Modelling in Physical Geography

Excercise 2

Konsta Happonen

Plotting

When doing analysis, it is crucial to get to know one's data. The fastest way to do this is "eyeballing", or visual inspection. Graphical representations of the data can be used to quickly gain information about the variation within and between variables. For this excercise we will be using one of the R default datasets called `brambles`, which contains the locations and age classes of raspberry stems in a square. For more information about the dataset, type `help(brambles)` on the command line.

```r
brambles <- boot::brambles # brambles is provided by the boot package.
str(brambles)
```

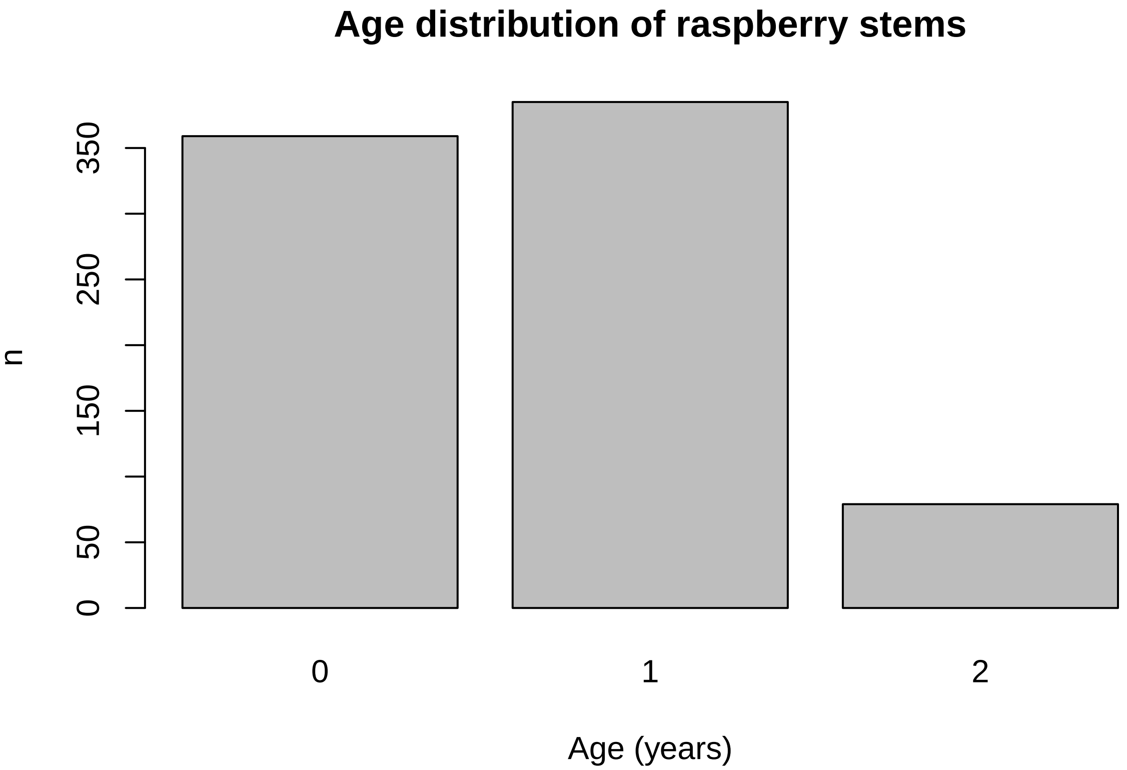
```
## 'data.frame':    823 obs. of  3 variables:
##  $ x  : num  0.677 0.676 0.681 0.683 0.776 0.794 0.944 0.948 0.983 0.986 ...
##  $ y  : num  0.001 0.022 0.031 0.038 0.028 0.033 0.011 0.01 0.077 0.084 ...
##  $ age: num  0 0 0 0 0 0 0 0 0 0 ...
```

One discrete variable

Barplots

To see the distribution of one discrete variable, we can use bar plots. We can treat the `brambles` age as a discrete variable, and plot it. The (horizontal) x-axis of the resulting plot lists all the unique ages in our data, and the (vertical) y-axis represents their counts. The arguments `main`, `xlab` and `ylab` control the plot title and axis labels. Use them to make your graph more informative.

```r
ages <- table(brambles$age) # Summarize the frequency of each age class
barplot(ages,
        main = "Age distribution of raspberry stems",
        xlab = "Age (years)",
        ylab = "n") # Plot them.
```



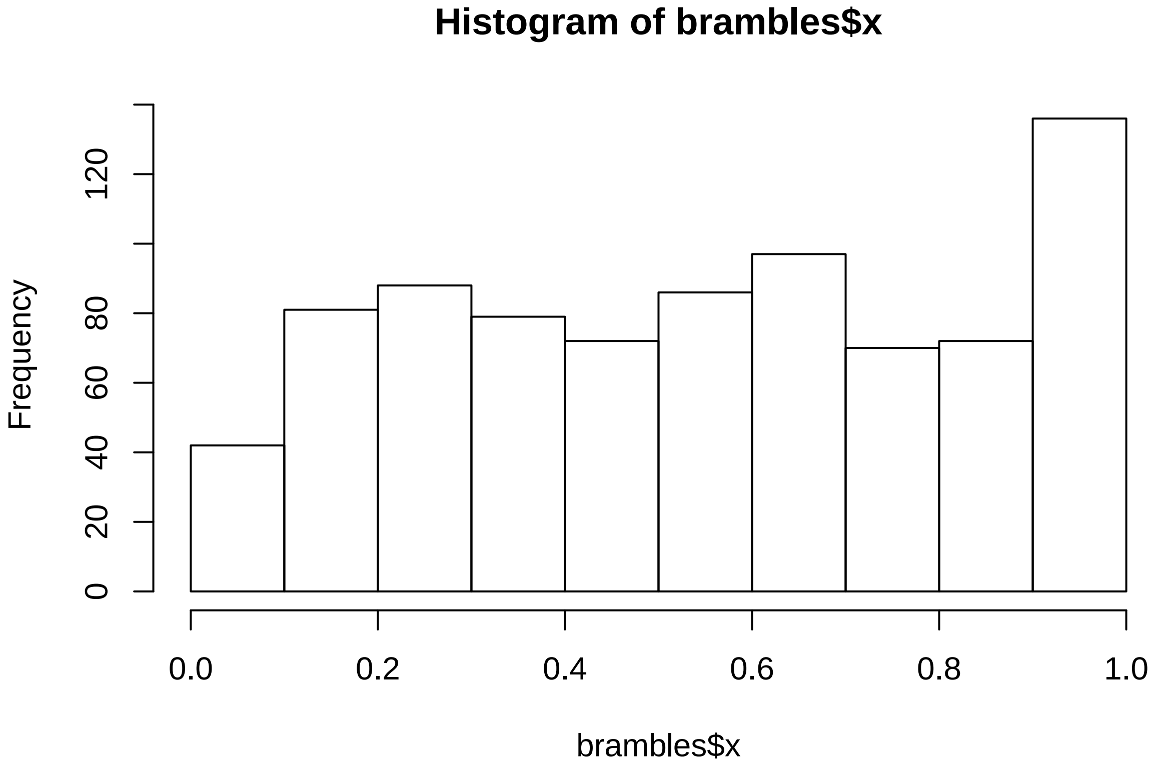
One continuous variable

The distribution of a continuous variable is most easily visualized by histograms and boxplots.

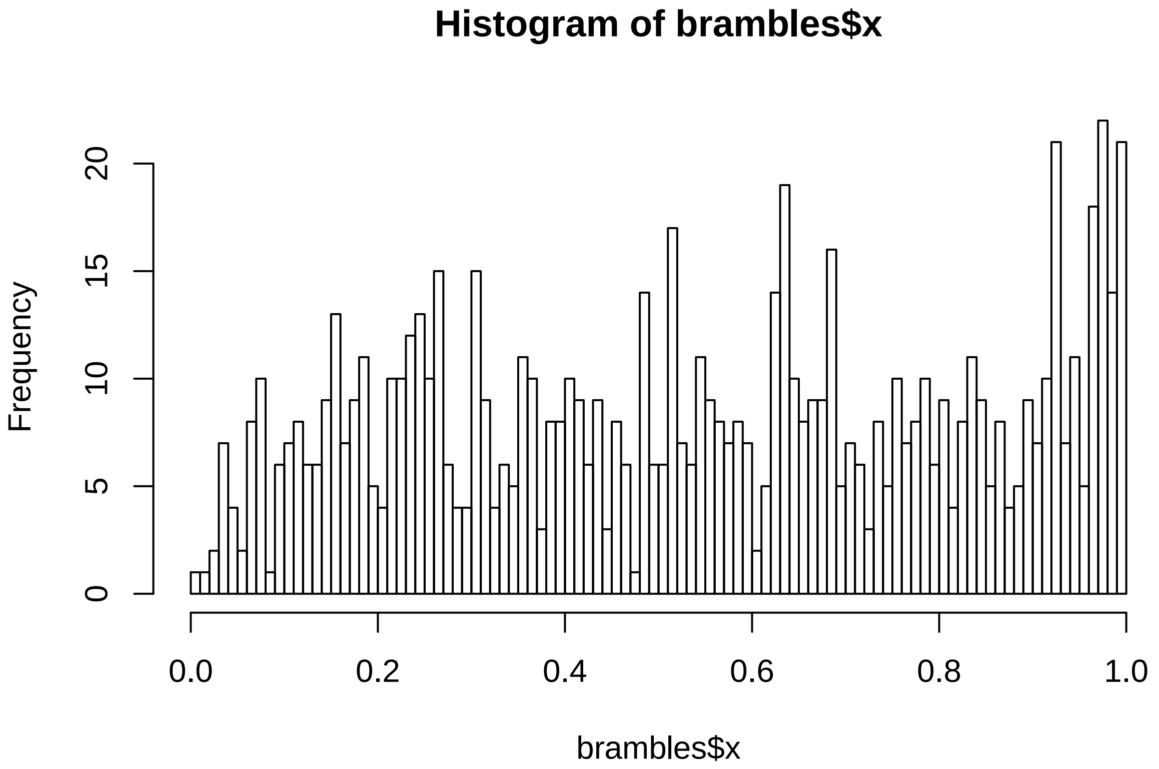
Histograms

A histogram discretizes the continuous variable to discrete classes, and then represents their frequencies with rectangles. The area of each rectangle is directly proportional to the frequency of that class. The argument `breaks` can be used to control the coarseness of the discretisation.

```
hist(brambles$x) # plot the distribution of raspberries along the x-axis
```



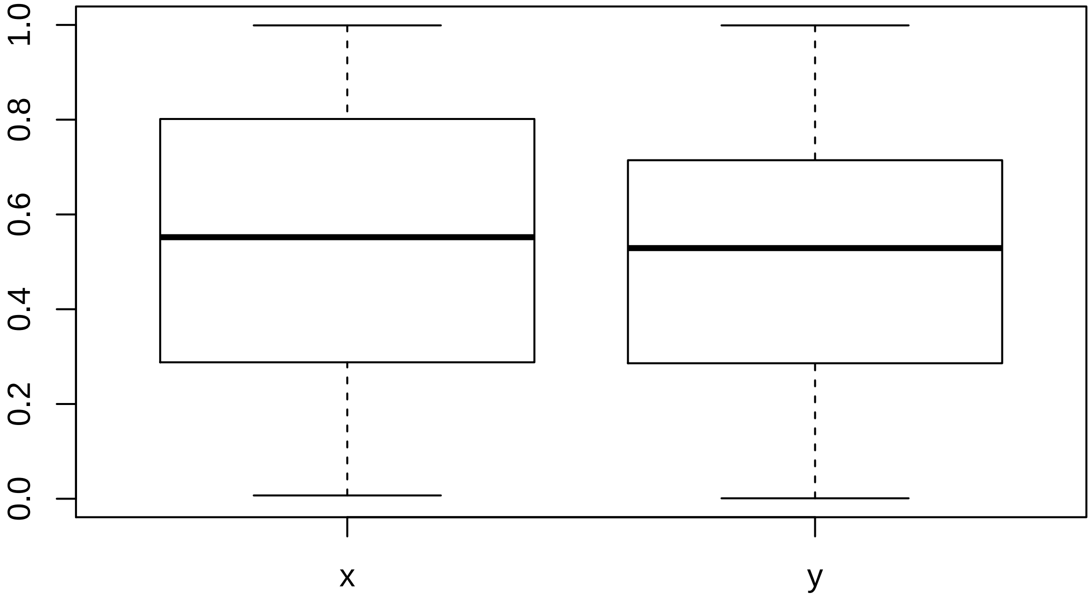
```r
hist(brambles$x, breaks=100)
```



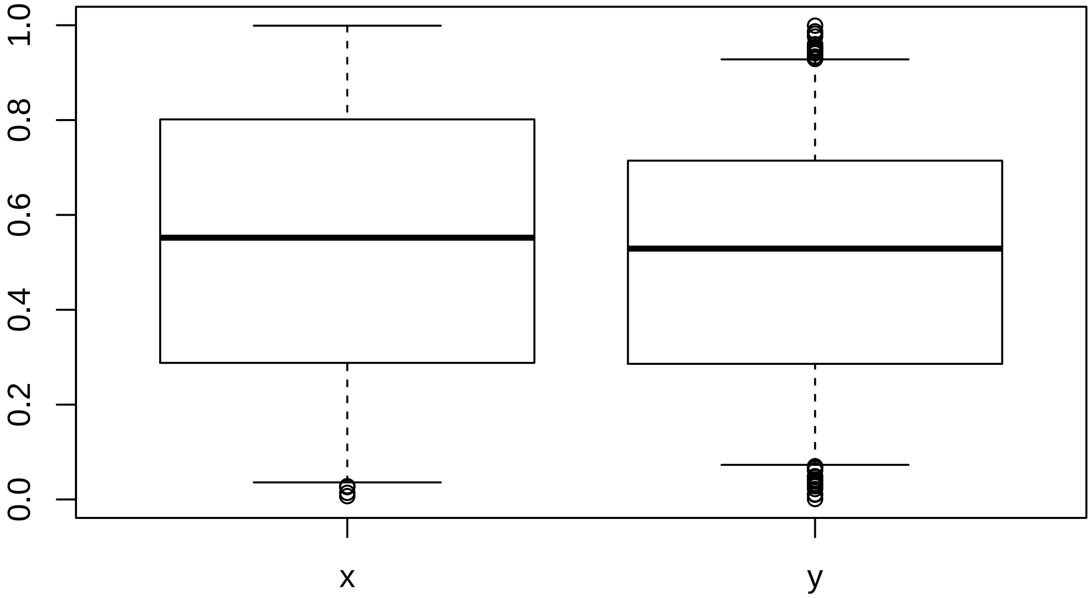
Boxplots

A box plot visualizes the *quartiles* of a set of numbers. Quartiles (Q1,Q2,Q3) are calculated by first arranging the values according to their value. Q2, also called the *median*, is the middle number between the minimum and maximum values. Accordingly, Q1 is the middle number between the minimum and Q2, and Q3 is the middle number between Q2 and the maximum. The boxplot displays the minimum, Q1, Q2,Q3, and the maximum. In addition, any data points more than 1.5 interquartile ranges (distance between Q1 and Q3) away from the nearest quantile are displayed as separate points. The definition of outliers can be tweaked with the argument `range`. Boxplots are especially useful for identifying outliers (abnormally extreme values, often indicative of some kind of error in data collection) and comparing multiple variables on the same scale.

```
boxplot(brambles[,c("x","y")])
```



```
boxplot(brambles[,c("x","y")], range=0.5)
```



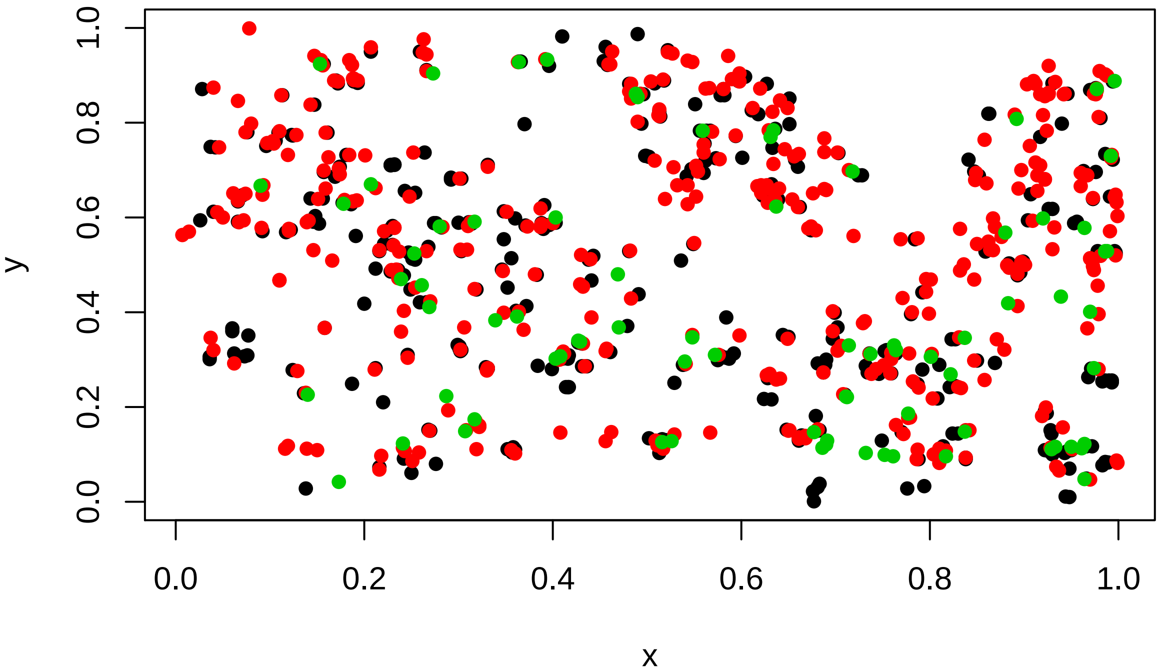
Two continuous variables

Often what is truly interesting in the relationship between our variables. To visualise these we can use scatterplots and lineplots

Scatterplots

In scatterplots, the values of two continuous values are mapped on the x- and y-axes. Each data point is represented a point in the plot. If the two variables are a “response” and an “explanatory” variable, the convention is to map the response on the y-axis. The argument `pch` can be used to change the point symbol, and the argument `col` to change the color of the points.

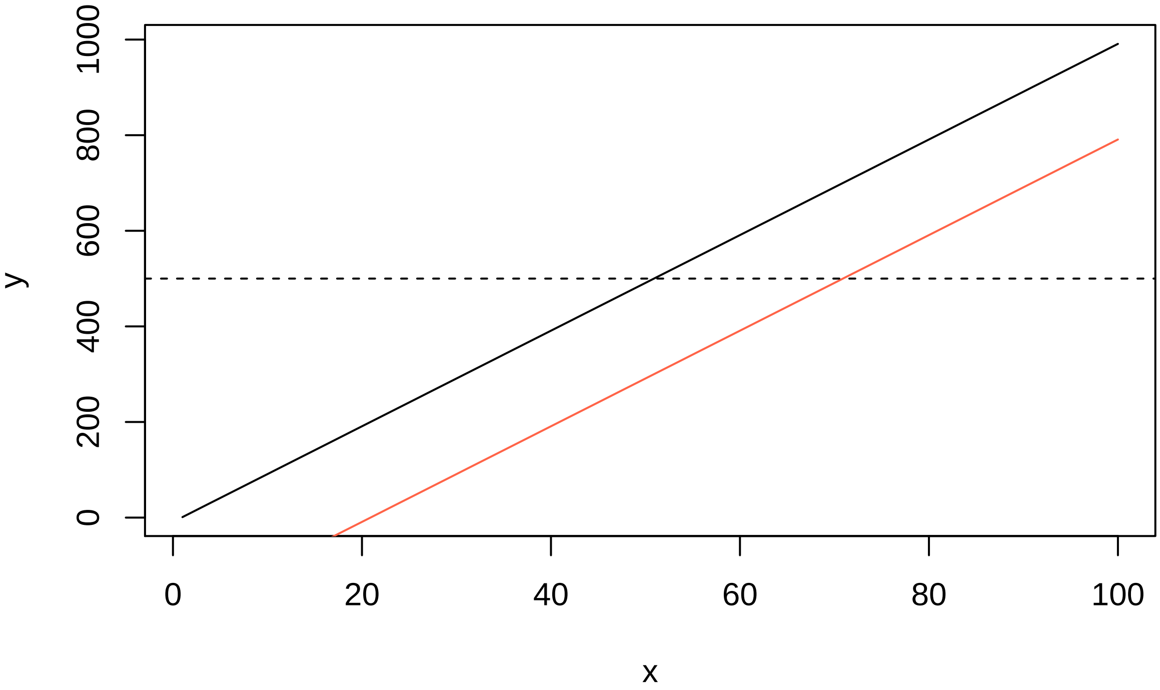
```r
## Plot the spatia distribution of data with filled circles, colored by age
plot(y~x, data = brambles, col = factor(brambles$age), pch = 16)
```



Lineplots

Lineplots are good for displaying trends. The `brambles` data is ill-suited for dempnstrating this, so we will create some trends. We will use the command `seq()` to create regular sequences of data. We specify `type = "l"` in the ploting command to create a line plot. We can then use `lines()` to add another line in the plot. We can also use `abline()` to create

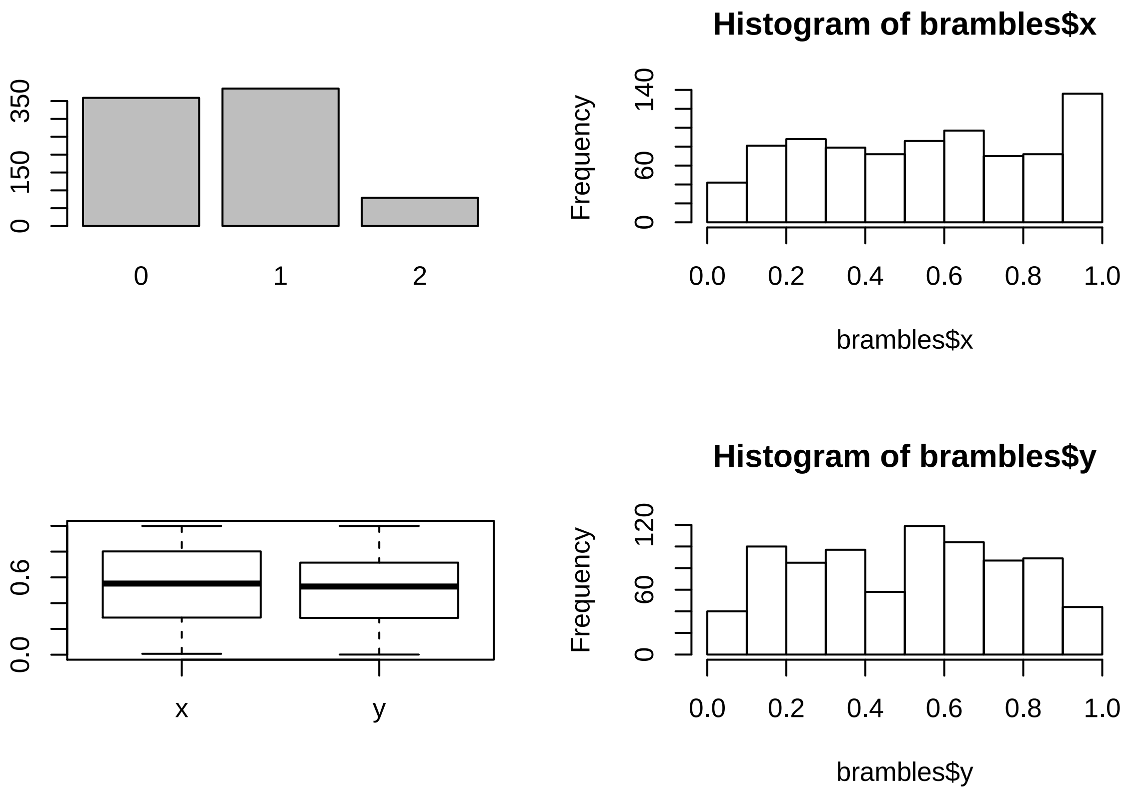
```r
x <- seq(from = 1, to = 100) # equal to 1:100
y <- seq(from = 1, to = 1000, by = 10)
plot(y~x, type = "l") # create a line plot
lines(y-200~x, col = "tomato") # add a new line
abline(h = 500, lty = 2) # add a horizontal line with a new linetype
```



Multiple plots in the same window

You can modify the options of the graphics window to display multiple plots at the same time. You do this with the `par(mfrow = c(rows,columns))` command, replacin `rows` and `columns` with suitable integers. To revert your options, close the graphics window or run the command `par(mfrow = c(1,1))`.

```
par(mfrow = c(2,2))
barplot(ages)
hist(brambles$x)
boxplot(brambles[,c("x","y")])
hist(brambles$y)
```



```r
par(mfrow = c(1,1))
```

Saving plots

To export your plots from R, use the commands `png()` to save as raster graphics, and `pdf()` to save as vector graphics. Prefer vector graphics, as they generally take less space and retain their resolution.

The `png()` takes as its arguments the width and height of the image in pixels. The default size is quite small, so do adjust.

```r
png(filename = "enigmatic_expression.png", width = 700, height = 700)
plot(x = c(2,8), y = c(7,7), xlim = c(0,10), ylim = c(0,10), col = "blue")
lines(c(6,4,3,4,6)~c(1,3,5,7,9), col = "red")
dev.off()
```

```
## X11cairo
##        2
```

The `pdf()` command's width and height arguments are in inches by default.

```r
pdf(file = "vital_organ.pdf", width = 7, height = 7)
plot(x = c(1:9),
     y = c(7,8.5,9,8,7,8,9,8.5,7),
     xlim = c(0,10),
     col = "red",
     type = "l",
     ylim = c(1,10))
lines(x = c(1,5,9), y = c(7,1,7) , col = "red")
dev.off()
```

```
## X11cairo
##        2
```

Important summary statistics

We have already been introduced to means, medians, and quartiles. We will now take a closer look of **quantiles** and **standard deviations**.

Quantiles

Quantiles are the break-points when dividing a continuous-valued sample that has been ordered by its value into frequency classes. Quartiles are a special case of quantiles, as are deciles (each class comprises 10% of the samples) and percentiles. Quantiles are calculated with the function `quantile()`. The argument `probs` defines the quantiles you want as fractions (percetage / 100%). For example, quantile(x, probs = 0.1) gets you the value that is bigger than 10% of x. Quantiles are important in statistical significance testing, as we shall later see.

```
our_sample <- (1:100)^2 # generate some data
our_sample
```

```
##   [1]     1     4     9    16    25    36    49    64    81   100   121
##  [12]   144   169   196   225   256   289   324   361   400   441   484
##  [23]   529   576   625   676   729   784   841   900   961  1024  1089
##  [34]  1156  1225  1296  1369  1444  1521  1600  1681  1764  1849  1936
##  [45]  2025  2116  2209  2304  2401  2500  2601  2704  2809  2916  3025
##  [56]  3136  3249  3364  3481  3600  3721  3844  3969  4096  4225  4356
##  [67]  4489  4624  4761  4900  5041  5184  5329  5476  5625  5776  5929
##  [78]  6084  6241  6400  6561  6724  6889  7056  7225  7396  7569  7744
##  [89]  7921  8100  8281  8464  8649  8836  9025  9216  9409  9604  9801
## [100] 10000
```

```
mean(our_sample)
```

```
## [1] 3383.5
```

```
median(our_sample)
```

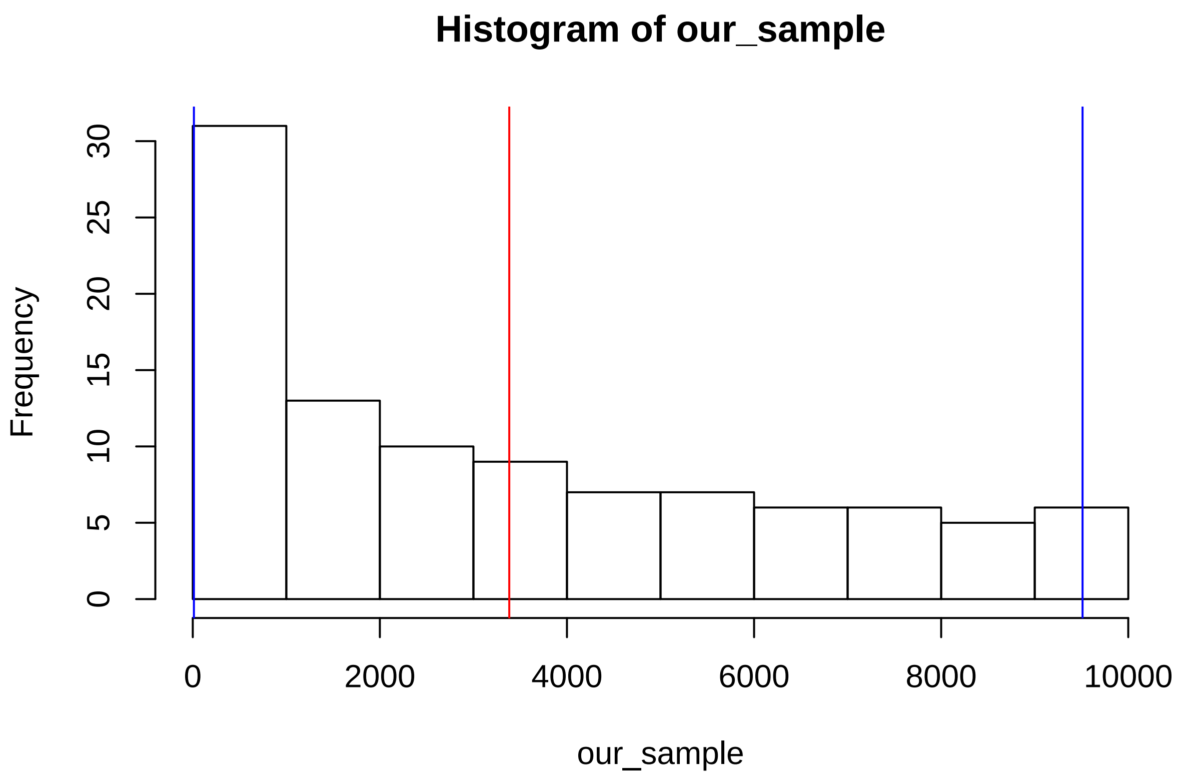
```
## [1] 2550.5
```

```
quantile(our_sample)# the default is to calculate minimum, maximum and quartiles.
```

```
##       0%      25%      50%      75%     100%
##     1.00   663.25  2550.50  5662.75 10000.00
## this range contains 95 % of our sample.
interval <- quantile(our_sample, p = c(0.025,0.975))
interval
```

```
##     2.5%    97.5%
##   12.325 9511.375
```

```
hist(our_sample)
abline(v=mean(our_sample), col = "red") # plot the mean with red
abline(v=interval, col = "blue") # plot the quartiles in blue
```



Standard deviation

Standard deviation is a measure of spread for continuous variables. Sample standard deviation is calculated by dividing the sum of squared deviations from the sample mean by `N-1` and taking the square root (`sqrt()`) of that. Or then by letting R do it for you with the function `sd()`.

```
sqrt(sum(((our_sample-mean(our_sample))^2))/(length(our_sample)-1))
```

```
## [1] 3024.356
```

```
sd(our_sample)
```

```
## [1] 3024.356
```

Generating data

It is surprising how often one needs to generate data. When producing random data, they are sampled from some *distribution*. In this excercise we will cover the uniform and normal distributions.

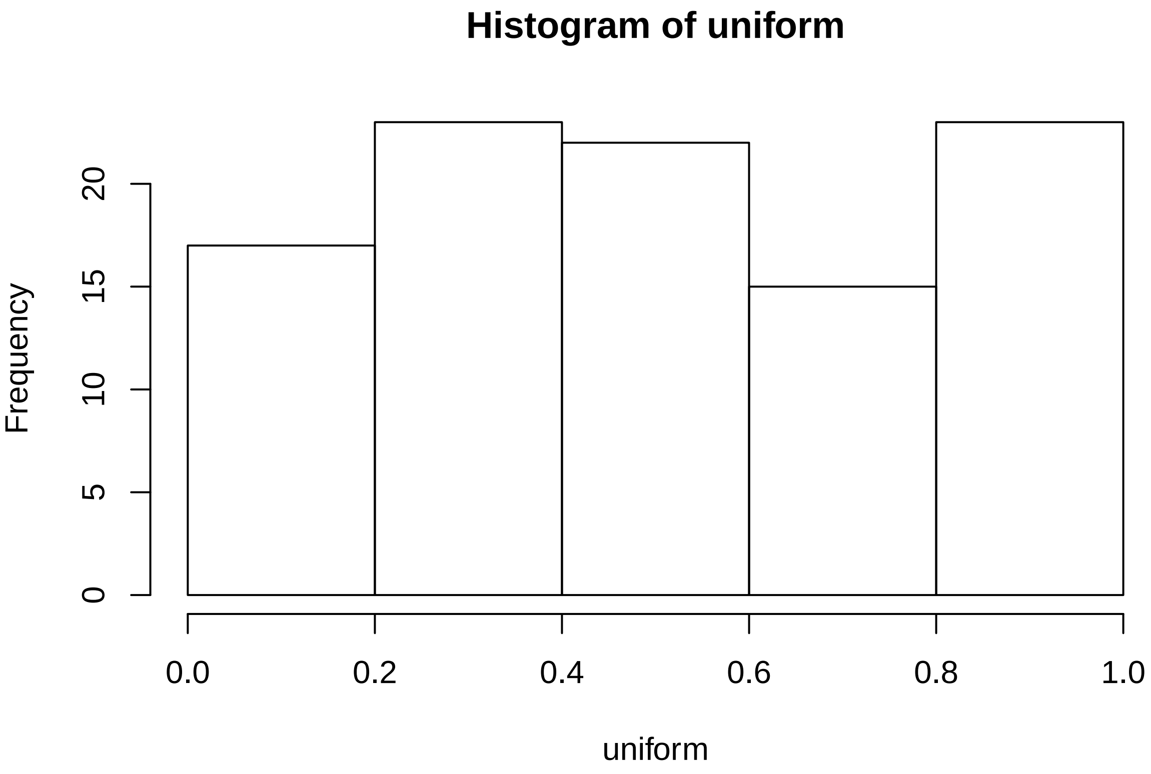
Uniform distribution

The uniform distribution is specified with only two parameters: the **minimum** and the **maximum**. As the name implies, the probability distribution between these two points is uniform; all values have exactly equal probability of being represented in the sample. Random samples from a uniform distribution are produced with the `runif()` command.

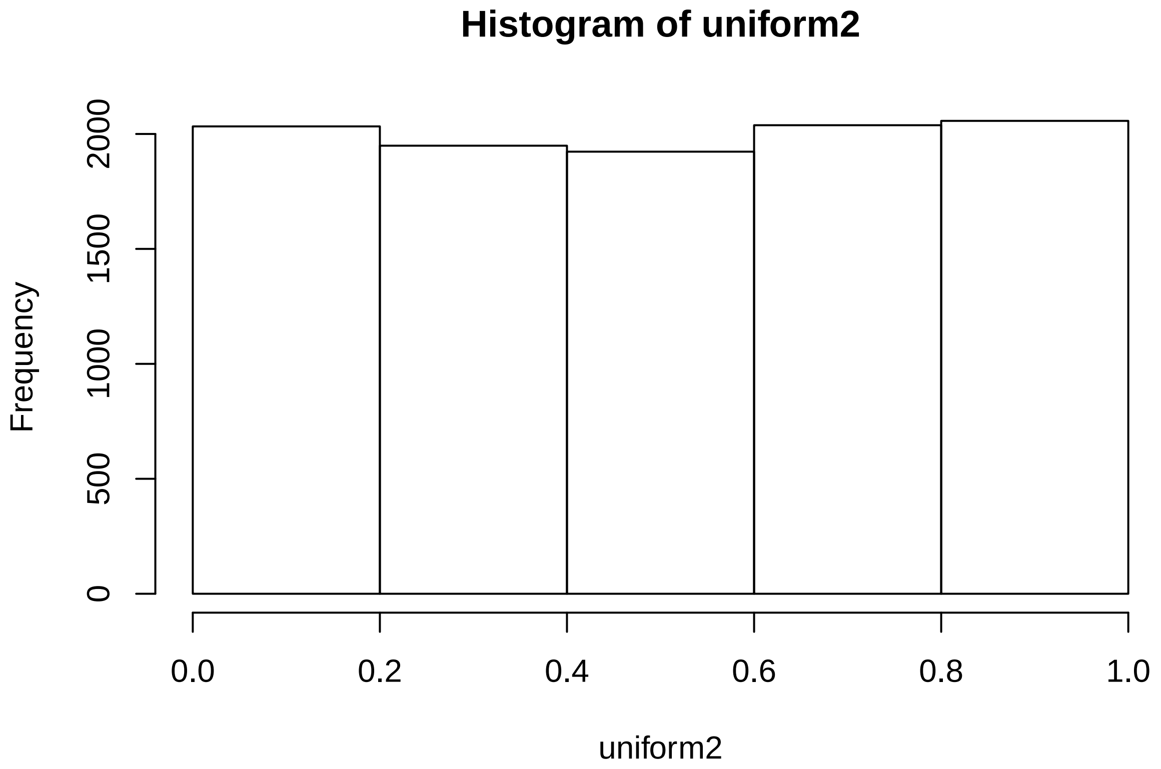
```
## 100 draws from a uniform(0,1) distribution.
uniform <- runif(n = 100, min = 0, max = 1)

## 10000 draws from a uniform(0,1) distribution.
uniform2 <- runif(n = 10000, min = 0, max = 1)

## The sample distribution is not exactly uniform because of sampling error
hist(uniform, breaks = 5)
```



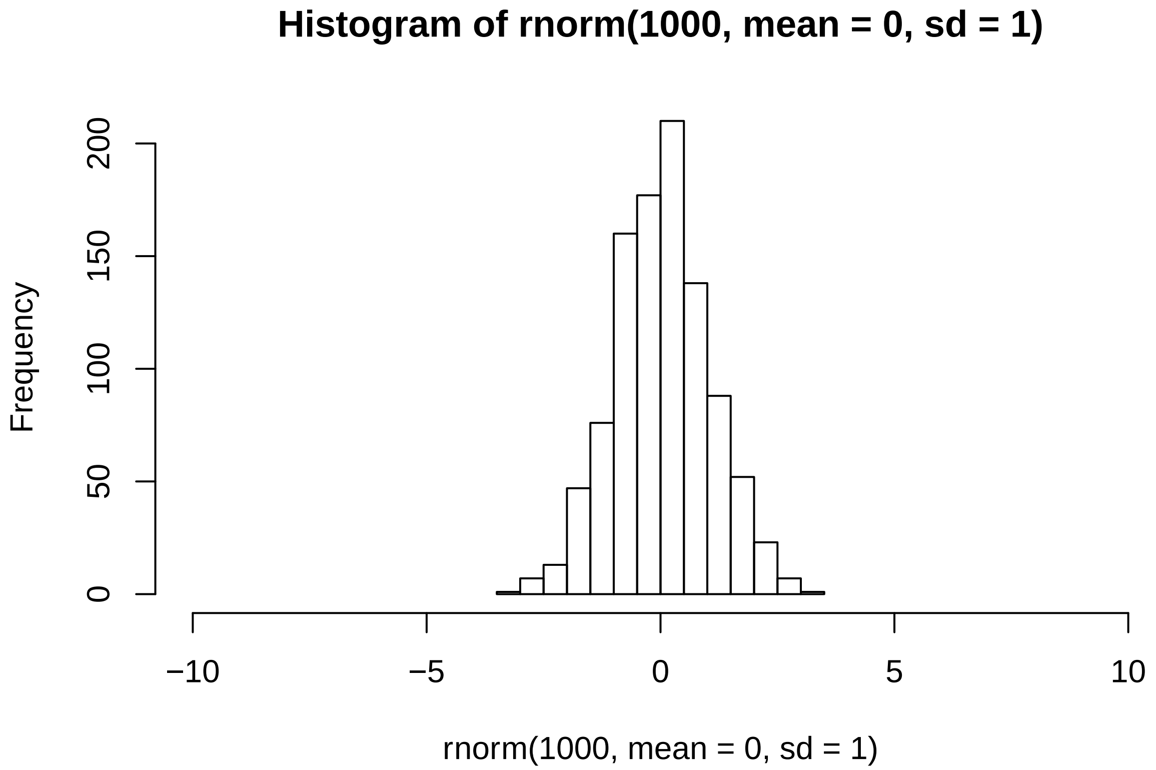
```
hist(uniform2, breaks = 5) # much closer
```



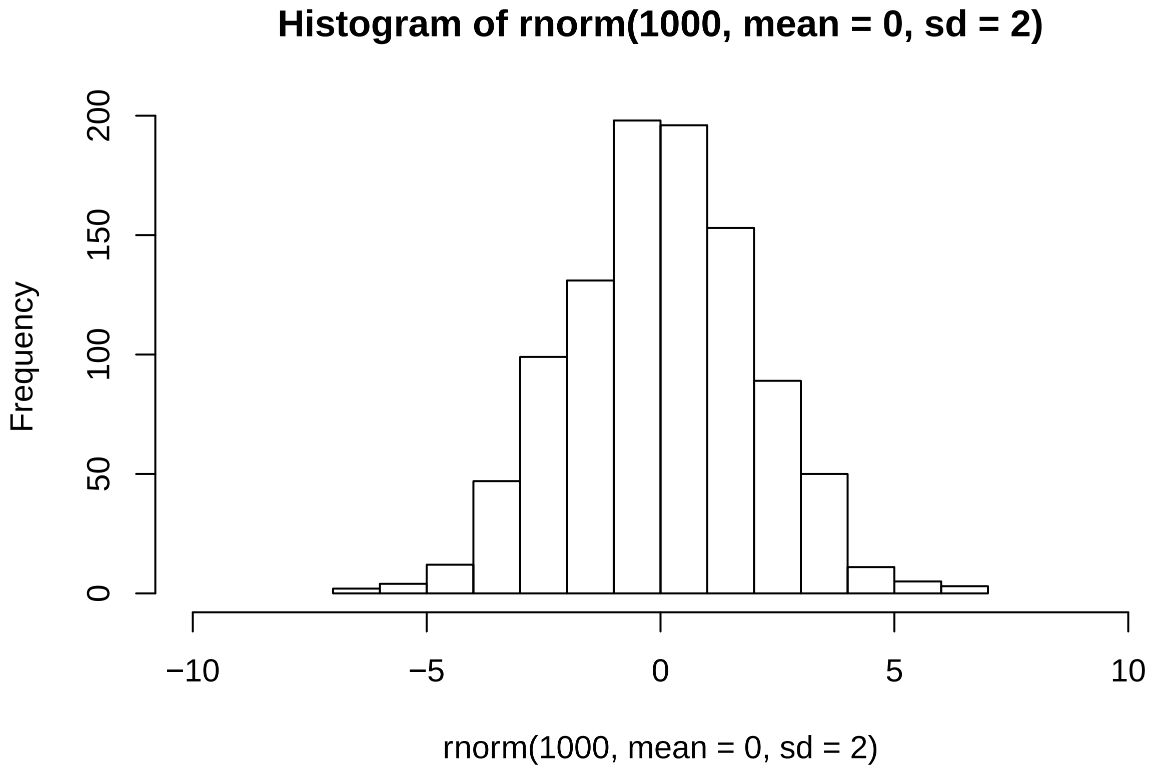
Normal distribution

The normal distribution is one of the most important distributions there is. It's probability distribution is the famous Gauss' bell-shaped curve. The normal distribution is parametrized with a mean and standard deviation parameter. The standard deviation is a *scale parameter*, which defines the variability of sampled values. Random normal deviates are produced with the command `rnorm()`. The normal distribution is especially important because of a thing called *the central limit theorem* (CLT). The CLT states that the means of samples taken from a distribution will follow a normal distribution almost irrespective of the shape of the original distribution. CLT is an important part of statistical significance testing, as we shall later see.

```
hist(rnorm(1000, mean = 0, sd = 1), xlim = c(-10,10)) # 1000 normal deviates, sd = 1
```



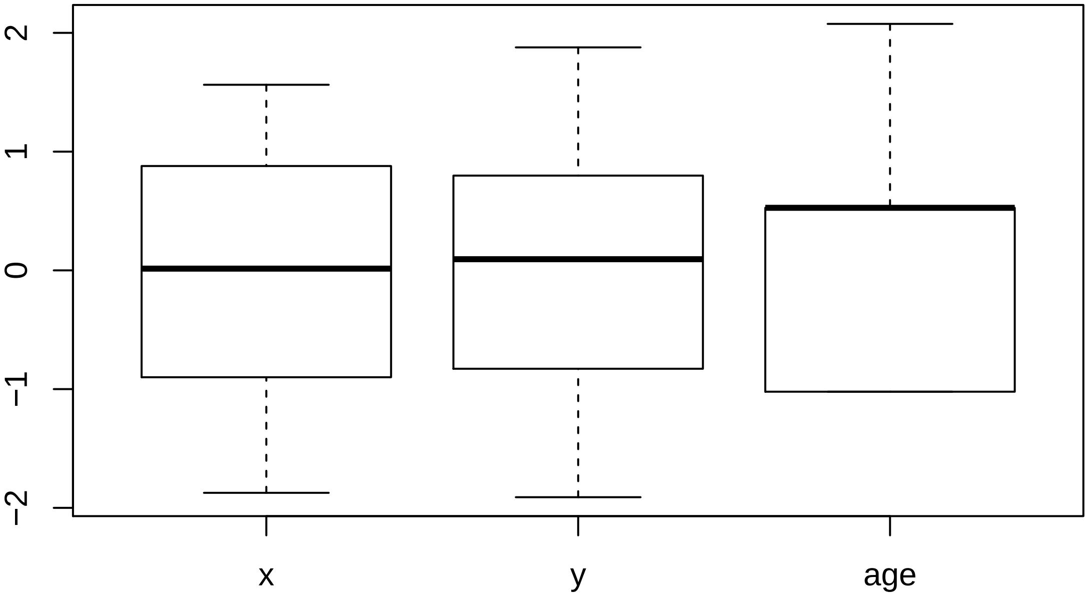
```
hist(rnorm(1000, mean = 0, sd = 2), xlim = c(-10,10)) # same, but sd 2. The distribution is wider
```



Scaling variables

Comparing the distribution of two variables is hard if they are not on the same scale. In this case it can help if you *normalize* or *scale* the variables. Scaling is done by substracting from each member of a sample the sample mean, and then dividing them by the sample standard deviation. The resulting variable will have zero mean and a standard deviation of one. Comparing variables on different scales thus becomes possible. Scaling can be done with the command `scale()`. Be aware, that because standard deviations are sensitive to outliers, scaling is as well. Scaling variables can be important in regression analysis, where it helps in the comparison and calculation of standardized effect sizes.

```
scaled_b <- scale(brambles) # Scale all variables in brambles
boxplot(scaled_b) # compare the variables on a common scale
```



Sampling

You can also randomly sample any object you have created in R with the command `sample()`. The main arguments are `x`, the sampled object, `size`, the sample size, `prob`, a vector of weights if some values are to be drawn more often than others. In addition, the argument `replace` defines whether to sample with or without replacement. Next, We will go through the differences of these two sampling methods.

Sampling without replacement

Sampling without replacement is the analogue of picking a piece of candy from a bag and eating it: you cannot eat (sample) the same candy twice. Thus when sampling without replacement, the maximum sample size is the number of candies (or observations or whatever).

```
sample(1:10, size = 5, replace = FALSE) # sample without replacement
```

```
## [1] 4 5 6 9 7
## sample(1:10, size = 11, replace = FALSE) # error: not enough samples
```

Sampling with replacement

Sampling with replacement on the other hand is comparable to picking a candy from the bag, inspecting it, putting it back again and reshuffling before taking another candy. Because the samples are replaced, there is no upper limit to the sample size. This also means that the same observation could be present in the sample more than once

```
sample(1:10, size = 5, replace = TRUE)
```

```
## [1]  7 10  7  2  6
```

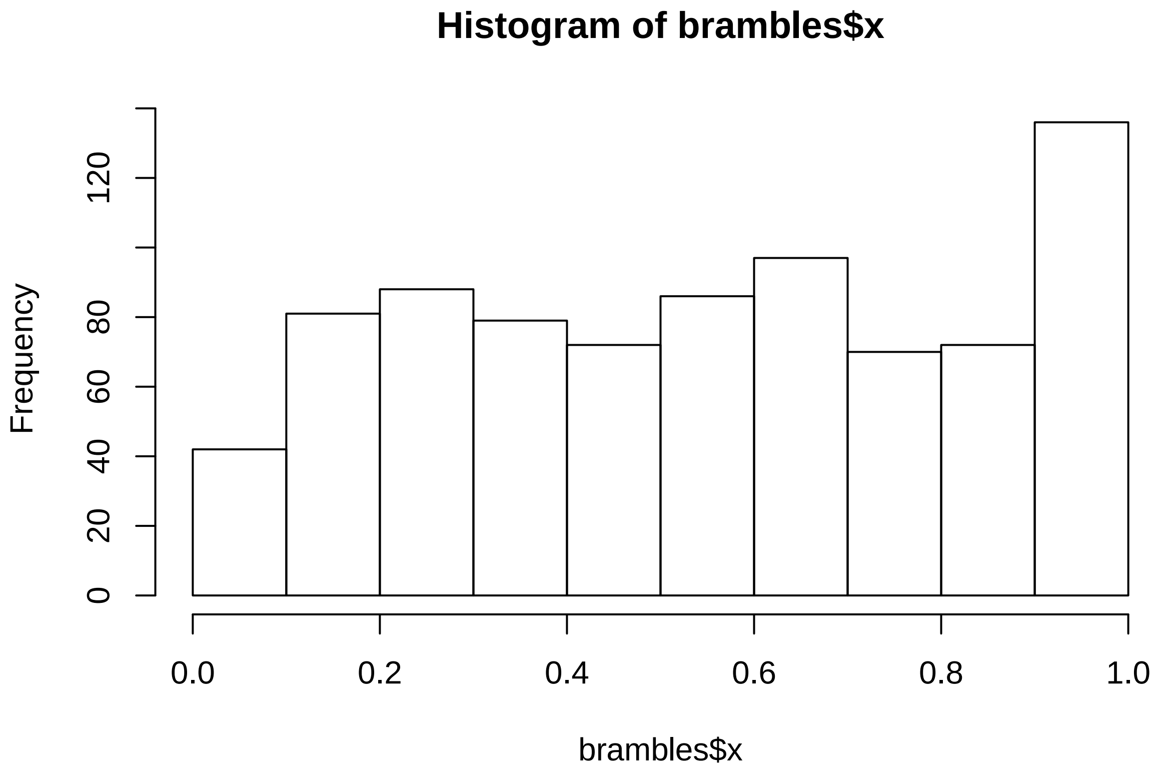
```
sample(1:10, size = 11, replace = TRUE)
```

```
##  [1]  8  9  5  9  1  6  3 10  7  5  1
```

Bootstrap confidence intervals

Let us look again at the spatial distribution of our raspberry stems. We want to know if there is a spatial trend in their distribution. More specifically, we want to know if the raspberries are evenly distributed from the left to the right, or if there are more raspberries on either side. For this we might investigate the mean of the x-coordinate of our raspberries. Since the possible range is 0–1, the value of the mean should be 0.5 if the raspberries are divided evenly on each side.

```
hist(brambles$x)
```



```r
mean(brambles$x)
```

```
## [1] 0.5478202
```

The histogram and mean show slightly more raspberries on the right. But is this evidence of a spatial trend? If by trend we mean that the process that generates raspberries has a tendency of generating them more on the right than on the left then no. Samples from a continuous distribution will **never** have exactly the same mean as the distribution they were drawn from (or more specifically, will have the same mean with infinitely samll probability). This is because of sampling error: a finite sample cannot represent the underlying distribution accurately. Small samples will, on average, differ more from the generating distribution. As sample size approaches infinity, sampling error tends to zero.

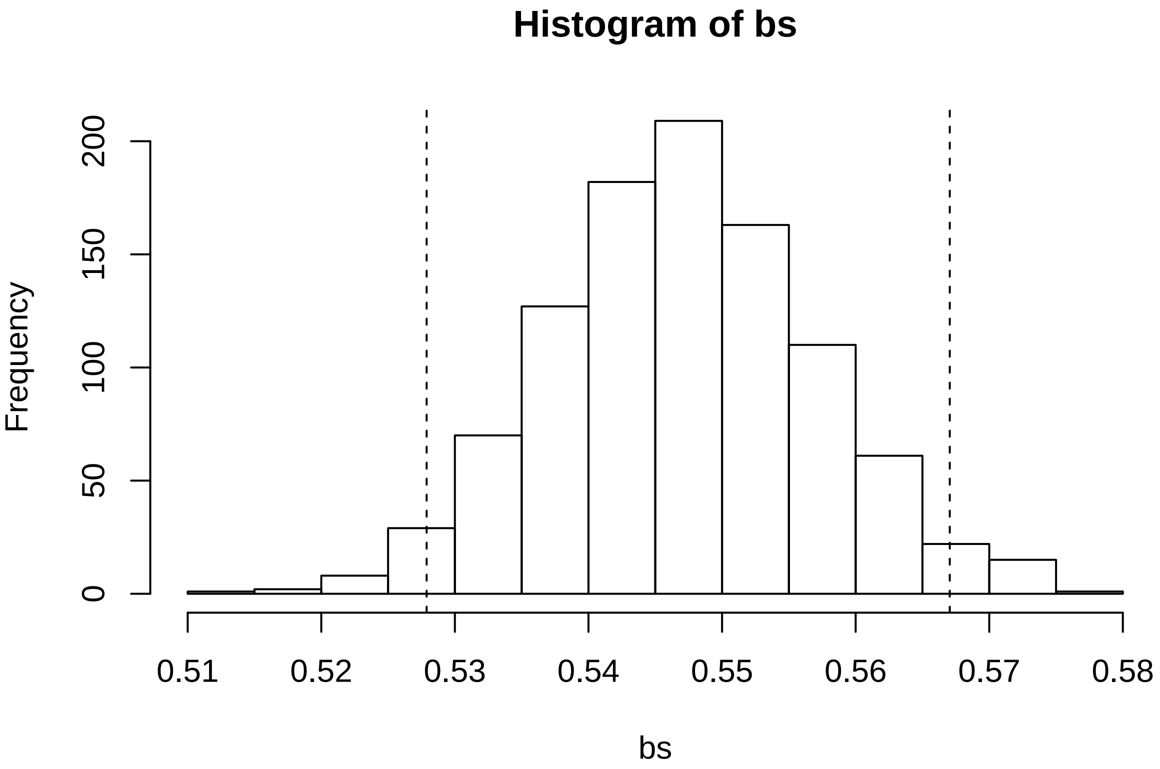
If we think our sample standard deviation accurately represents the standard deviation of the underlying distribution, and that raspberries are distributed identically and individually in our square, we can use bootstrapping to calculate the sampling distribution of the mean.

Bootstrapping is effectively just sampling with replacement, but doing it many times. For this we can use the R command `replicate()`. The argument `n` specifies how many times we want to replicate the command, and the argument `expr` specifies the command to be replicated. The argument `simplify` tells the command in what form we want the return values from our replicated command. We sample with replacement from the x-coordinates 999 times, and calculate the means of those samples. For each replicate, we set the sample size to the original sample size. We concatenate these samples to the original mean for a total of 1000 samples.

```r
## Bootstapping the mean
## default sample size for sample() is the original sample size.
bs <- replicate(n = 999, expr = mean(sample(brambles$x,replace = TRUE)))
bs <- c(bs, mean(brambles$x))
```

`bs` now contains the bootstrapped *sampling distribution of the mean*. We can now take quantiles of this distribution to produce *bootstrapped confidence intervals* for the mean x-coordinate of our raspberries. For example, if we want to calculate 95% confidence intervals, we exclude the 2.5% most extreme values from both ends.

```r
confs <- quantile(bs, probs = c(0.025,0.975))
hist(bs)
abline(v = confs, lty = 2)
```



The true mean of the distribution is, with 95% certainty, located somewhere between the dashed lines. The value 0.5 (our *null hypothesis*) is not contained in this interval. We can thus conclude (with 95% certainty) that raspberries have a tendency of being on the right.