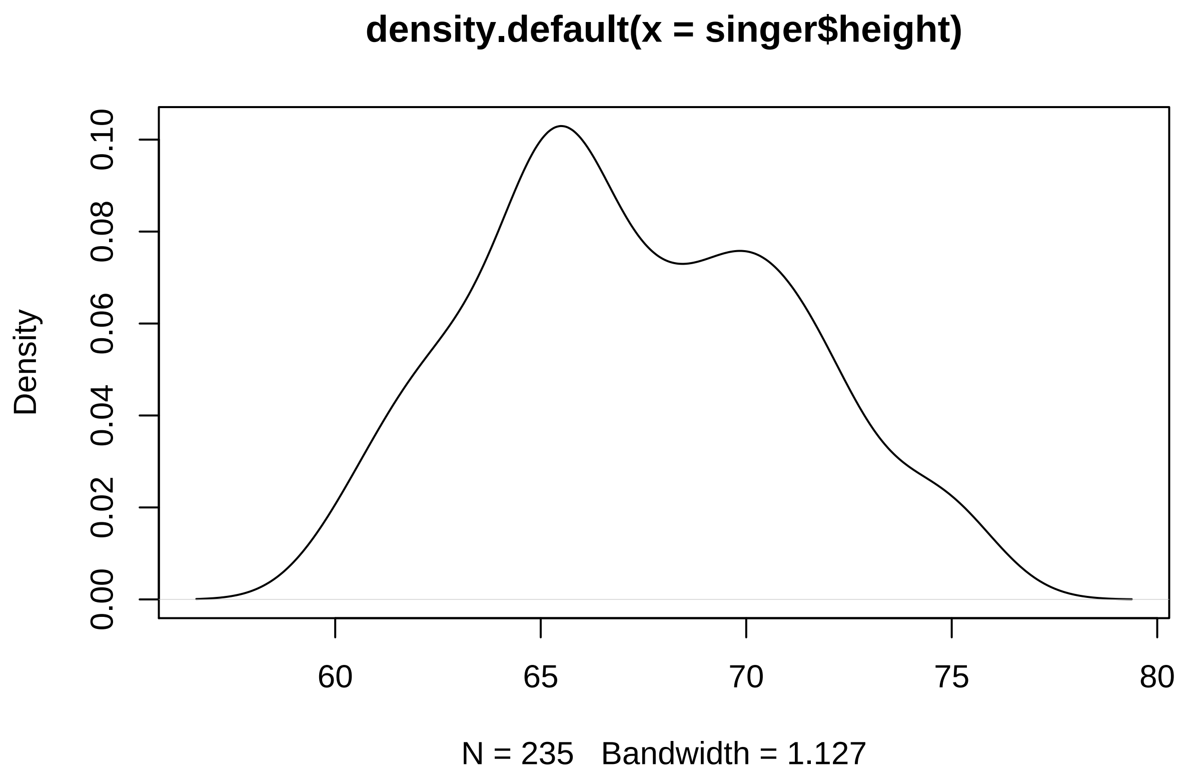
So how does the data depart from normality? To get a rough idea of the PDF, we can draw a histogram of the data .

```
hist(singer$height,
     breaks=59.5:76.5)
```



The histogram is a perfectly OK graph to draw. However it does suffer from the limitation that it's "blocky," whereas the (unrounded) distribution of heights should be smooth. The **kernel density plot** can be thought of as a smooth version of the histogram. Here's the density plot for the singer heights:

```r
plot(density(singer$height))
```



Very roughly speaking, the density plot puts down a "kernel" (lump) of probability at each observed x, then adds the kernels together. See what happens if you build the density plot up kernel by kernel:

```
plot(density(singer$height[1:2]))
plot(density(singer$height[1:3]))
plot(density(singer$height[1:4]))
plot(density(singer$height[1:5]))
plot(density(singer$height[1:10]))
plot(density(singer$height[1:30]))
plot(density(singer$height[1:100]))
plot(density(singer$height))
```

Read Trosset chapter 7.4 for the technical details (or take a nonparametric statistics course.) We'll just focus on interpreting the plots. It looks like the distribution of heights might have two peaks. Maybe these correspond to women and men? We can look at the two sets of heights separately:

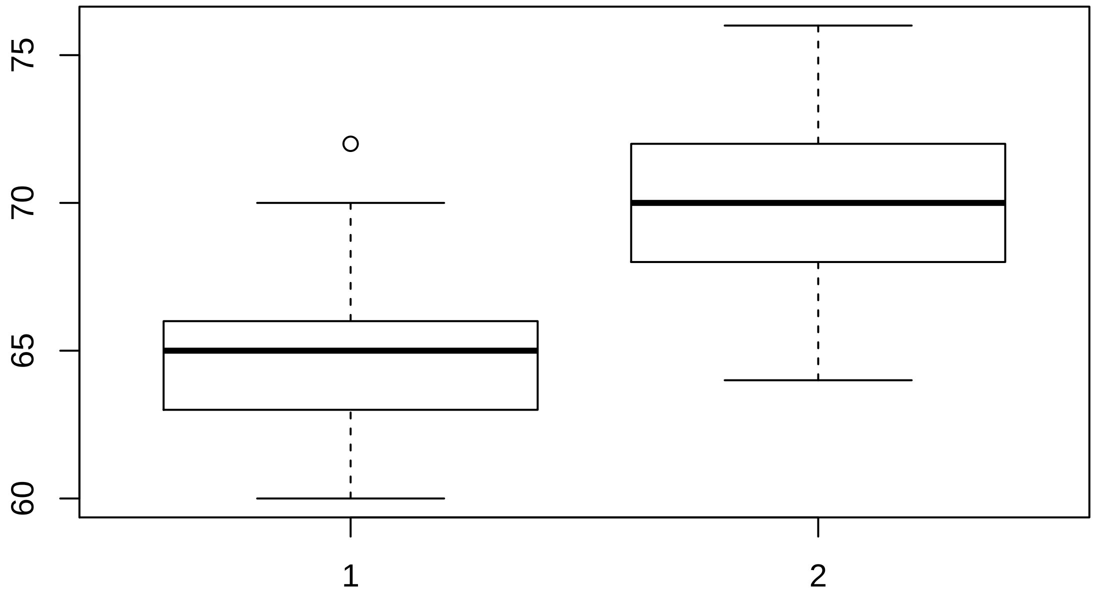
```
womens.heights = singer$height[1:128]
mens.heights = singer$height[129:235]
summary(womens.heights)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   60.00   63.00   65.00   64.73   66.00   72.00
```

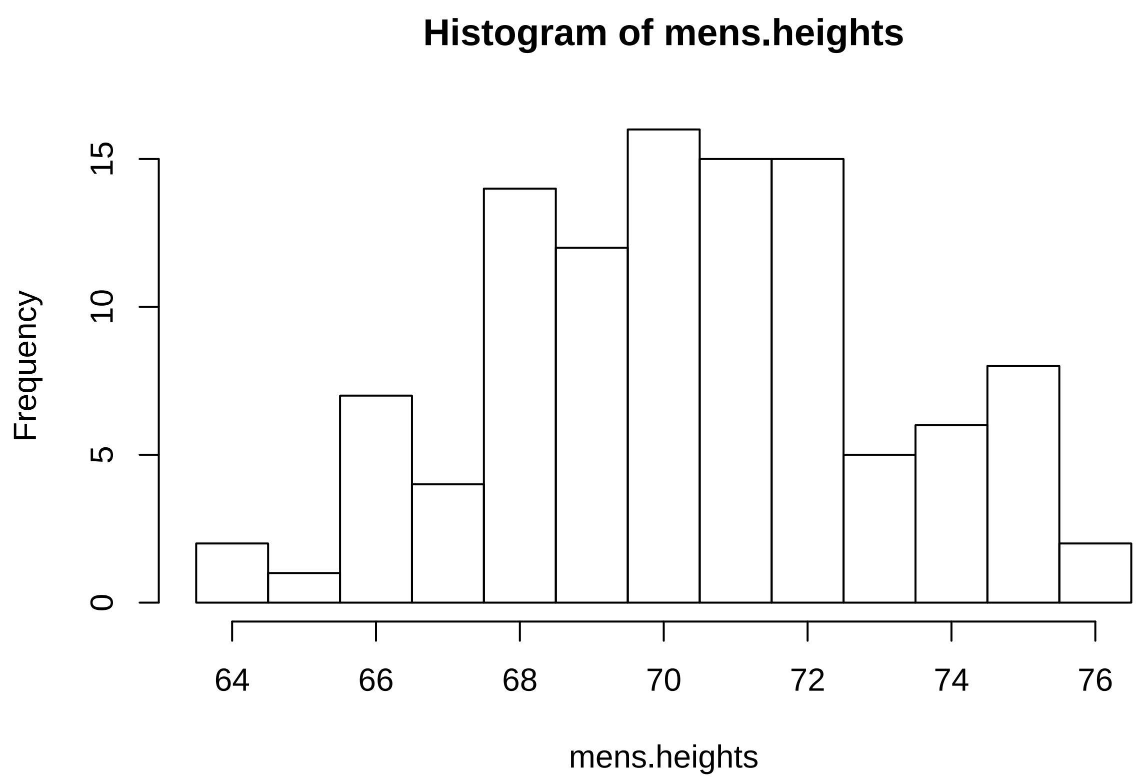
```
summary(mens.heights)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   64.00   68.00   70.00   70.36   72.00   76.00
```

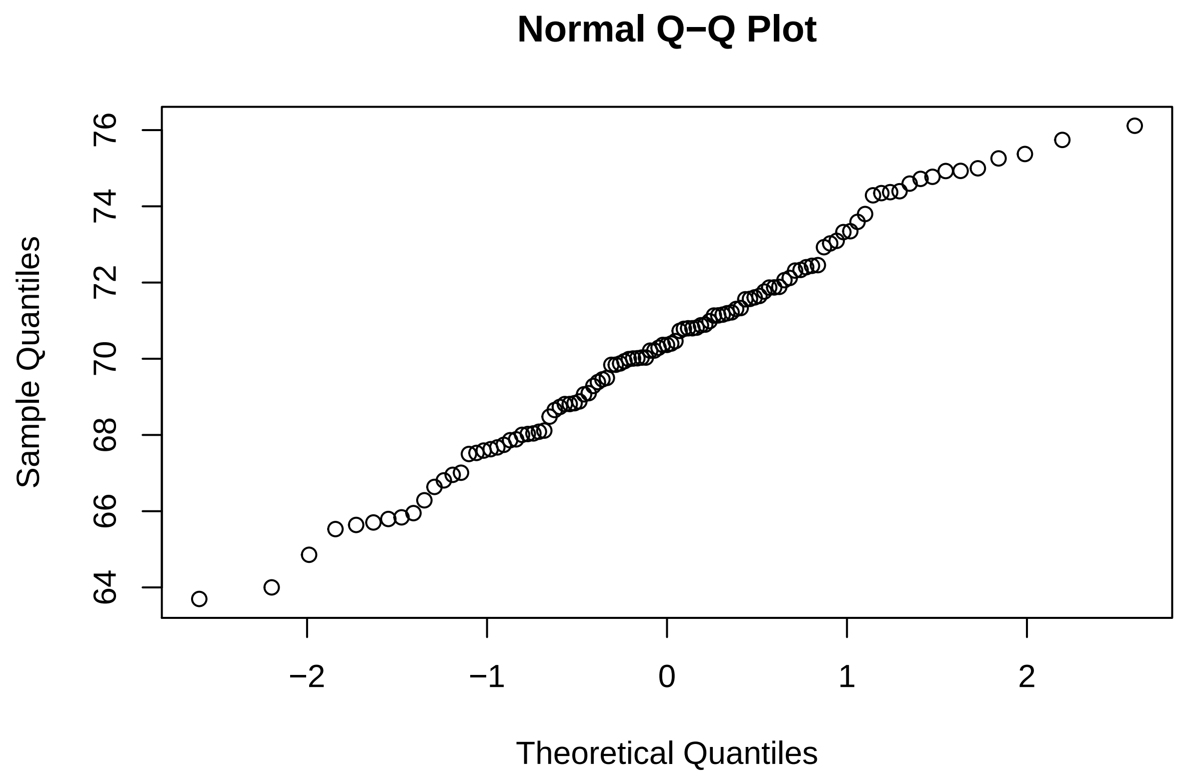
```
boxplot(womens.heights, mens.heights)
```



```
hist(mens.heights, breaks=63.5:76.5)
```



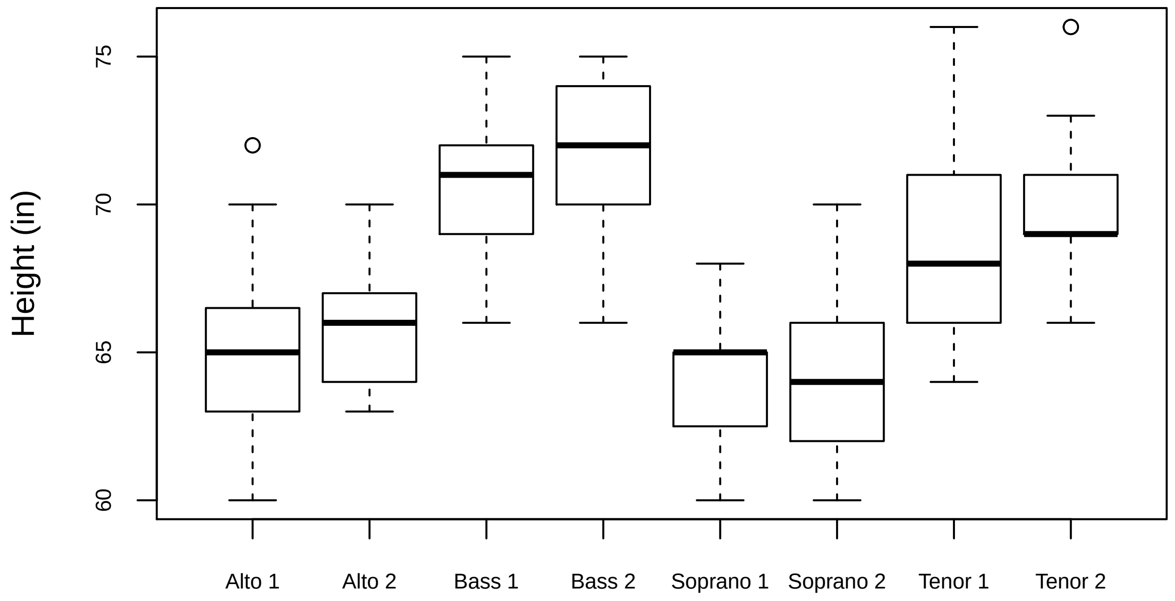
```
qqnorm(mens.heights+
  runif(107, min=-0.5, max=0.5))
```



The men's heights look a lot closer to normal. You can check whether this is also true of the women's heights.

Finally, we could look at the height split up across the eight vocal parts:

```r
boxplot(singer$height~singer$voice.part,
  cex.axis=0.7, #makes x labels smaller
  ylab="Height (in)")
```



Stereogram fusion times

A stereogram is an image seen in 3D by presenting different 2D images to the left and right eyes. An experiment was carried out to determine whether giving visual information changed the time required for people to see the 3D image. In the file `stereograms.txt`, group 1 was a control group (no visual information) and group 2 was a treatment group (visual information.)

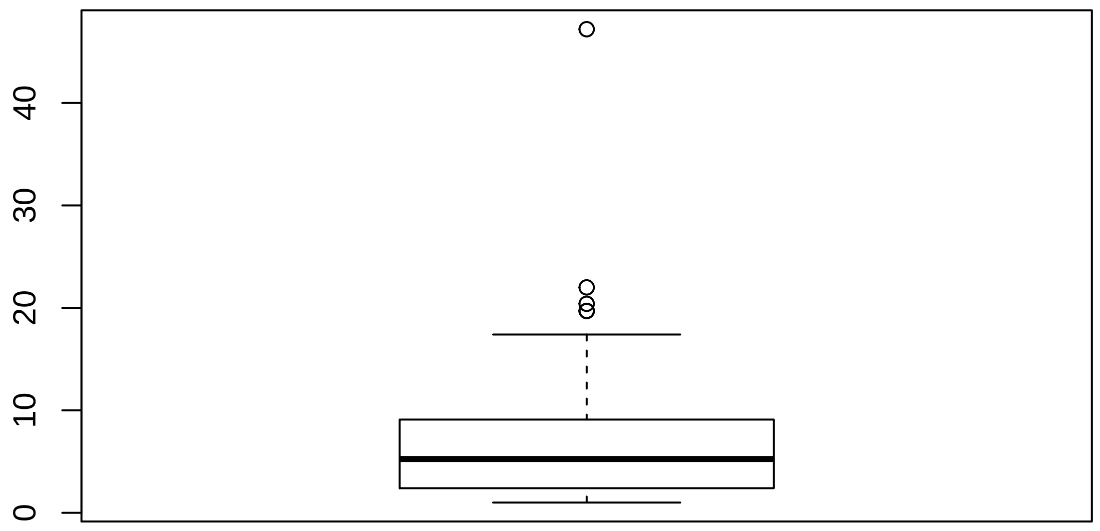
```r
stereograms = read.table("stereograms.txt",
  header=TRUE)
head(stereograms)
```

```
##   time group
## 1 47.2     1
## 2 22.0     1
## 3 20.4     1
## 4 19.7     1
## 5 17.4     1
## 6 14.7     1
```

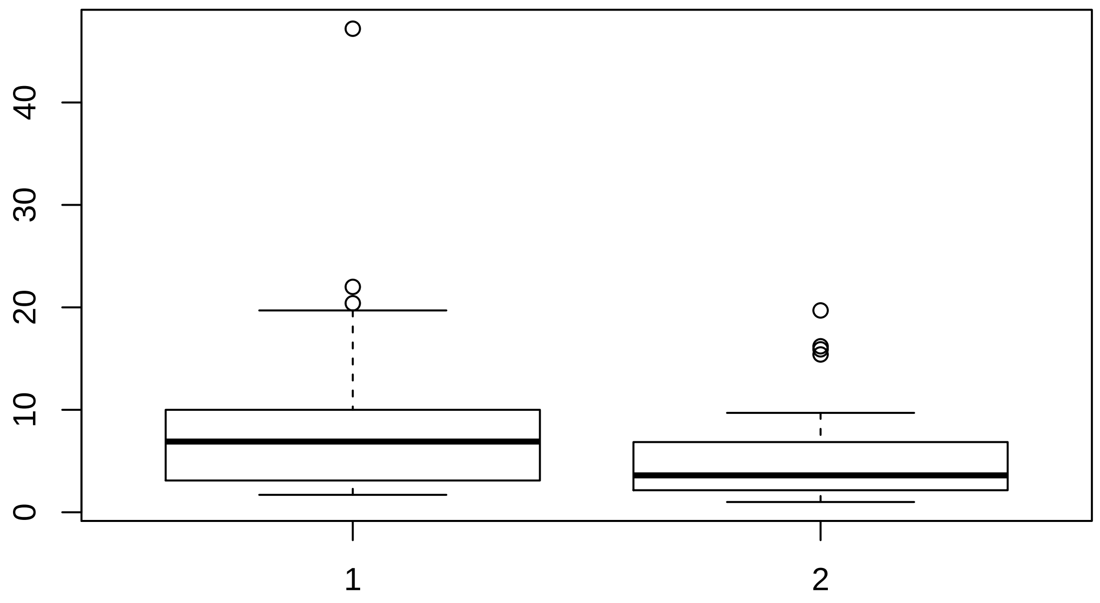
```r
summary(stereograms)
```

```
##       time             group      
##  Min.   : 1.000   Min.   :1.000  
##  1st Qu.: 2.475   1st Qu.:1.000  
##  Median : 5.250   Median :1.000  
##  Mean   : 7.210   Mean   :1.449  
##  3rd Qu.: 9.050   3rd Qu.:2.000  
##  Max.   :47.200   Max.   :2.000
```

```r
boxplot(stereograms$time)
```

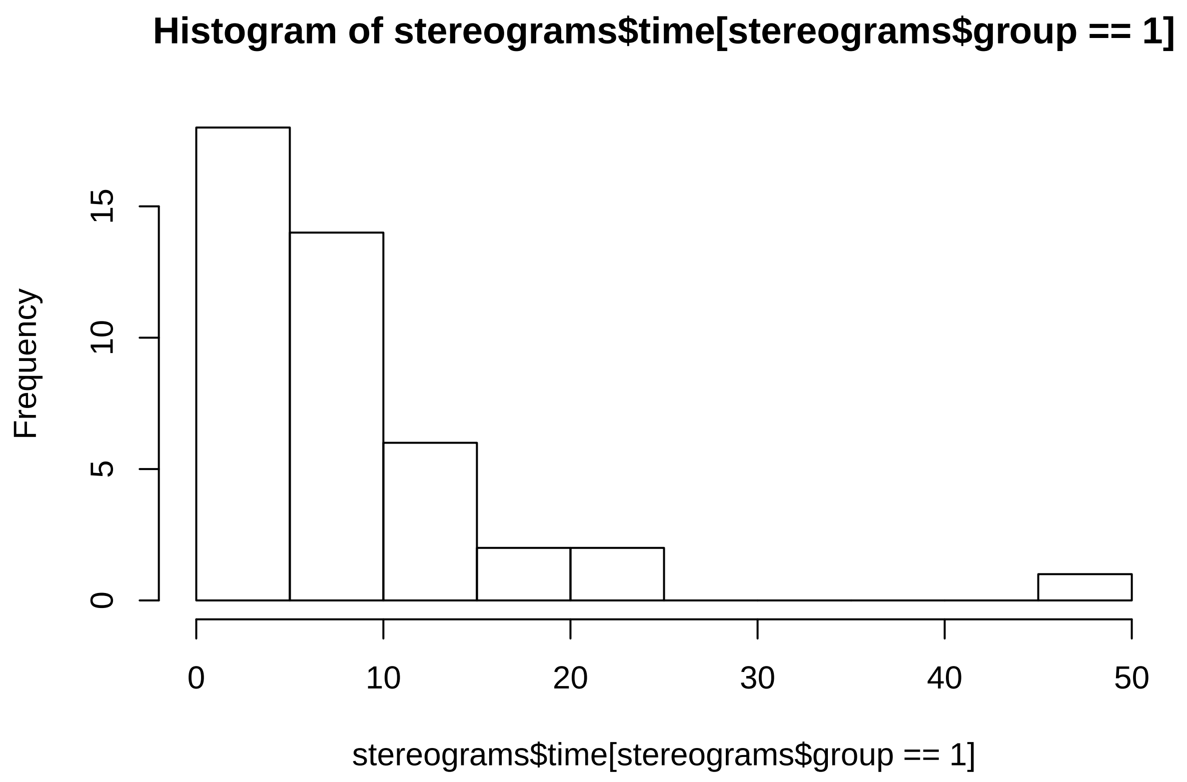


```
boxplot(stereograms$time~stereograms$group)
```

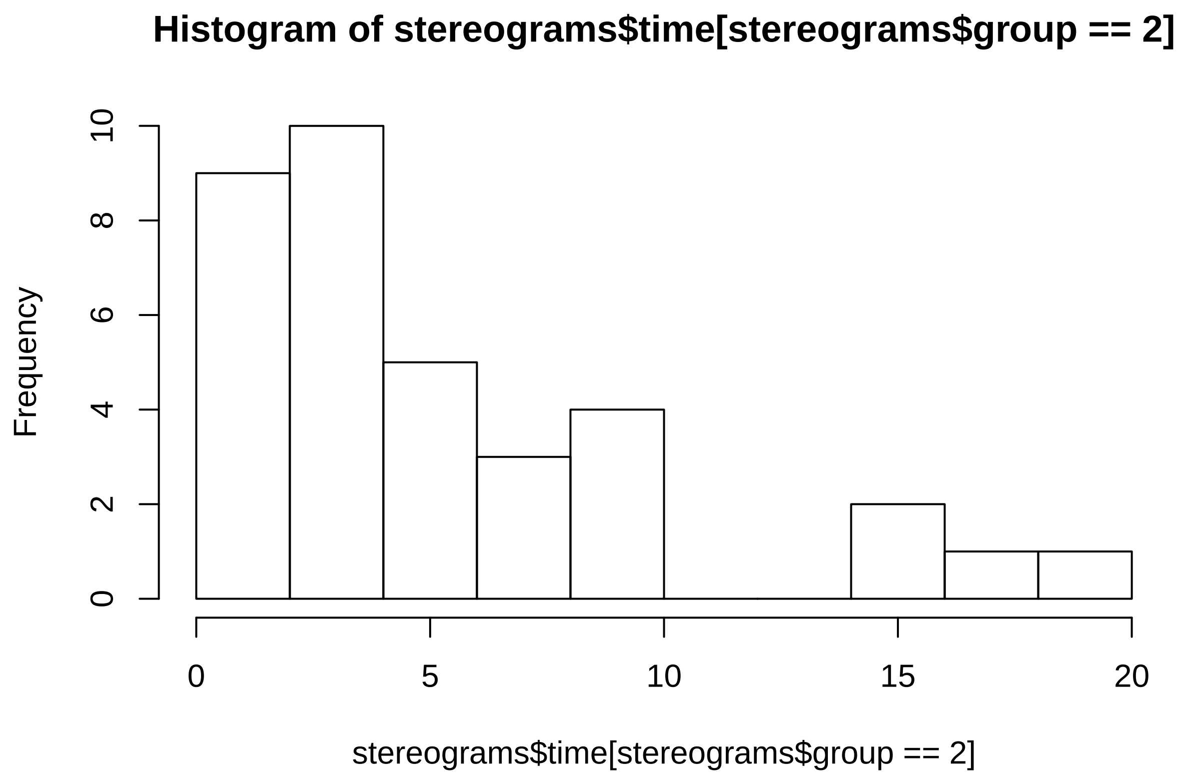


It's clear that the two sets of data have very different distributions.

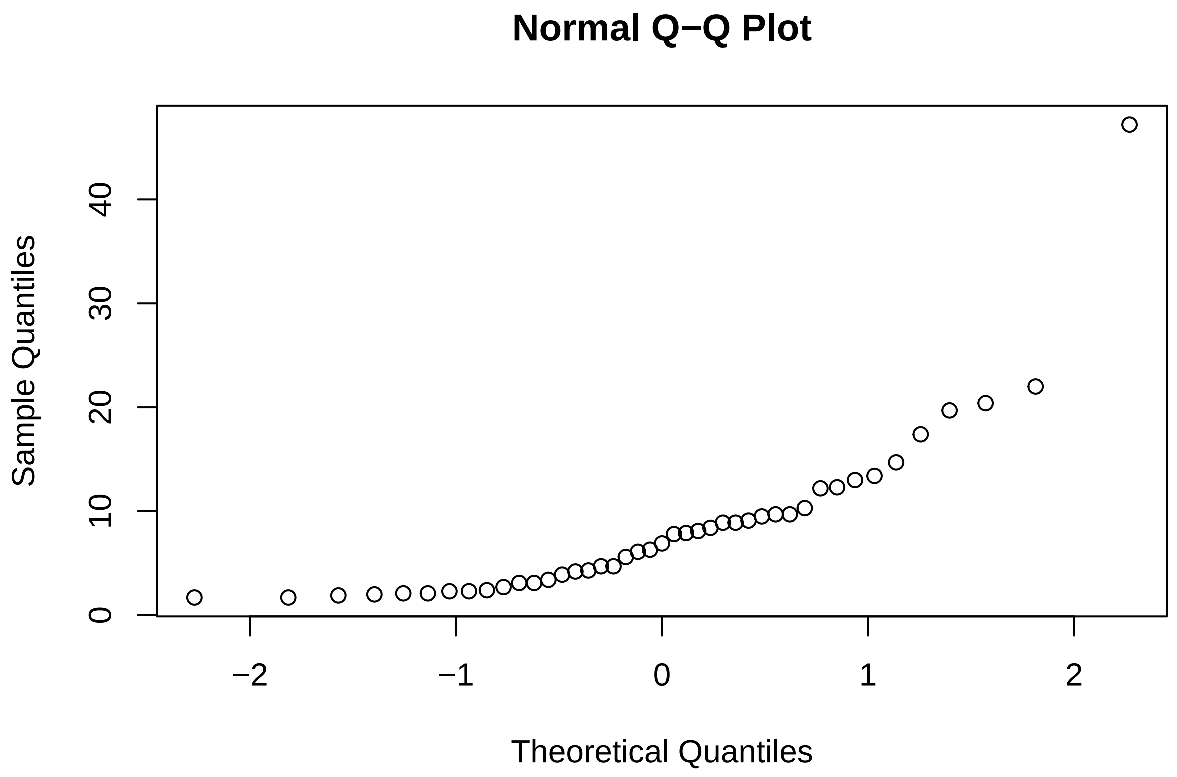
```
hist(stereograms$time[stereograms$group==1])
```



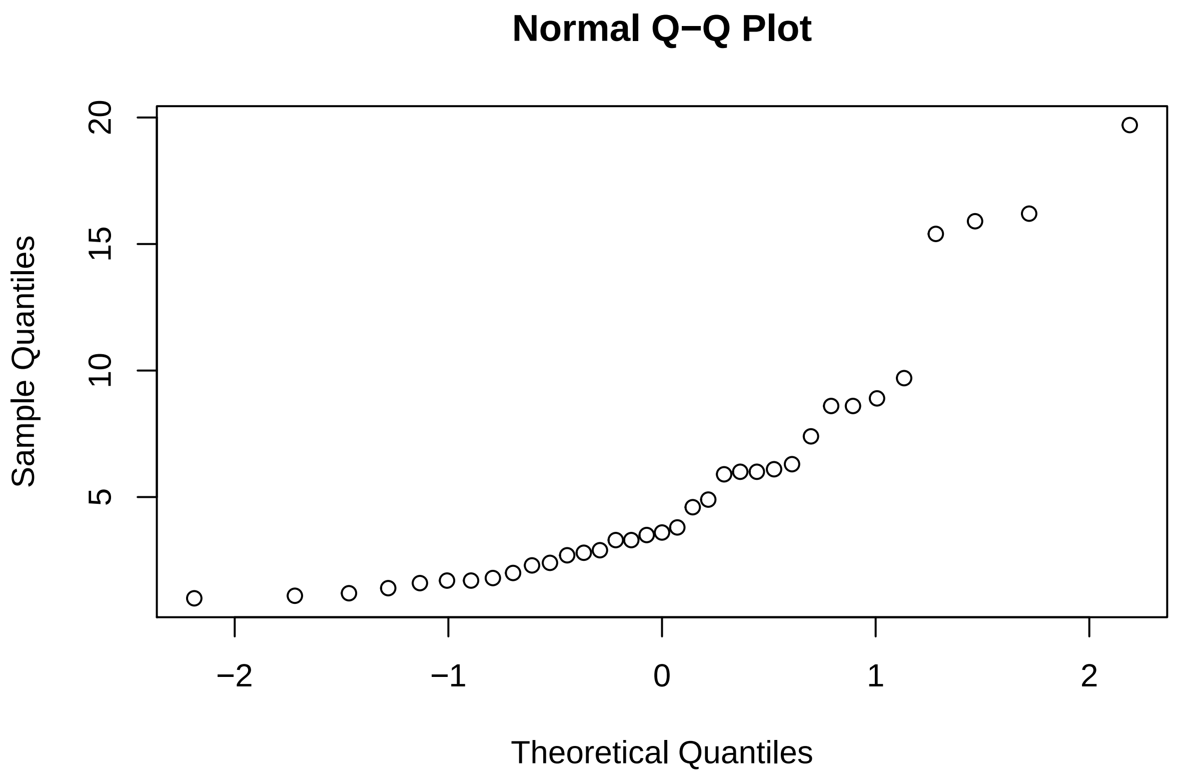
```
hist(stereograms$time[stereograms$group==2])
```



```
qqnorm(stereograms$time[stereograms$group==1])
```

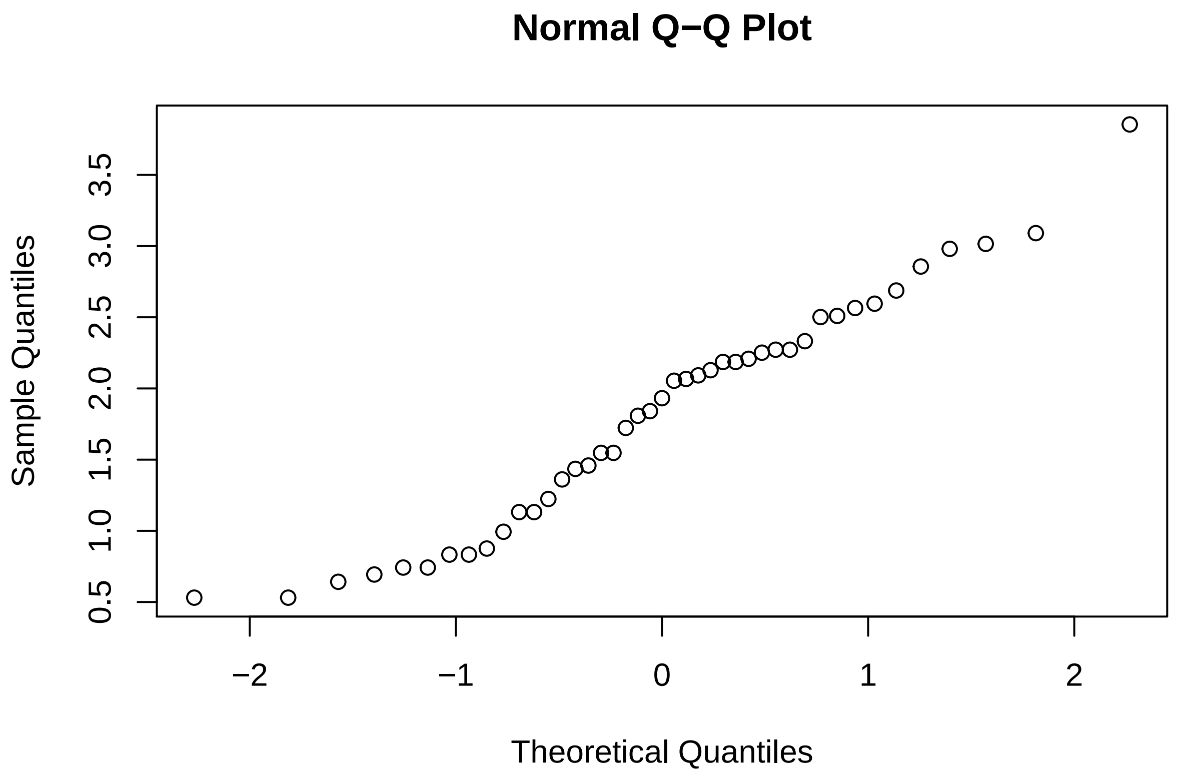


```
qqnorm(stereograms$time[stereograms$group==2])
```

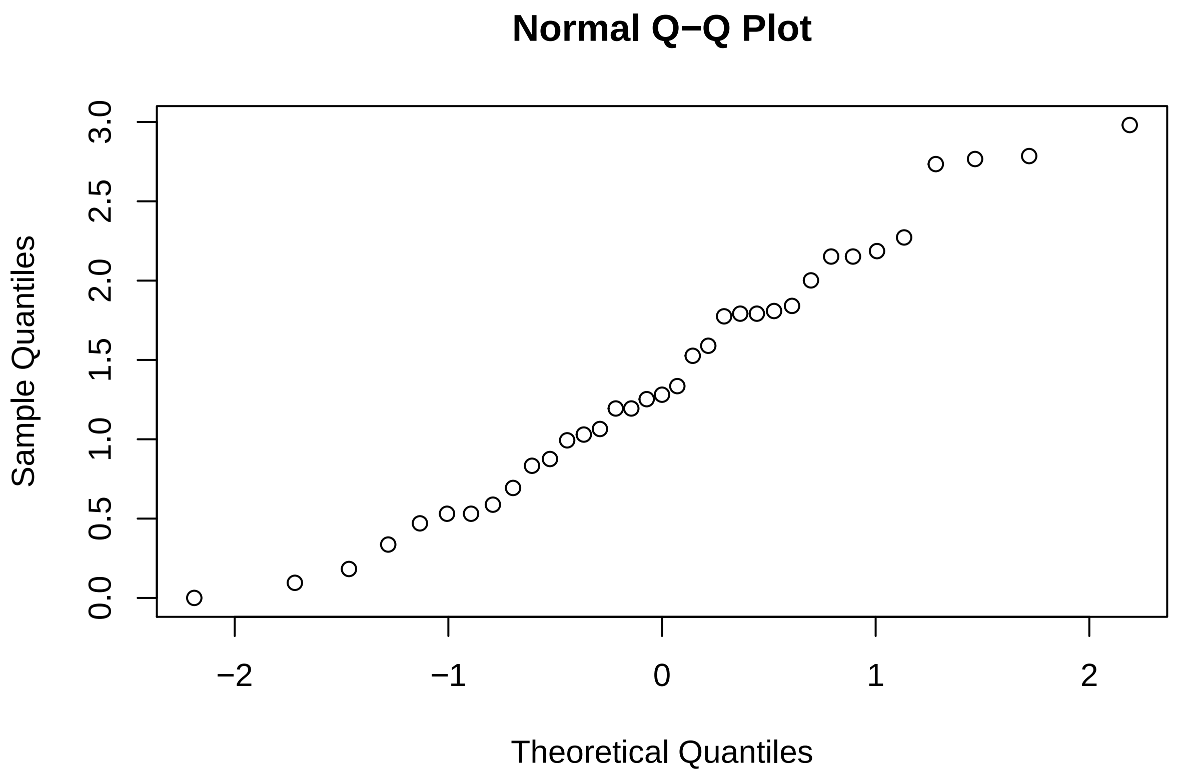


Both distributions are skewed, and certainly not normal. With right-skewed positive data, a useful trick is to take (natural) logs: the logs are often much closer to having a normal distribution, and are possible to interpret (effects that are additive on the log scale are multiplicative on the original scale.)

```
qqnorm(log(stereograms$time[stereograms$group==1]))
```



```
qqnorm(log(stereograms$time[stereograms$group==2]))
```

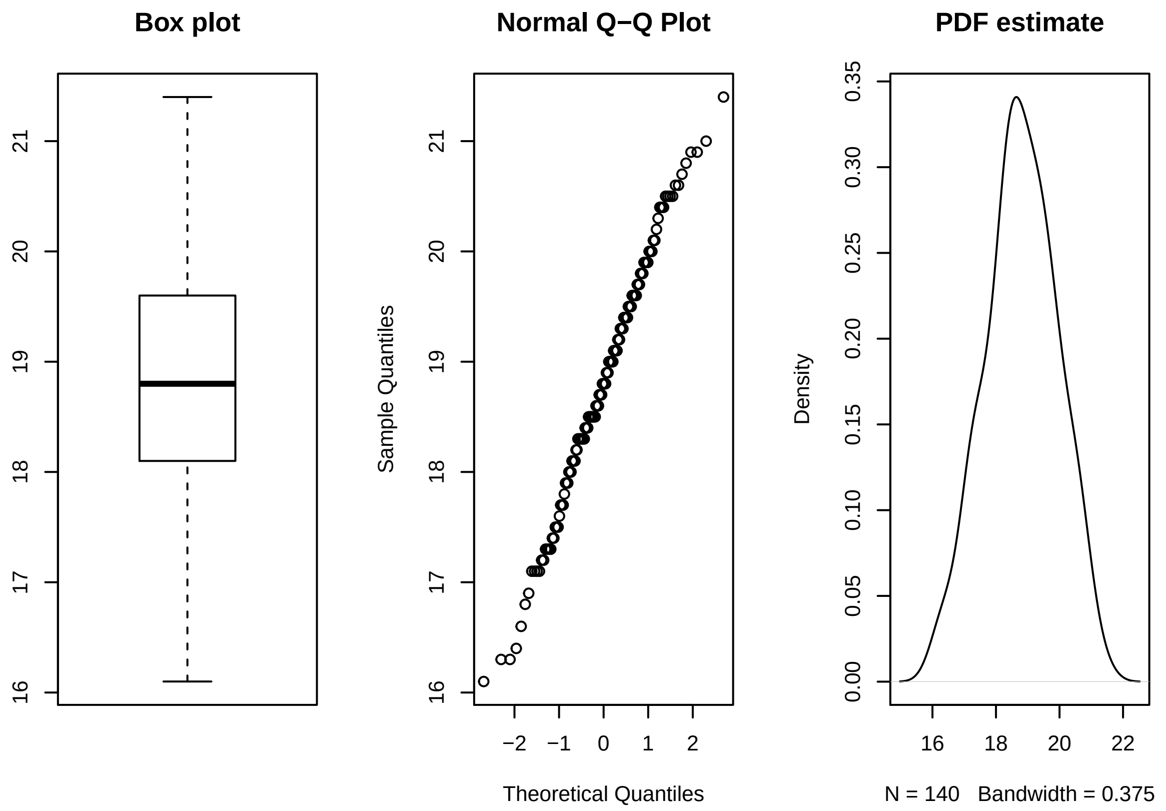


We have **log transformed** the data. Other transformations like square roots can occasionally be useful, but they often do not have a straightforward interpretation. Trosset discusses transformations in his chapter 7.6, and his concluding advice is wise: “never transform unless there is a good reason to do so.”

Case study: Are forearm lengths normally distributed?

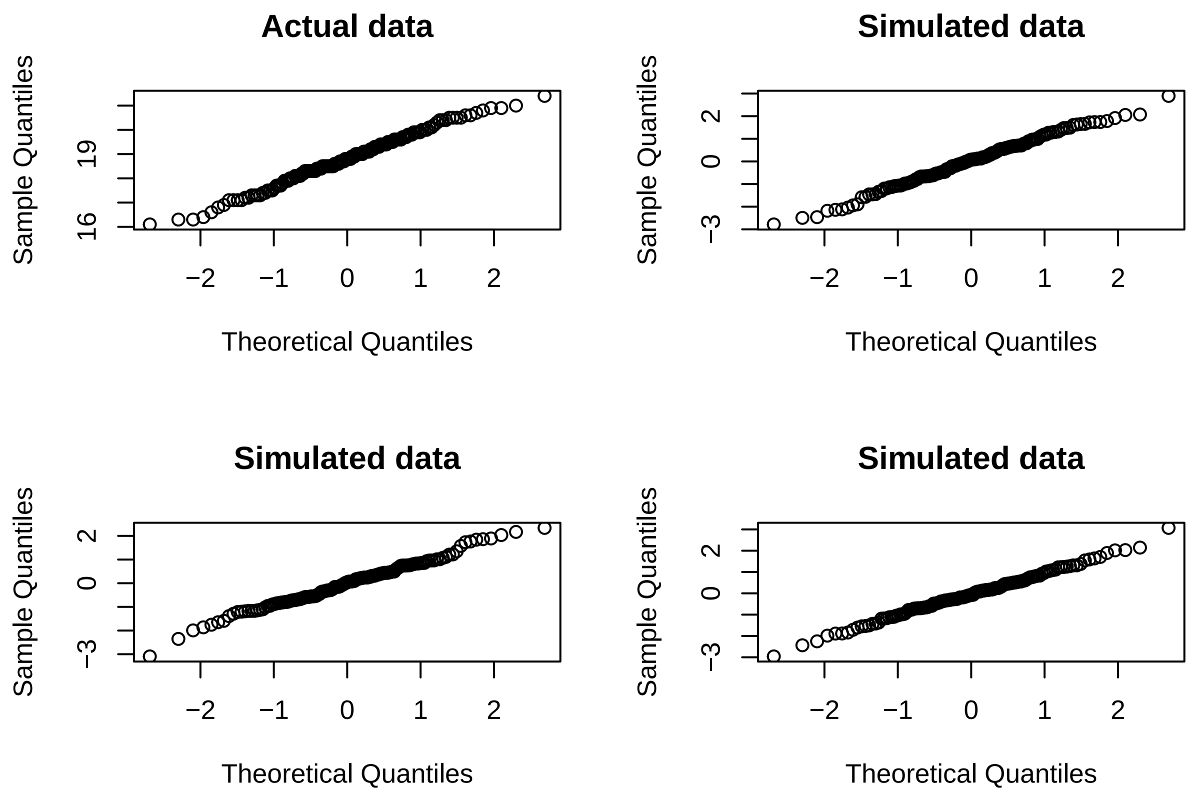
We reproduce the example of Trosset 7.5 here with minimal annotation. See Trosset for commentary.

```
# Read in data
forearms = scan("http://mypage.iu.edu/~mtrosset/StatInfeR/Data/forearms.dat")
# Create three side-by-side plots
par(mfrow=c(1,3))
# To switch back to a single plot:
# par(mfrow=c(1,1))
# Draw boxplot, QQ plot, density plot
boxplot(forearms, main="Box plot")
qqnorm(forearms)
plot(density(forearms), main="PDF estimate")
```



```
# Write a function to calculate IQR/SD
iqrsd = function(x){
  x.mean = mean(x)
  x.var = mean(x^2) - x.mean^2
  q = as.vector(quantile(x, probs=c(.25, .75)))
  x.iqr = q[2] - q[1]
  return(x.iqr / sqrt(x.var))
}
# Compare QQ plot of data to QQ plots of simulated data
```

```r
par(mfrow=c(2,2))
qqnorm(forearms, main="Actual data")
n = length(forearms)
qqnorm(rnorm(n), main="Simulated data")
qqnorm(rnorm(n), main="Simulated data")
qqnorm(rnorm(n), main="Simulated data")
```



```r
# IQR/SD of data
iqrsd(forearms)
```

```
## [1] 1.343526
```

```r
# IQR/SD of ten simulated data set
replicate(10, iqrsd(rnorm(n)))
```

```
##  [1] 1.628660 1.213643 1.476569 1.185520 1.359488 1.235385 1.440083
##  [8] 1.335868 1.381225 1.417593
```