Simple Statistics functions in R

Susan Holmes

In this session, we'll learn how to simulate data with R using random number generators and how to use some of the most useful statistical functions.

Getting started

When wanting to produce the same results with a random number generator it is important to set a starting point. This is important if you want to reproduce the results of a simulation or algorithm, and is very important in debugging.

```
setwd("~/RWork")
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Taking a subsample

```
x <- c(1.9, 4.0, 4.4, 7.2, 3.8, 8.3, 8.7, 5.4, 8.8)
sample(x,3)
```

```
## [1] 3.8 8.8 4.0
```

```
sample(x,3)
```

```
## [1] 7.2 5.4 4.0
```

```r
sample(x,3)
```

```
## [1] 4.0 3.8 5.4
```

Question Why doesn't this always give the same answer?

Randomly generate 3 numbers from 1 to 9:

```r
sample(9,3)
```

```
## [1] 2 6 9
```

```r
sample(9,3)
```

```
## [1] 3 1 9
```

Underneath the box: a RNG (random number generator)

```r
runif(3)
```

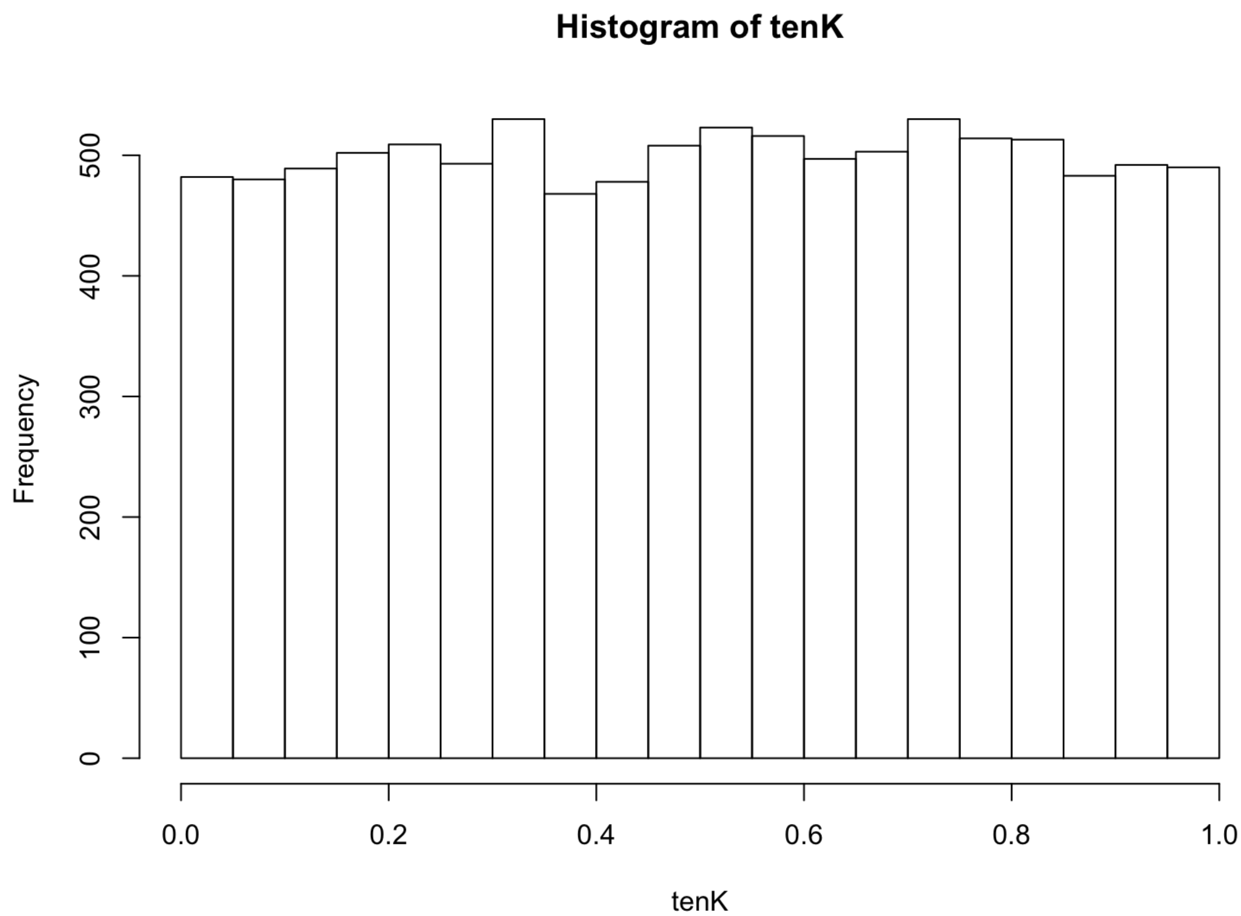
```
## [1] 0.2616396 0.8103531 0.8115878
```

```r
runif(3)
```

```
## [1] 0.5394167 0.7769816 0.5154705
```

```r
tenK<-runif(10000)
```

```r
hist(tenK)
```



Now try:

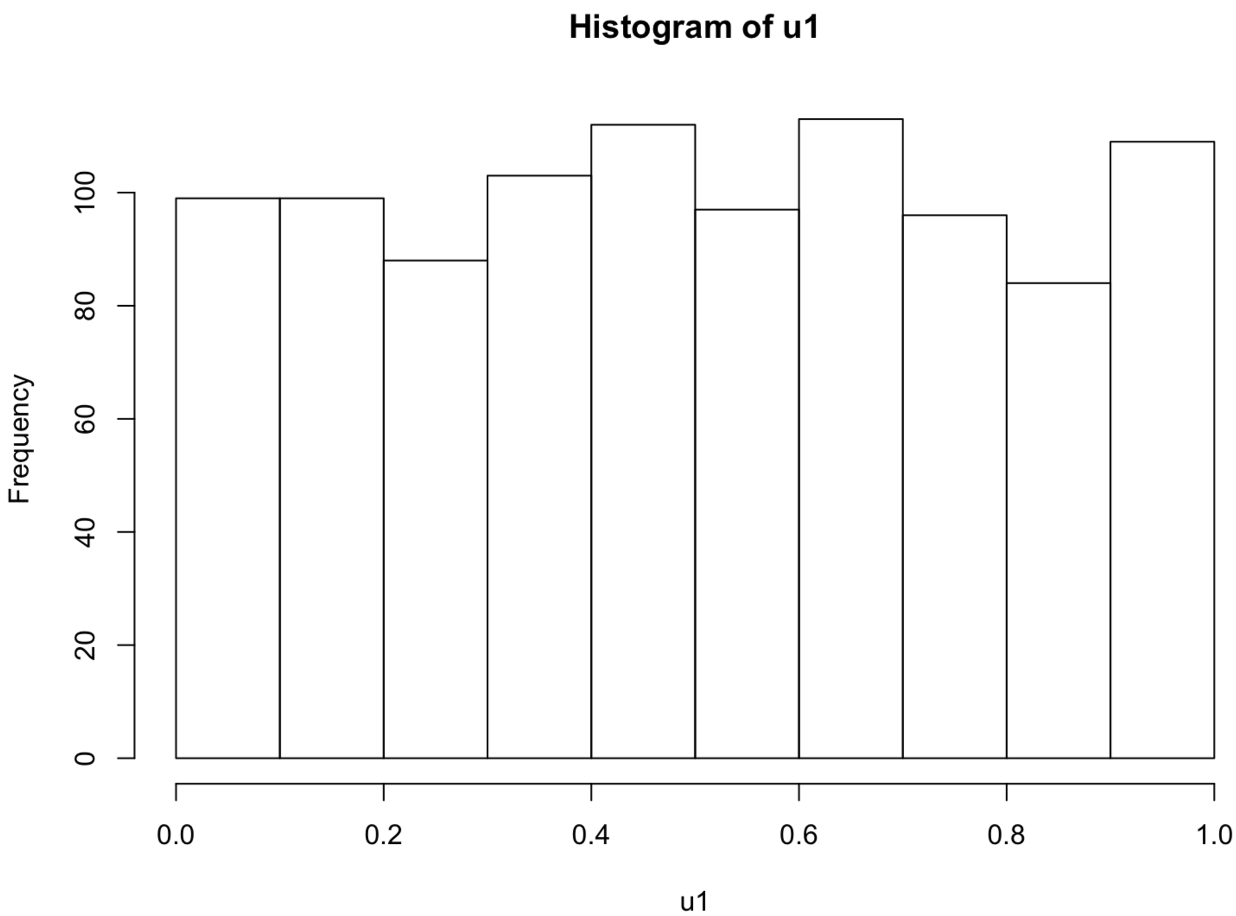
```
set.seed(198911)
vecu=runif(100)
mean(vecu)
```

```
## [1] 0.4724584
```

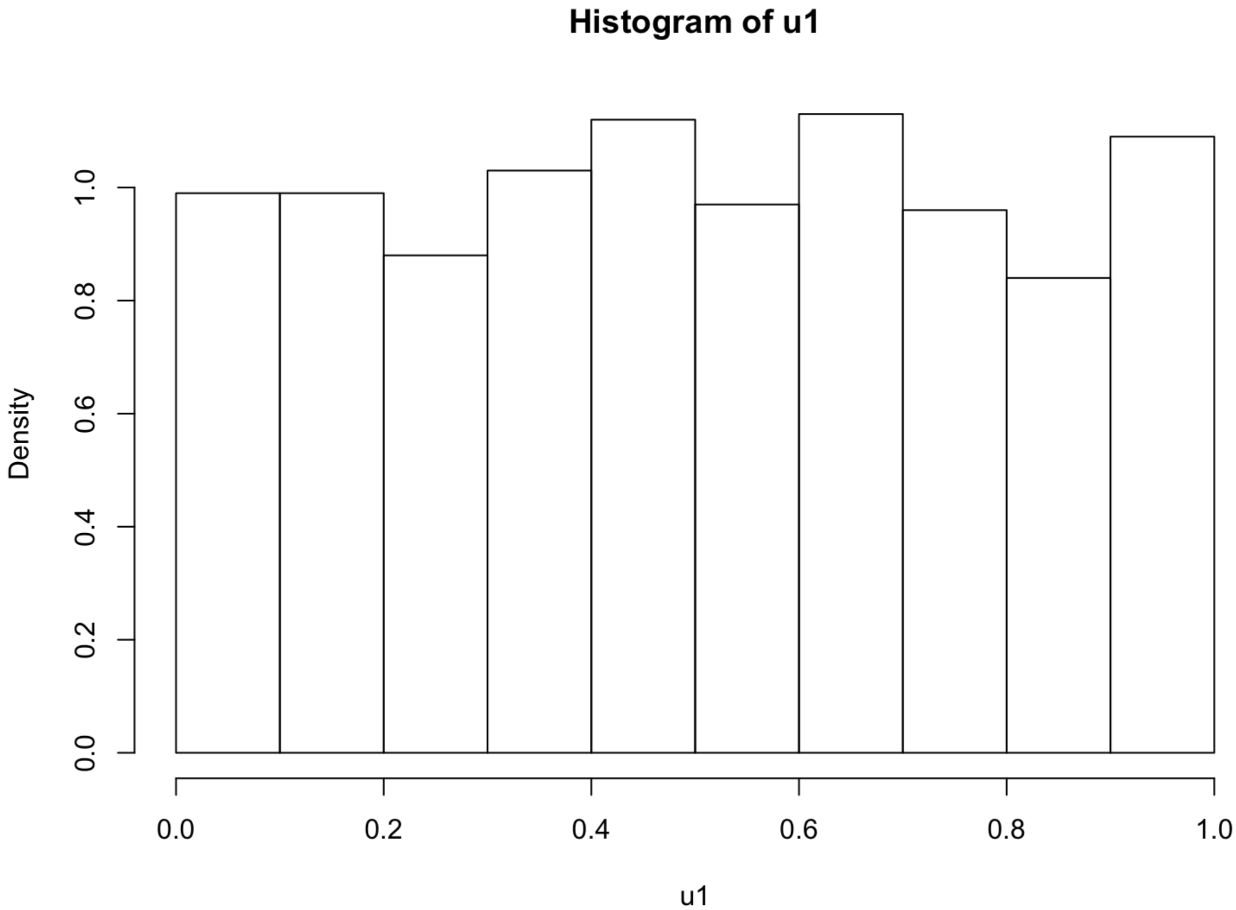
```
set.seed(198911)
vecu=runif(100)
mean(vecu)
```

```
## [1] 0.4724584
```

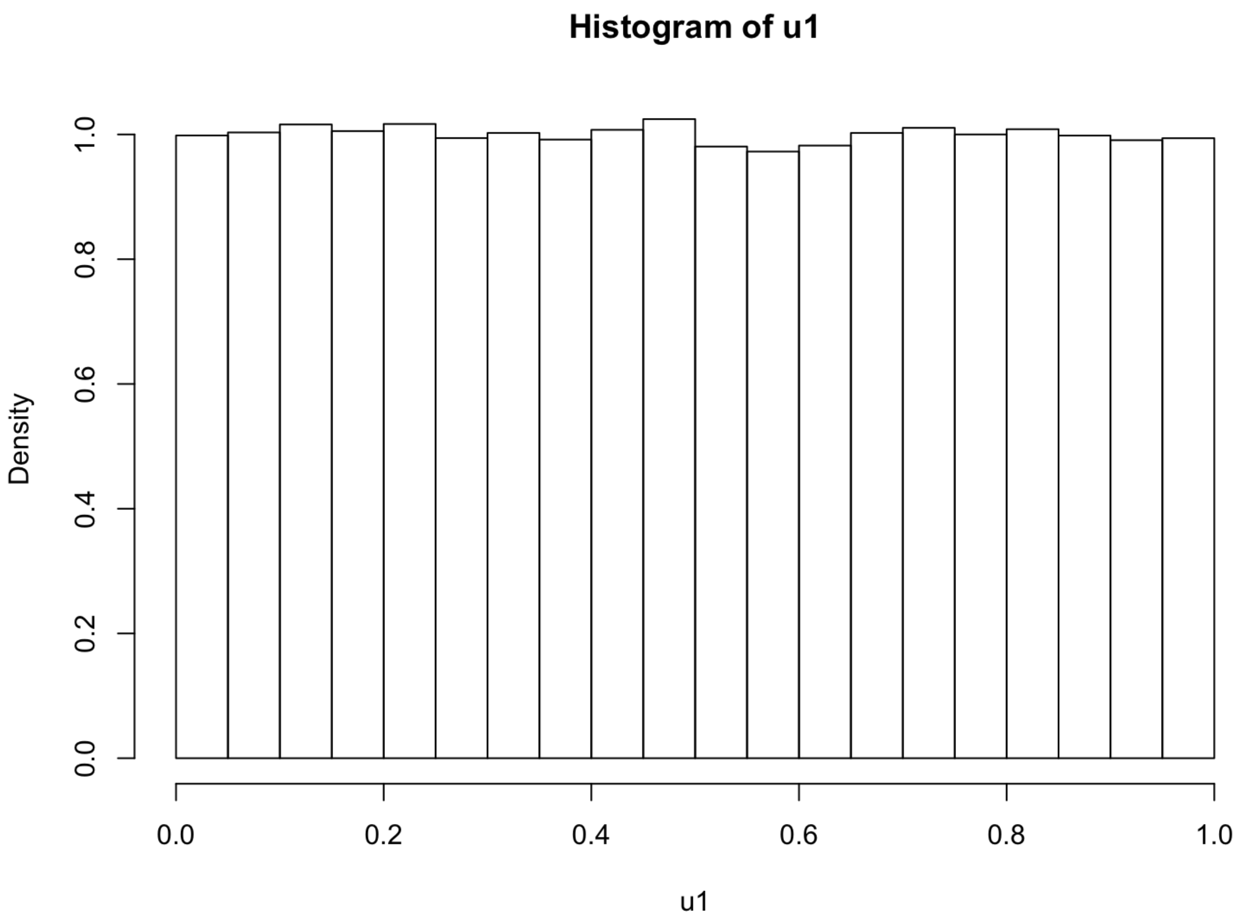
```
u1=runif(1000)
hist(u1)
```




```
hist(u1,freq=FALSE)
```




```
u1=runif(100000)
hist(u1,freq=FALSE)
```




```r
vals = seq(0,1,length=51)
head(vals)
```

```
## [1] 0.00 0.02 0.04 0.06 0.08 0.10
```

```r
punif(vals)
```

```
##  [1] 0.00 0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16 0.18 0.20 0.22 0.24 0.26
## [15] 0.28 0.30 0.32 0.34 0.36 0.38 0.40 0.42 0.44 0.46 0.48 0.50 0.52 0.54
## [29] 0.56 0.58 0.60 0.62 0.64 0.66 0.68 0.70 0.72 0.74 0.76 0.78 0.80 0.82
## [43] 0.84 0.86 0.88 0.90 0.92 0.94 0.96 0.98 1.00
```

```r
dunif(vals)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
qunif(vals)
```

```
##  [1] 0.00 0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16 0.18 0.20 0.22 0.24 0.26
## [15] 0.28 0.30 0.32 0.34 0.36 0.38 0.40 0.42 0.44 0.46 0.48 0.50 0.52 0.54
## [29] 0.56 0.58 0.60 0.62 0.64 0.66 0.68 0.70 0.72 0.74 0.76 0.78 0.80 0.82
## [43] 0.84 0.86 0.88 0.90 0.92 0.94 0.96 0.98 1.00
```

For a uniform, the distribution function is

$$P(X \leq a) = a$$

This is a special property of the Uniform and is not generally true.

We call the value q25 such that

$$P(X \leq q25) = 0.25$$

the 25th percentile or the first quartile.

Question Do you know what we call the value such that

$$P(X \leq m) = 0.5$$

An aside about efficiency: using the apply function

In general, we want to avoid `for` loops in R since that is slower than working with functions such as `apply()`.

We will generate 5 samples from a uniform and sum them together. This is `sum(runif(n=nuni))`.

The function `replicate()` allows us to do this many times with very little code. Here, we do this sum 50,000 times to get an idea of the distribution.

```
reps <- 50000
nuni <- 5
set.seed(0)
system.time(
  sum5 <- replicate(reps, sum(runif(nuni)))
  ) # replicate
```

```
##    user  system elapsed
##   0.214   0.013   0.260
```

```
head(sum5)
```

```
## [1] 3.015391 3.334659 1.515444 3.357100 2.701522 1.918804
```

Efficiency can be measured

There are different ways to do these simulations.

Look at the `help(replicate)` and you can see that there are various functions `sapply()`, `lapply()`, and `vapply()`. These are related to the functions `apply()` and `tapply()`.

Here is how you might do the same thing with `sapply()`. You can plot this using the same commands above.

```
set.seed(0)
reps <-10000
system.time(x1 <- sapply(1:reps, function(i){sum(runif(n=nuni))})) # simple apply
```

```
##    user  system elapsed
##   0.034   0.000   0.036
```

```
head(x1)
```

```
## [1] 3.015391 3.334659 1.515444 3.357100 2.701522 1.918804
```

```
set.seed(0)
system.time(x1 <- lapply(1:reps, function(i){sum(runif(n=nuni))})) # List apply
```

```
##    user  system elapsed
##   0.066   0.001   0.067
```

```
head(x1)
```

```
## [[1]]
## [1] 3.015391
##
## [[2]]
## [1] 3.334659
##
## [[3]]
## [1] 1.515444
##
## [[4]]
## [1] 3.3571
##
## [[5]]
## [1] 2.701522
##
## [[6]]
## [1] 1.918804
```

When we apply a very simple function (e.g., a sum), the fastest way is often to just make a matrix of all the simulations and then apply that function to the matrix appropriately.

The functions `rowSums()` and `colSums()` are particularly efficient at this.

```
set.seed(0)
system.time(sum5 <- apply(matrix(runif(n=nuni*reps), nrow=nuni),2,sum)) # apply on a matrix
```

```
##    user  system elapsed
##   0.019   0.000   0.020
```

```r
head(sum5)
```

```
## [1] 3.015391 3.334659 1.515444 3.357100 2.701522 1.918804
```

```r
set.seed(0)
system.time(sum5 <- colSums(matrix(runif(n=nuni*reps), nrow=nuni))) # using colSums
```

```
##    user  system elapsed 
##   0.002   0.000   0.002
```

```r
head(sum5)
```

```
## [1] 3.015391 3.334659 1.515444 3.357100 2.701522 1.918804
```

Question What is the range of Sum5 ?

```r
summary(sum5)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  0.4874  2.0529  2.5032  2.5025  2.9549  4.7007
```

Question Do you think all the values in the range are equally likely?

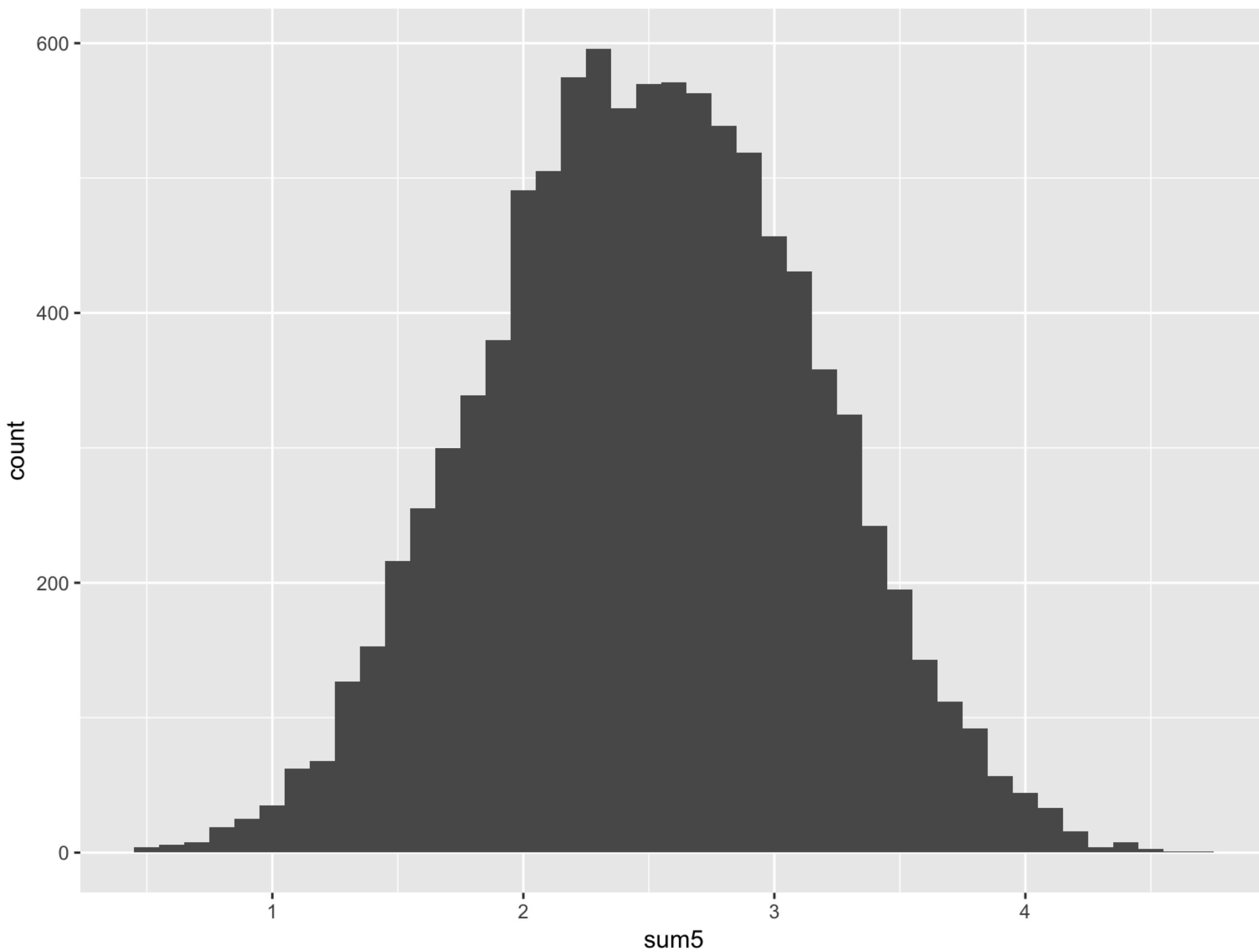
We can make a histogram of the simulation and compare it to other distributions.

A ggplot histogram

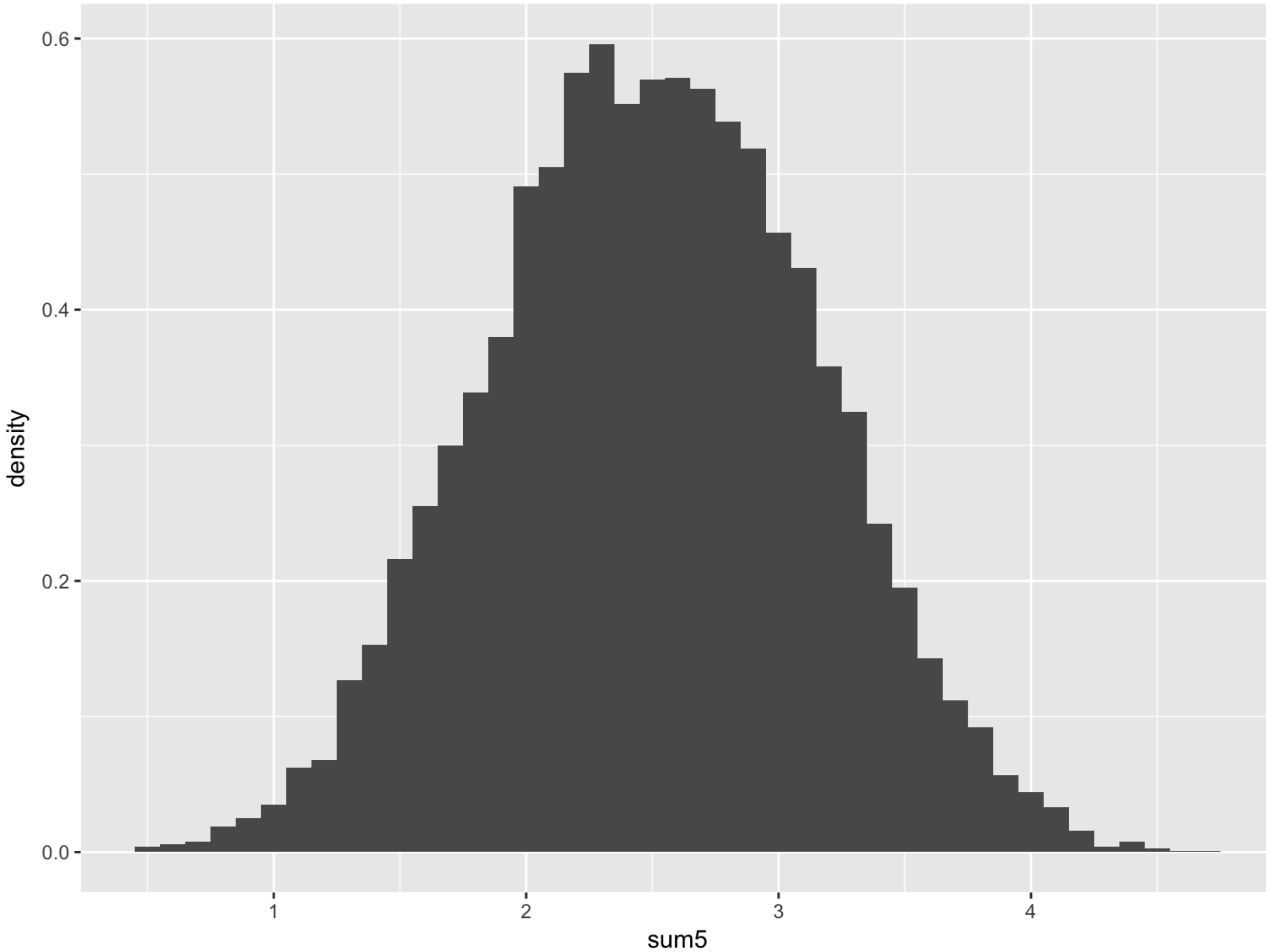
Look up ggplot cookbook

Find how to call a histogram in ggplot.

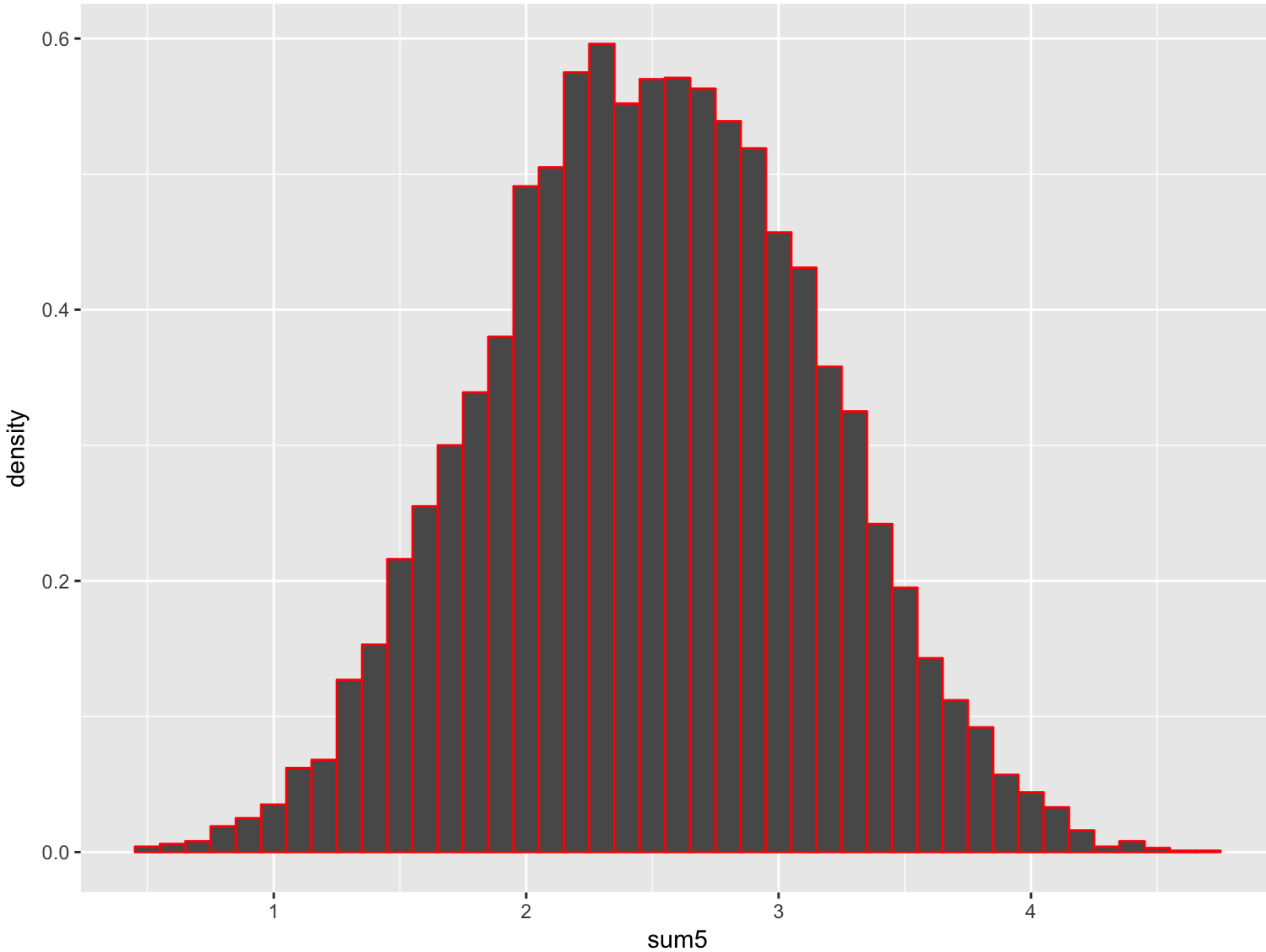
```
require(ggplot2)
d5 <-data.frame(sum5)
ggplot(d5, aes(sum5)) +
  geom_histogram(binwidth =0.1)
```



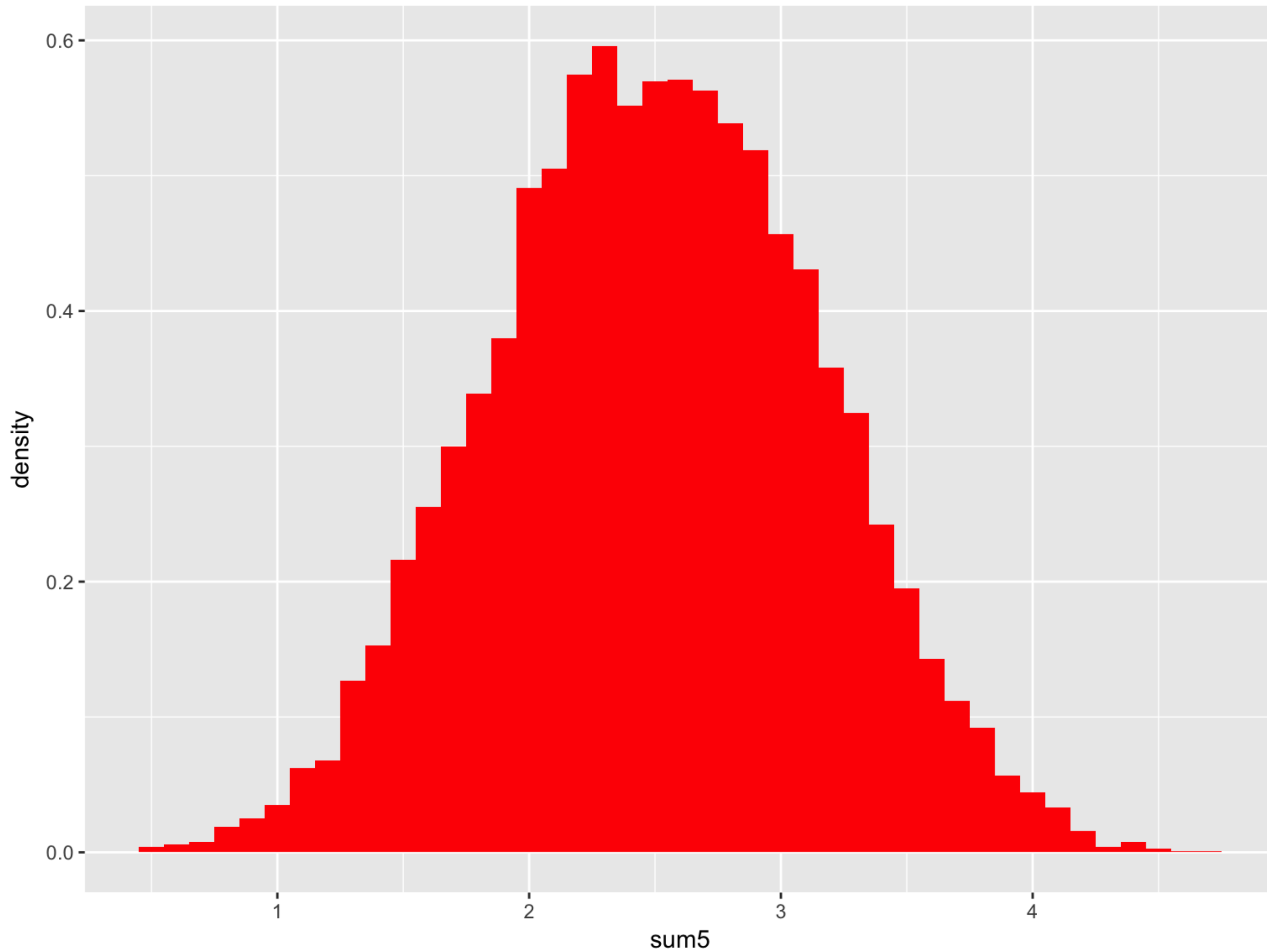

```
ggplot(d5, aes(sum5)) +
  geom_histogram(aes(y=..density..),binwidth =0.1)
```




```r
ggplot(d5, aes(sum5)) +
  geom_histogram(aes(y=..density..),binwidth =0.1,color="red")
```




```
ggplot(d5, aes(sum5)) +
  geom_histogram(aes(y=..density..),binwidth =0.1,fill="red")
```



Question

Let U_1, U_2, U_3 all come from a uniform(0,1) distribution. Let $M = \max(U_1, U_2, U_3)$. Estimate (to 3 significant digits) the probability $\mathbb{P}(M > 0.75)$. Do not use a for loop for any of this question.

Computing a probability by Monte Carlo

Estimate (to 3 significant digits) the probability $\mathbb{P}(\max(U_1, U_2, U_3) > 0.75)$.

```
B <- 1000000
m3 <- matrix(runif(3000000),ncol=1000000,nrow=3)
sum(apply(m3,2,max)>0.75)/1000000
```

```
## [1] 0.578239
```

Answer: 0.578.

Many random variable distributions

```
?Distributions
```

Generating normal random variables

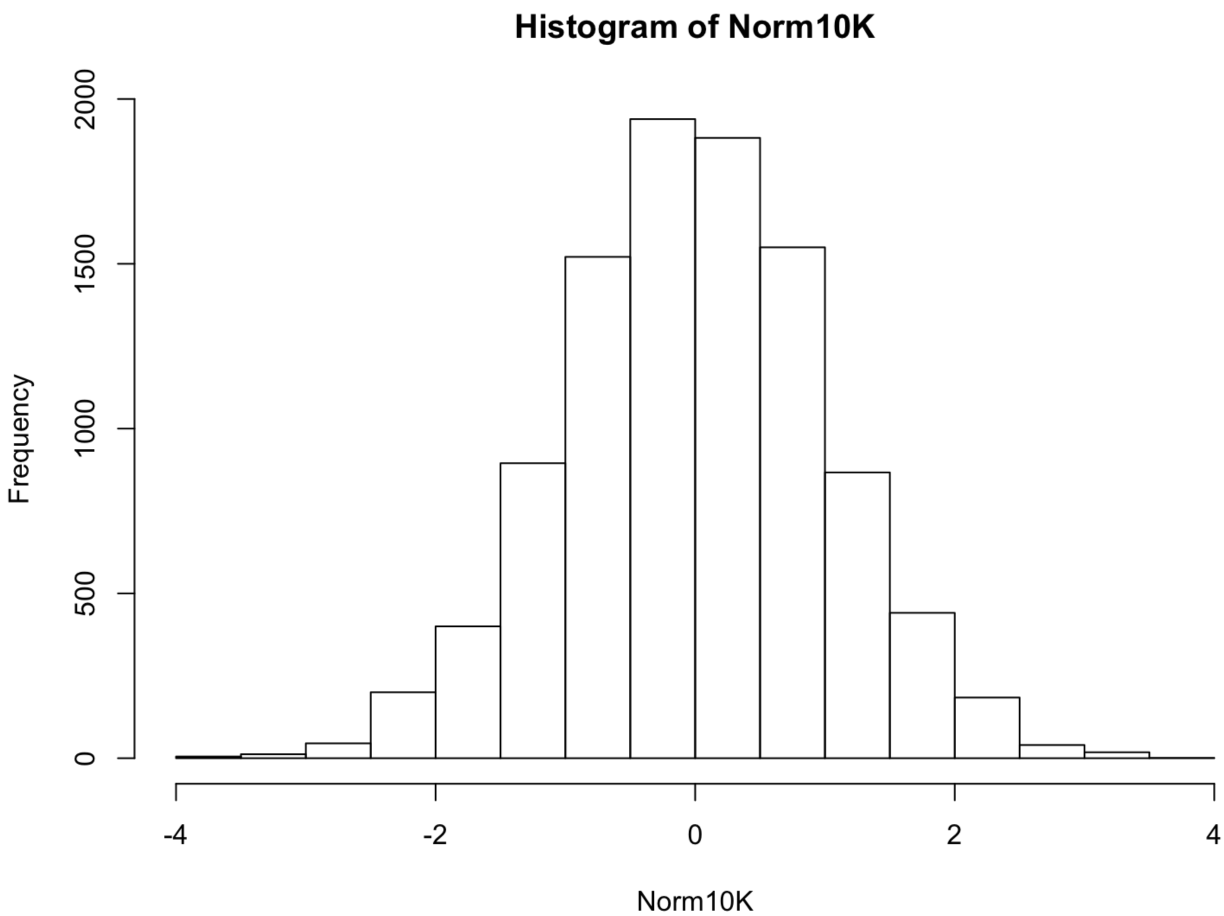
```
Norm10K <-rnorm(10000)
mean(Norm10K)
```

```
## [1] -0.001394208
```

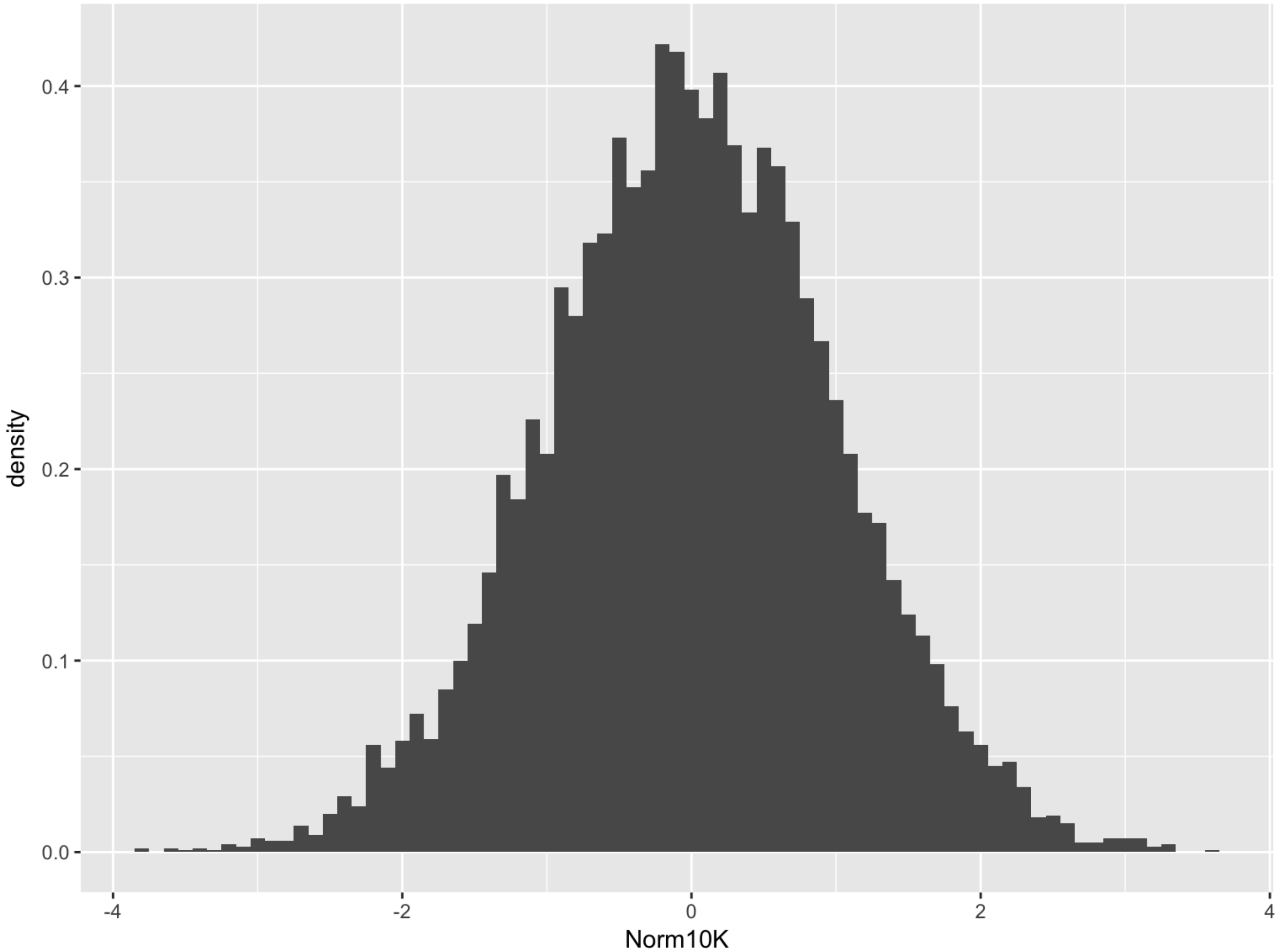
```
sd(Norm10K)
```

```
## [1] 1.005359
```

```
hist(Norm10K)
```




```
ggplot(data.frame(Norm10K), aes(x = Norm10K)) +
  geom_histogram(aes(y=..density..), binwidth =0.1)
```




```r
qnorm(0.25)
```

```
## [1] -0.6744898
```

```r
qnorm(0)
```

```
## [1] -Inf
```

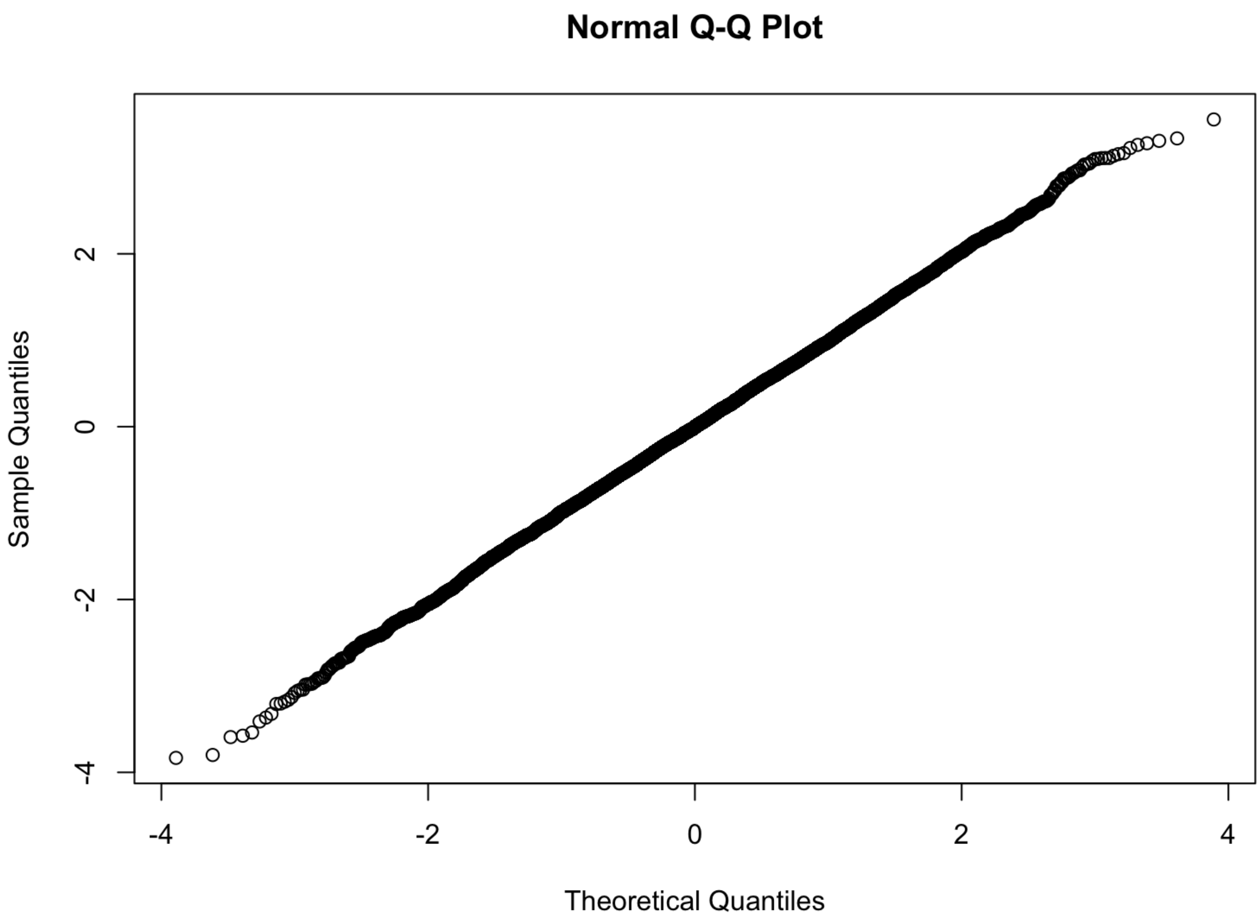
```r
qnorm(0.75)
```

```
## [1] 0.6744898
```

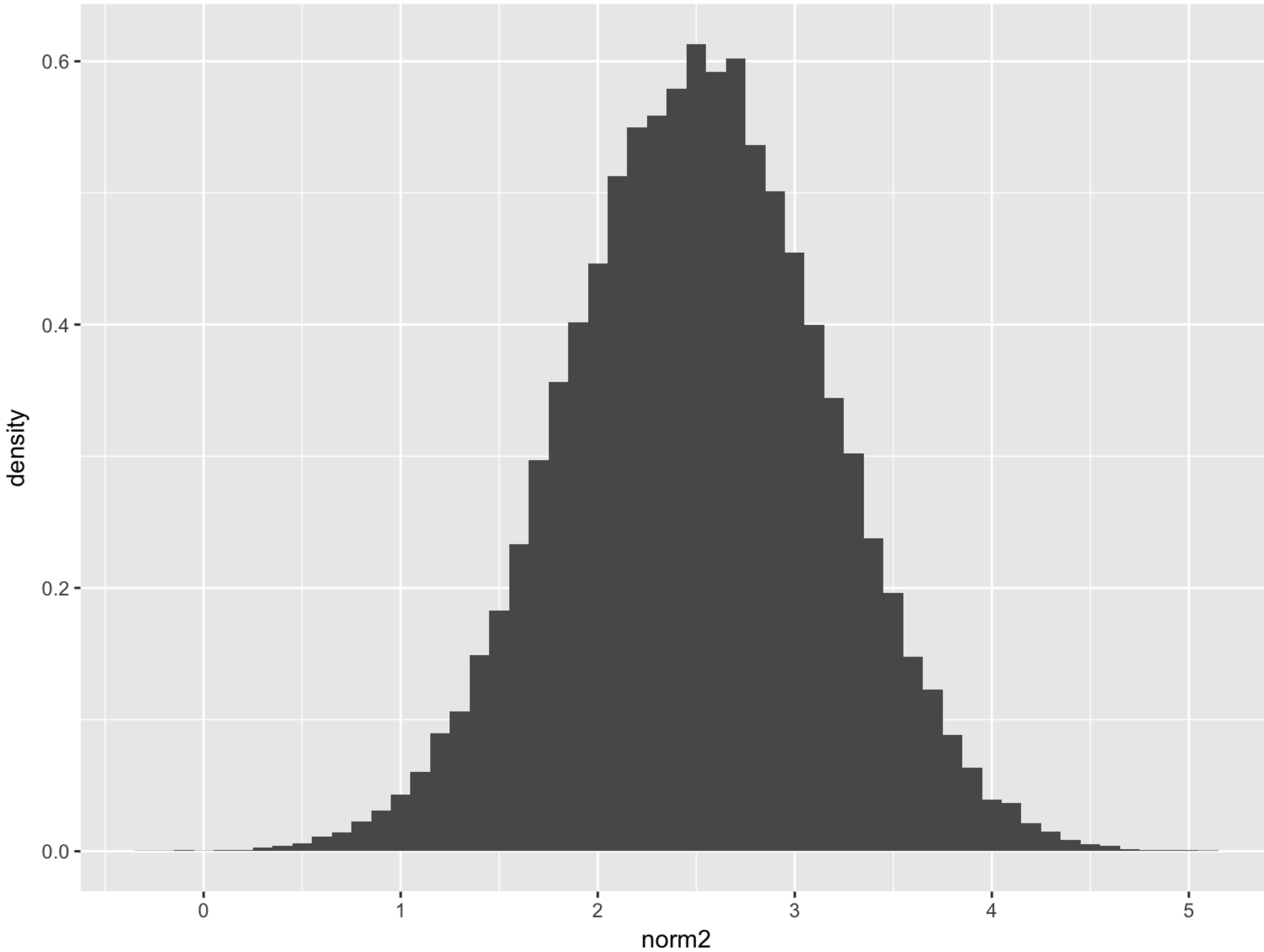
Quantiles versus quantiles

A useful way of comparing distributions:

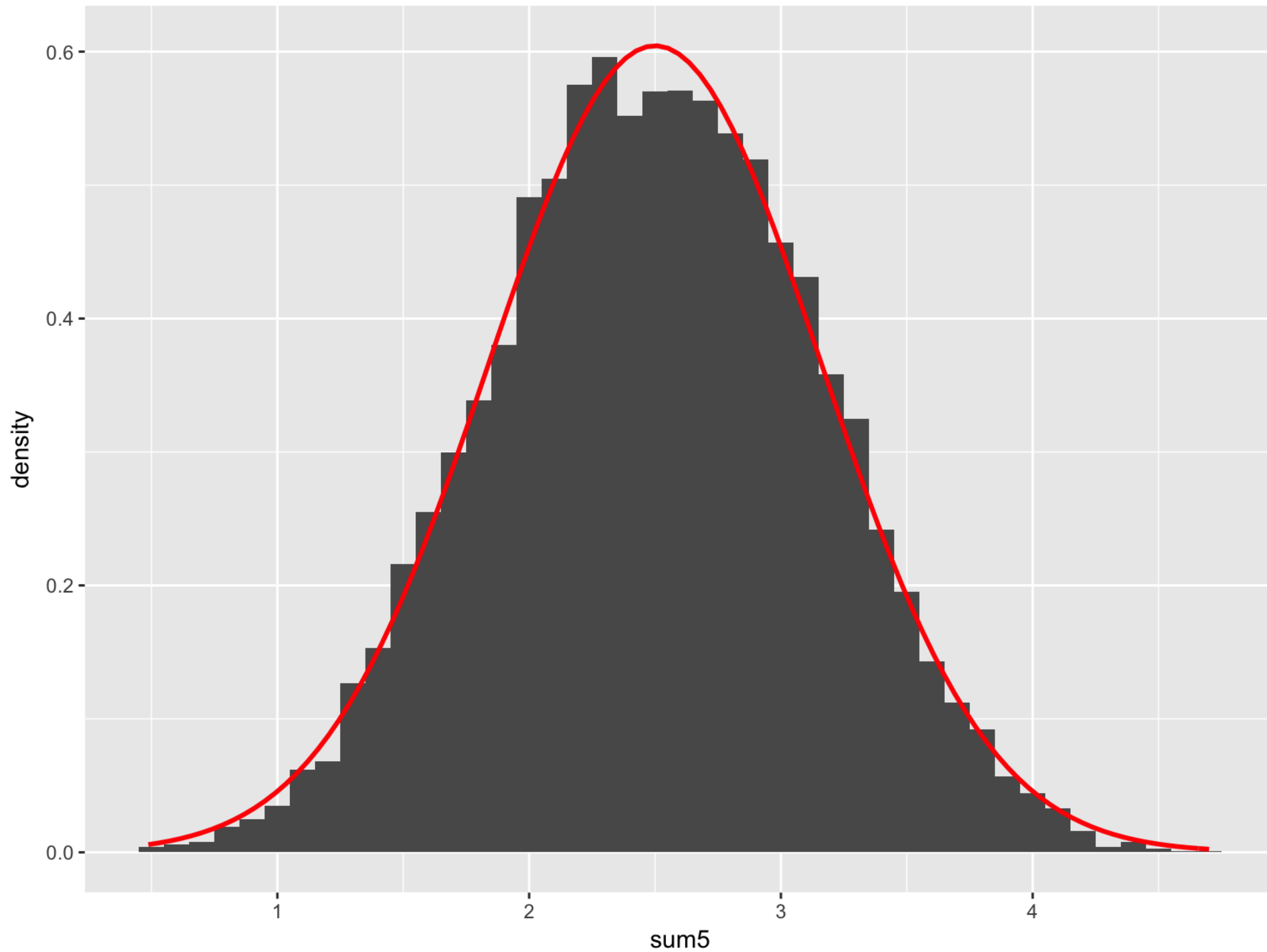
```r
qqnorm(Norm10K)
```




```r
norm2=rnorm(50000,2.5,0.66)
ggplot(data.frame(norm2), aes(x = norm2)) +
  geom_histogram(aes(y=..density..), binwidth =0.1)
```

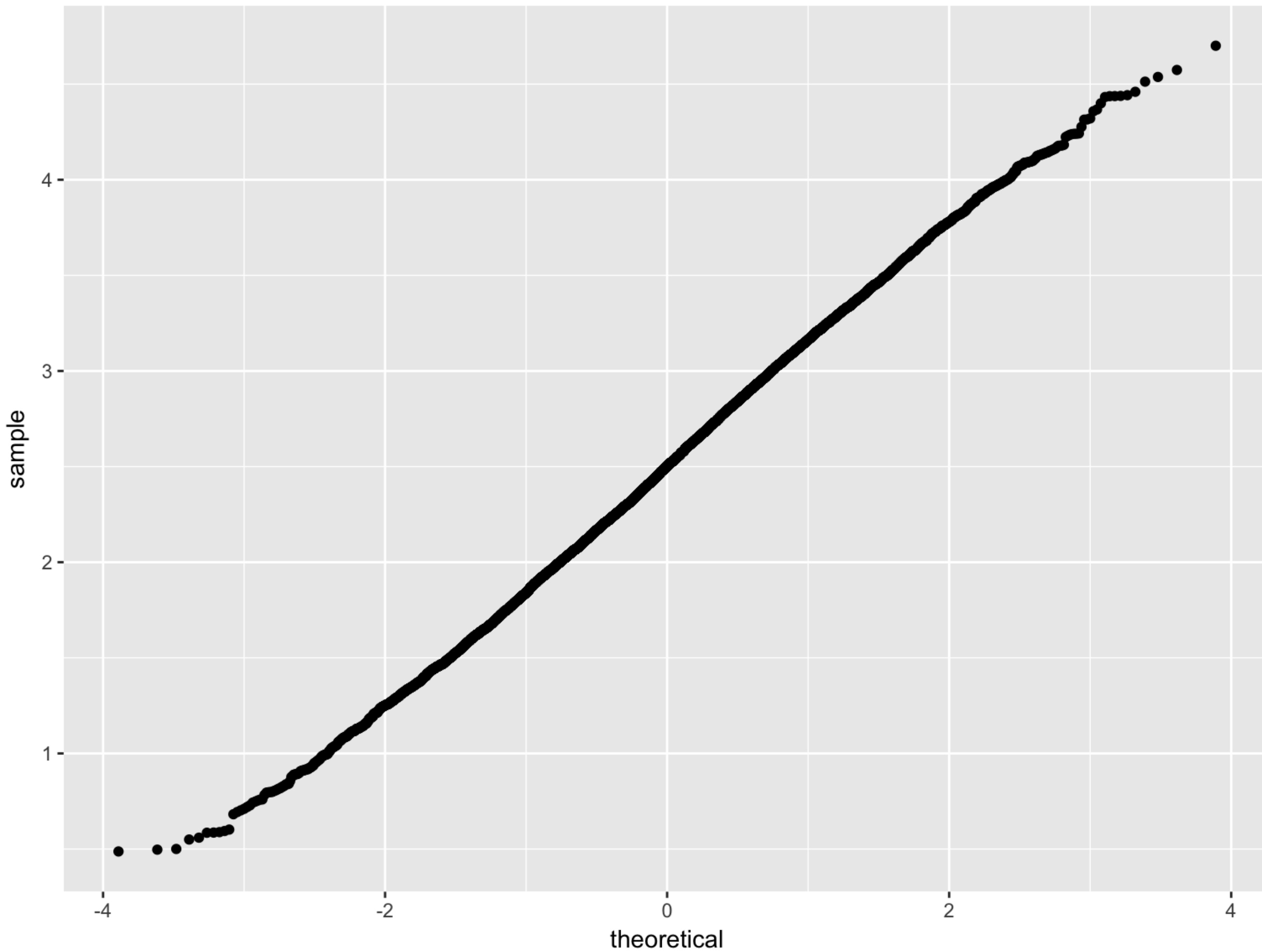



```
ggplot(data.frame(sum5), aes(x = sum5)) +
  geom_histogram(aes(y=..density..),binwidth =0.1) +
stat_function(fun=function(x) dnorm(x,mean=2.5,0.66),
                color="red", size=1)
```

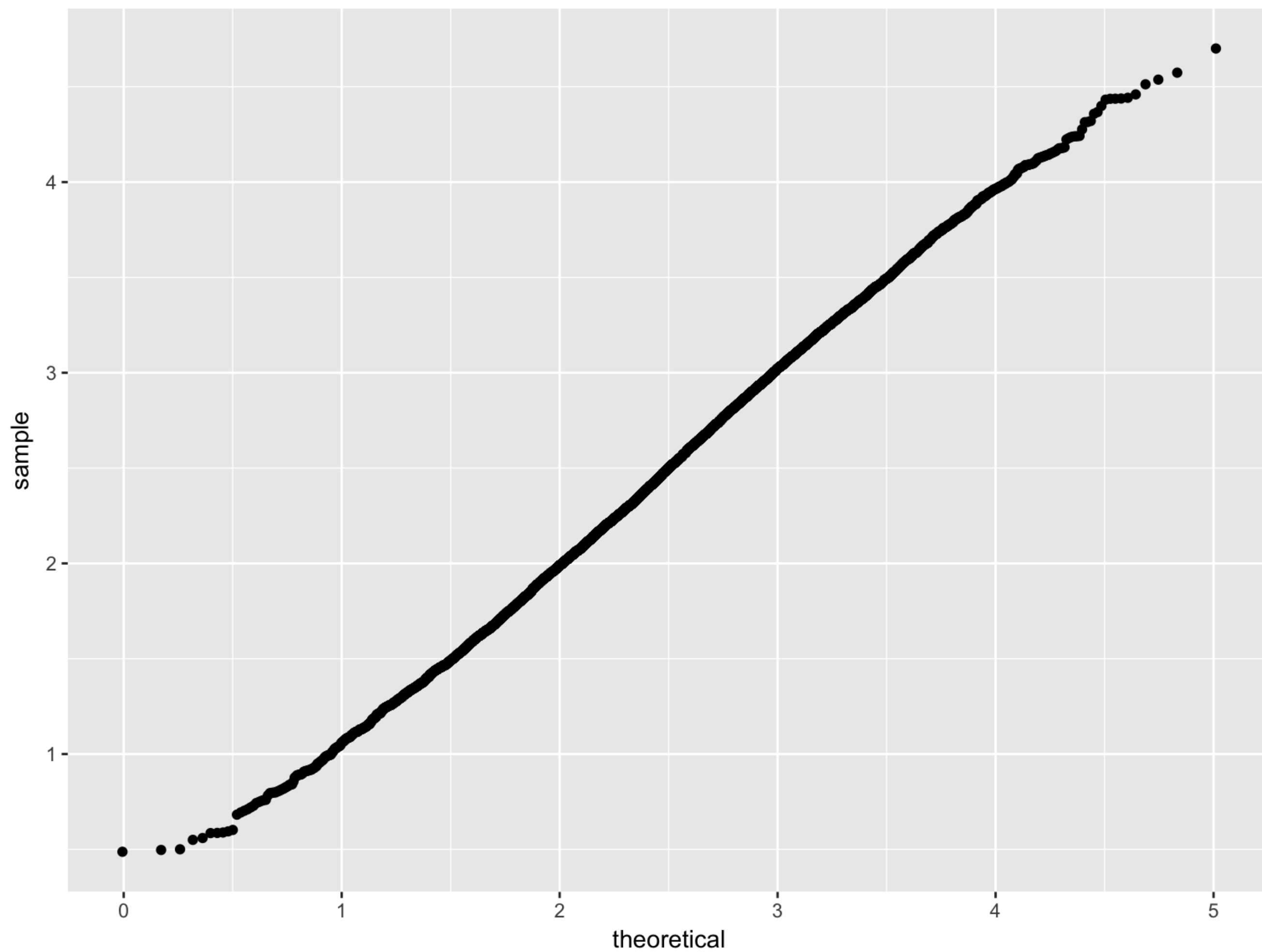


Making a quantile-quantile plot with ggplot.

```
df5=data.frame(sum5)
ggplot(df5, aes(sample = sum5)) +
  geom_point(stat = "qq")
```




```
ggplot(df5, aes(sample = sum5)) +
  stat_qq(distribution = qnorm, dparams = list(mean(sum5),sd(sum5)))
```



Question

Let $Z_{(n)}$ be maximum of n standard normal observations. Estimate what n should be so that $\mathbb{P}(Z_{(n)} > 4) = 0.25$.

```
n <-100
mat10k=matrix(rnorm(1000*n),ncol=n)
maxs = apply(mat10k,1,max)
summary(maxs)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.502   2.217   2.467   2.517   2.766   4.614
```

```
n <-200
mat10k=matrix(rnorm(1000*n),ncol=n)
maxs = apply(mat10k,1,max)
summary(maxs)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.762   2.459   2.701   2.749   2.984   4.584
```

```
n <-1000
mat10k=matrix(rnorm(1000*n),ncol=n)
maxs = apply(mat10k,1,max)
summary(maxs)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.446   2.976   3.197   3.236   3.437   4.465
```

```
n <-8800
mat10k=matrix(rnorm(1000*n),ncol=n)
maxs = apply(mat10k,1,max)
summary(maxs)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    3.104   3.593   3.759   3.803   3.984   5.137
```

Question

Let $X_1, \ldots, X_n$ be Poisson random variables with parameter $\lambda = 0.5$ and where $n = 21$. Estimate the probability that the sample mean is greater than the sample median.

```
resp = matrix(rpois(100000*21,lambda=0.5),nrow=100000)
means = apply(resp,2,mean)
medians = apply(resp,2,median)
```

Testing

How do we test the difference between two samples?

```
?t.test
sleep
```

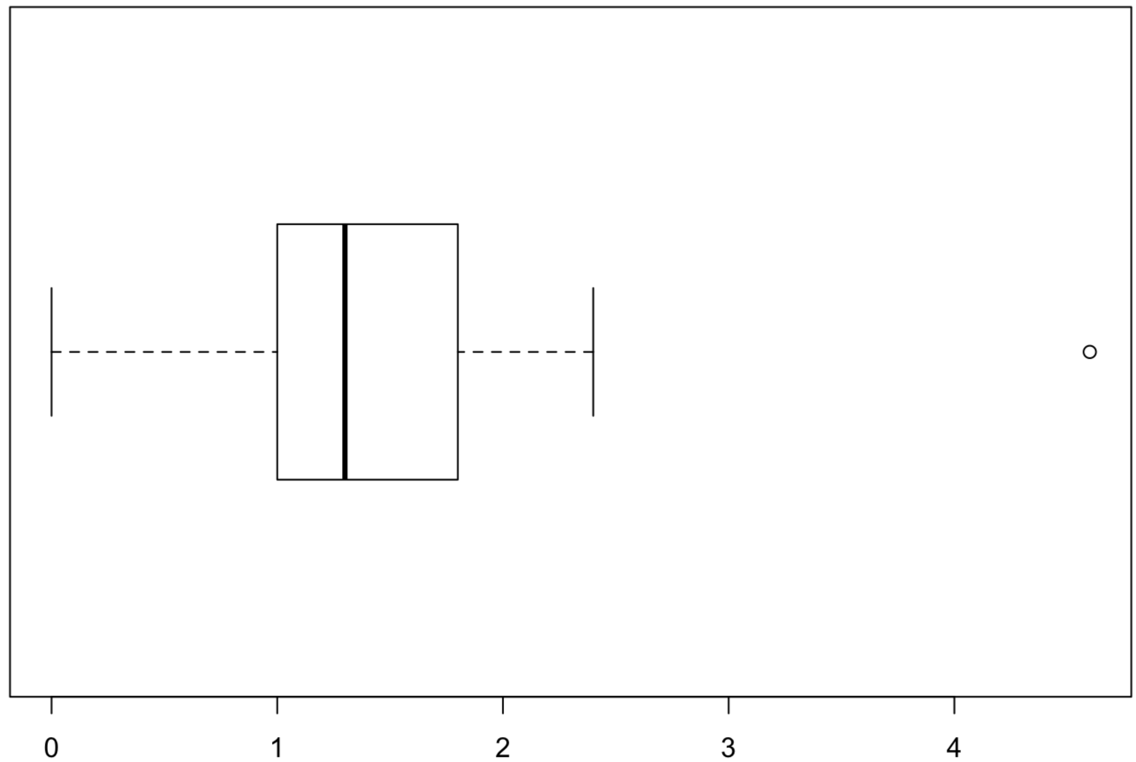
```
##    extra group ID
## 1    0.7     1  1
## 2   -1.6     1  2
## 3   -0.2     1  3
## 4   -1.2     1  4
## 5   -0.1     1  5
## 6    3.4     1  6
## 7    3.7     1  7
## 8    0.8     1  8
## 9    0.0     1  9
## 10   2.0     1 10
## 11   1.9     2  1
## 12   0.8     2  2
## 13   1.1     2  3
## 14   0.1     2  4
## 15  -0.1     2  5
## 16   4.4     2  6
## 17   5.5     2  7
## 18   1.6     2  8
## 19   4.6     2  9
## 20   3.4     2 10
```

```
?sleep
```

```
attach(sleep)
sleep1 <- extra[group == 2] - extra[group == 1]
sleep1
```

```
##  [1] 1.2 2.4 1.3 1.3 0.0 1.0 1.8 0.8 4.6 1.4
```

```
boxplot(sleep1, horizontal = TRUE)
```



Let's do a t.test for this data.

```
t.test(sleep1)
```

```
##
##  One Sample t-test
##
## data:  sleep1
## t = 4.0621, df = 9, p-value = 0.002833
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.7001142 2.4598858
## sample estimates:
## mean of x
##      1.58
```

```
res=t.test(sleep1)
res$p.value
```

```
## [1] 0.00283289
```

```r
t.test(extra~group,paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  extra by group
## t = -4.0621, df = 9, p-value = 0.002833
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.4598858 -0.7001142
## sample estimates:
## mean of the differences
##                   -1.58
```

```r
t.test(extra~group,paired=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  extra by group
## t = -1.8608, df = 17.776, p-value = 0.07939
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.3654832  0.2054832
## sample estimates:
## mean in group 1 mean in group 2
##            0.75            2.33
```

Question What is the difference?

Testing multiple groups

```r
load("birthn.RData")
birthn[1:4,]
```

```
##   year month date_of_month day_of_week births
## 1 2000     1             1           6   9083
## 2 2000     1             2           7   8006
## 3 2000     1             3           1  11363
## 4 2000     1             4           2  13032
```

```r
library(dplyr)
sumsperday<- birthn %>%
 group_by(day_of_week) %>%
 summarise(sum=sum(births)) %>%
 arrange()
sumsperday
```

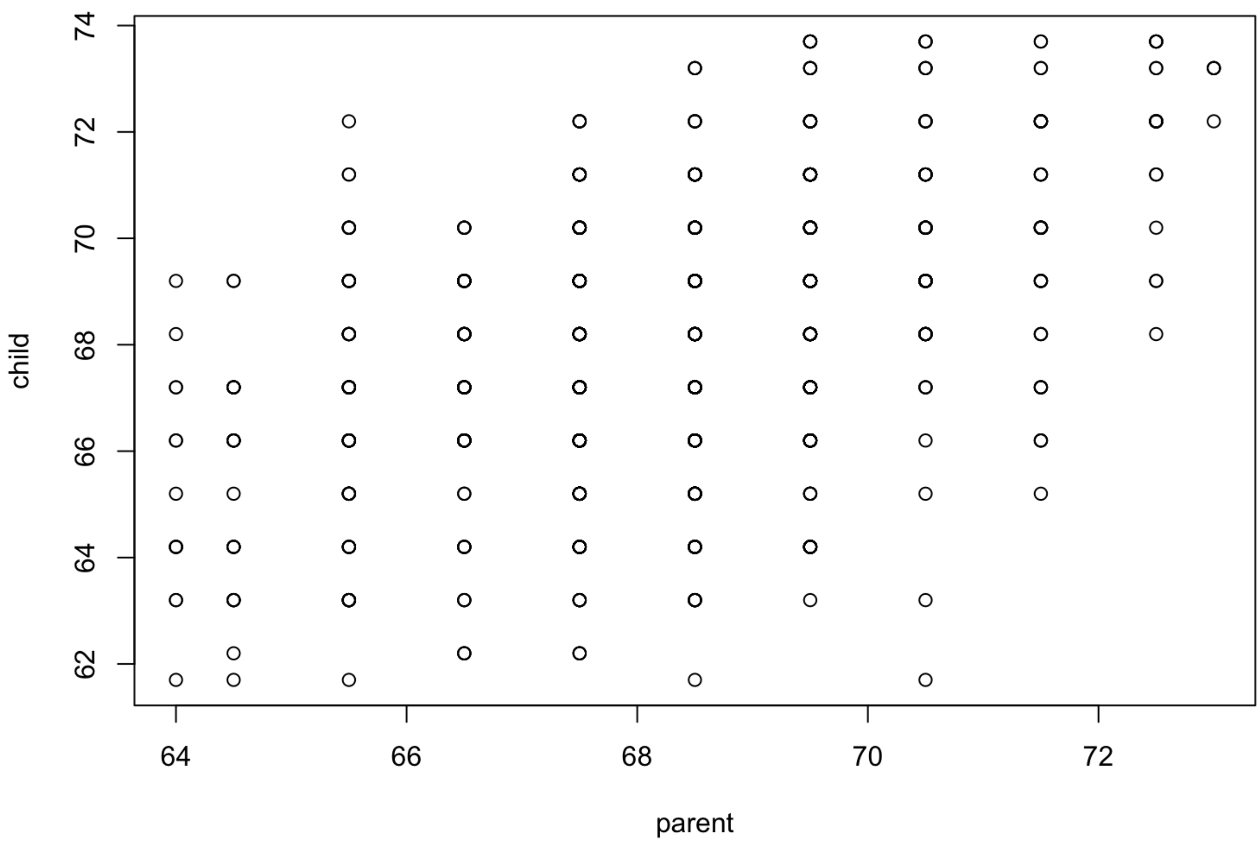
```
## # A tibble: 7 × 2
##   day_of_week      sum
##         <int>    <int>
## 1           1  9316001
## 2           2 10274874
## 3           3 10109130
## 4           4 10045436
## 5           5  9850199
## 6           6  6704495
## 7           7  5886889
```

```r
chisq.test(sumsperday$sum)
```

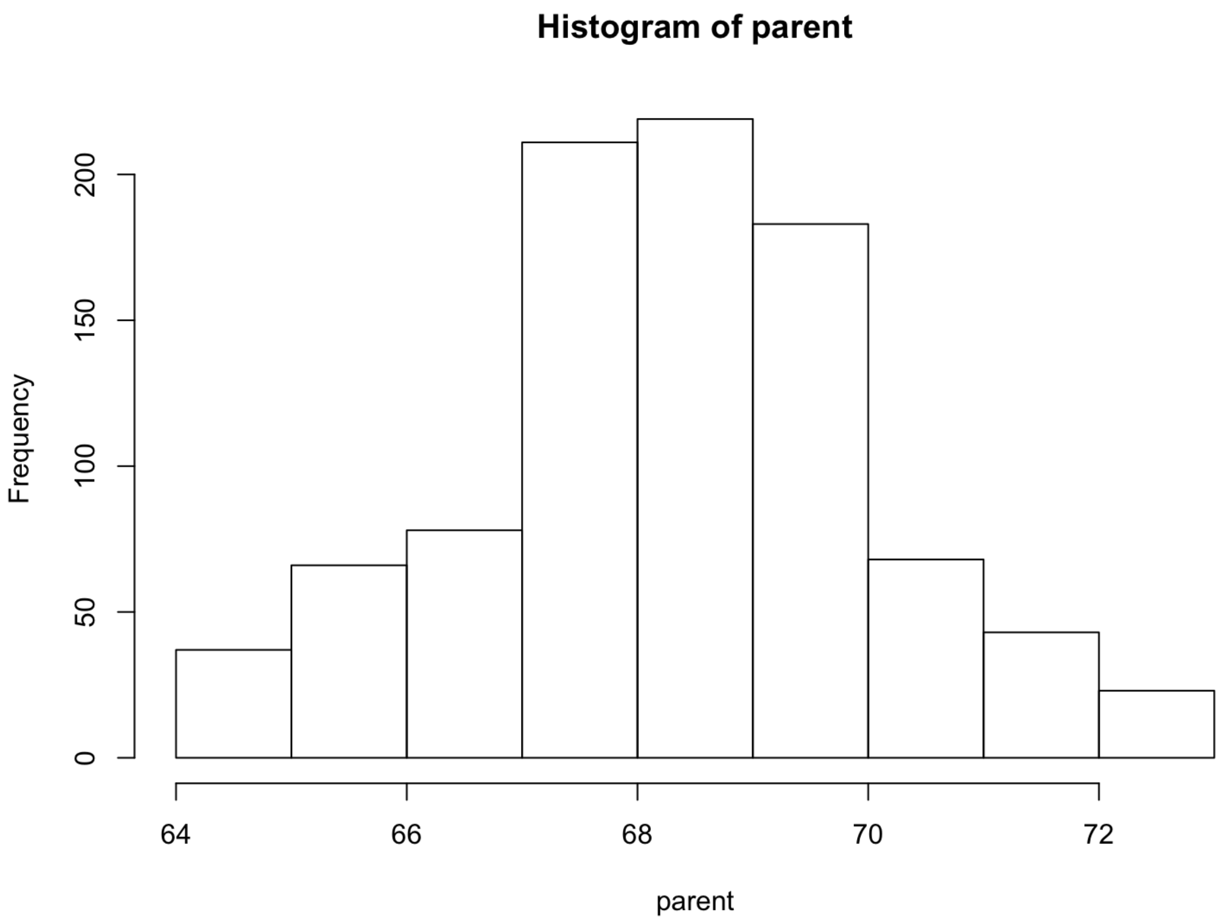
```
## 
##  Chi-squared test for given probabilities
## 
## data:  sumsperday$sum
## X-squared = 2210500, df = 6, p-value < 2.2e-16
```

Regression

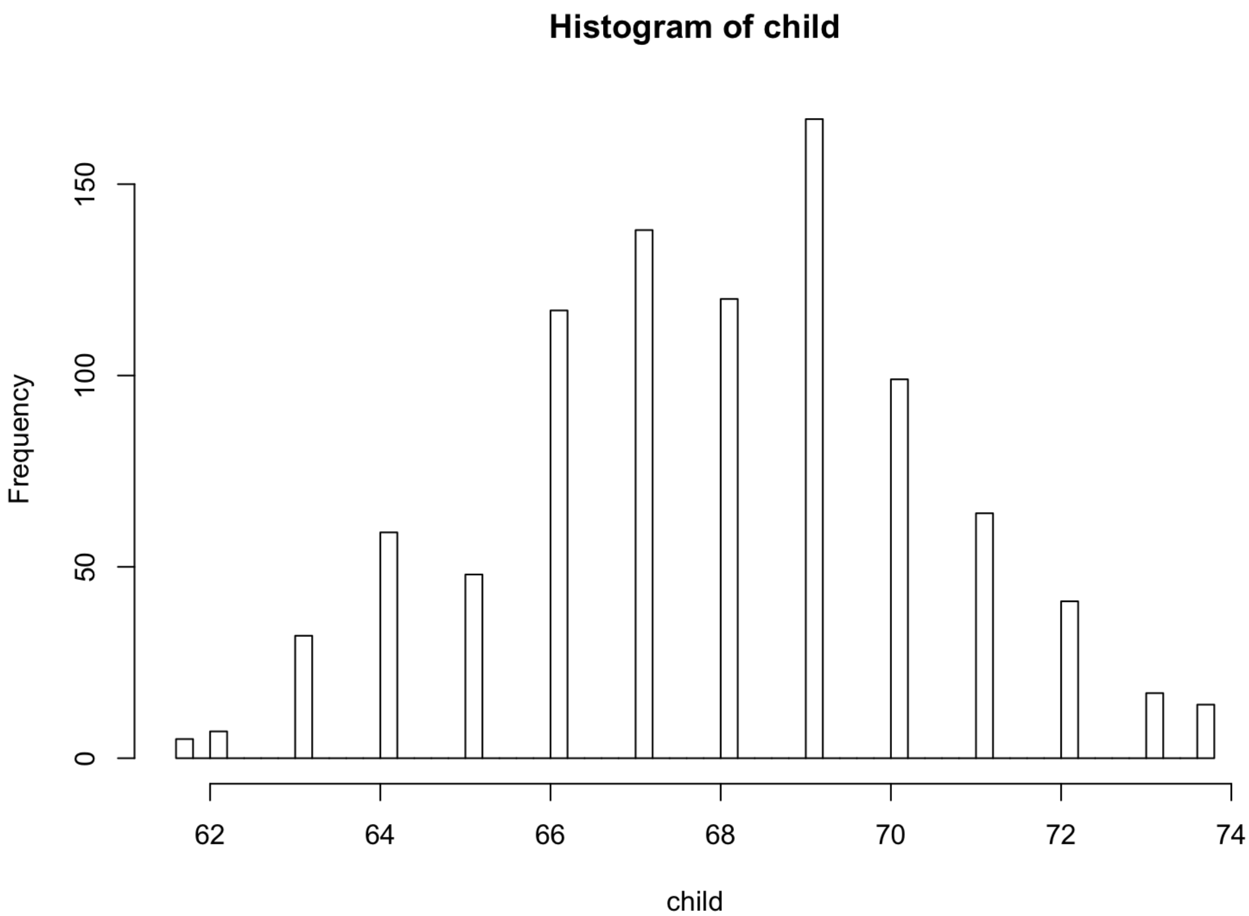
```
library(HistData)
?Galton
attach(Galton)
plot(parent,child)
```



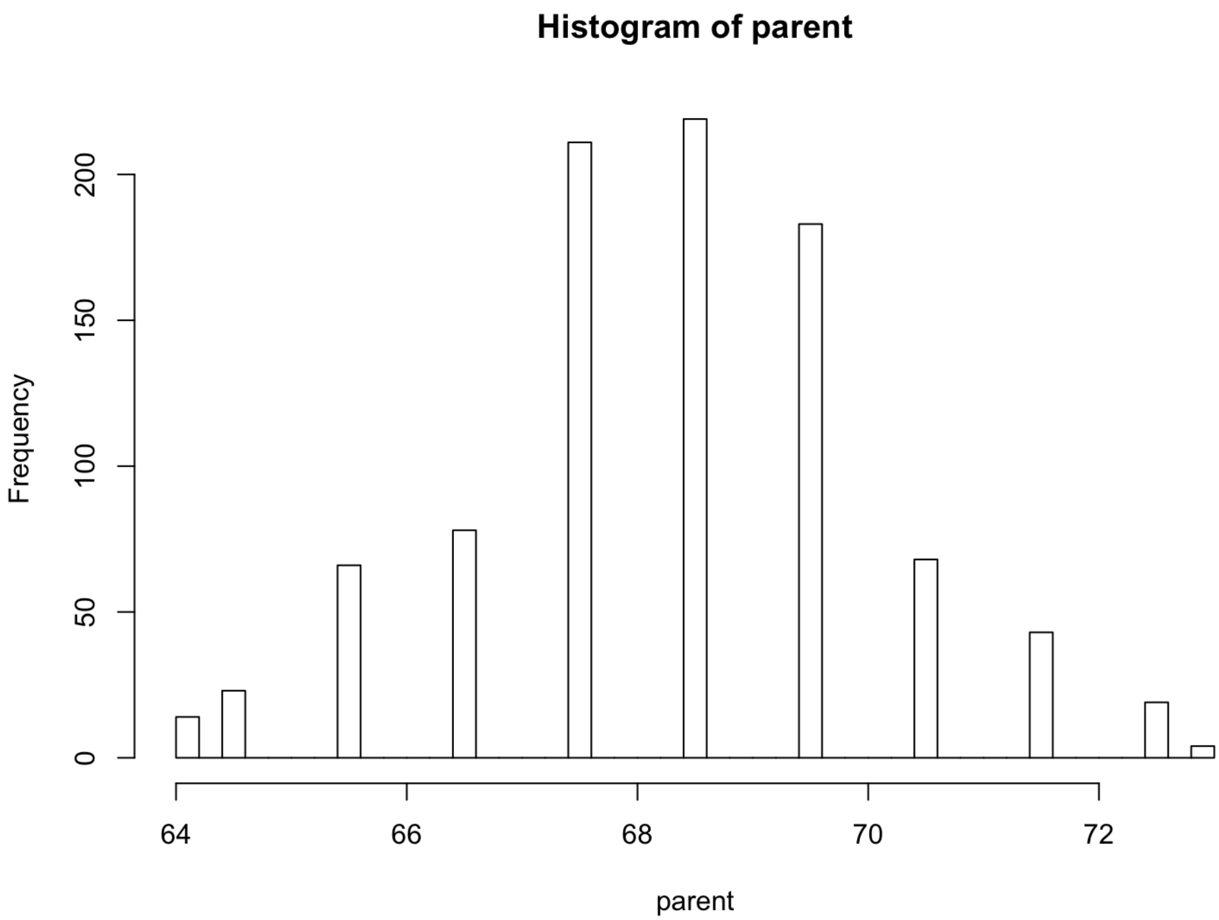

```
hist(parent)
```



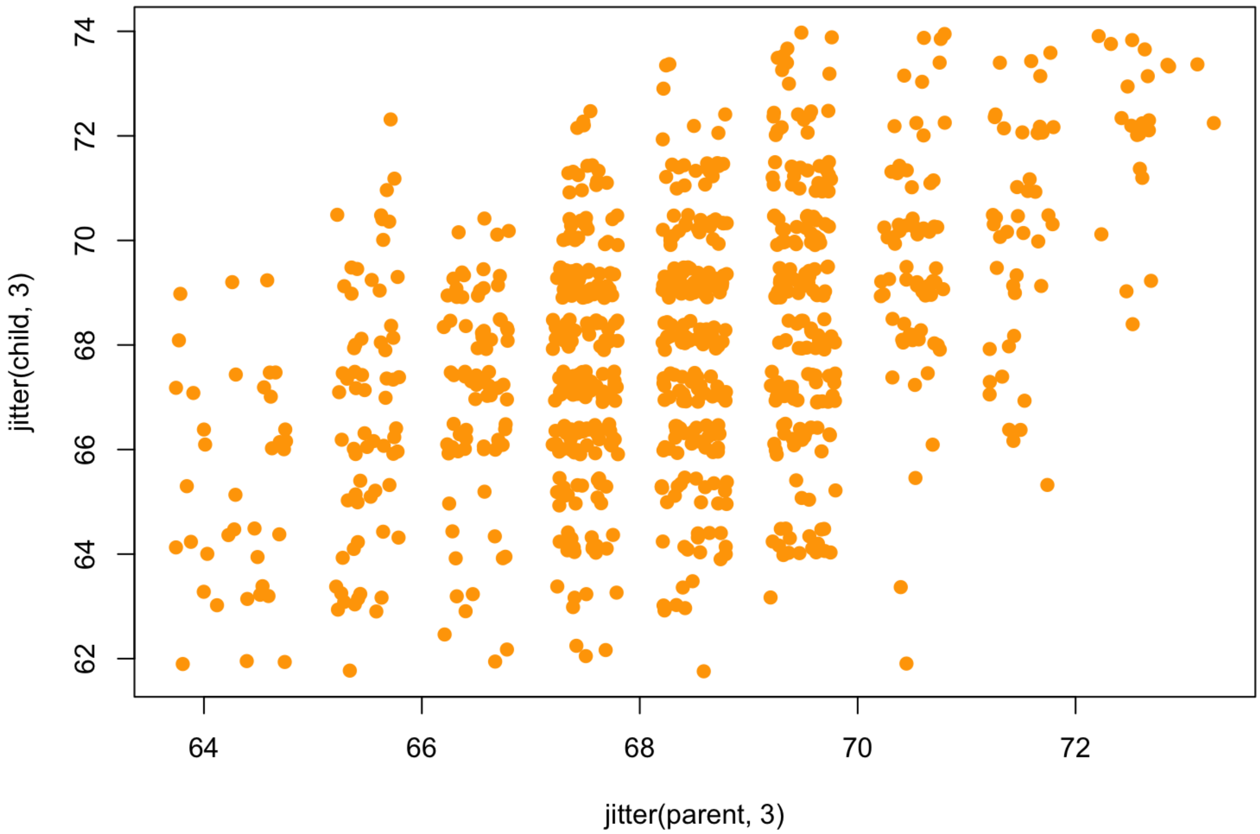

```
hist(child,breaks=50)
```




```
hist(parent,breaks=50)
```




```
plot(jitter(parent,3),jitter(child,3),pch=19,col="orange")
```



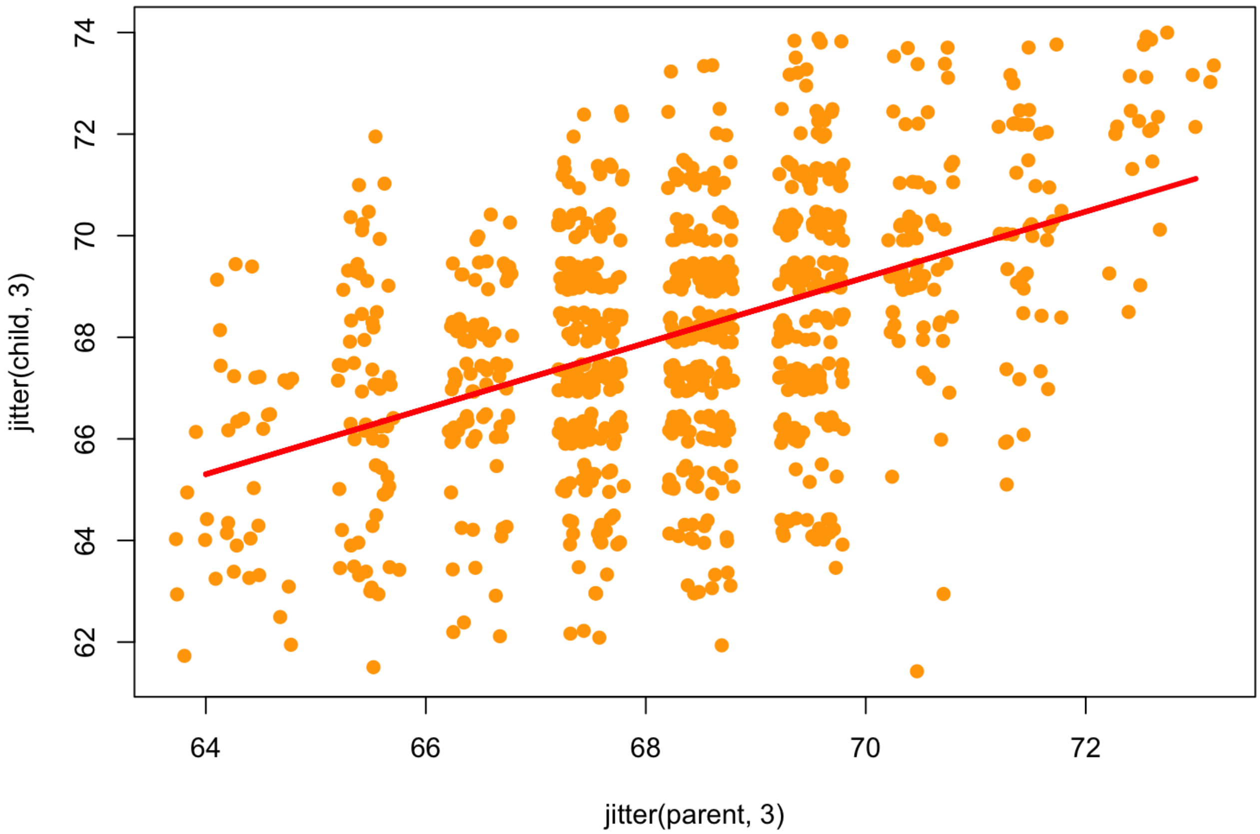

```r
cor(parent,child)
```

```
## [1] 0.4587624
```

```r
lm(child~parent)
```

```
##
## Call:
## lm(formula = child ~ parent)
##
## Coefficients:
## (Intercept)       parent
##     23.9415       0.6463
```

```r
reslm <- lm(child ~ parent)
plot(jitter(parent,3),jitter(child,3),pch=19,col="orange")
lines(parent,reslm$fitted,col="red",lwd=3)
```



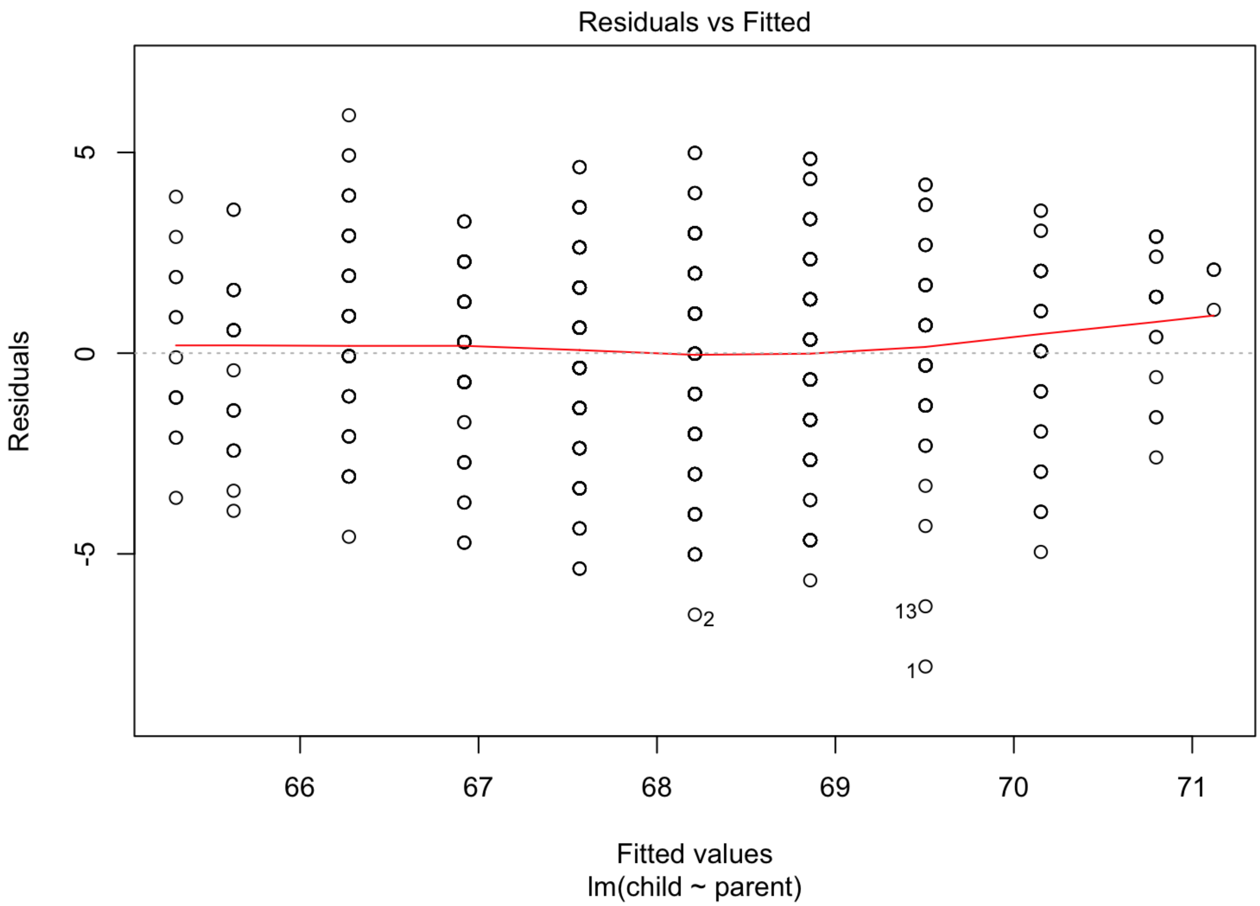
Complex objects returned by many statistical functions

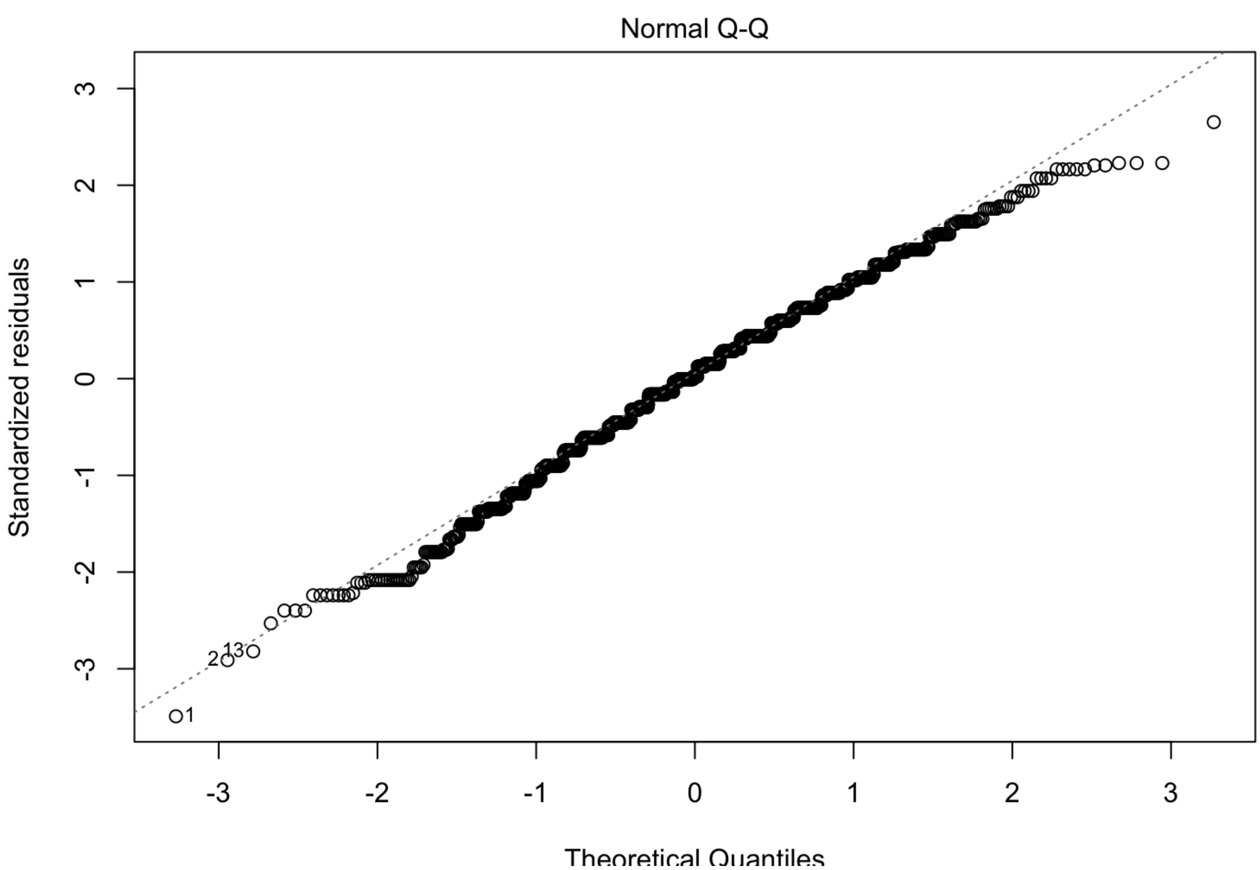
```
str(reslm)
```

```
## List of 12
##  $ coefficients : Named num [1:2] 23.942 0.646
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "parent"
##  $ residuals    : Named num [1:928] -7.81 -6.51 -4.57 -3.93 -3.6 ...
##   ..- attr(*, "names")= chr [1:928] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:928] -2074.19 -35.17 -4.66 -4.12 -3.86 ...
##   ..- attr(*, "names")= chr [1:928] "(Intercept)" "parent" "" "" ...
##  $ rank         : int 2
##  $ fitted.values: Named num [1:928] 69.5 68.2 66.3 65.6 65.3 ...
##   ..- attr(*, "names")= chr [1:928] "1" "2" "3" "4" ...
##  $ assign       : int [1:2] 0 1
##  $ qr           :List of 5
##   ..$ qr    : num [1:928, 1:2] -30.4631 0.0328 0.0328 0.0328 0.0328 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:928] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:2] "(Intercept)" "parent"
##   .. ..- attr(*, "assign")= int [1:2] 0 1
```

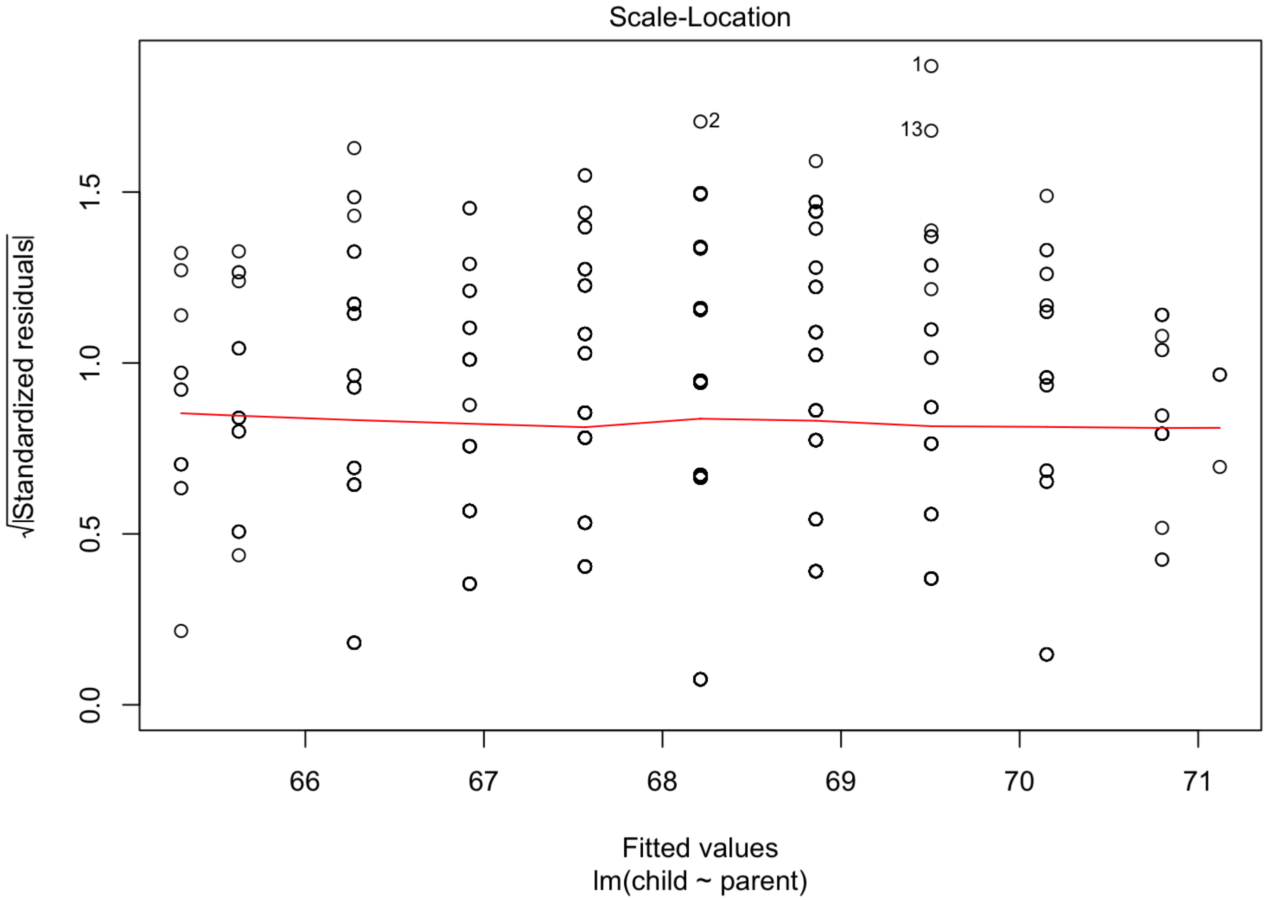
```
##   ..$ qraux: num [1:2] 1.03 1
##   ..$ pivot: int [1:2] 1 2
##   ..$ tol  : num 1e-07
##   ..$ rank : int 2
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 926
##  $ xlevels      : Named list()
##  $ call         : language lm(formula = child ~ parent)
##  $ terms        :Classes 'terms', 'formula'  language child ~ parent
##   .. ..- attr(*, "variables")= language list(child, parent)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "child" "parent"
##   .. .. .. ..$ : chr "parent"
##   .. ..- attr(*, "term.labels")= chr "parent"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(child, parent)
##   .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:2] "child" "parent"
##  $ model        :'data.frame':   928 obs. of  2 variables:
##   ..$ child : num [1:928] 61.7 61.7 61.7 61.7 61.7 62.2 62.2 62.2 62.2 62.2 ...
##   ..$ parent: num [1:928] 70.5 68.5 65.5 64.5 64 67.5 67.5 67.5 66.5 66.5 ...
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language child ~ parent
##   .. .. ..- attr(*, "variables")= language list(child, parent)
##   .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:2] "child" "parent"
##   .. .. .. .. ..$ : chr "parent"
##   .. .. ..- attr(*, "term.labels")= chr "parent"
##   .. .. ..- attr(*, "order")= int 1
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. .. ..- attr(*, "predvars")= language list(child, parent)
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. .. ..- attr(*, "names")= chr [1:2] "child" "parent"
##  - attr(*, "class")= chr "lm"
```

```
plot(reslm)
```

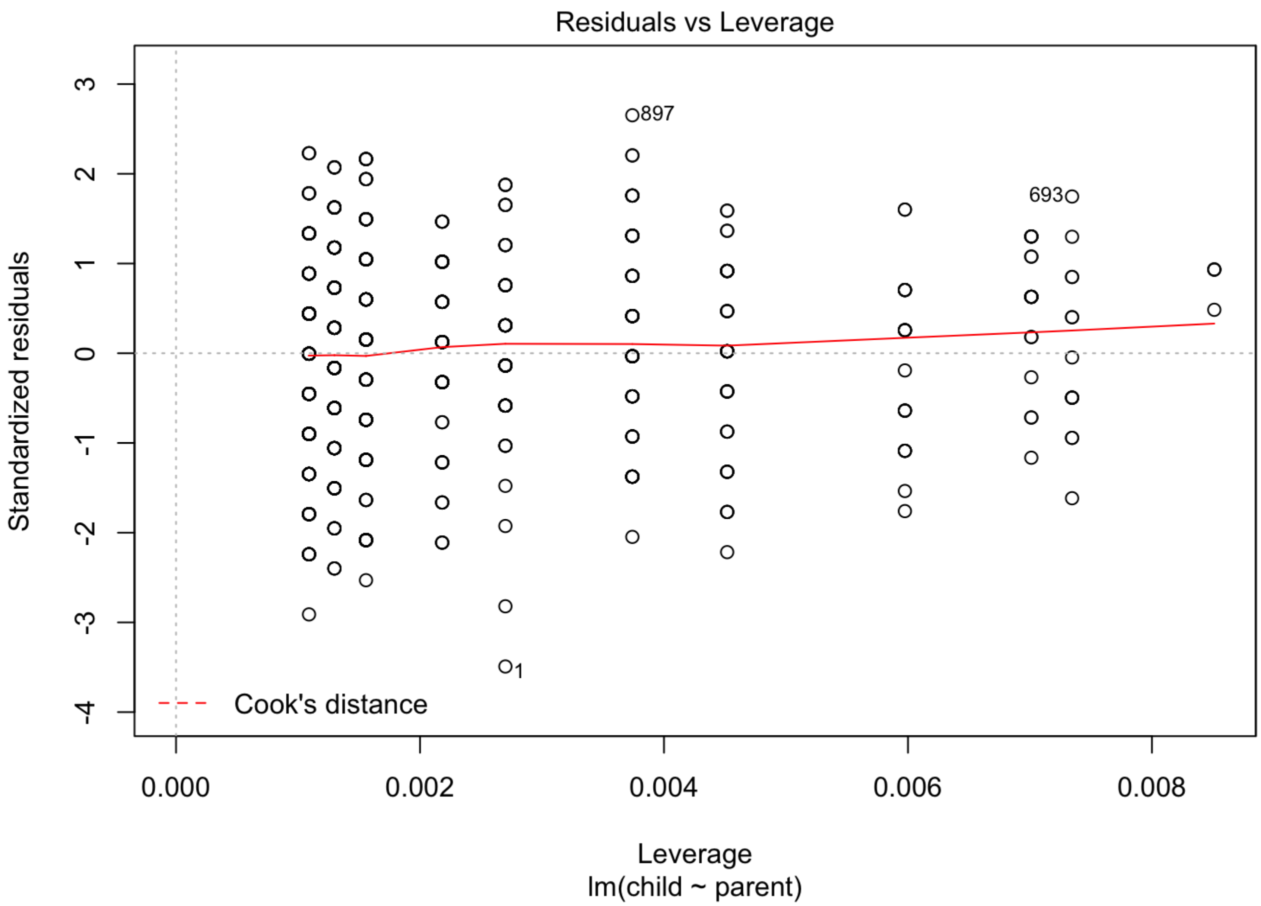

Residuals vs Fitted
Residuals
Fitted values
lm(child ~ parent)
5
0
-5
66
67
68
69
70
71
2
13
1


Normal Q-Q
Standardized residuals
Theoretical Quantiles
3
2
1
0
-1
-2
-3
2
13
1

Theoretical Quantiles
lm(child ~ parent)



Scale-Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
66
67
68
69
70
71
Fitted values
lm(child ~ parent)
1
2
13


Residuals vs Leverage
Standardized residuals
3
2
1
0
-1
-2
-3
-4
897
693
1
Cook's distance
0.000
0.002
0.004
0.006
0.008
Leverage
lm(child ~ parent)

Summary of this Session:

- The `sample` function generates random subsamples of the data.
- It uses a uniform RNG (random number generator) that generates numbers between 0 and 1 at random.
- Other distributions available include the normal, poisson,….
- Some simple tests available are the `t.test`, `chisq.test`.
- `attach` a data set so that all the variables are known to the system as we work with them.
- Output of the functions are often lists with components that can be captured as necessary by their `names`.
- Example of `lm` output for linear regression.
- Along the way, we used a few more of the possibilities of ggplot2 and dplyr.

Followup activity

Visit the RStudio cheatsheets and download the ggplot2 cheatsheet