# Transcript of Mick Crawley's R course 2010
# Imperial College London, Silwood Park

Emanuel G Heitlinger

## Second and third day

### R as a calculator

Modulo and interger quotients

```
> 119%%13
```

```
[1] 2
```

```
> 119%/%13
```

```
[1] 9
```

Rounding down and rounding up to next integer

```
> floor(5.7)
```

```
[1] 5
```

```
> ceiling(5.7)
```

```
[1] 6
```

Creating a own function for rounding

```
> my.round <- function(x) floor(x + 0.5)
> my.round(5.7)
```

```
[1] 6
```

```
> my.round(5.4)
```

```
[1] 5
```

Generating repeats and repeats of levels

```
> rep(5.3, 17)

 [1] 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3 5.3

> gl(6, 3, 18)

 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6
Levels: 1 2 3 4 5 6

> rep(1:6, 1:6)

 [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6

> rep(c("monoecious", "dioecious", "hermaphrodite", "agamic"),
+     c(3, 2, 7, 3))

 [1] "monoecious"    "monoecious"    "monoecious"    "dioecious"
 [5] "dioecious"     "hermaphrodite" "hermaphrodite" "hermaphrodite"
 [9] "hermaphrodite" "hermaphrodite" "hermaphrodite" "hermaphrodite"
[13] "agamic"        "agamic"        "agamic"
```

## Vectors in R

Making a function for variance and see how it compares with the builtin function using .Internal (C-code). Thats not in the course program but I am a bit further here.

```
> y <- c(7, 5, 7, 2, 4, 6, 1, 6, 1, 6, 2, 6)
> my.var <- function(y) sum((y - mean(y))^2)/(length(y) - 1)
> my.own.var <- function(y) sum((y - sum(y)/length(y))^2)/(length(y) -
+     1)
> system.time(my.own.var(rep(y, 1e+06)))

   user  system elapsed
  0.646   0.173   0.819

> system.time(my.var(rep(y, 1e+06)))

   user  system elapsed
  0.545   0.180   0.725

> system.time(var(rep(y, 1e+06)))

   user  system elapsed
  0.340   0.076   0.416

> my.var(y) == my.own.var(y)

[1] TRUE

> my.var(y) == var(y)

[1] FALSE
```

They are the same only to machine precision.

## Sorting, ordering, etc...

The different functions available: rank, sort and order

```
> houses <- read.table("houses.txt", header = TRUE)
> attach(houses)
> ranks <- rank(Price)
> sorted <- sort(Price)
> ordered <- order(Price)
> view <- data.frame(Price, ranks, sorted, ordered)
> view
```

```
   Price ranks sorted ordered
1    325  12.0     95        9
2    201  10.0    101        6
3    157   5.0    117       10
4    162   6.0    121       12
5    164   7.0    157        3
6    101   2.0    162        4
7    211  11.0    164        5
8    188   8.5    188        8
9     95   1.0    188       11
10   117   3.0    201        2
11   188   8.5    211        7
12   121   4.0    325        1
```

```
> Location[order(Price)]
```

```
 [1] Reading     Staines     Winkfield   Newbury     Bracknell   Camberley
 [7] Bagshot     Maidenhead  Warfield    Sunninghill Windsor     Ascot
12 Levels: Ascot Bagshot Bracknell Camberley Maidenhead Newbury ... Winkfield
```

```
> Location[rev(order(Price))]
```

```
 [1] Ascot       Windsor     Sunninghill Warfield    Maidenhead  Bagshot
 [7] Camberley   Bracknell   Newbury     Winkfield   Staines     Reading
12 Levels: Ascot Bagshot Bracknell Camberley Maidenhead Newbury ... Winkfield
```

Order is the most useful as it returns the indexes of the original vector in an order that would sort the original vector.

```
> worms <- read.table("worms.txt", header = TRUE)
> attach(worms)
> by(worms, Vegetation, mean)[1:3]
```

```
$Arable
  Field.Name         Area        Slope   Vegetation      Soil.pH         Damp
          NA     3.866667     1.333333           NA     4.833333     0.000000
Worm.density
    5.333333

$Grassland
  Field.Name         Area        Slope   Vegetation      Soil.pH         Damp
          NA    2.9111111    3.6666667           NA    4.1000000    0.1111111
```

```
Worm.density
   2.4444444

$Meadow
  Field.Name         Area        Slope   Vegetation      Soil.pH         Damp
          NA     3.466667     1.666667           NA     4.933333     1.000000
Worm.density
    6.333333
```

By can also be used in modells, supplieing a user defined function to by.

```
> by(worms, Vegetation, function(x) lm(Worm.density ~ Soil.pH,
+     data = x))[1:3]

$Arable

Call:
lm(formula = Worm.density ~ Soil.pH, data = x)

Coefficients:
(Intercept)      Soil.pH
     -9.689        3.108


$Grassland

Call:
lm(formula = Worm.density ~ Soil.pH, data = x)

Coefficients:
(Intercept)      Soil.pH
    -15.041        4.265


$Meadow

Call:
lm(formula = Worm.density ~ Soil.pH, data = x)

Coefficients:
(Intercept)      Soil.pH
         31           -5
```

Selecting at random using the function sample can be useful in bootstrapping.

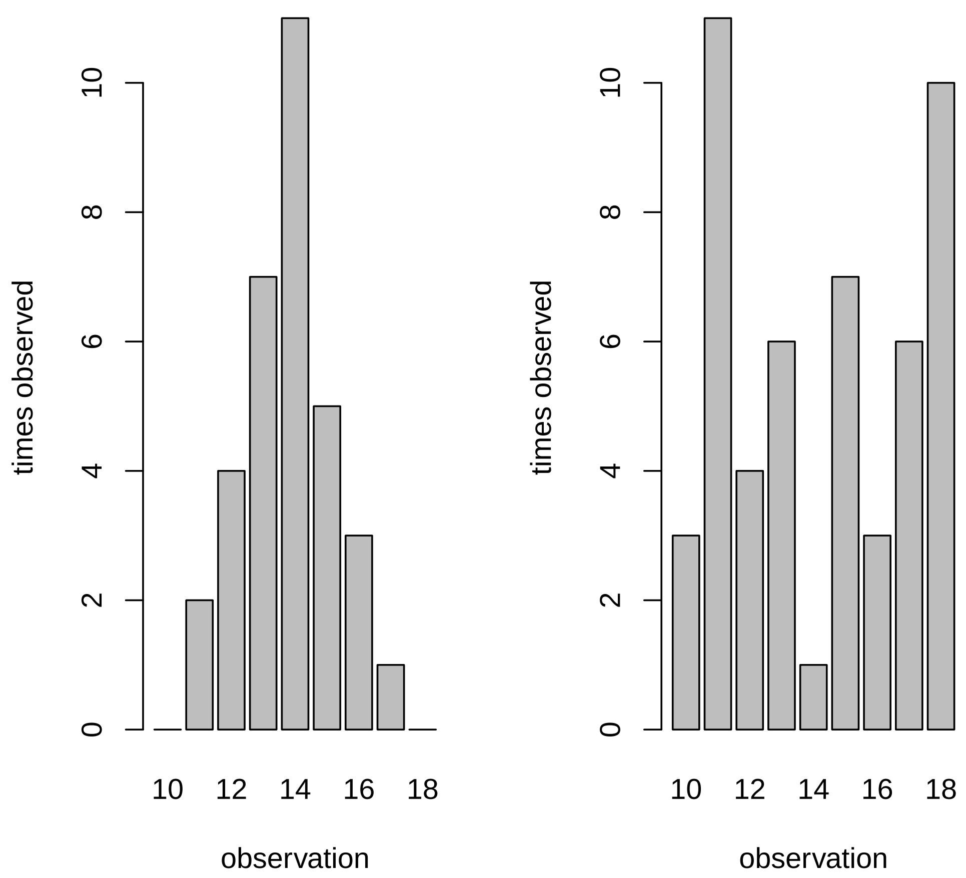
```
> worms[sample(1:20, 5), ]

       Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
15     Pond.Field  4.1     0     Meadow     5.0  TRUE            6
3   Nursery.Field  2.8     3  Grassland     4.3 FALSE            2
11    Garden.Wood  2.9    10      Scrub     5.2 FALSE            8
6        Oak.Mead  3.1     2  Grassland     3.9 FALSE            2
17      Cheapside  2.2     8      Scrub     4.7  TRUE            4
```

## Measures of central tendency

Distributions can be bimodal:

```
> dist <- read.table("mode.txt", header = TRUE)
> par(mfrow = c(1, 2))
> barplot(dist$fx, names = as.character(dist$x), ylab = "times observed",
+     xlab = "observation")
> barplot(dist$fy, names = as.character(dist$x), ylab = "times observed",
+     xlab = "observation")
> y <- rep(dist$x, dist$fx)
```



### The median

Write a function for the median and compare it to the original.

```
> my.median <- function(y) {
+     if (length(y%%2 == 0)) {
+         mean(y[length(y)/2], y[(length(y)/2) + 1])
+     }
+     else y[ceiling(length(y)/2)]
+ }
> system.time(my.median(rep(y, 1e+06)))

   user  system elapsed
  2.014   0.359   2.373

> system.time(median(rep(y, 1e+06)))

   user  system elapsed
  1.075   0.397   1.472

> my.median(y) == median(y)

[1] TRUE
```

### The geometric mean

```
> my.gm <- function(y) prod(y)^(1/length(y))
> my.good.gm <- function(y) exp(sum(log(y))/length(y))
> system.time(my.gm(rep(y, 1e+06)))

   user  system elapsed
  0.723   0.178   0.900

> system.time(my.good.gm(rep(y, 1e+06)))

   user  system elapsed
  2.739   0.445   3.186

> my.gm(y) == my.good.gm(y)

[1] FALSE
```

Surprisingly the first way was faster, anyways the second way is better because the huge product in the first way (my.gm) can be a challenge to machine precision.

### The harmonic mean

For ratios like speed.

```
> my.ham <- function(x) 1/mean(1/x)
> my.ham(c(1, 2, 4, 1))
```

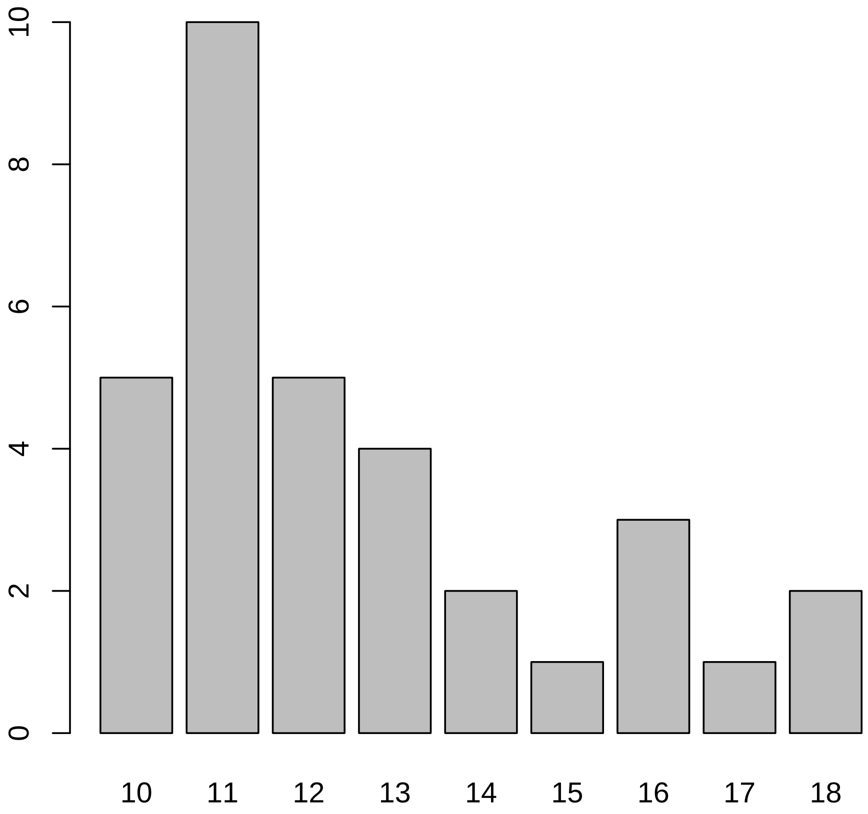
```
[1] 1.454545
```

### Summary for measures of central tendency

Back to our distribution example, plus some fooling around with cat and paste to present it nicely.

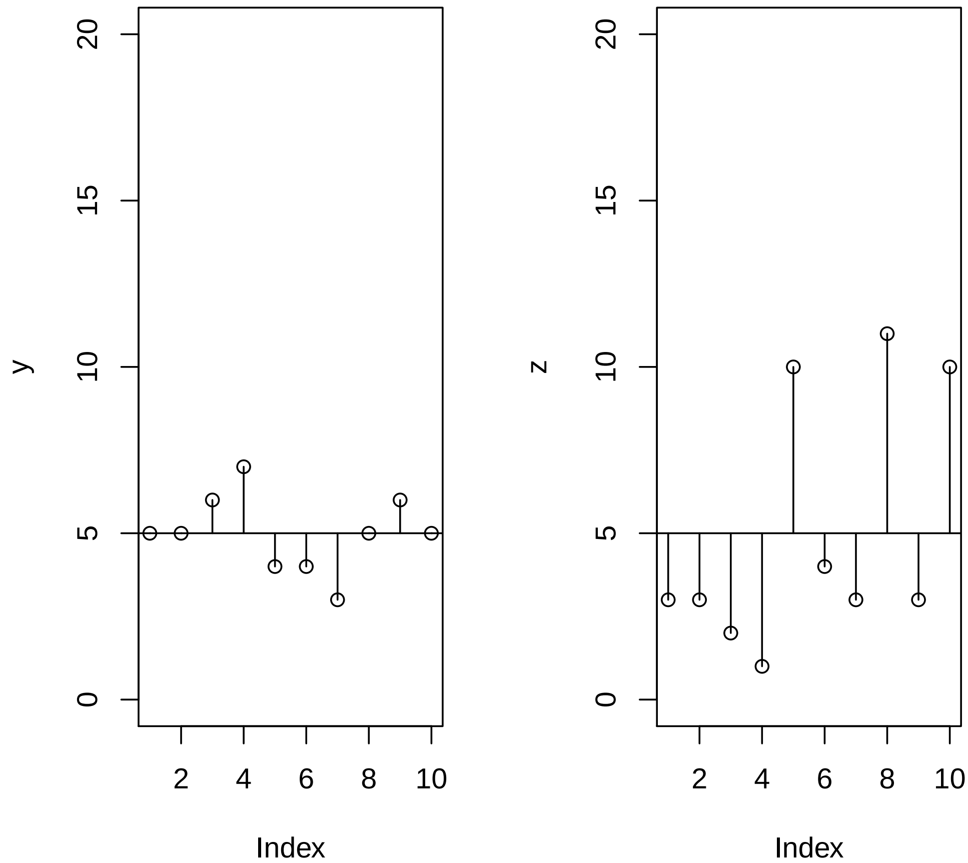
```
> par(mfrow = c(1, 1))
> barplot(dist$fz, names = as.character(dist$x))
> w <- rep(dist$x, dist$fz)
> cat(paste("Arithmetic mean = ", mean(w)), (paste("Geometric mean = ",
+     my.good.gm(w))), (paste("Harmonic mean = ", my.ham(w))),
+     (paste("Median = ", median(w))), sep = "\n")
```

```
Arithmetic mean =  12.6060606060606
Geometric mean =  12.4038006797634
Harmonic mean =  12.2203740784901
Median =  12
```


0
2
4
6
8
10
10
11
12
13
14
15
16
17
18

## Measures of variation

```
> y <- c(5, 5, 6, 7, 4, 4, 3, 5, 6, 5)
> z <- c(3, 3, 2, 1, 10, 4, 3, 11, 3, 10)
> par(mfrow = c(1, 2))
> plot(y, ylim = c(0, 20))
> abline(mean(y), 0)
> join <- function(i) lines(c(i, i), c(y[i], mean(y)))
> bin <- sapply(1:length(y), join)
> plot(z, ylim = c(0, 20))
> abline(mean(z), 0)
> join <- function(i) lines(c(i, i), c(z[i], mean(y)))
> bin <- sapply(1:length(z), join)
```




**The sum of squares!**

$\sum (y - \bar{y})^2$

To make this a general measure it is devided by the degrees of freedom, we get: **The variance!**

$s^2 = \dfrac{\sum (y - \bar{y})^2}{n - 1}$

## Using variance

### Measuring unrelieability

One such measure is **the standard error of the mean**: $SE_{\bar{y}} = \sqrt{\frac{s^2}{n}}$

Best practice for scientific papers is to use: The mean of y was 5 ± 0.37 (1 s.e., n = 10).

Wow, look at the source code! That's really best of the best practice using Sweave's in line Sexpressions...

### Estimiating confidence

The function qt() holds the values associated with certain probabilities and degrees of freedom for student's t (similarly qnorm for the normal distributi on).

Now the confidence intervals are given as follows:

$CI_{95\%} = t_{(\alpha=0.025,d.f.10)}\sqrt{\frac{s^2}{n}}$

That means you give the confidence interval as follows in a paper: The mean of y was 5 ± 0.83 (95% C.I., n = 10). But better use the standard error!

## Robust estimators

### Quantiles

Create an object holding 10000 normally distributed values. Look at the Quantiles, a non-parametric estimate comparable to the mean. The second argument vector gives here 2.5% and 97.5% of the values.

```
> quantile(rnorm(1000), c(0.025, 0.975))

     2.5%     97.5%
-1.969259  1.872692

> quantile(rnorm(10000), c(0.025, 0.975))

     2.5%     97.5%
-1.965511  1.953520
```

A sample of 10000 produces a better estimate of the tails of a distribution. We know that the Quantiles of the standard normal distribution should be ±1.96. See:

```
> qnorm(c(0.025, 0.975))
```

```
[1] -1.959964  1.959964
```

**MAD**

The variance has the same problem as the arithmetic mean, it is extremely sensible to outliers. There is also the **Median Absolute Deviation (MAD)**, which is a more robust estimator (comparable to the median for the central tendency).

```
> goo <- c(3, 4, 6, 4, 5, 2, 4, 5, 1, 5, 4, 6)
> c(mean = mean(goo), sd = sd(goo))
```

```
    mean       sd
4.083333 1.505042
```

```
> c(median = median(goo), mad = mad(goo))
```

```
median    mad
4.0000 1.4826
```

So long no outlier, mean and median are close, so are standard deviation and mad.

```
> goo1 <- c(goo, 100)
> c(mean = mean(goo1), sd = sd(goo1))
```

```
    mean       sd
11.46154 26.64149
```

```
> c(median = median(goo1), mad = mad(goo1))
```

```
median    mad
4.0000 1.4826
```

If we add a outliemer mean and standard deviation are strongly affected median and mad not.

We can use this to scan for outliers making an arbitrary decission of a four fold difference between sd and mad showing outliers:

```
> my.outlier <- function(x) {
+     if (sd(x) > 4 * mad(x))
+         print("Outliers present")
+     else print("Deviation reasonable")
+ }
> my.outlier(goo)
```

```
[1] "Deviation reasonable"
```

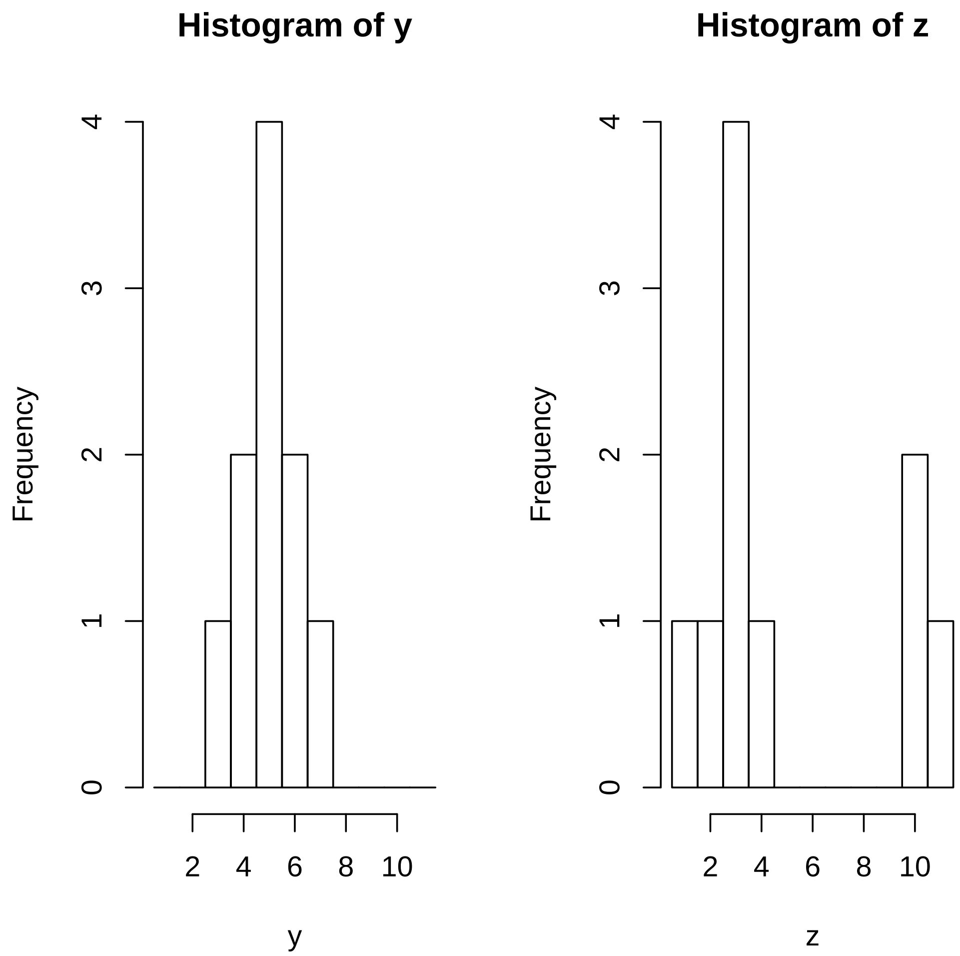
```
> my.outlier(goo1)
```

```
[1] "Outliers present"
```

## Comparing two samples

A nonparametric test is Wilcox rank sum test. Clearly advised for our data y and z from before:

```
> par(mfrow = c(1, 2))
> hist(y, breaks = c(0.5:11.5))
> hist(z, breaks = c(0.5:11.5))
```




We can do a rank sum test as follows “by hand”.

```
> combi <- c(y, z)
> sample <- c(rep("Y", length(y)), rep("Z", length(z)))
> rank.combi <- rank(combi)
> tapply(rank.combi, sample, sum)

  Y   Z
121  89
```

The values could now be compared in tables. But as we know how it works we can also now the builtin function.

```
> wilcox.test(y, z)
```

```
        Wilcoxon rank sum test with continuity correction

data:  y and z
W = 66, p-value = 0.2349
alternative hypothesis: true location shift is not equal to 0
```

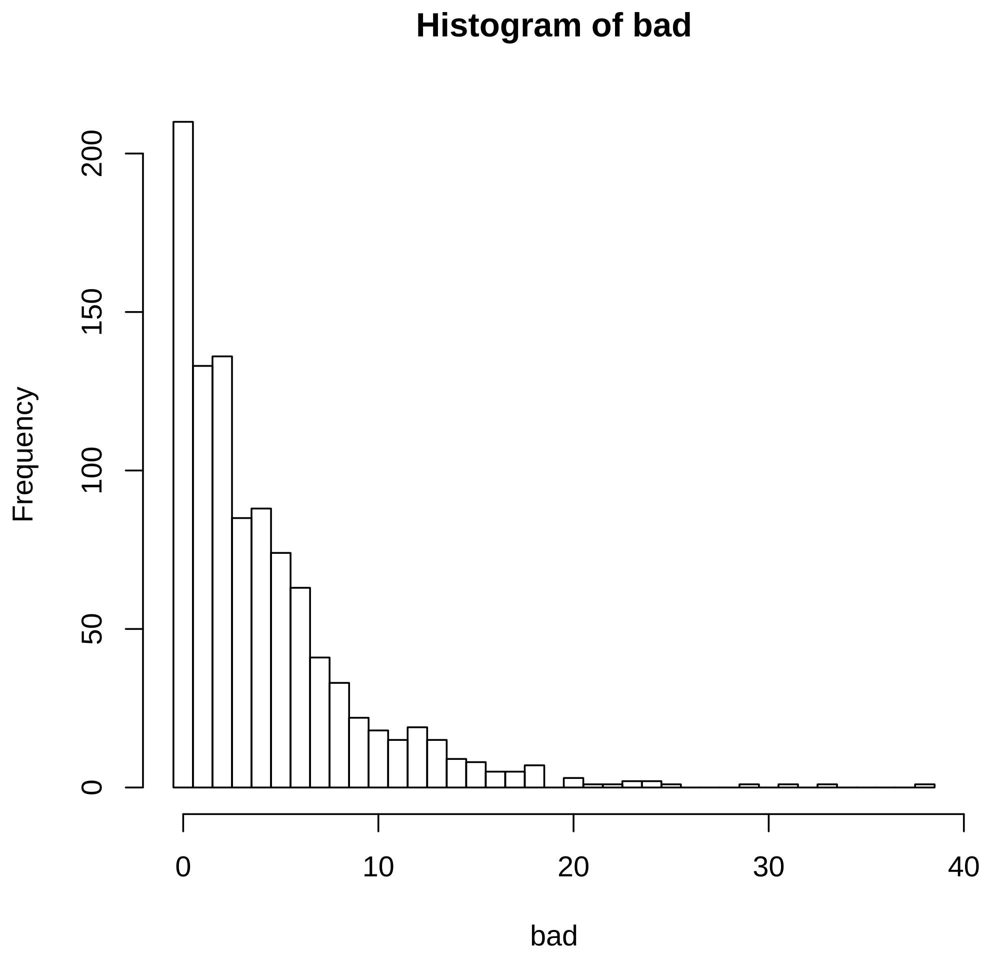
The p value is much bigger than any critical value so we accept the null hypothesis.

## Central limit theorem

**For a distribution with a finite variance the mean of a random sample from that distribution tends to be normally distributed**

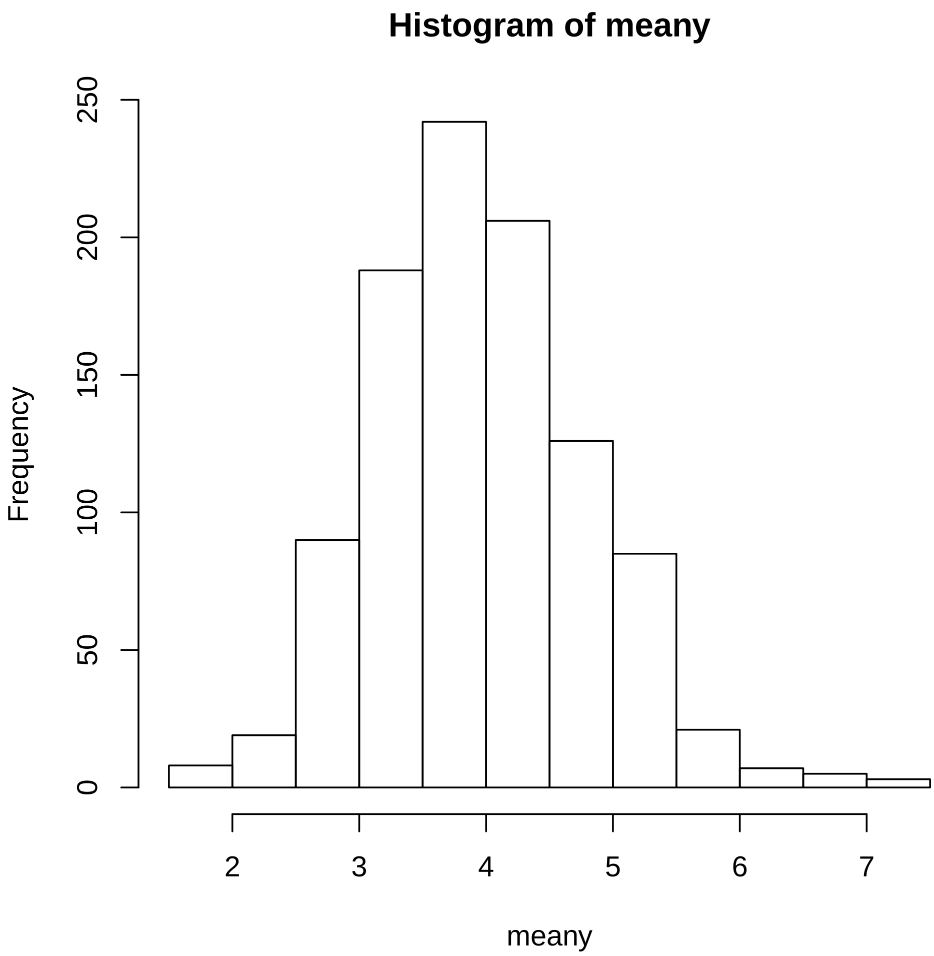
To demonstrate we generate random sample from a "badly behaved" negative binomial distribution.

```
> bad <- rnbinom(1000, 1, 0.2)
> hist(bad, breaks = -0.5:38.5)
```




Now we take 1000 different sets of 30 samples from the some distribution.

```
> meany <- numeric(1000)
> for (i in 1:1000) {
+     y <- rnbinom(30, 1, 0.2)
+     meany[i] <- mean(y)
+ }
> hist(meany)
```




Looks very normal!