Mixture Models

Brandon Kozak

10/10/2019

```r
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------------------------

## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ---------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(ggplot2)
library(vcd)
```

```
## Loading required package: grid
```

Intro

Key idea in this chapter will be techniques for dealing with heterogeneity.

Goals

- Learn how to generate our own mixture models from normal populations.
- Learn about the Expectation-Maximization algorithm.
- Look at a special type of mixture called zero-inflation.
- Look at the empirical cumulative distribution
- Build the Laplace distribution
- Look at the gamma-Poisson distribution
- See how mixture models enable us to choose data transformations.

4.2 Finite mixtures

4.2.1 Our first example

We will demo-straight a mixture model with two equal-size components.

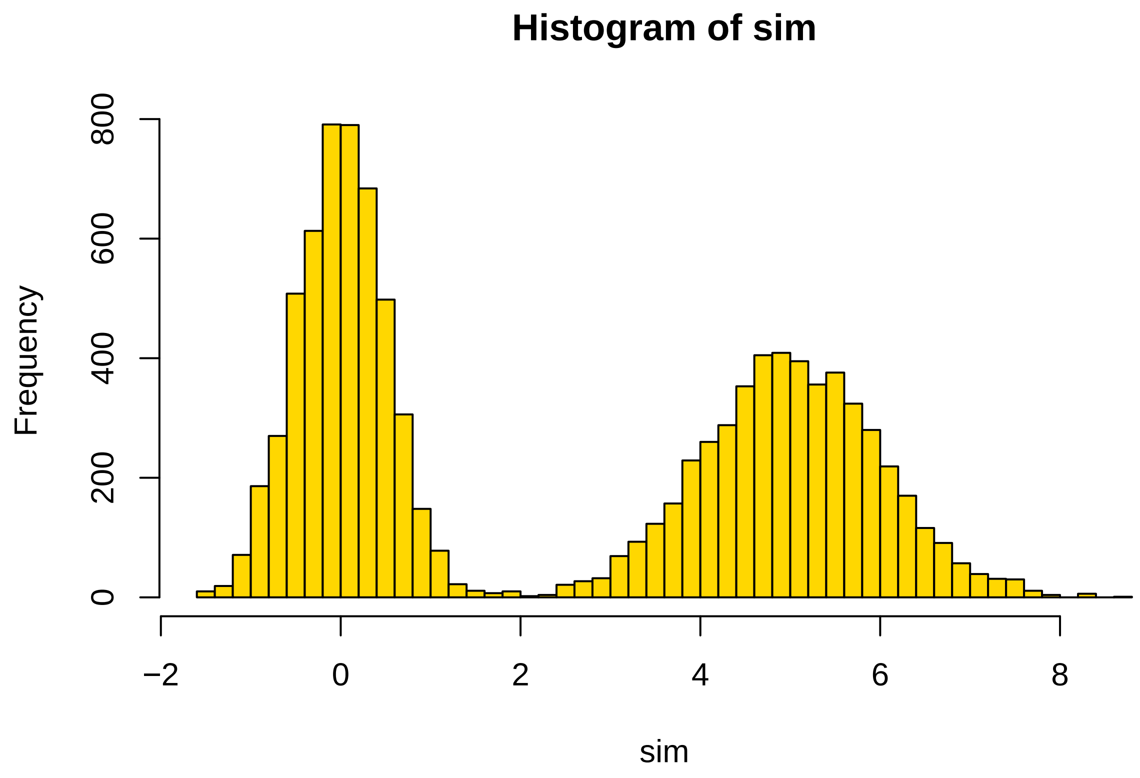
First let us generate 10000 coin flips

```r
set.seed(1234)
coin_flips_1 = runif(10000) > .5
table(coin_flips_1)
```

```
## coin_flips_1
## FALSE  TRUE
##  4990  5010
```

Now based on the outcome of the coin flip, we will sample from a different normal distribution

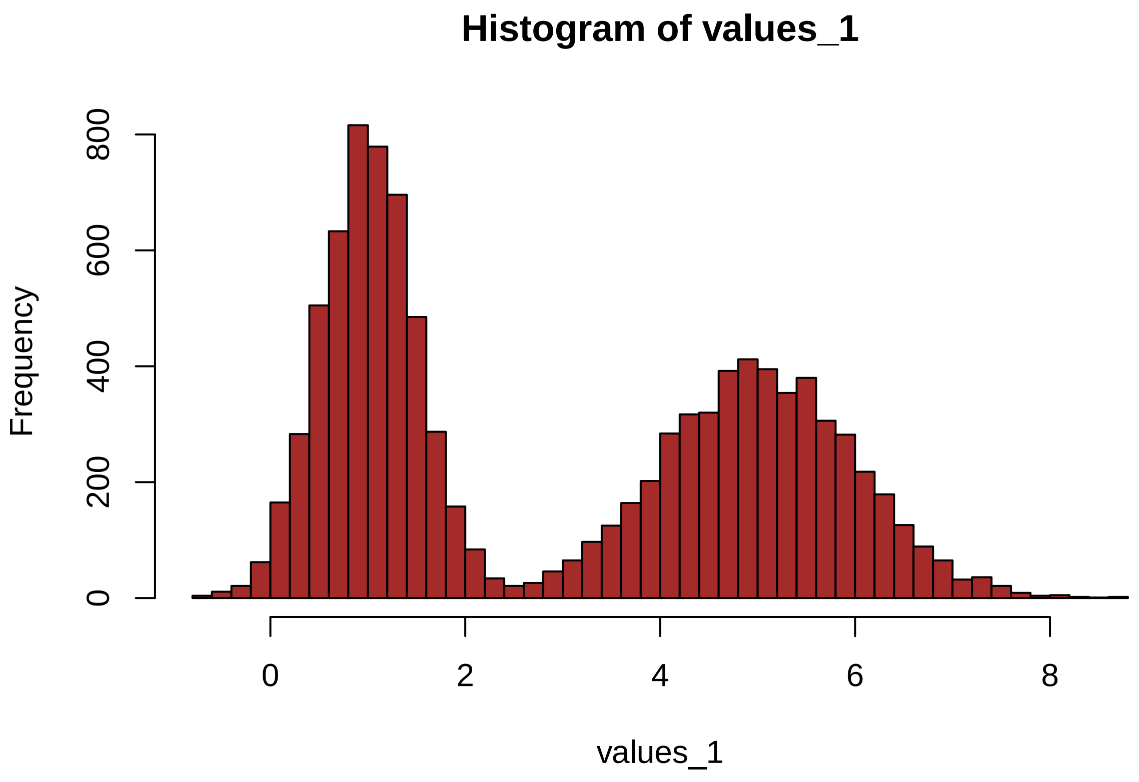
```r
gen_bi_norm = function(flip, mu1=1, mu2=4, sd1=.5, sd2=.5){

  ifelse(flip, rnorm(1,mu1,sd1), rnorm(1,mu2,sd2))

}

sim = vapply(coin_flips_1,gen_bi_norm,0,5,.5,1,FUN.VALUE=1)

hist(sim,50,col="gold")
```



Can we do this without vapply?

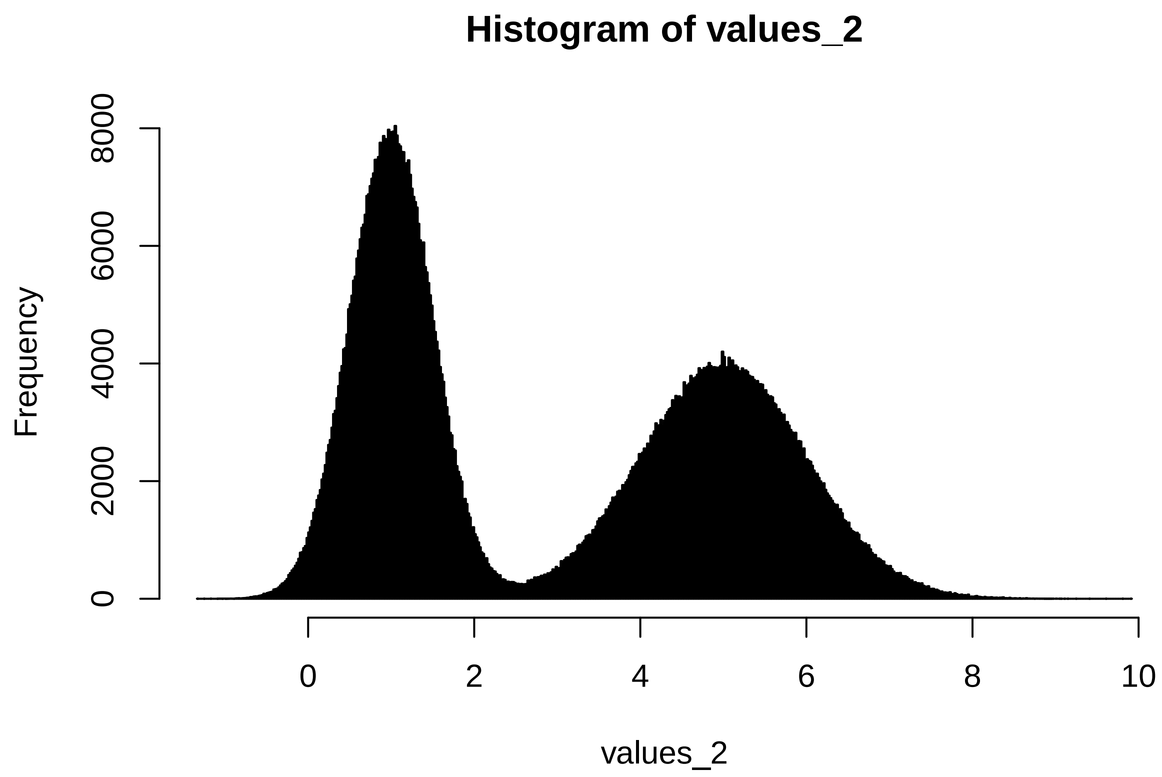
```r
mu = c(1,5)
sigma = c(.5,1)
```

```r
set.seed(1234)
values_1 = rnorm(length(coin_flips_1),
               mean = ifelse(coin_flips_1, mu[1], mu[2]),
               sd = ifelse(coin_flips_1, sigma[1], sigma[2]))
hist(values_1,50,col="brown")
```



Now that we have improved code, lets see what happens when we flip 1 million coins and plot a histogram with 500 bins

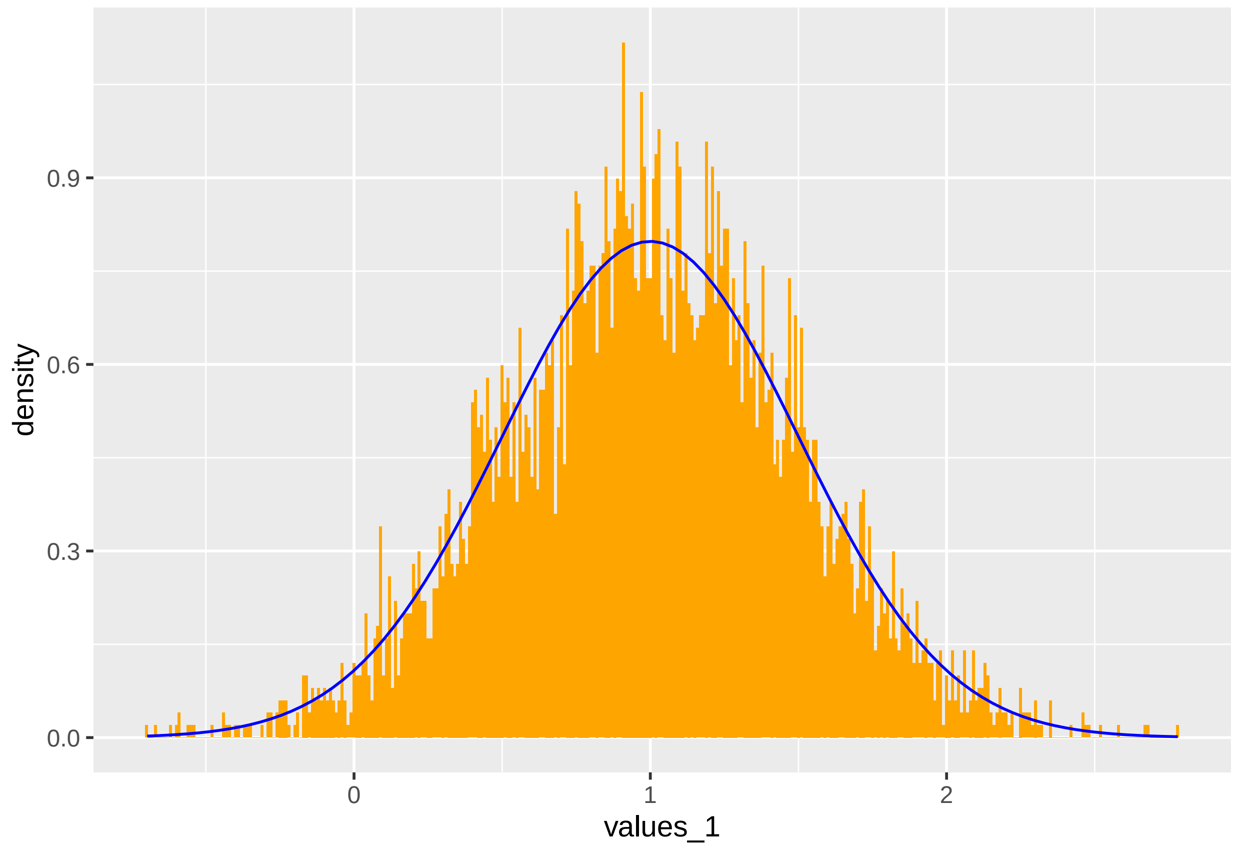
```r
coin_flips_2 = runif(1000000) > .5
mu = c(1,5)
sigma = c(.5,1)
set.seed(1234)
values_2 = rnorm(length(coin_flips_2),
               mean = ifelse(coin_flips_2, mu[1], mu[2]),
               sd = ifelse(coin_flips_2, sigma[1], sigma[2]))
hist(values_2,500)
```



With this many flips and bins we see something that resembles a pdf.

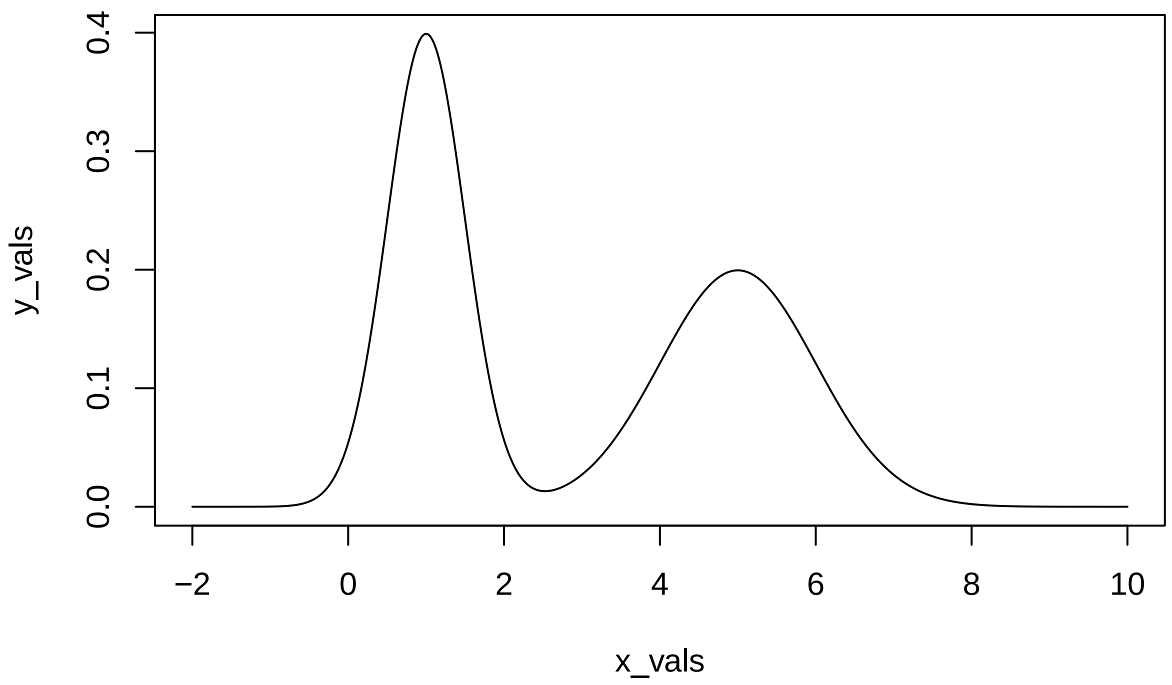
How close will one of the parts resemble a normal dist?

```
flips = tibble(coin_flips_1,values_1)

flips %>% filter(coin_flips_1==TRUE) %>% ggplot(aes(x=values_1)) +
  geom_histogram(aes(y=..density..), fill="orange", binwidth = .01) +
  stat_function(fun = dnorm,
                args = list(mean = mu[1], sd = sigma[1]),
                color = "blue")
```



How can we generate the theoretical dist?

```
x_vals = seq(-2,10,.001)

y_vals = (.5 * dnorm(x_vals, mu[1], sigma[1])) + (.5 * dnorm(x_vals, mu[2], sigma[2]))


plot(x_vals, y_vals, type="l")
```



4.2.2 Discovering the hidden class labels

Here we will be using the expectation-maximization (EM) algorithm to infer the value of the hidden groupings.

Two key components to the procedure:

- pretending to know the probability with which each observation belongs to a component and then estimating the distribution parameters of the components
- pretending to know the parameters of the component distributions and estimating the probability with which each observation belongs to the components

That is we measure a variable Y on a series of objects that we think come from two groups, but are not sure what the labels are. We start by adding a hidden/latent variable to the data set.

Our goal is to find values of U and the unknown parameters of the underlying densities. We will use MLE to estimate the parameters that make the data Y the most likely.

Let's look at an example. Say we have two unfair coins, whose probabilities of heads are p1 = .125 and p2 =.25. With probability π we pick coin 1, with probability $1 - \pi$, coin 2. Then we toss the coin twice and record the number of heads, call it K.

a.) Simulate 100 instances of this procedure with $\pi = \frac{1}{8}$ and compute the contingency tale of K.

```
n = 100
pi = 1/8
```

```r
p1 = .125
p2 = .25

simulate_flip = function(pi,p1,p2){
  rand_nums = runif(3)

  if(rand_nums[1]<pi){
    return(sum(rand_nums[2:3]<p1))
  }
  else{
    return(sum(rand_nums[2:3]<p2))
  }
}
set.seed(1234)
# True = Head, False = Tails
coin_flips = replicate(100,simulate_flip(pi,p1,p2))
table(K=coin_flips)
```

```
## K
##  0  1  2
## 59 35  6
```

b.)

```r
set.seed(1234)
# True = Head, False = Tails
coin_flips = replicate(100,simulate_flip(1/4,p1,p2))
table(K=coin_flips)
```

```
## K
##  0  1  2
## 61 33  6
```

c.)

I do not believe there is any way to obtain/approximate the probabilities by just using K.

Mixture of normal's

Let's start by generating data from a mixture of two normal with unknown means, but variance of 1 for each.

```r
set.seed(1234)
# We are not suppose to know these!
mus = c(-0.5, 1.5)
# Theses are our labels
u = sample(2, 100, replace = TRUE)
# and our outcome values, we set the mean based on the labels
y = rnorm(length(u), mean = mus[u])
# Store in to a tibble
duy = tibble(u, y)
# and lets take a look!
head(duy)
```

```
## # A tibble: 6 x 2
##       u      y
##   <int>  <dbl>
## 1     2 -0.306
## 2     2  0.918
## 3     2  0.391
## 4     2  0.485
## 5     1 -0.662
## 6     2  2.06
```

Now we can use MSE on each labeled data separately. Hence, since we know the data comes from a normal distribution, we can use x-bar to estimate the mean for each group.

```
duy %>% group_by(u) %>% summarise(mean = mean(y))
```

```
## # A tibble: 2 x 2
##       u   mean
##   <int>  <dbl>
## 1     1 -0.479
## 2     2  1.62
```

We can take a look at packages meant for EM, like mixtools.

```
library("mixtools")
```

```
## mixtools package, version 1.1.0, Released 2017-03-10
## This package is based upon work supported by the National Science Foundation under Grant No. SES-0518

##
## Attaching package: 'mixtools'

## The following object is masked from 'package:grid':
##
##     depth
```

```
set.seed(1234)
y = c(rnorm(100, mean = -0.2, sd = 0.5),
      rnorm( 50, mean =  0.5, sd =   1))
gm = normalmixEM(y, k = 2, lambda = c(0.5, 0.5),
     mu = c(-0.01, 0.01), sigma = c(1, 1))
```

```
## number of iterations= 151
```

```
gm$lambda
```

```
## [1] 0.3747345 0.6252655
```

```
gm$mu
```

```
## [1] -0.5042110  0.2825762
```

```
gm$sigma
```

```
## [1] 0.2746151 0.7768337
```

```
gm$loglik
```

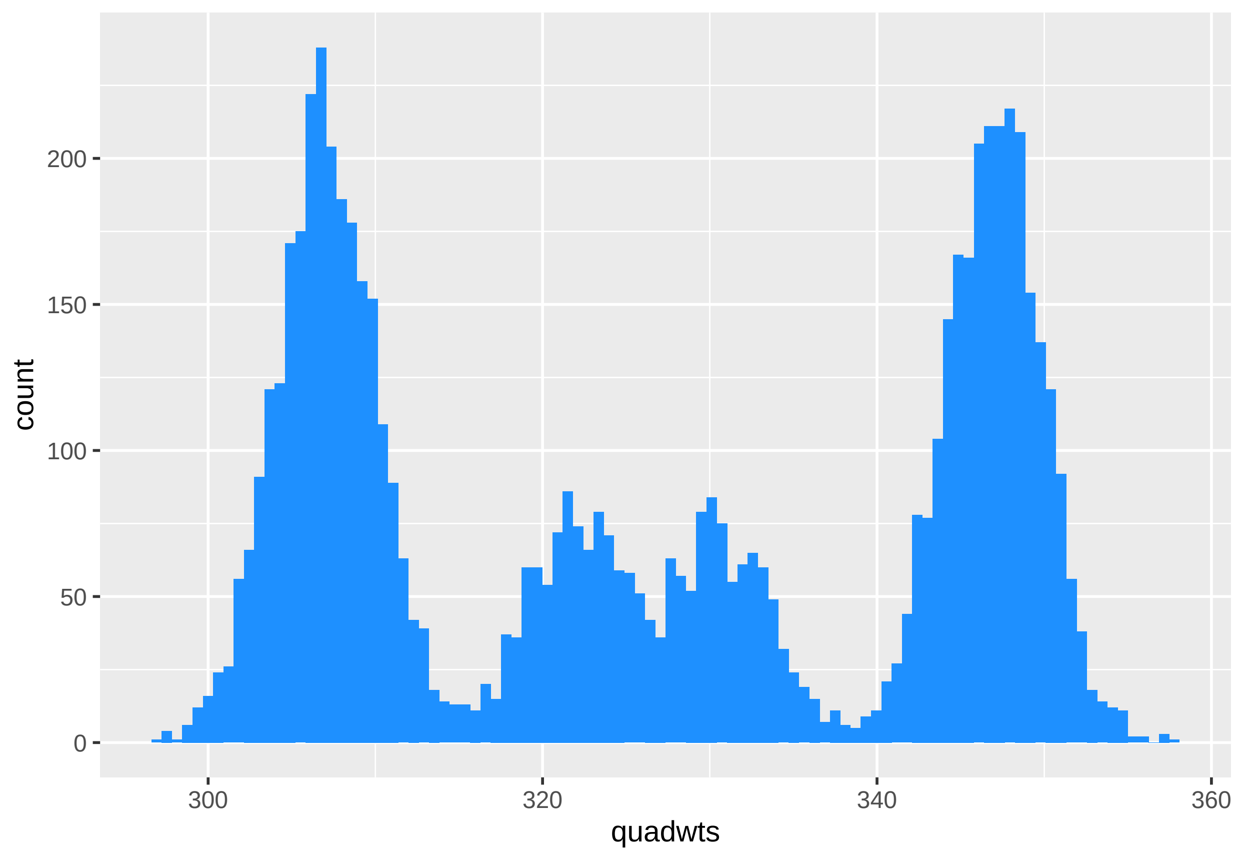
```
## [1] -156.4786
```

Models for zero inflated data

Will investigate this later, need to make sure I have the proper files. # {r} # library(mosaics) # library(mosaicsExample) # # constructBins(infile = system.file(file.path("extdata", "wgEncodeSydhTfbsGm12878Stat1StdAlnRep1_chr22_sorted.bam"), package="mosaicsExample"), # fileFormat = "bam", outfileLoc = "../data/", # byChr = FALSE, useChrfile = FALSE, chrfile = NULL, excludeChr = NULL, # PET = FALSE, fragLen = 200, binSize = 200, capping = 0) # constructBins(infile = system.file(file.path("extdata", "wgEncodeSydhTfbsGm12878InputStdA package="mosaicsExample"), # fileFormat = "bam", outfileLoc = "../data/", # byChr = FALSE, useChrfile = FALSE, chrfile = NULL, excludeChr = NULL, # PET = FALSE, fragLen = 200, binSize = 200, capping = 0) # datafiles = c("../data/wgEncodeSydhTfbsGm12878Stat1StdAlnRep1_chr2 # "../data/wgEncodeSydhTfbsGm12878InputStdAlnRep1_chr22_sorted.bam_fragL200_bin200.txt") # binTFBS = readBins(type = c("chip","input"), fileName = datafiles) # #

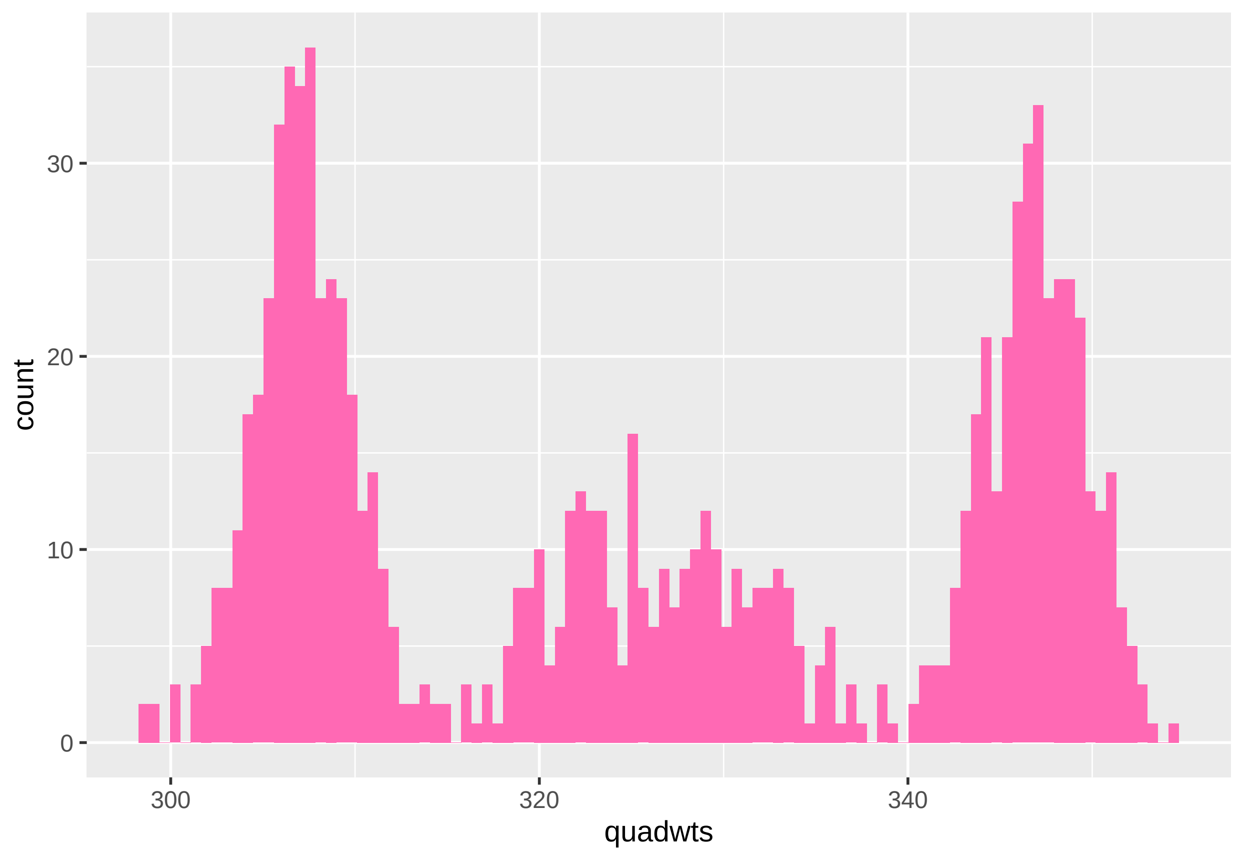
4.2.4 More than two components

```
set.seed(1234)
# effecitvely the means
masses = c(A =  331, C =  307, G =  347, T =  322)
# the chance of certain types
probs  = c(A = 0.12, C = 0.38, G = 0.36, T = 0.14)
# total number of nucleotides
N  = 7000
# constant sd for each type
sd = 3
# using the probabilities, sample with replacment N times
nuclt   = sample(length(probs), N, replace = TRUE, prob = probs)
# Now generate the data using the means/masses
quadwts = rnorm(length(nuclt),
                mean = masses[nuclt],
                sd   = sd)
# and plot
ggplot(tibble(quadwts = quadwts), aes(x = quadwts)) +
  geom_histogram(bins = 100, fill = "dodgerblue")
```



What if we set N to 1000?

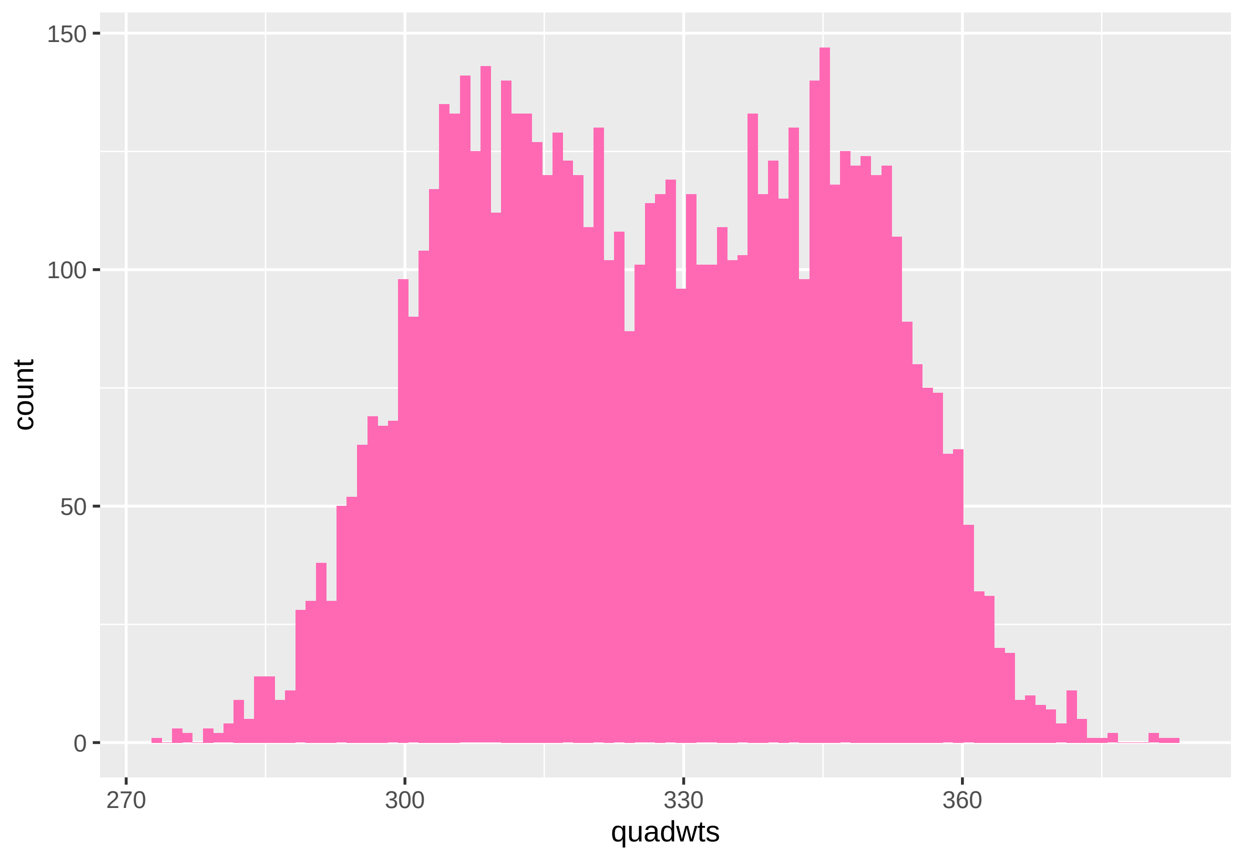
```
set.seed(1234)
N = 1000
nuclt   = sample(length(probs), N, replace = TRUE, prob = probs)
# Now generate the data using the means/masses
quadwts = rnorm(length(nuclt),
                mean = masses[nuclt],
                sd   = sd)
# and plot
ggplot(tibble(quadwts = quadwts), aes(x = quadwts)) +
  geom_histogram(bins = 100, fill = "hotpink")
```



We see that the histogram look less defined.

What about if we leave N=7000 but sd=10?

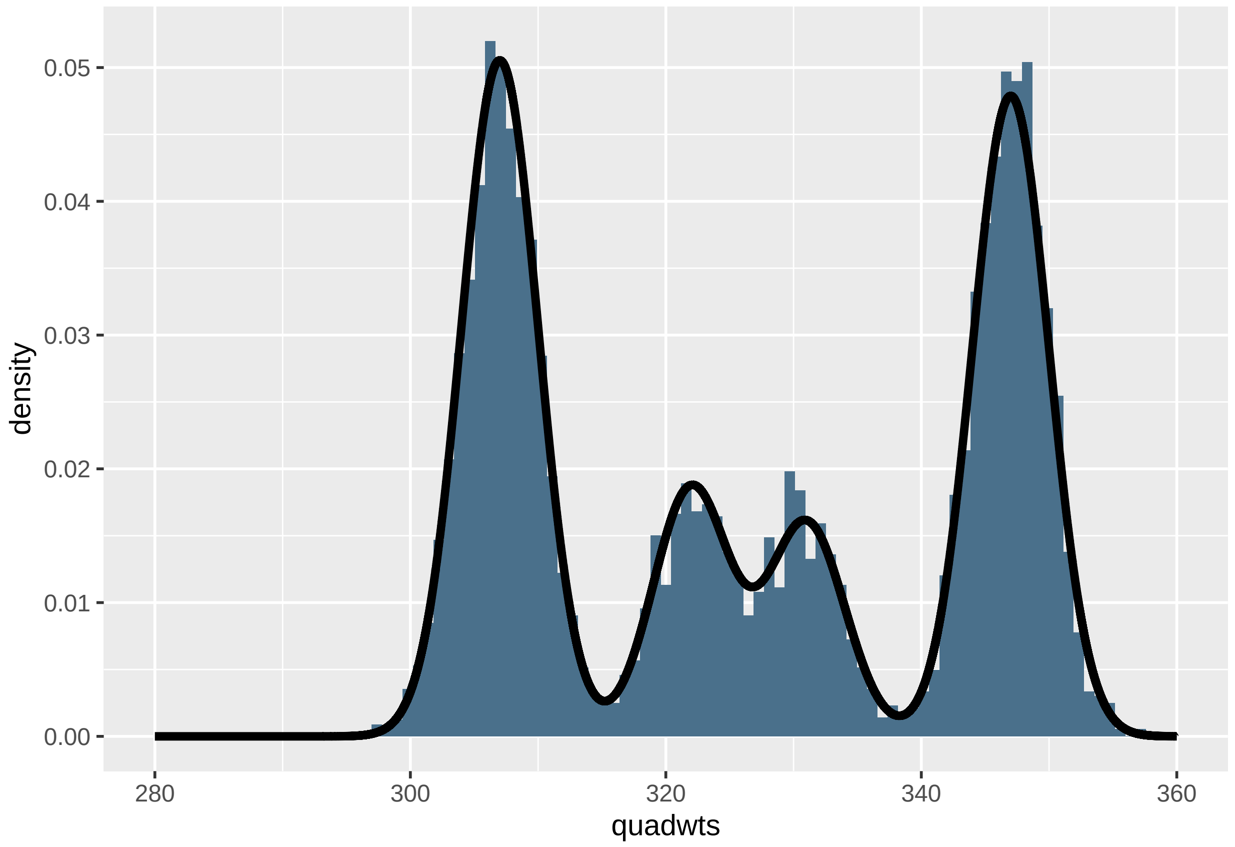
```r
set.seed(1234)
N = 7000
sd=10
nuclt   = sample(length(probs), N, replace = TRUE, prob = probs)
# Now generate the data using the means/masses
quadwts = rnorm(length(nuclt),
                mean = masses[nuclt],
                sd   = sd)
# and plot
ggplot(tibble(quadwts = quadwts), aes(x = quadwts)) +
  geom_histogram(bins = 100, fill = "hotpink")
```



We see that since the sd is higher, each individual dist is more spread out causing them to “leak” into each other. This makes it harder to tell that there are 4 components at play here.

Can we plot the theoretical density curve?

```
set.seed(1234)
N = 7000
sd=3
nuclt   = sample(length(probs), N, replace = TRUE, prob = probs)
# Now generate the data using the means/masses
quadwts = rnorm(length(nuclt),
                mean = masses[nuclt],
                sd   = sd)
# Generate theoretical values

x_vals = seq(280,360,.001)

y_vals = probs[[1]] * dnorm(x_vals,masses[[1]],sd) +
         probs[[2]] * dnorm(x_vals,masses[[2]],sd) +
         probs[[3]] * dnorm(x_vals,masses[[3]],sd) +
         probs[[4]] * dnorm(x_vals,masses[[4]],sd)

den_data = tibble(x_vals,y_vals)

# and plot
ggplot(tibble(quadwts = quadwts), aes(x = quadwts, y=..density..)) +
  geom_histogram(bins = 100, fill = "skyblue4", position = "identity") +
  geom_line(data=den_data,aes(x=x_vals,y=y_vals),color="black",size=1.5)
```



4.3 Empirical distributions and the nonparmaertic bootstrap

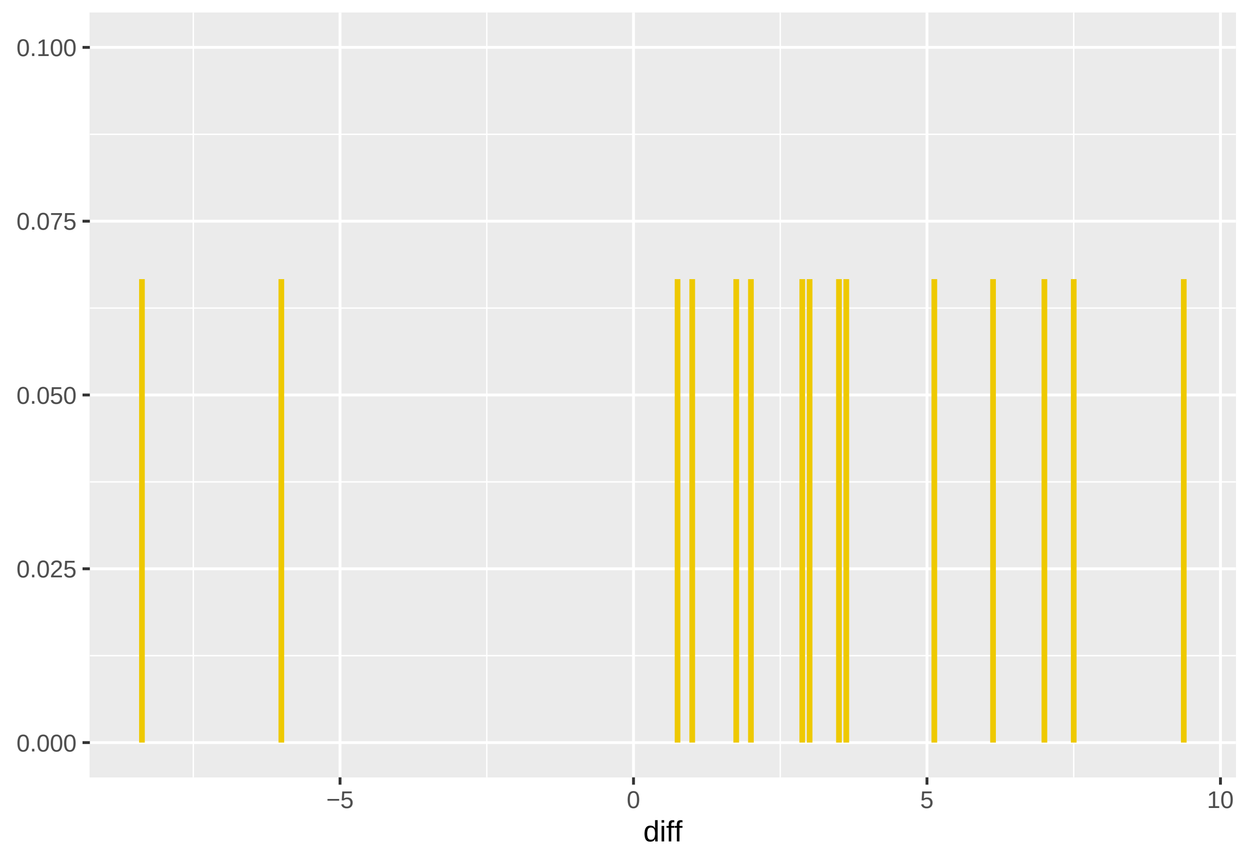
It's often very difficult to obtain the exact sampling distribution of a statistic. In theses cases we can use boot strap to gain an approximation to this sampling distribution.

Lets use this idea to obtain the sampling dist of the median using Darwin's Zea Mays data set.

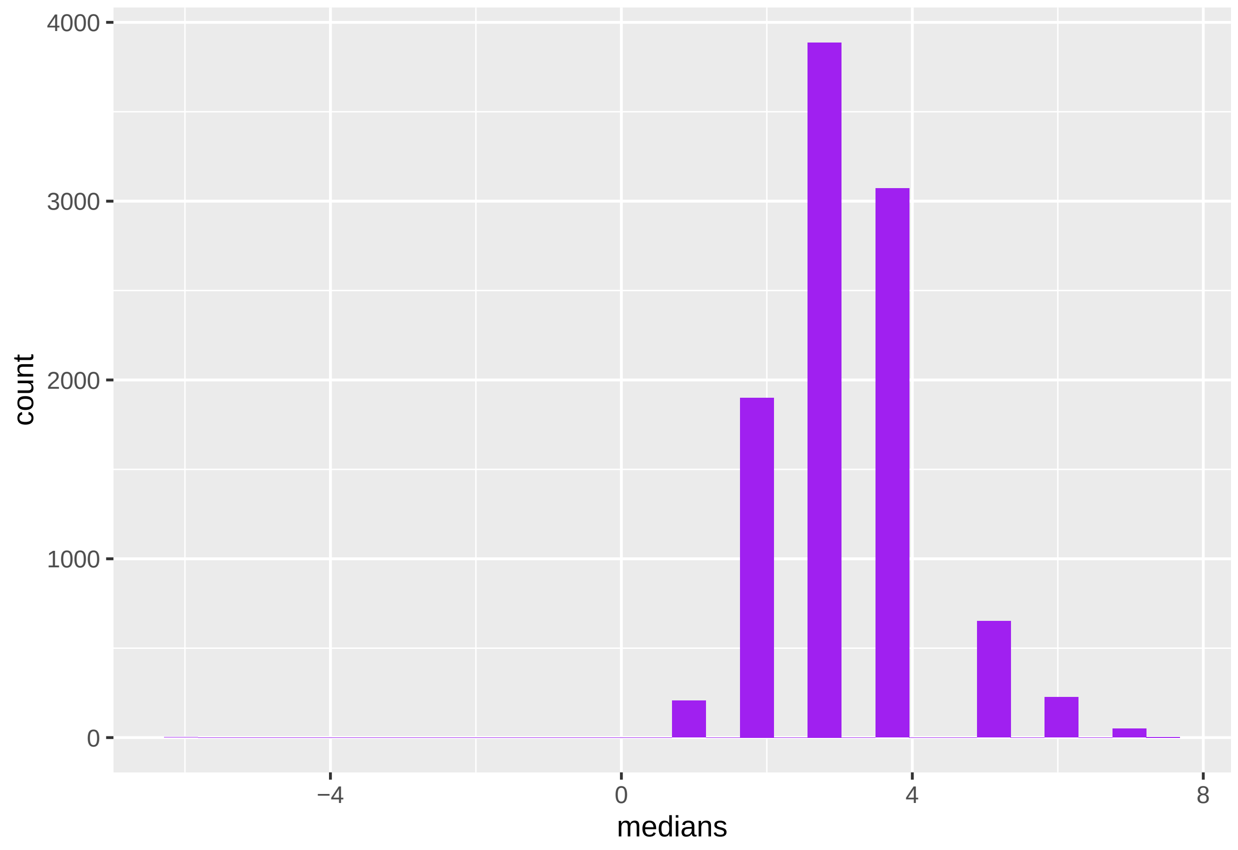
```
set.seed(1234)
library("HistData")
ZeaMays$diff
```

```
##  [1]  6.125 -8.375  1.000  2.000  0.750  2.875  3.500  5.125  1.750  3.625
## [11]  7.000  3.000  9.375  7.500 -6.000
```

```
ggplot(ZeaMays, aes(x = diff, ymax = 1/15, ymin = 0)) +
  geom_linerange(size = 1, col = "gold2") + ylim(0, 0.1)
```



```
# Boot strap samples to find the sampling distrubtion of the median
B = 10000
meds = replicate(B, {
  i = sample(15, 15, replace = TRUE)
  median(ZeaMays$diff[i])
})
ggplot(tibble(medians = meds), aes(x = medians)) +
  geom_histogram(bins = 30, fill = "purple")
```

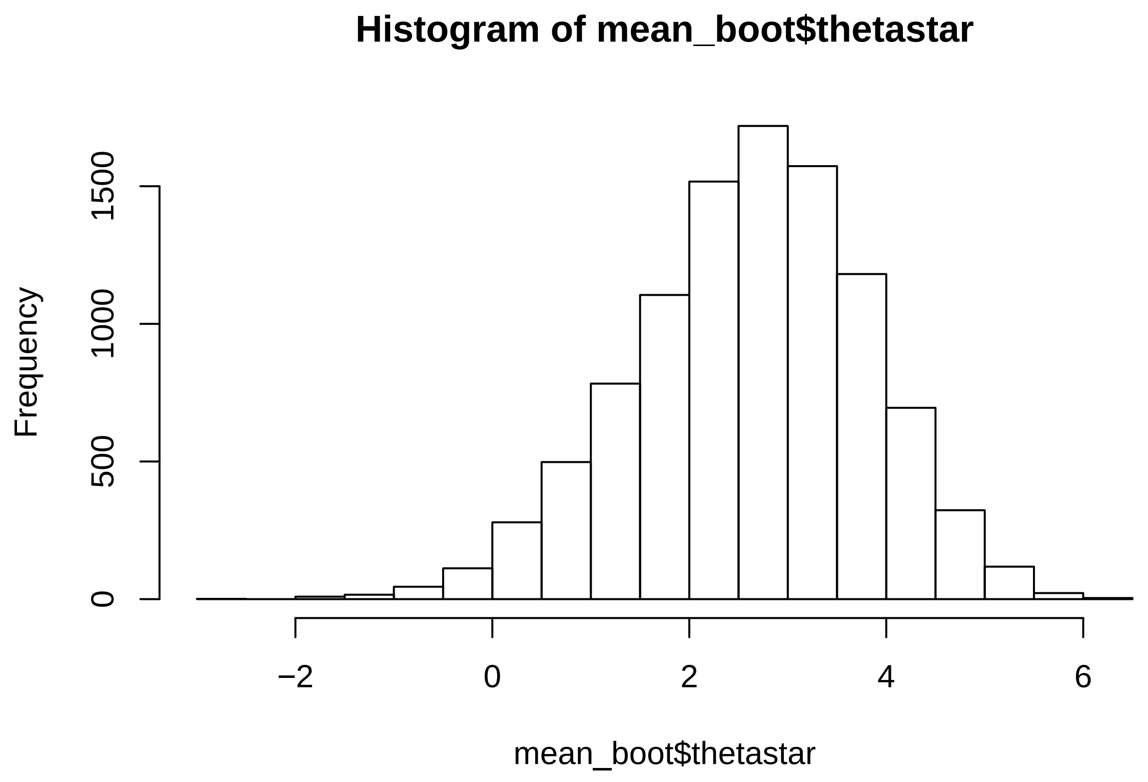


Then based on the boot strap samples, we can get a 99 percent CI for the median.

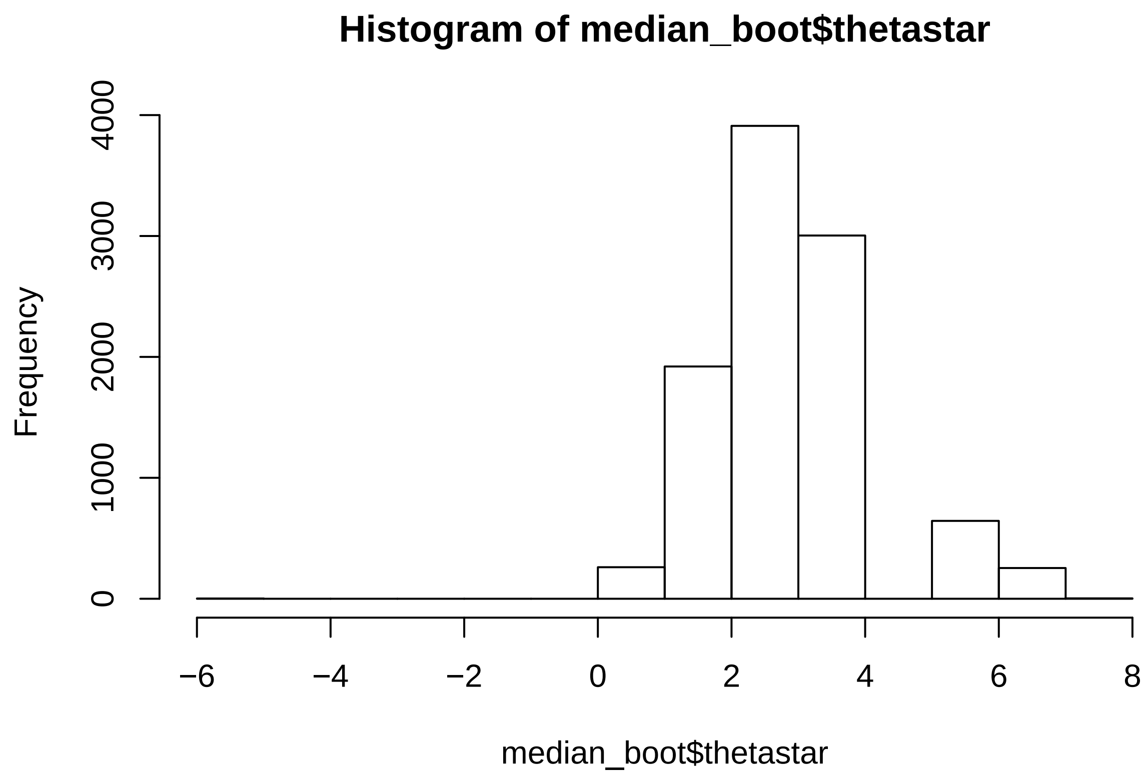
```
ci = quantile(meds,c(.01,.99))
```

We can do the same with the bootstrap package. Lets also look at the sample mean too!

```
set.seed(1234)
library("bootstrap")
mean_boot = bootstrap(ZeaMays$diff, B, mean)
median_boot = bootstrap(ZeaMays$diff, B, median)

hist(mean_boot$thetastar)
```



```r
hist(median_boot$thetastar)
```



```r
cat("\n99% CI Mean\n")
```

```
##
## 99% CI Mean
```

```r
quantile(mean_boot$thetastar,c(.01,.99))
```

```
##       1%      99%
## -0.34175  5.15000
```

```r
cat("\n99% CI Median\n")
```

```
##
## 99% CI Median
```

```r
quantile(median_boot$thetastar,c(.01,.99))
```

```
##    1%   99%
## 1.000 6.125
```

More on boot strap, one of the questions that comes up is **How many bootstap samples are possible given a sample size of n**.

To answer this we seek a little bit of combinatorics and the idea of **stars and bars**.

First we see that taking a sample with replacement is equivalent to placing n indistinguishable balls into n distinguishable bins.

Lets consider the n=3 case:

ooo||

Is one way we can place the balls, if we look a bit further we can see that we are really arranging 3 balls and 2 bars.

If we consider how many ways to arrange the bars, we have $\binom{2+3}{3} = 10$ ways, but notice that by doing this, we automatically have placed the balls and thus this number actually tells us the number of ways to arrange both the bars and the balls, and thus the number of distinct boot strap samples.

We can make this more general and write the formula as $\binom{n+n-1}{n} = \binom{2n-1}{n}$

Thus, for n=15 we get $\binom{29}{15} = 77,558,760$

Another question is **what types of errors happen when we do bootstrap?**

We classify the error into two groups:

- 1. Simulation error, this error comes from the nature of re-sampling, we can minimize this error by increasing the number of boot strap samples.
- 2. The error that comes from using the sample rather than the population. This error can not be lowered, but the idea is that we assume we have a good sample from the start, in order keep this error small. However, if we do not have a good sample, then there is nothing we can do to reduce this type of error.

4.4 Infinite mixtures

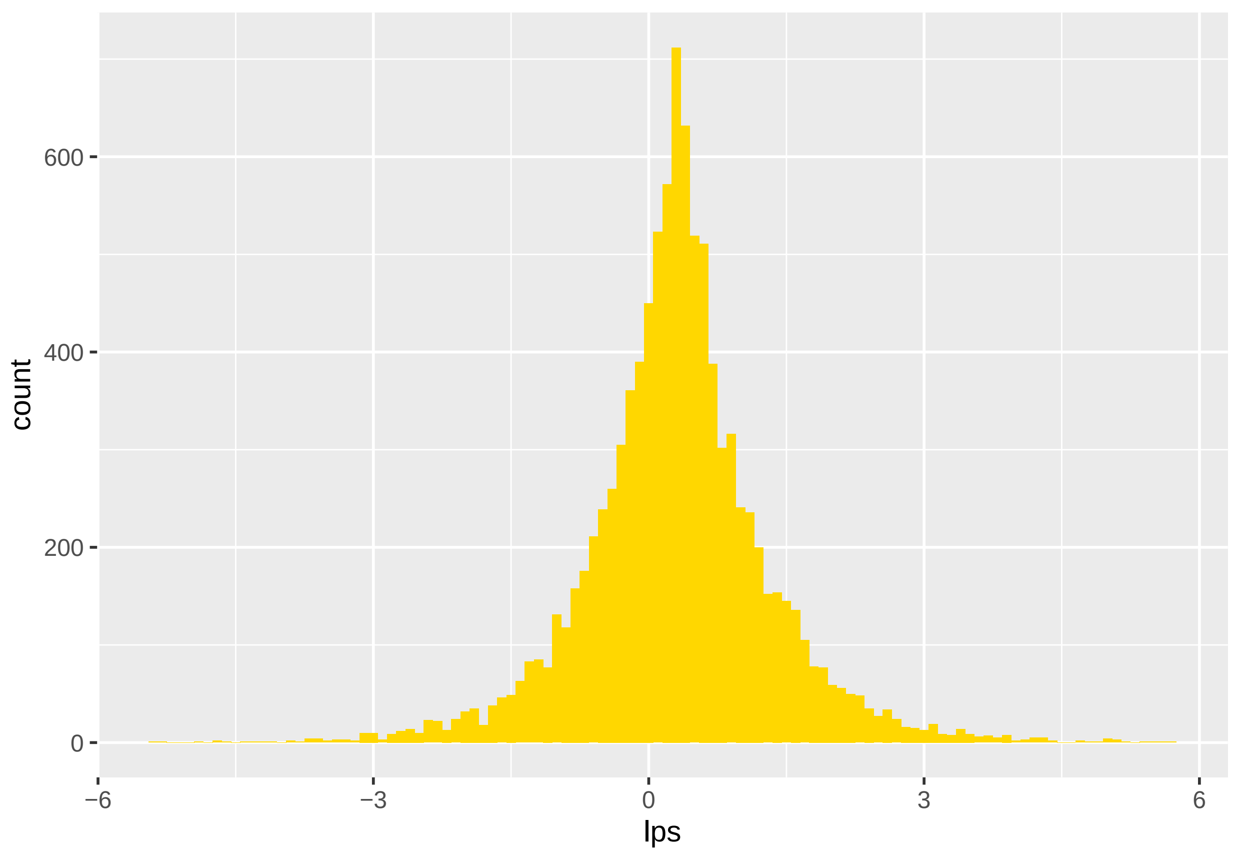
Key idea here is that we can allow as many labels as there are observations. If the number of mixture components is as big as (or bigger than) the number of observations, then we say we have an infinite mixture.

4.4.1 The case for normal

4.4.1 Infinite mixture of normal

Consider the following 2-level data generating scheme

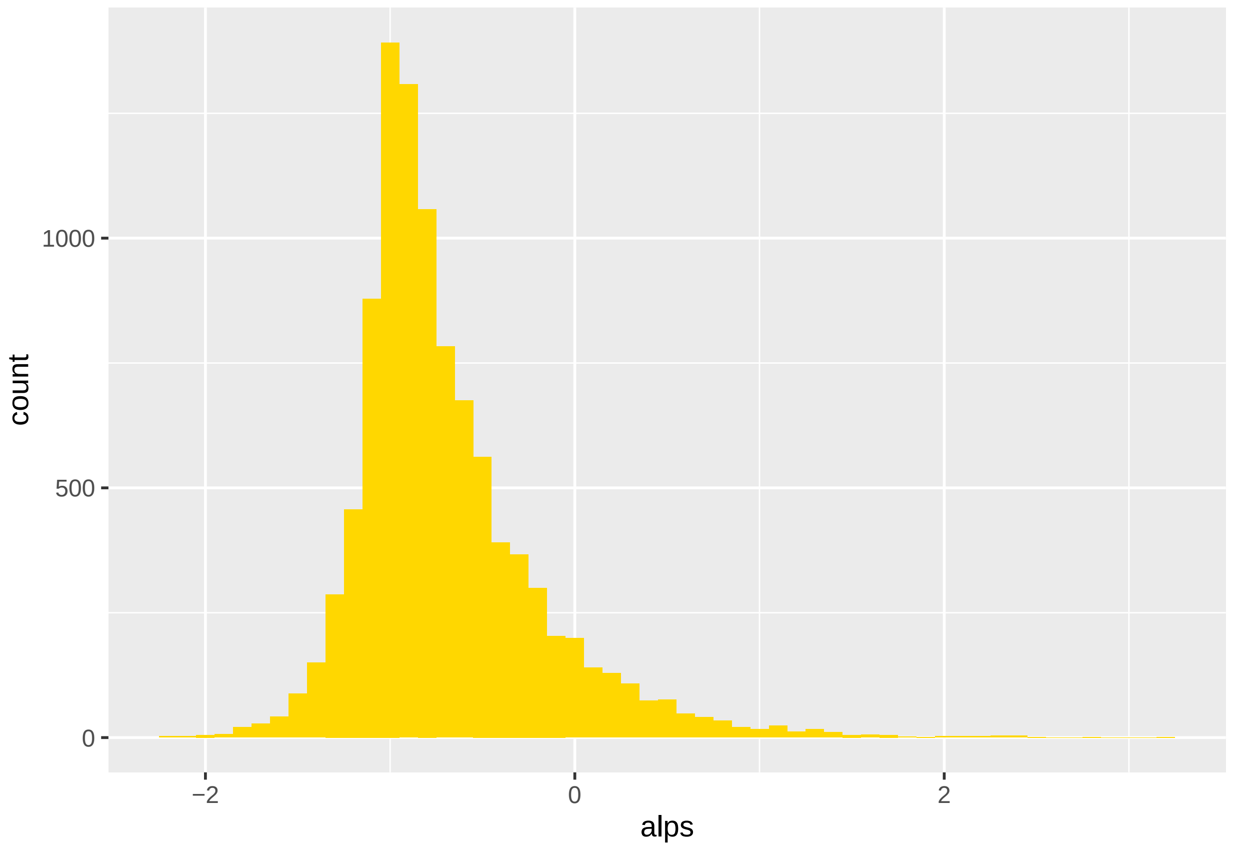
```r
set.seed(1234)
# Level One, create a sample from an exponential dist
w = rexp(10000, rate = 1)
# Level Two, take values from a Normal(.3, sqrt(w))
mu  = 0.3
lps = rnorm(length(w), mean = mu, sd = sqrt(w))
ggplot(data.frame(lps), aes(x = lps)) +
  geom_histogram(fill = "gold", binwidth = 0.1)
```



Turns out that this is known as the Laplace distribution.

We can also look at the asymmetric Laplace

```r
set.seed(1234)
# Constants to transform our mean and varaince
mu = 0.3; sigma = 0.4; theta = -1
w  = rexp(10000, 1)
alps = rnorm(length(w), theta + mu * w, sigma * sqrt(w))
ggplot(tibble(alps), aes(x = alps)) +
  geom_histogram(fill = "gold", binwidth = 0.1)
```



I believe the question here refers to log transformations.

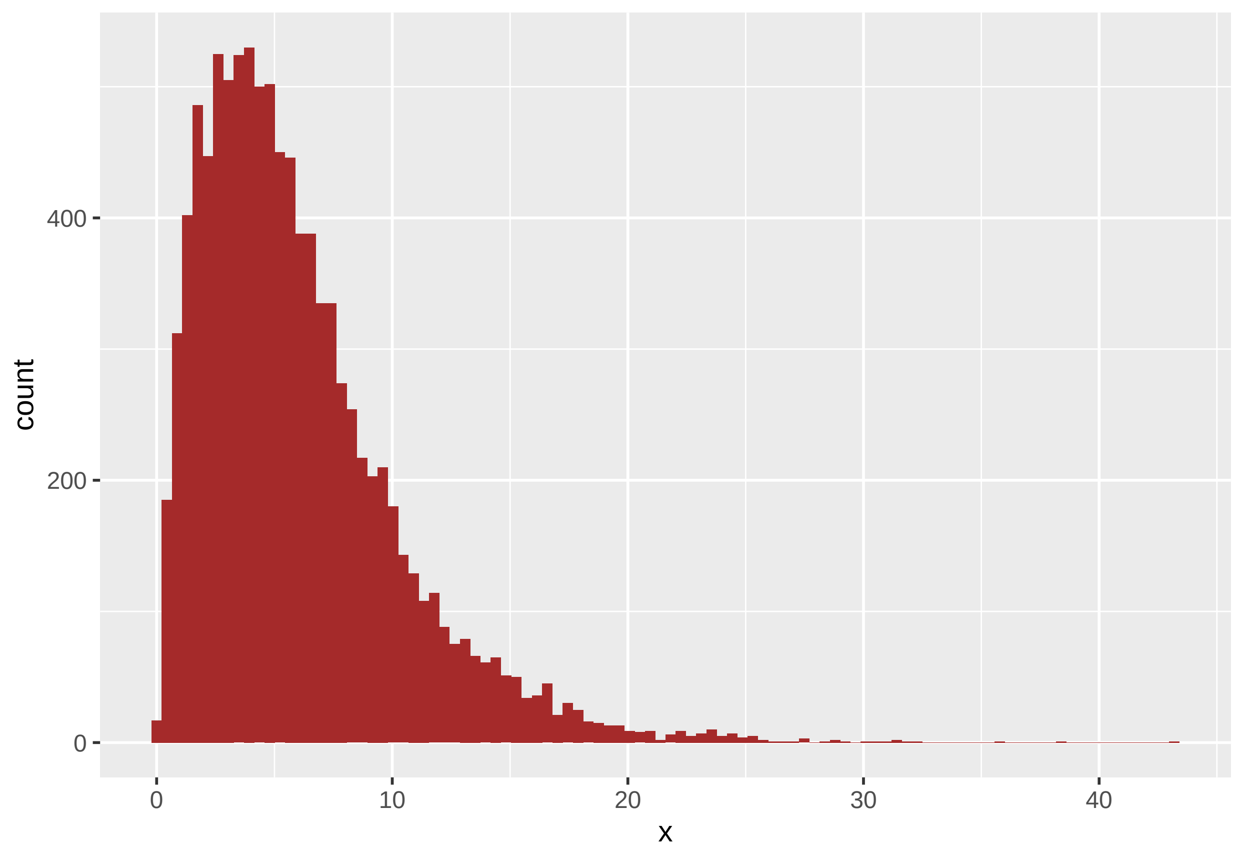
4.4.2 The Poisson case

General idea is that we allow our λ parameter to be a random variable that follows some distribution. We will see more of this in chapter 8.

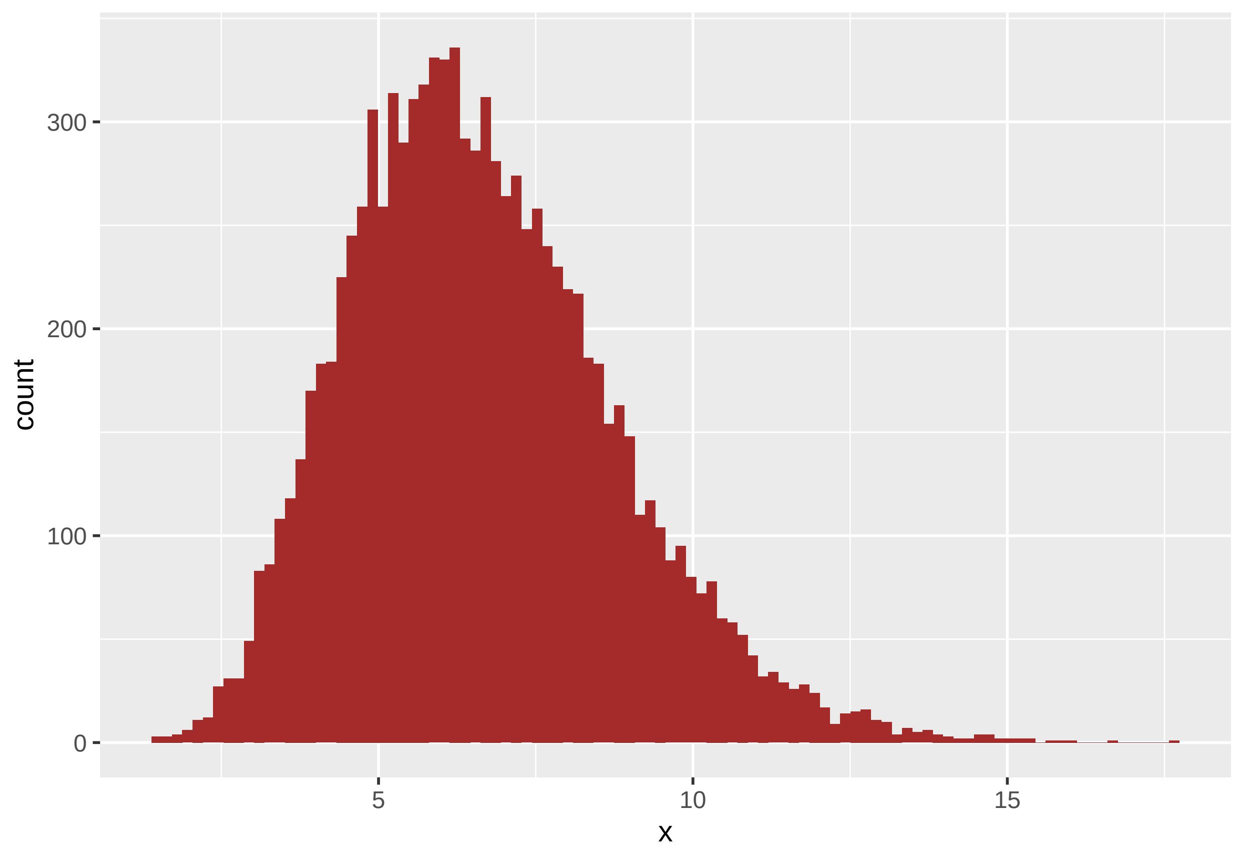
4.4.3 The Gamma case

Here we talk about the Gamma distribution, it has two parameters α and β and takes values on $(0, \inf)$ Here are some examples

```r
set.seed(1234)
ggplot(tibble(x = rgamma(10000, shape = 2, rate = 1/3)),
   aes(x = x)) + geom_histogram(bins = 100, fill= "brown")
```



```
ggplot(tibble(x = rgamma(10000, shape = 10, rate = 3/2)),
   aes(x = x)) + geom_histogram(bins = 100, fill= "brown")
```

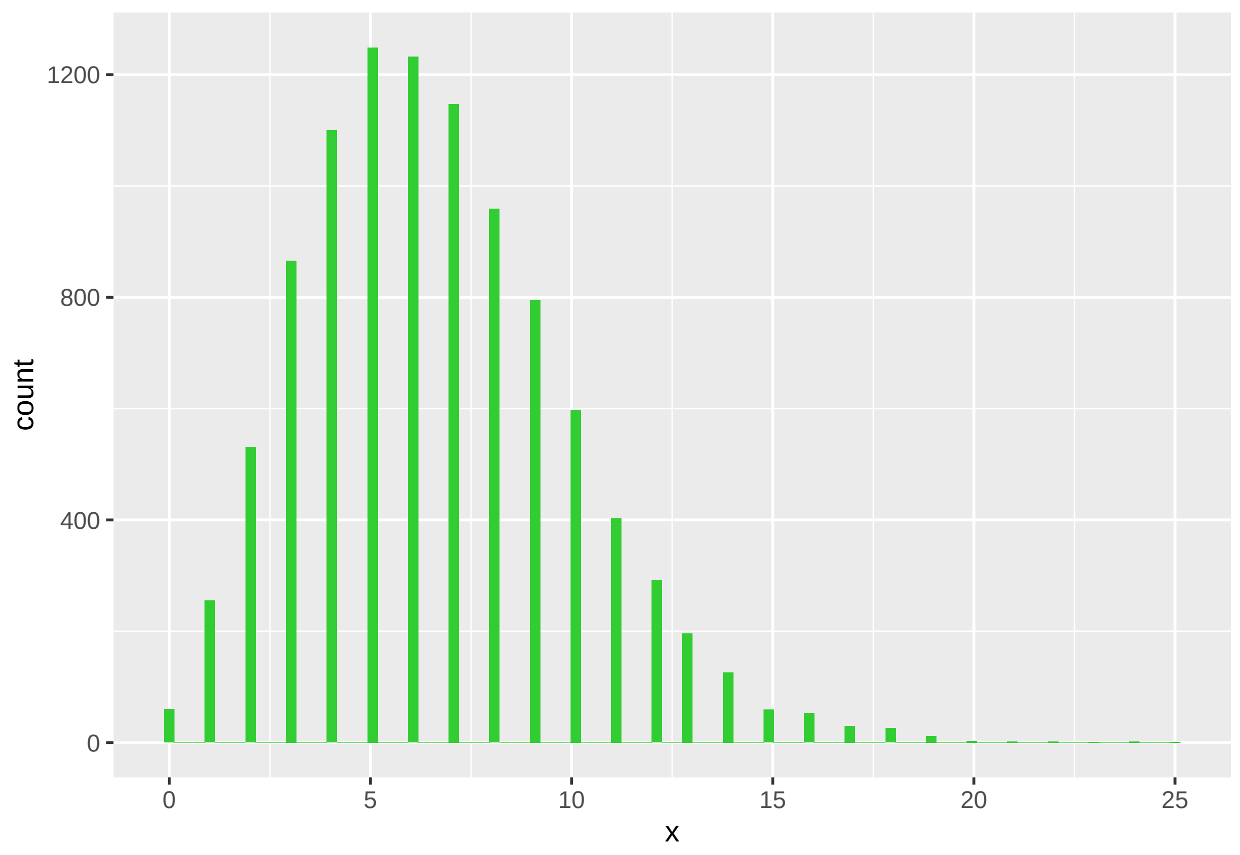


Now, to sample from a Gamma-Poisson mixture, we do two steps:

- 1.) Generate a set of parameters $\lambda_1, \lambda_2, \ldots$ from a gamma distribution.
- 2.) Use each parameter to generate a set of Poisson(λ_i) random variables.

For example

```r
set.seed(1234)
# our parameters
lambda = rgamma(10000, shape = 10, rate = 3/2)
# our values
gp = rpois(length(lambda), lambda = lambda)
# and the plot!
ggplot(tibble(x = gp), aes(x = x)) +
  geom_histogram(bins = 100, fill= "limegreen")
```



Question time!

a.) The end values are discrete, even though we start with continuous parameters.

b.) It can be shown that this process is the same as taking from a **Negative Binomial Distribution**.

Analytic derivations

- I'll leave this for my stat courses.

4.4.4 Variance stabilizing transformations

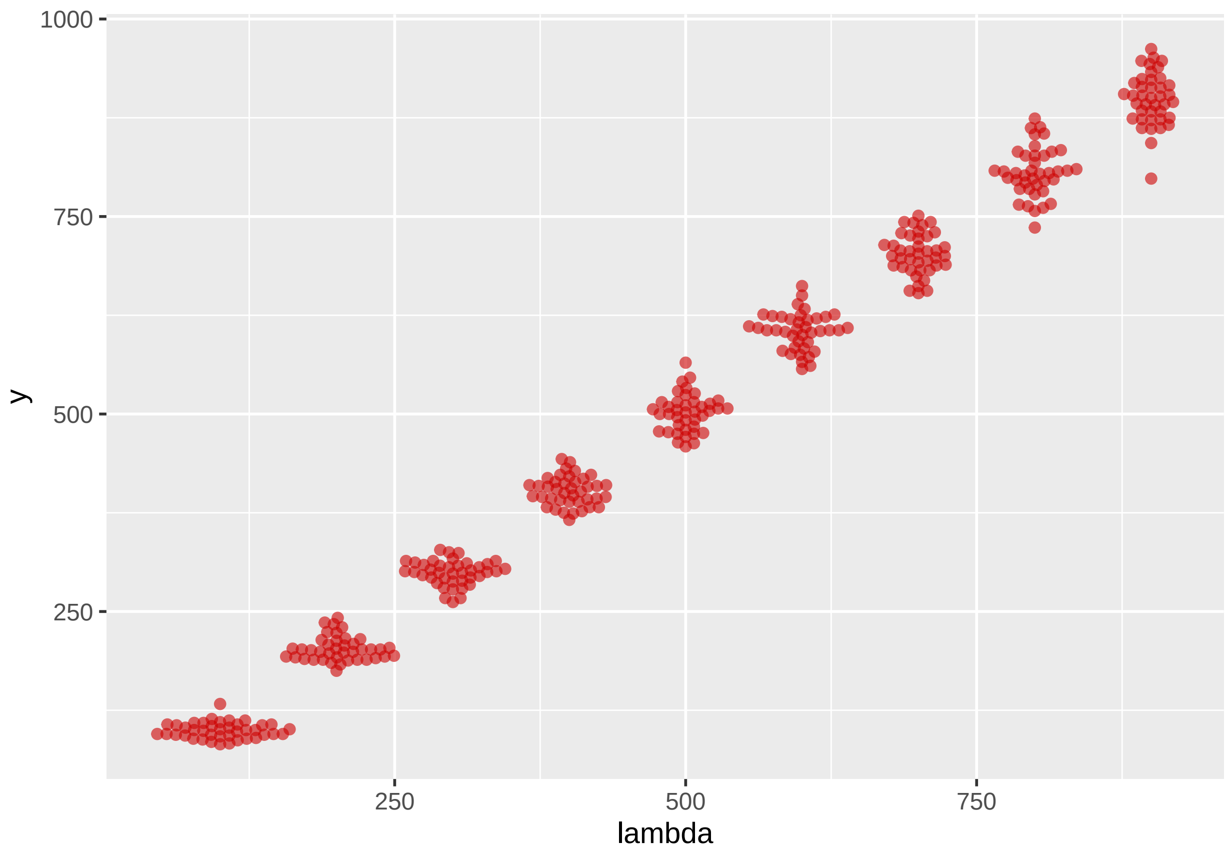
Key issue, how much variability is there between repeated measurement?

If the variability differs by alot, it becomes harder to test for true differences.

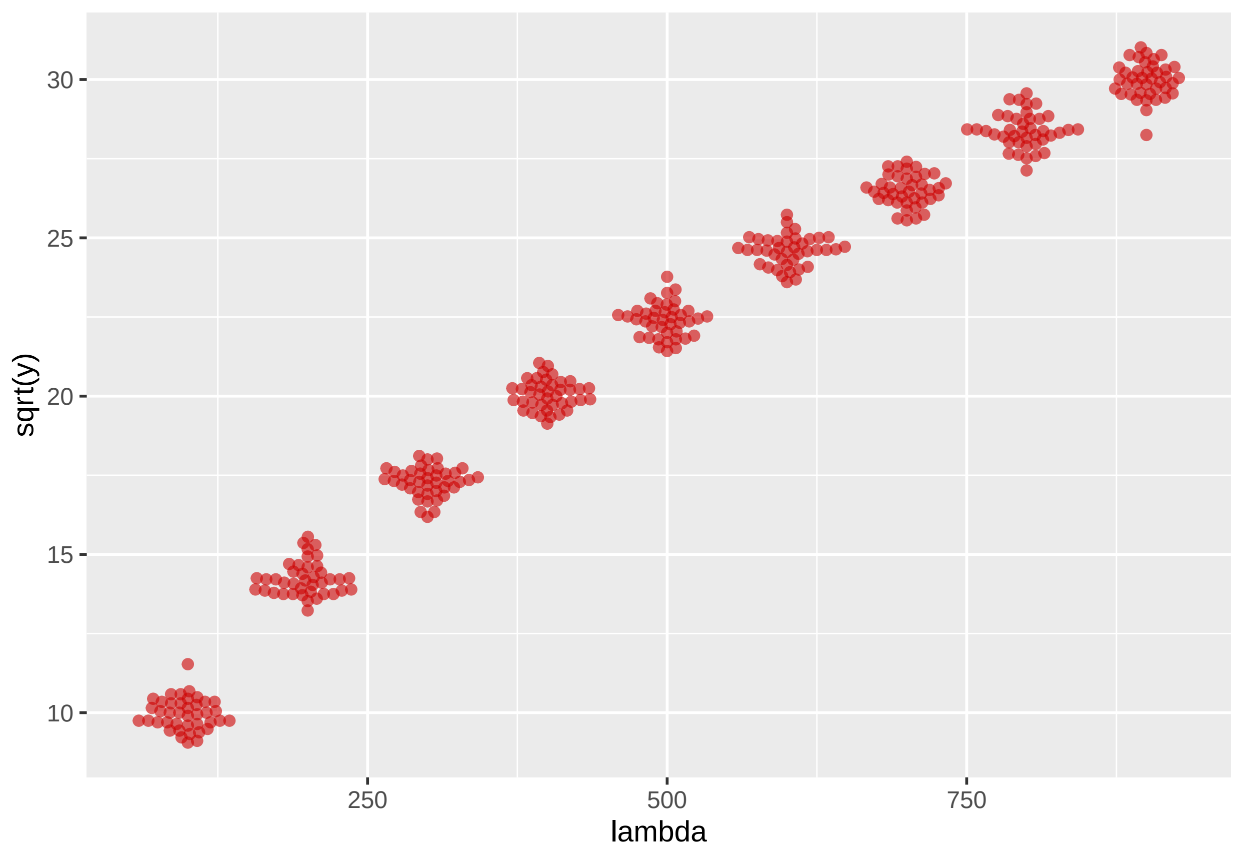
One way to overcome this is by using **Variance stabilizing transformations**

```r
set.seed(1234)
# Values of lambda
lambdas = seq(100, 900, by = 100)

# Now for each lambda, generate 40 random pois values.
simdat = lapply(lambdas, function(l)
    tibble(y = rpois(n = 40, lambda=l), lambda = l)
  ) %>% bind_rows

library("ggbeeswarm")
```

```
ggplot(simdat, aes(x = lambda, y = y)) +
  geom_beeswarm(alpha = 0.6, color = "red3")
```



```
ggplot(simdat, aes(x = lambda, y = sqrt(y))) +
  geom_beeswarm(alpha = 0.6, color = "red3")
```



We can try the transformation $2\sqrt{y}$ and then look at the numbers

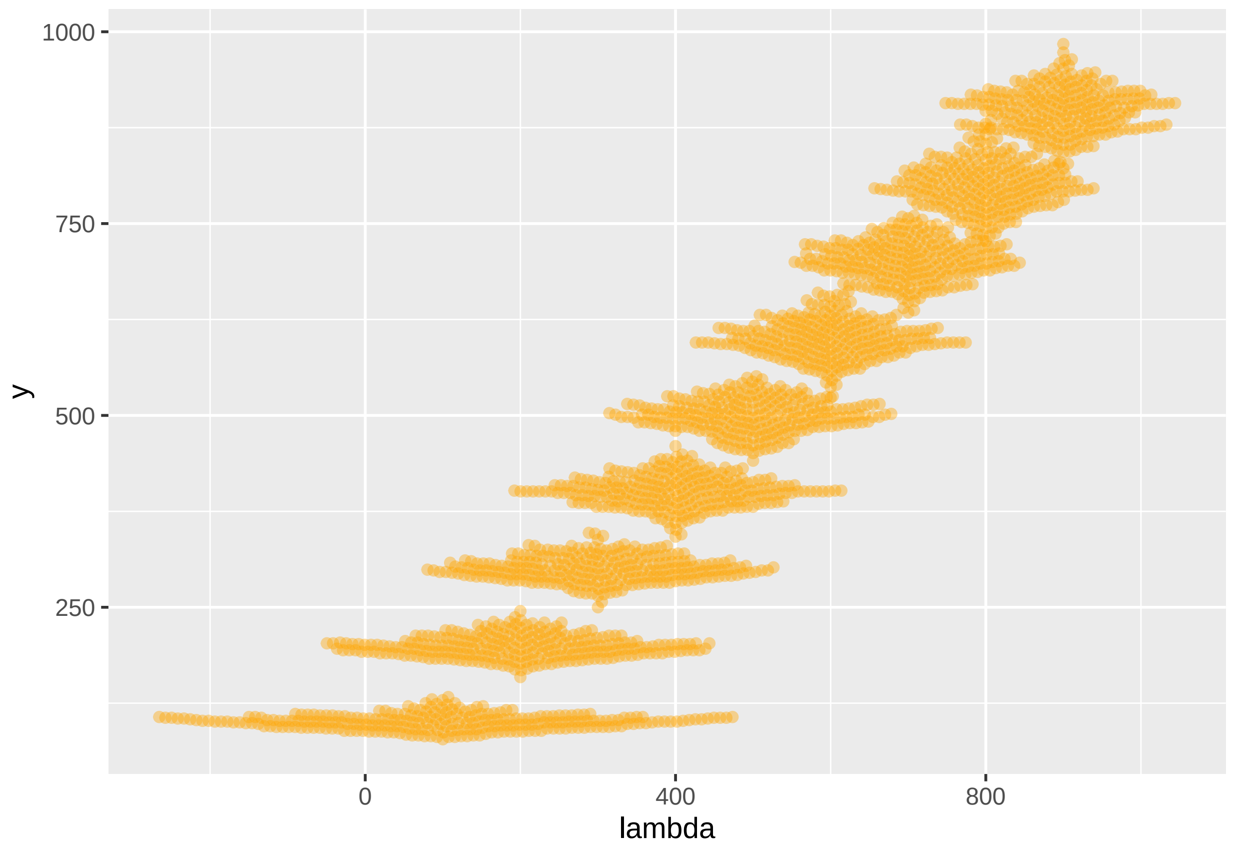
```
summarise(group_by(simdat, lambda), sd(y), sd(2*sqrt(y)))
```

```
## # A tibble: 9 x 3
##   lambda `sd(y)` `sd(2 * sqrt(y))`
##    <dbl>   <dbl>             <dbl>
## 1    100    10.1             0.996
## 2    200    15.4             1.06
## 3    300    15.5             0.903
## 4    400    18.3             0.913
## 5    500    23.6             1.05
## 6    600    23.6             0.959
## 7    700    25.6             0.965
## 8    800    31.2             1.10
## 9    900    32.9             1.10
```

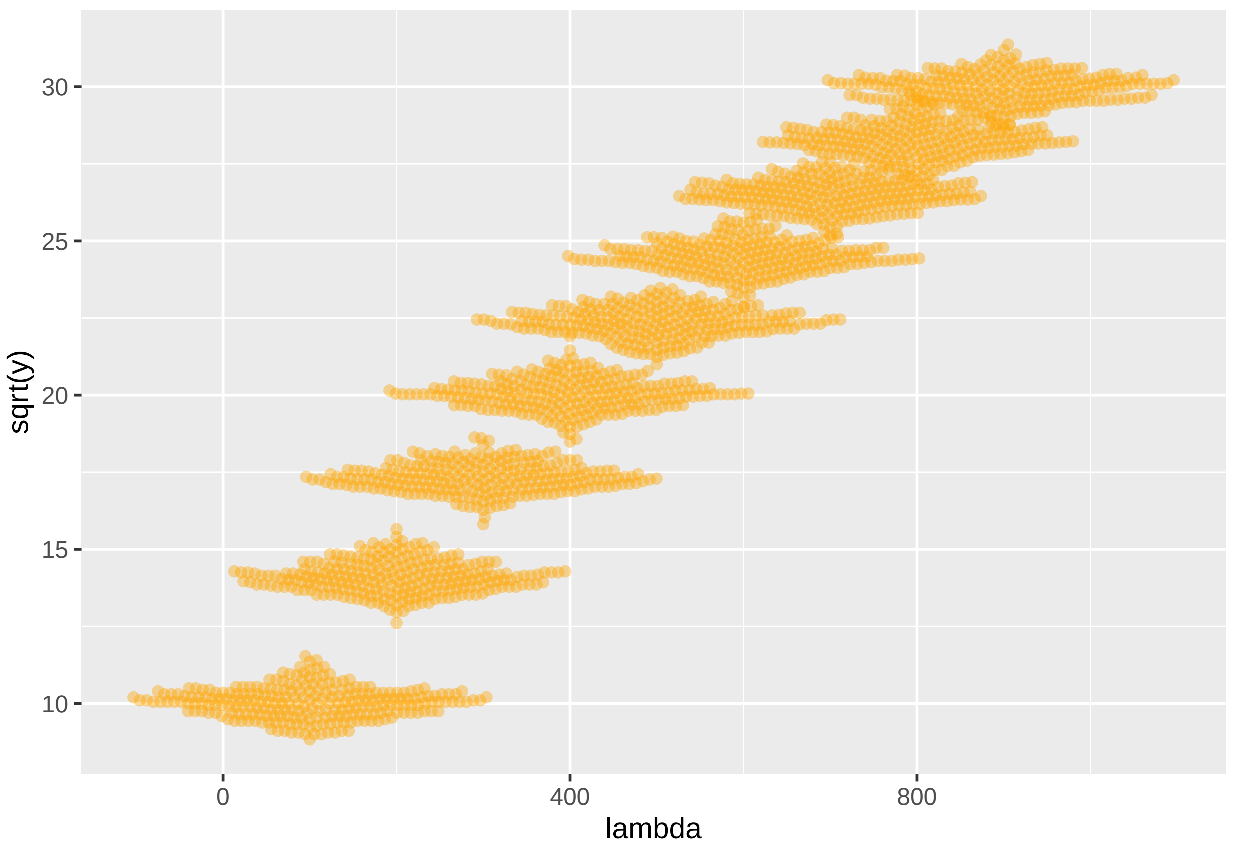
Lets also try to increase the number of replicates.

```
set.seed(1234)
simdat = lapply(lambdas, function(l)
    tibble(y = rpois(n = 250, lambda=l), lambda = l)
  ) %>% bind_rows

library("ggbeeswarm")
```

```
ggplot(simdat, aes(x = lambda, y = y)) +
  geom_beeswarm(alpha = 0.4, color = "orange")
```



```
ggplot(simdat, aes(x = lambda, y = sqrt(y))) +
  geom_beeswarm(alpha = 0.4, color = "orange")
```



```r
summarise(group_by(simdat, lambda), sd(y), sd(2*sqrt(y)))
```

```
## # A tibble: 9 x 3
##   lambda `sd(y)` `sd(2 * sqrt(y))`
##    <dbl>   <dbl>             <dbl>
## 1    100    9.91             0.984
## 2    200   14.5              1.03
## 3    300   16.4              0.947
## 4    400   20.8              1.04
## 5    500   21.4              0.957
## 6    600   24.6              1.01
## 7    700   25.4              0.959
## 8    800   29.8              1.05
## 9    900   28.5              0.951
```

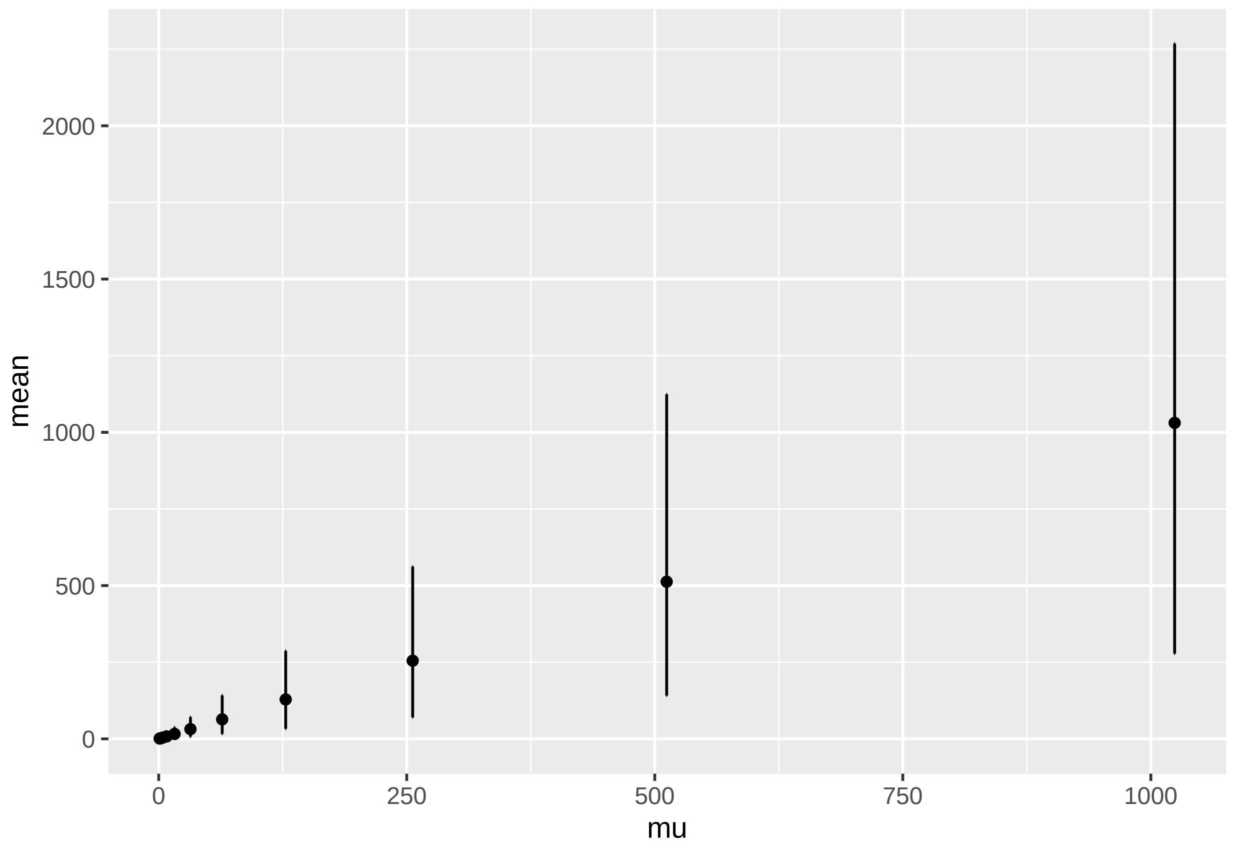
Ah! We see how helpful theses transformations can be!

We can also go back to the gamma-Poisson distribution.

```r
set.seed(1234)
# Generate some mu values
muvalues = 2^seq(0, 10, by = 1)
# For each mu, generate 1000 random binoimals with size 4.
# Then also obtain means, sd's and 95% CI for the output values.
simgp = lapply(muvalues, function(mu) {
  u = rnbinom(n = 1e4, mu = mu, size = 4)
  tibble(mean = mean(u), sd = sd(u),
         lower = quantile(u, 0.025),
         upper = quantile(u, 0.975),
         mu = mu)
  } ) %>% bind_rows
# Take a look
head(as.data.frame(simgp), 2)
```

```
##     mean       sd lower upper mu
## 1 1.0075 1.121949     0     4  1
## 2 2.0033 1.735624     0     6  2
```

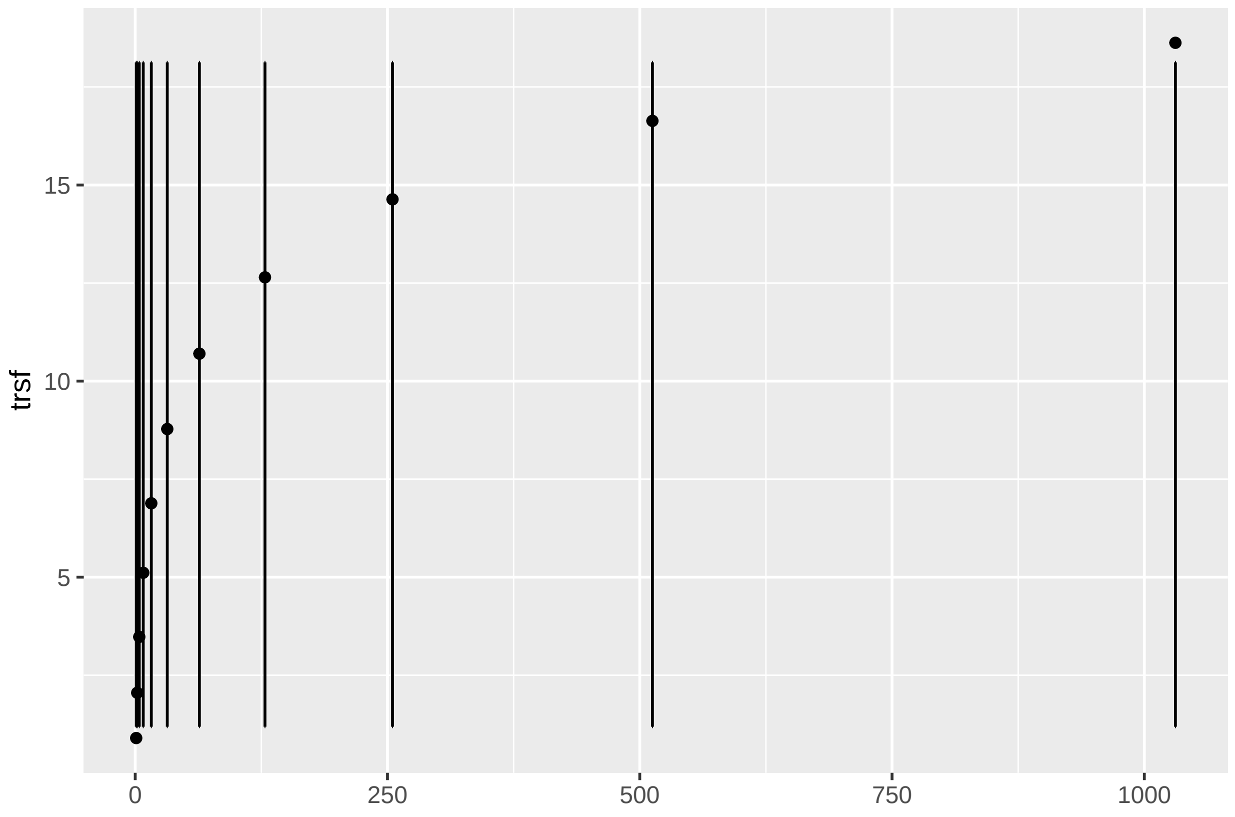
```r
# And plot
ggplot(simgp, aes(x = mu, y = mean, ymin = lower, ymax = upper)) +
  geom_point() + geom_errorbar()
```



Is there any transformations we can apply to stabilize the variance?

```r
set.seed(1234)
simgp = mutate(simgp,
  slopes = 1 / sd,
  trsf   = cumsum(slopes * mean),
           lower = quantile(trsf, 0.025),
         upper = quantile(trsf, 0.975),)
```

```r
ggplot(simgp, aes(x = mean, y = trsf,min = lower, ymax = upper)) +
  geom_point() + geom_errorbar() + xlab("")
```

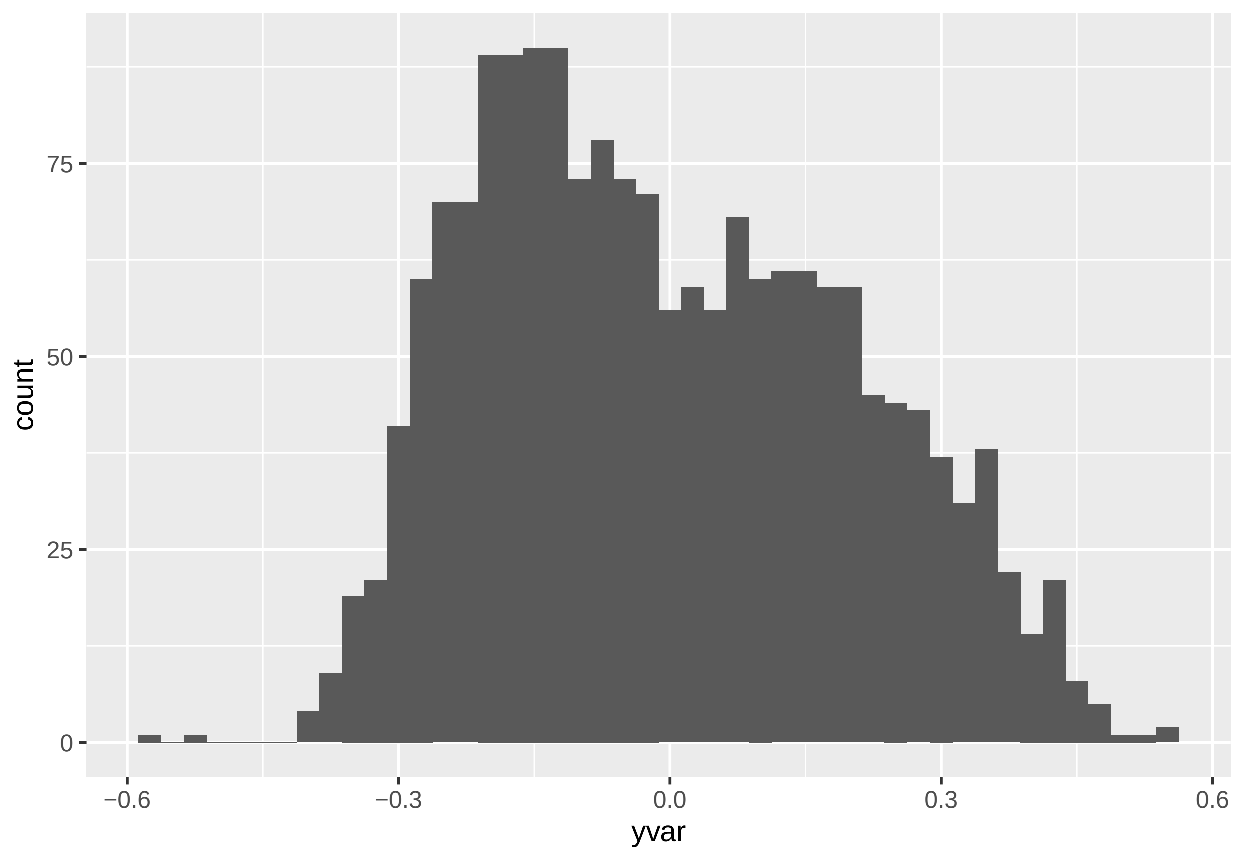


We talk about the delta method. A theoretical approach to obtain theses transformations, but it looks pretty complicated for now.

Exercises

1.)

```r
set.seed(1234)
# Load data
yvar = readRDS(here::here("Data","Myst.rds"))$yvar
# plot dist
ggplot(tibble(yvar), aes(x = yvar)) + geom_histogram(binwidth=0.025)
```



```r
# Now, generate probabilities of membership
pA = runif(length(yvar))
pB = 1 - pA

# Iteration
iter = 0
# current log likelihood
loglik = -Inf
# current change in the log likelihood
delta = +Inf
# tolerance level
tolerance = 1e-3
# Min and Max number of iterations
miniter = 50; maxiter = 1000
```

```r
set.seed(1234)
# Our EM algorithmn
while((delta > tolerance) && (iter <= maxiter) || (iter < miniter)) {
  # E steps
  lambda = mean(pA)
  muA = weighted.mean(yvar, pA)
  muB = weighted.mean(yvar, pB)
  sdA = sqrt(weighted.mean((yvar - muA)^2, pA))
  sdB = sqrt(weighted.mean((yvar - muB)^2, pB))
  # M steps
  phiA = dnorm(yvar, mean = muA, sd = sdA)
  phiB = dnorm(yvar, mean = muB, sd = sdB)
  pA   = lambda * phiA
  pB   = (1 - lambda) * phiB
  ptot = pA + pB
  pA   = pA / ptot
  pB   = pB / ptot

  loglikOld = loglik
  loglik = sum(log(pA))
  delta = abs(loglikOld - loglik)
  iter = iter + 1
}
param = tibble(group = c("A","B"), mean = c(muA,muB), sd = c(sdA,sdB))
param
```

```
## # A tibble: 2 x 3
##   group   mean     sd
##   <chr>  <dbl>  <dbl>
## 1 A      0.147 0.150
## 2 B     -0.169 0.0983
```

The E steps calculate a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters.

The maximization step computes parameters that maximize the expected log likelihood that we found in the E steps

Tolerance is the minimum amount of difference between the log likelihood from one iteration to another before we stop the algorithm.

miniter and maxiter and the min and max number of iterations, theses of course take priority over the tolerance.

```
set.seed(1234)
# Compare using mixtools

gm = normalmixEM(yvar, k = 2)
```

```
## number of iterations= 288
```

```
gm$lambda
```

```
## [1] 0.4756152 0.5243848
```

```
gm$mu
```

```
## [1] -0.1693592  0.1473215
```

```
gm$sigma
```

```
## [1] 0.09826527 0.14978797
```

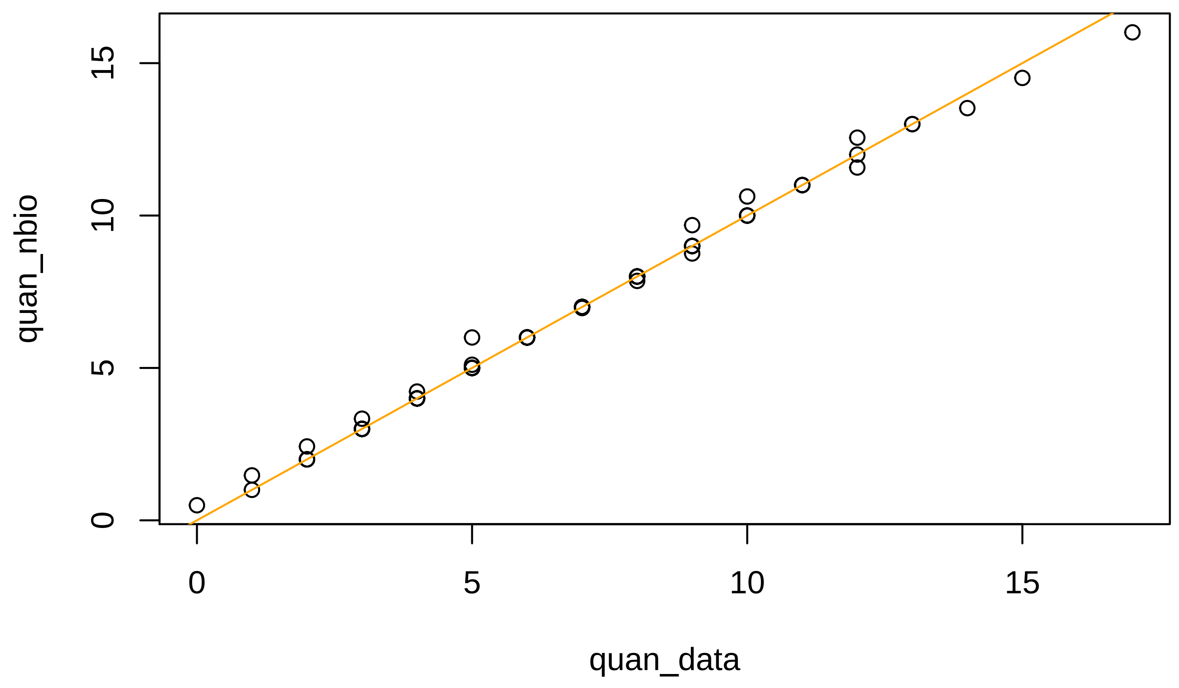
```r
gm$loglik
```

```
## [1] 417.2218
```

Same parameters, less iterations.

2.)

```r
# our gamma parameters
a=10
b=3/2
#How many quantiles
qs = ppoints(100)

# qs us the simulated data from 4.4.2, then we obtain the quantiles
quan_data = quantile(gp,qs)

# Now obtain theoritical quantiles of the gamma-possion dist (NB(a,b/b+1))
quan_nbio = quantile(qnbinom(p=qs,a,(b/(b+1))),qs)

# plot
plot(quan_data,quan_nbio)
abline(a =0, b=1, col="orange")
```



Nice! Our simulated data is really close to the theoretical data!

3.)

a.)

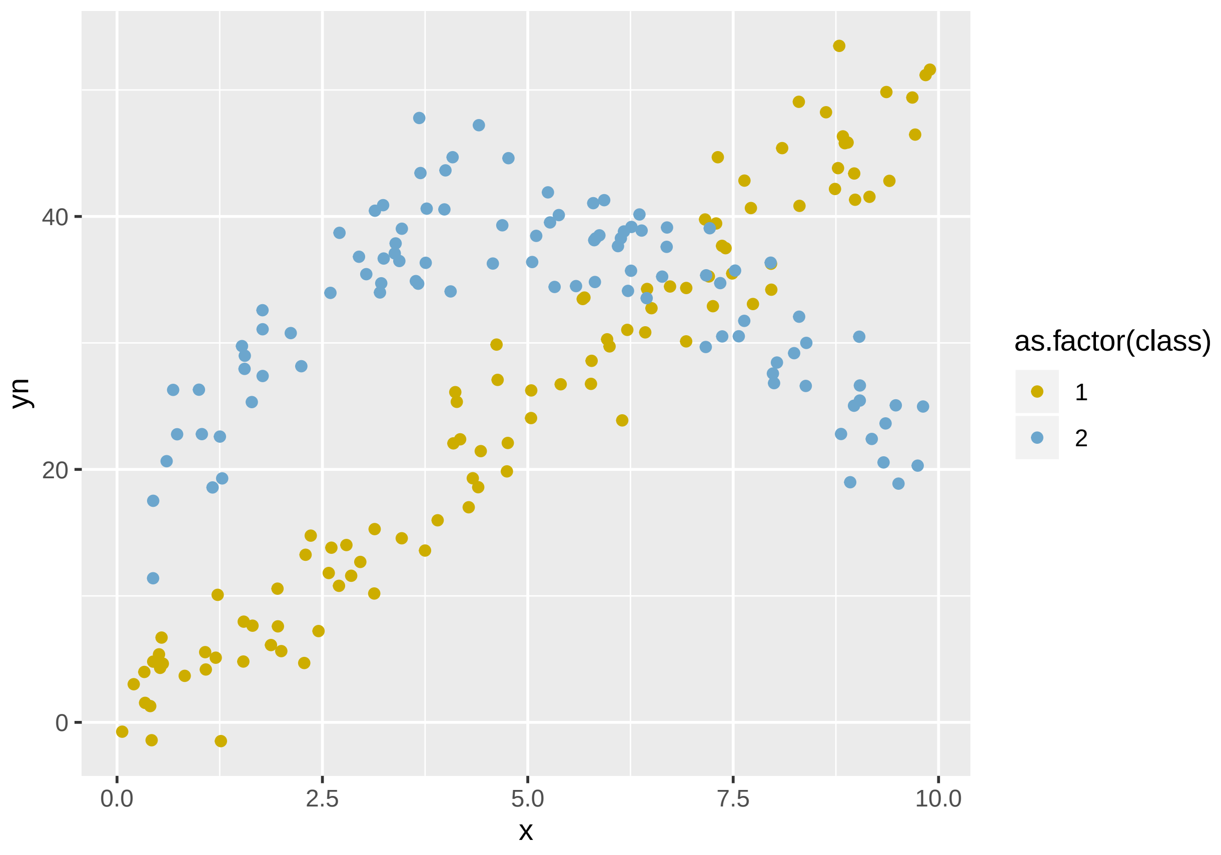
```r
library("flexmix")
```

```
## Loading required package: lattice
```

```r
data("NPreg")

head(NPreg)
```

```
##          x        yn yp yb class id1 id2
## 1 4.176633 22.380379  4  0     1   1   1
## 2 1.201631  5.111575  3  0     1   1   1
## 3 2.295006 13.251058  9  0     1   2   1
## 4 5.965868 30.285240  3  1     1   2   1
## 5 2.358083 14.764508  4  0     1   3   2
## 6 7.637061 42.833760  1  1     1   3   2
```

```r
NPreg %>% ggplot(aes(x=x,y=yn,color=as.factor(class))) +
  geom_point() +
  scale_color_manual(values = c("gold3","skyblue3"))
```



Looks class one was generated from a linear model and class two from a quadratic model.

b.)

```r
m1 = flexmix(yn ~ x + I(x^2), data = NPreg, k = 2)

fitted = m1@cluster

truth_table = tibble(real_class = NPreg$class,
                     fitted_class = fitted,
                     correct_esti = (real_class==fitted_class))

head(truth_table,20)
```

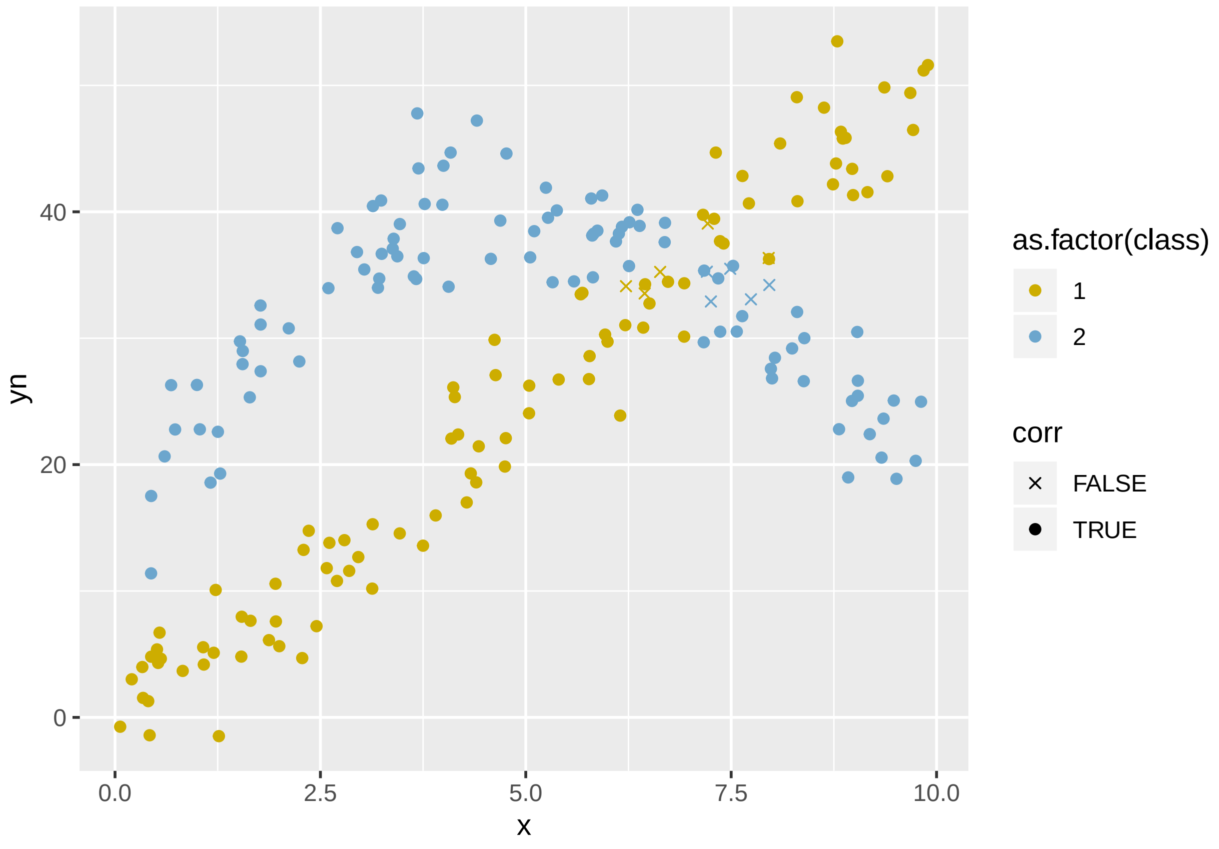
```
## # A tibble: 20 x 3
##    real_class fitted_class correct_esti
##         <int>        <int> <lgl>
##  1          1            1 TRUE
##  2          1            1 TRUE
##  3          1            1 TRUE
##  4          1            1 TRUE
##  5          1            1 TRUE
##  6          1            1 TRUE
##  7          1            1 TRUE
##  8          1            1 TRUE
##  9          1            1 TRUE
## 10          1            1 TRUE
## 11          1            1 TRUE
## 12          1            1 TRUE
## 13          1            1 TRUE
## 14          1            1 TRUE
## 15          1            2 FALSE
## 16          1            1 TRUE
## 17          1            1 TRUE
## 18          1            1 TRUE
## 19          1            2 FALSE
## 20          1            1 TRUE
```

```r
tail(truth_table,20)
```

```
## # A tibble: 20 x 3
##    real_class fitted_class correct_esti
##         <int>        <int> <lgl>
##  1          2            2 TRUE
##  2          2            2 TRUE
##  3          2            2 TRUE
##  4          2            2 TRUE
##  5          2            1 FALSE
##  6          2            1 FALSE
##  7          2            2 TRUE
##  8          2            2 TRUE
##  9          2            2 TRUE
## 10          2            2 TRUE
## 11           2             2 TRUE
## 12           2             2 TRUE
## 13           2             2 TRUE
## 14           2             2 TRUE
## 15           2             2 TRUE
## 16           2             2 TRUE
## 17           2             2 TRUE
## 18           2             2 TRUE
## 19           2             2 TRUE
## 20           2             2 TRUE
```

c.)

```
data = tibble(x=NPreg$x,yn=NPreg$yn,class=as.factor(m1@cluster),corr=truth_table$correct_esti)

data %>% ggplot(aes(x=x,y=yn,color=as.factor(class),shape=corr)) +
  geom_point() +
  scale_color_manual(values = c("gold3","skyblue3")) +
  scale_shape_manual(values = c("cross","circle"))
```



4.)

A MODEL OF LARGE-SCALE PROTEOME EVOLUTION RICARD V. SOLÉ, ROMUALDO PASTOR-SATORRAS, ERIC SMITH and THOMAS B. KEPLER https://www.worldscientific.com/doi/abs/10.1142/S021952590200047X

Bayesian mixture model based clustering of replicated microarray data M. Medvedovic1, K.Y. Yeung and R.E. Bumgarner

https://watermark.silverchair.com/bth068.pdf?token=AQECAHi208BE49Ooan9kkhW_Ercy7Dm3ZL_
9Cf3qfKAc485ysgAAAnYwggJyBgkqhkiG9w0BBwagggJjMIICXwIBADCCAlgGCSqGSIb3DQEHATAeBglghkgBZQMEAS4
nTfbfm5amBBZs1l9Ybq1dwuSnnIl3wDKn8hftWiUgfDQfpPnfV3hyRctIJUJVlUodNRYIvDmSc3HwbA59eJvi-5iYL5EmTrT
Xd00EmM_1mUJz7D0d13DqaqBVGpp1d6Mfj5caDF2SsN5reOW3i1a2_sCENMYW3bg1lRUkL3poE2pQyoxPDYZUQrzLX
0CtxNeJZprJ6q7b9S7CL6jCjaPnWnOakDyo0DVWujS140b96yws6wdQ67tWbnXpPdwDovqoDaj8QcH-RWreMRpMhgxsRZ
DOG5CwpMHetwIdjonbs5wPFXhjQYdMU4WhE-UBuLq2JFqRsF5EWn8nzZoLAC4qkFfcVNkyzxqO0MXJjZ9lyuxAIVlGU
twky9L-lOUY6L1ltFj-21mG0lJzX62mKUz3z3zbh_-W4yRB6jYvnFZAPojfQH7Zs--XWEu-5LJTRVHWHVrGiEeIsj9zf6PlH
T0Pf7yPXIbKclkqbGwvsnJHt5FTtj1fkoLI