

Class-4-Summary

Jenna G. Tichon

15/10/2019

4. Mixture Models

4.2 Finite mixtures

Coin flip experiment

```
# Flip a fair coin 10000 times and store as T or F
coinflips <- (runif(1000,0,1)>0.5)
```

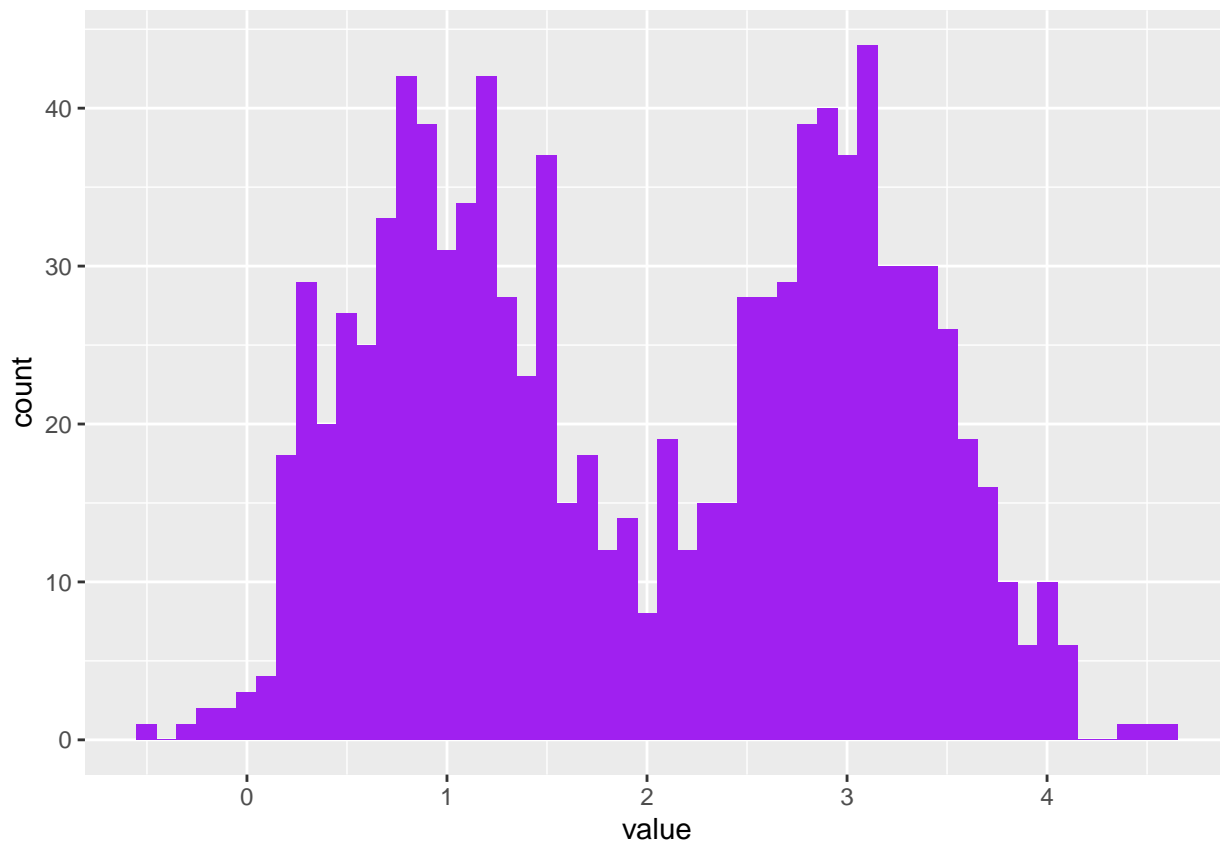
```
# Make a summary table
table(coinflips)
```

```
## coinflips
## FALSE TRUE
##    498    502
```

Coin flip followed by generating from either a $N(1,0.5)$ or a $N(3,0.5)$ distribution

```
#Function to simulate one flip
oneFlip <- function(fl, mean1 = 1, mean2 = 3, sd1 = 0.5, sd2 = 0.5){
  #If heads use distribution 1, else use distribution 2
  if(fl){
    rnorm(1,mean1,sd1)
  } else {
    rnorm(1, mean2, sd2)
  }
}
```

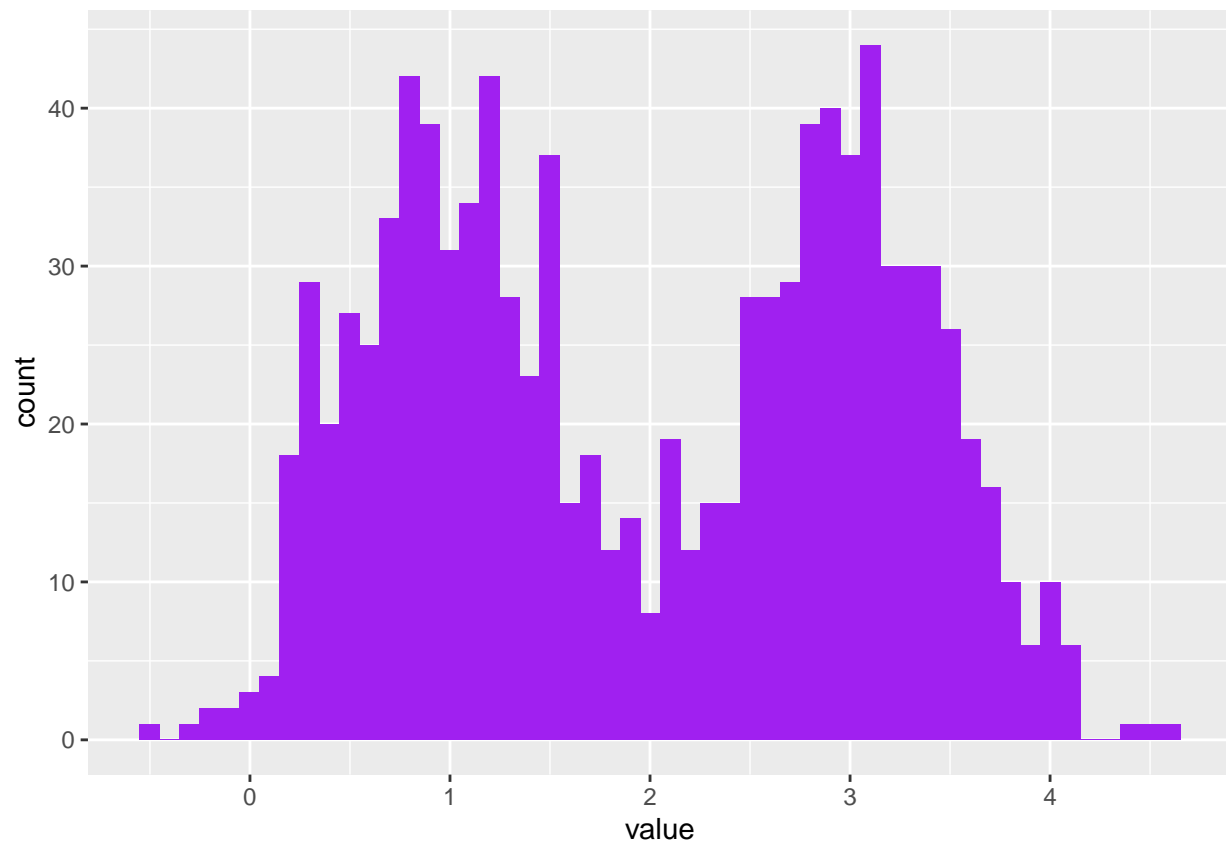
```
#Make a histogram using the 10000 coinflips in coinflips
fairmix = vapply(coinflips, oneFlip, numeric(1))
ggplot(tibble(value = fairmix), aes(x = value)) +
  geom_histogram(fill = "purple", binwidth = 0.1)
```



Q 4.1

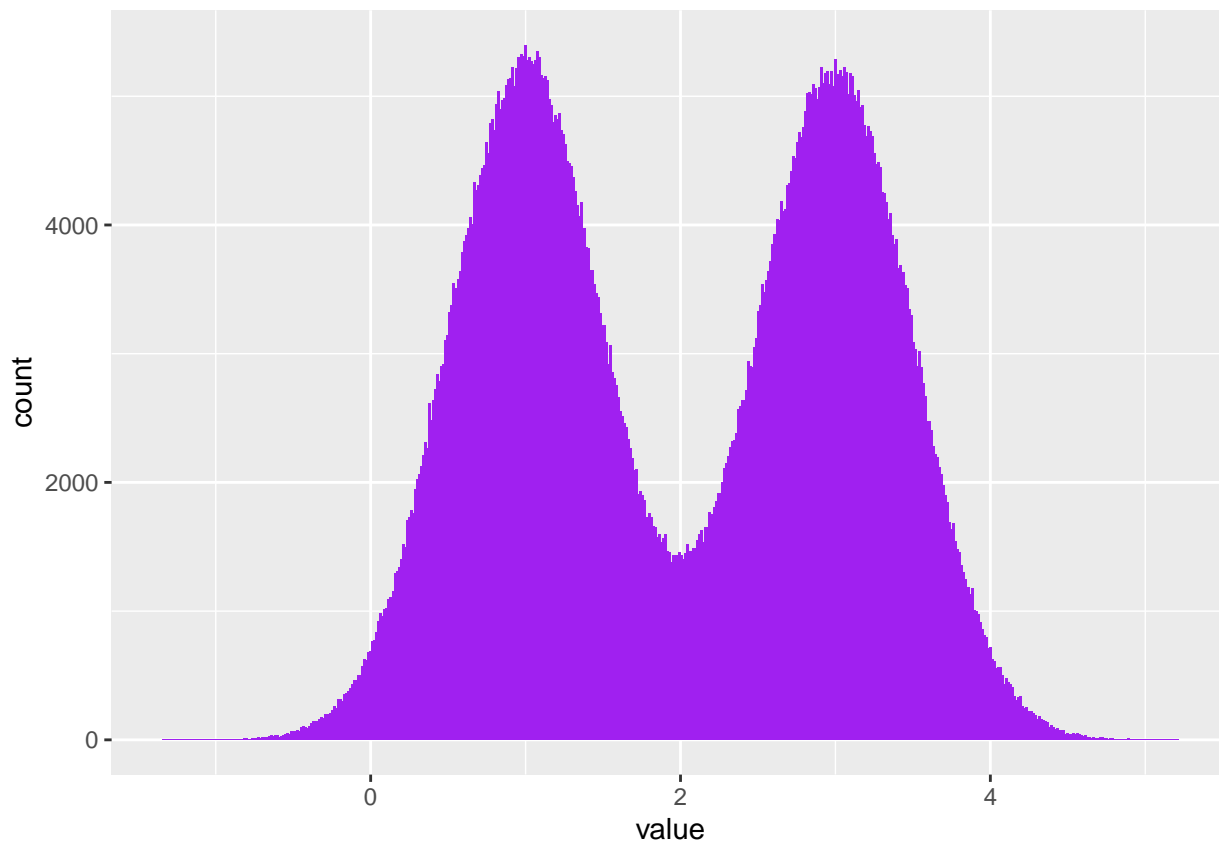
! Typo ! The standard deviation suddenly changes

```
means = c(1, 3)
sds    = c(0.5, 0.5)
values = rnorm(length(coinflips),
               mean = ifelse(coinflips, means[1], means[2]),
               sd    = ifelse(coinflips, sds[1], sds[2]))
ggplot(tibble(value = fairmix), aes(x = value)) +
  geom_histogram(fill = "purple", binwidth = 0.1)
```



Q 4.2

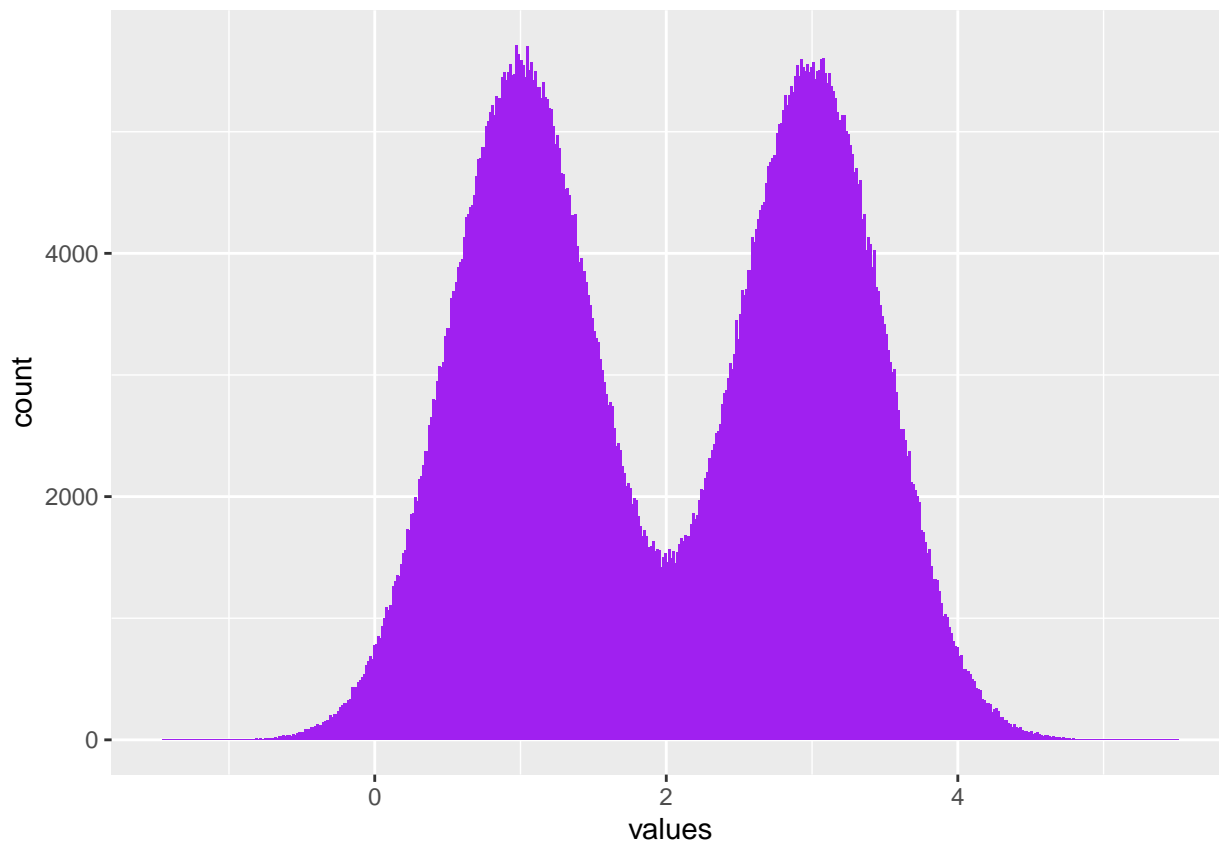
```
means = c(1, 3)
sds    = c(0.5, 0.5)
values = rnorm(1000000,
              mean = ifelse(coinflips, means[1], means[2]),
              sd    = ifelse(coinflips, sds[1], sds[2]))
ggplot(tibble(value = values), aes(x = value)) +
  geom_histogram(fill = "purple", bins = 500)
```



Data becomes less sparse and looks more continuous

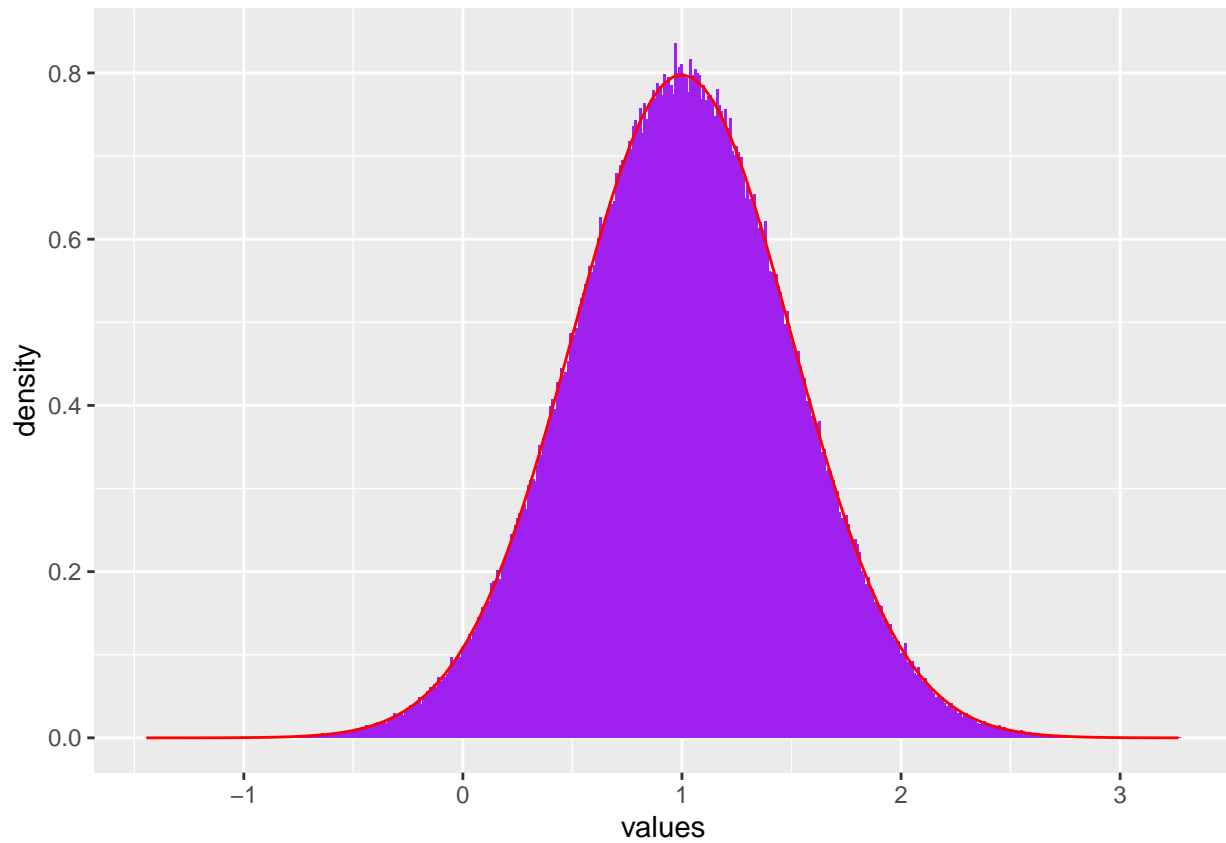
Text solution:

```
fair = tibble(  
  coinflips = (runif(1e6) > 0.5),  
  values = rnorm(length(coinflips),  
                 mean = ifelse(coinflips, means[1], means[2]),  
                 sd   = ifelse(coinflips, sds[1],  sds[2])))  
ggplot(fair, aes(x = values)) +  
  geom_histogram(fill = "purple", bins = 500)
```



Q 4.3

```
#Take the tibble coinflips and sort out just those marked as fair = TRUE  
#Make a histogram of those coin flips and overlay the density of normal for the fair coin flips  
ggplot(dplyr::filter(fair, coinflips), aes(x = values)) +  
  geom_histogram(aes( y = ..density..), fill = "purple",  
                 binwidth = 0.01) +  
  stat_function(fun = dnorm,  
               args = list(mean = means[1], sd = sds[1]), color = "red")
```

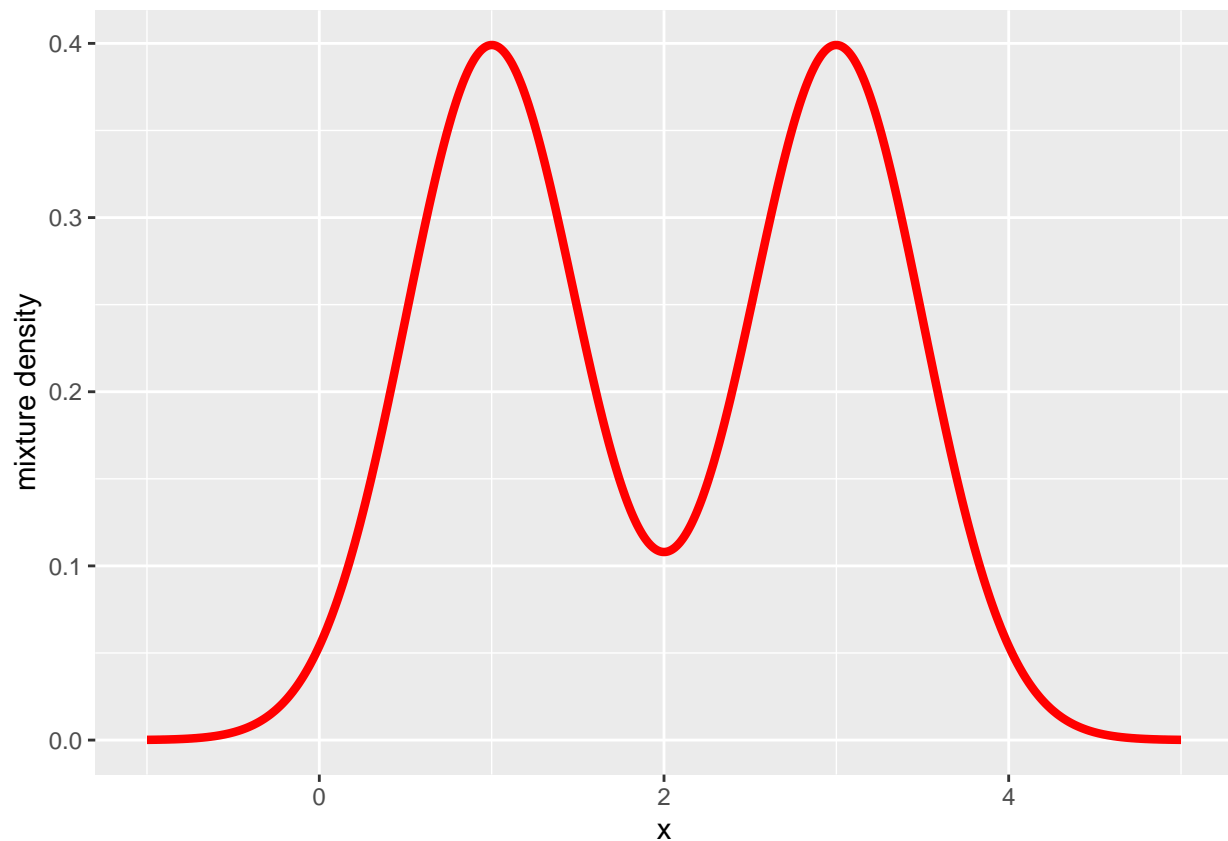


The density curve is

$$f(x) = \frac{1}{2}\phi_1(x) + \frac{1}{2}\phi_2(x)$$

Plotting the density:

```
fairtheory = tibble(  
  x = seq(-1, 5, length.out = 1000),  
  f = 0.5 * dnorm(x, mean = means[1], sd = sds[1]) +  
      0.5 * dnorm(x, mean = means[2], sd = sds[2]))  
ggplot(fairtheory, aes(x = x, y = f)) +  
  geom_line(color = "red", size = 1.5) + ylab("mixture density")
```



4.2.2 Discovering the hidden class labels

u is unobserved group label. y is observed data from two unknown groups. The joint density of y and u is

$$f_{\theta}(y, u) = f_{\theta}(y|u)f_{\theta}(u)$$

In this last example $\theta = (\mu_1, \mu_2, \sigma_1, \sigma_2, \lambda)$ where λ is the mixture fraction $\lambda = 0.5$.

Q 4.6

Experiment: - With prob π flip coin 1 with $p_1 = 0.125$, with probability $1 - \pi$ flip coin 2 with $p_2 = 0.25$ - Toss coin twice - Record number of heads K

(a)

```
#Function to simulate the experiment once
kflips<-function(p1=0.125, p2=0.25, pi=(1/8)){
  coin<-rbinom(1,1,pi)
  if(coin == 1){
    rbinom(1,2,p1)
  } else{
    rbinom(1,2,p2)
  }
}

#Apply this 100 times and make a contingency table
```

```
k<-replicate(100, kflips())
table(k)
```

```
## k
##  0  1  2
## 63 35  2
```

(b) Redo with $\pi=0.25$

```
k<-replicate(100, kflips(pi=0.25))
table(k)
```

```
## k
##  0  1  2
## 58 37  5
```

Mixture of two normals:

```
#Set mu's
mus = c(-.5, 1.5)

#Make string of 1, 2's
u = sample(2, 100, replace = TRUE)

#Generate random normals where the mean is either 1 or 2 depending on mus vector
y = rnorm(length(u), mean = mus[u])
duy = tibble(u, y)
head(duy)
```

```
## # A tibble: 6 x 2
##       u       y
##   <int> <dbl>
## 1     1 -1.29
## 2     1  1.28
## 3     2  2.74
## 4     1 -0.793
## 5     2  1.50
## 6     1 -0.170
```

We can estimate the μ 's by dividing the y 's by the u , then get the mean

```
#Group duy vector by u then make summary of means based on group
group_by(duy, u) %>% summarize(mean(y))
```

```
## # A tibble: 2 x 2
##       u `mean(y)`
##   <int>   <dbl>
## 1     1  -0.386
## 2     2   1.33
```

```
#Using aggregate function
agg.data<-aggregate(as.data.frame(y), by=list(u), FUN=mean)
print(agg.data)
```

```
##   Group.1      y
## 1      1 -0.3862981
## 2      2  1.3310188
```


If we don't know labels, we can put probabilities on possible u 's and sum over all u s to get our marginal probabilities.

$$\text{marglike}(\theta; y) = f(y|\theta) = \sum_u f(y, u|\theta) du$$

At the current iteration, mark an $*$, put best current guesses in $\theta^* = (\mu_1^*, \mu_2^*, \lambda^*)$. Put these into an expectation function.

$$E^*(\theta) = E_{\theta^*, Y} [\log p(u, y|\theta^*)] = \sum_u p(u|y, \theta^*) \log p(u, y|\theta^*)$$

The θ that maximizes $E^*(\theta)$ is the maximization step. Continue until marginal gains are made on likelihood.

4.2.3 Models for zero inflated data

ALERT!!: What's this about source code file to get binsTFBS for this chapter?

```
{r} # bincts = print(binTFBS) # ggplot(bincts, aes(x = tagCount))
+ #   geom_histogram(binwidth = 1, fill = "forestgreen") #
```

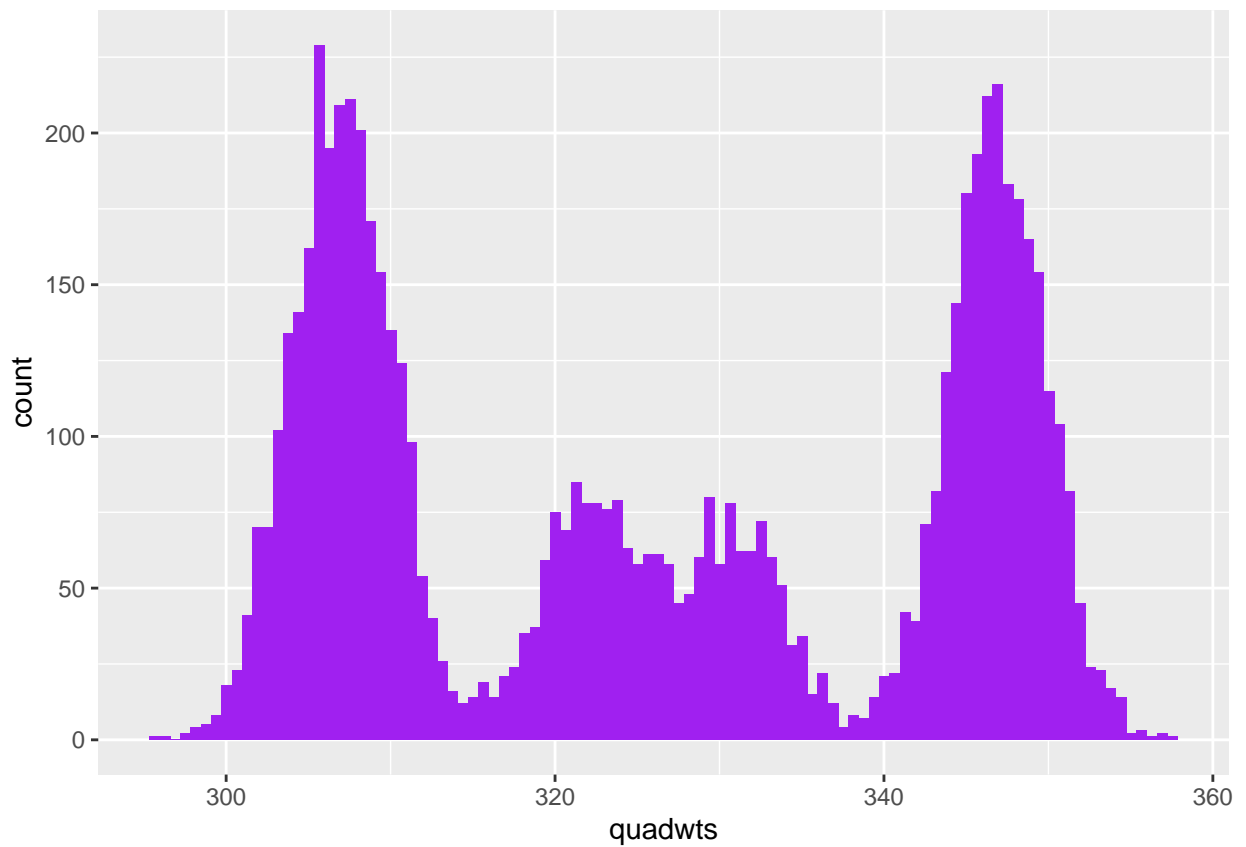
Q 4.9

```
{r} # bincts = print(binTFBS) # ggplot(bincts, aes(x = tagCount))
+ scale_y_log10() + #   geom_histogram(binwidth = 1, fill =
"forestgreen") #
```

4.2.4 More than two components

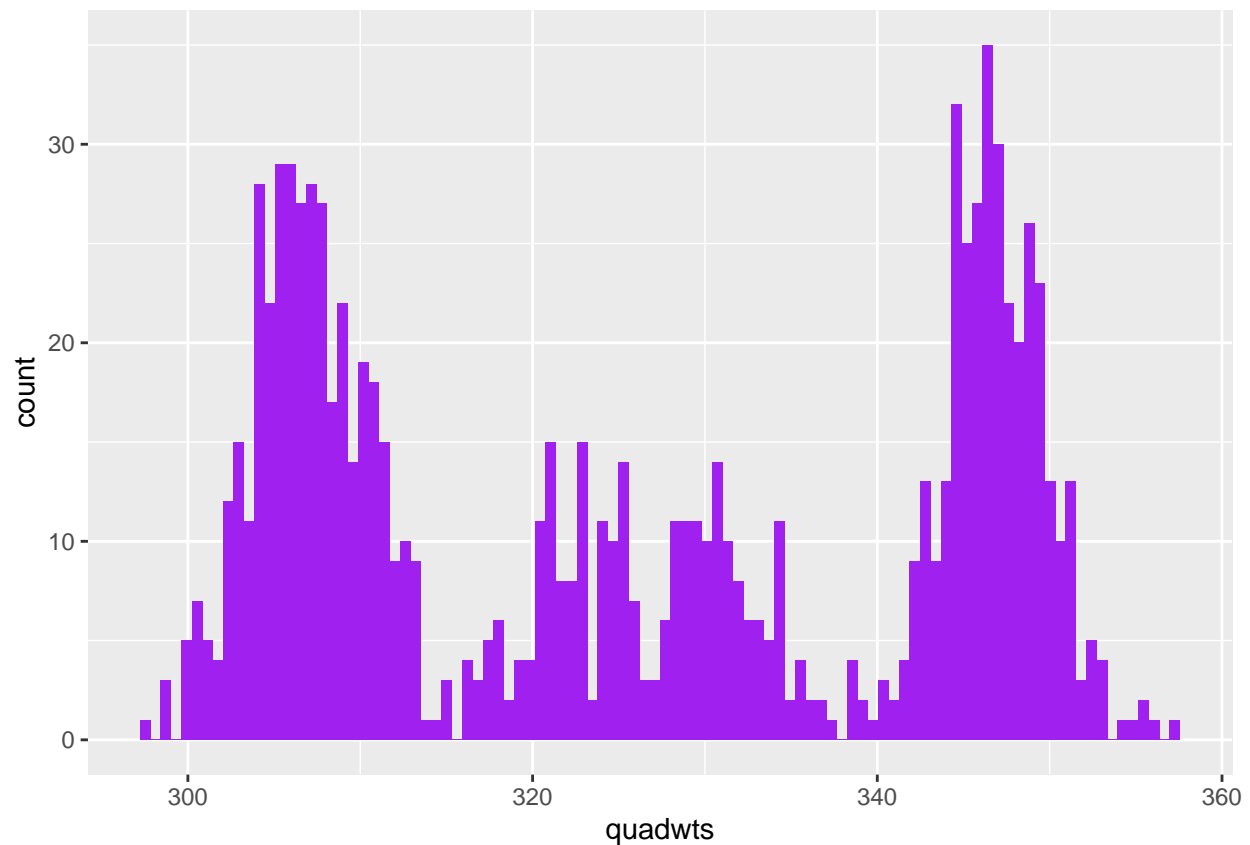
Three components, N=7000, different weights, sd=3

```
masses = c(A = 331, C = 307, G = 347, T = 322)
probs  = c(A = 0.12, C = 0.38, G = 0.36, T = 0.14)
N      = 7000
sd     = 3
nuclt  = sample(length(probs), N, replace = TRUE, prob = probs)
quadwts = rnorm(length(nuclt),
                mean = masses[nuclt],
                sd    = sd)
ggplot(tibble(quadwts = quadwts), aes(x = quadwts)) +
  geom_histogram(bins = 100, fill = "purple")
```



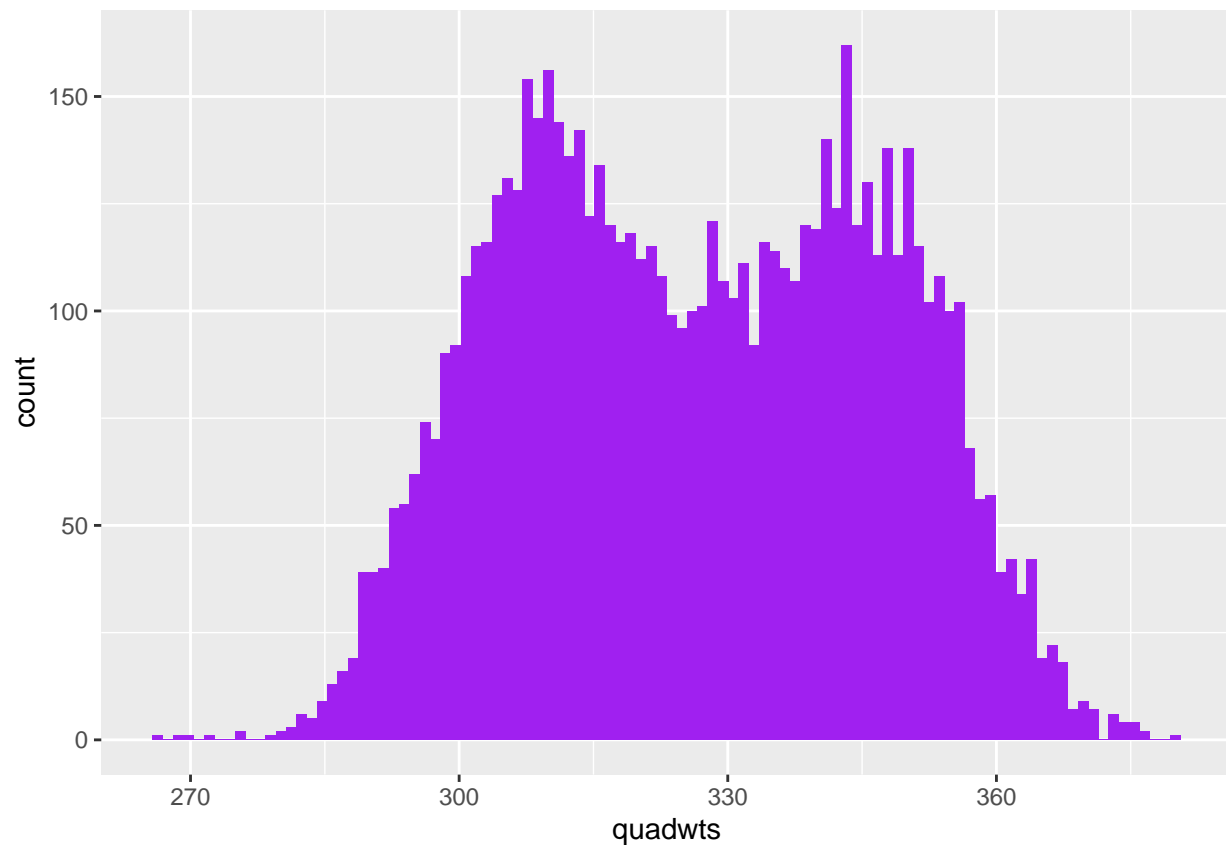
Repeat with 1000

```
masses = c(A = 331, C = 307, G = 347, T = 322)
probs  = c(A = 0.12, C = 0.38, G = 0.36, T = 0.14)
N      = 1000
sd     = 3
nuclt  = sample(length(probs), N, replace = TRUE, prob = probs)
quadwts = rnorm(length(nuclt),
                mean = masses[nuclt],
                sd    = sd)
ggplot(tibble(quadwts = quadwts), aes(x = quadwts)) +
  geom_histogram(bins = 100, fill = "purple")
```



Change sd to 10

```
masses = c(A = 331, C = 307, G = 347, T = 322)
probs  = c(A = 0.12, C = 0.38, G = 0.36, T = 0.14)
N      = 7000
sd     = 10
nuclt  = sample(length(probs), N, replace = TRUE, prob = probs)
quadwts = rnorm(length(nuclt),
                mean = masses[nuclt],
                sd    = sd)
ggplot(tibble(quadwts = quadwts), aes(x = quadwts)) +
  geom_histogram(bins = 100, fill = "purple")
```



4.3 Empirical Distributions and the Nonparametric Bootstrap

```
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 = "forestgreen") + ylim(0,0.1)
```

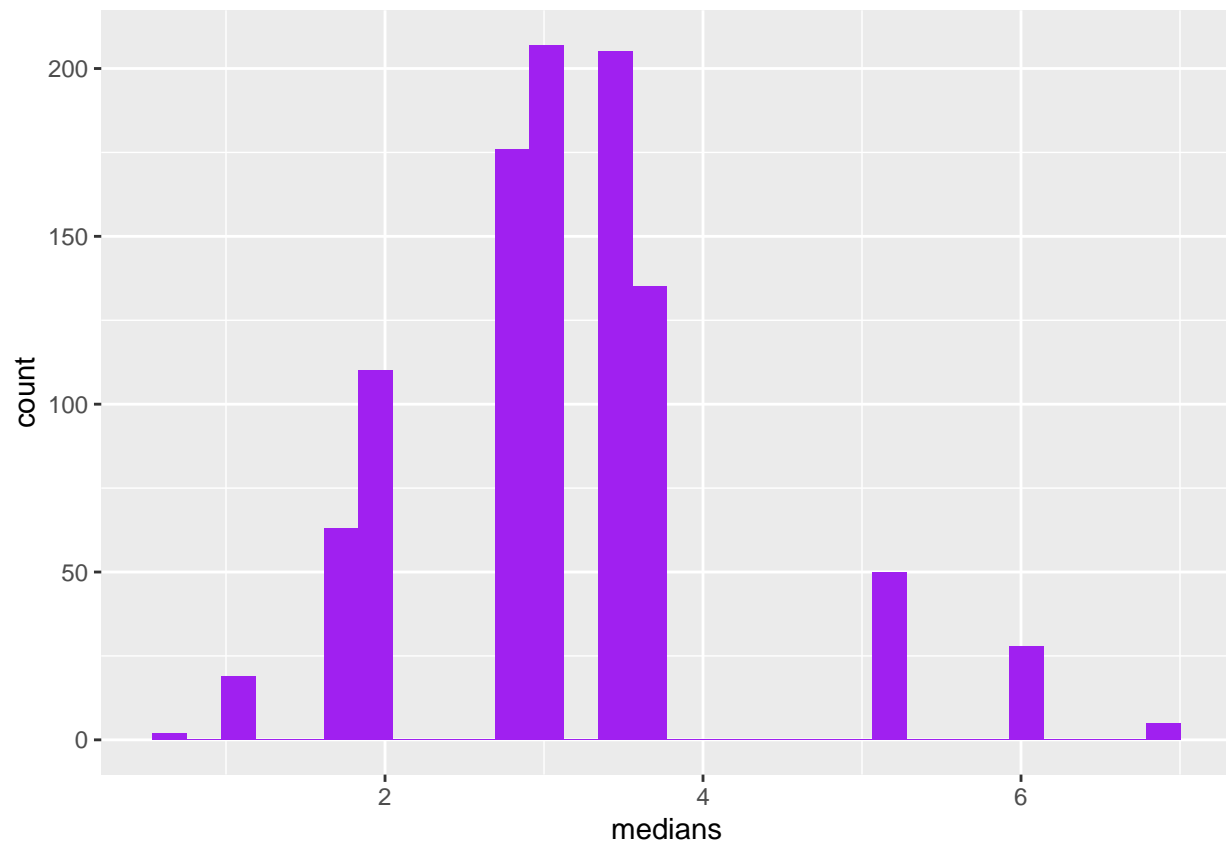


Use nonparametric bootstrap to find the sampling distribution of the median:

```
#1000 replications
B = 1000

#Vector of Medians. Replicate B times, calculate
meds = replicate(B, {
  #sample 1:15 (diff had length 15) 15 times with replacement
  i = sample(15, 15, replace = TRUE)
  #find the median of the vector of randomly sampled with replacement observations from diff
  median(ZeaMays$diff[i])
})

#Plot medians
ggplot(tibble(medians = meds), aes(x = medians)) +
  geom_histogram(bins = 30, fill = "purple")
```



Q4.13

99% confidence interval doesn't overlap 0

```
meds[5]
```

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

```
meds[995]
```

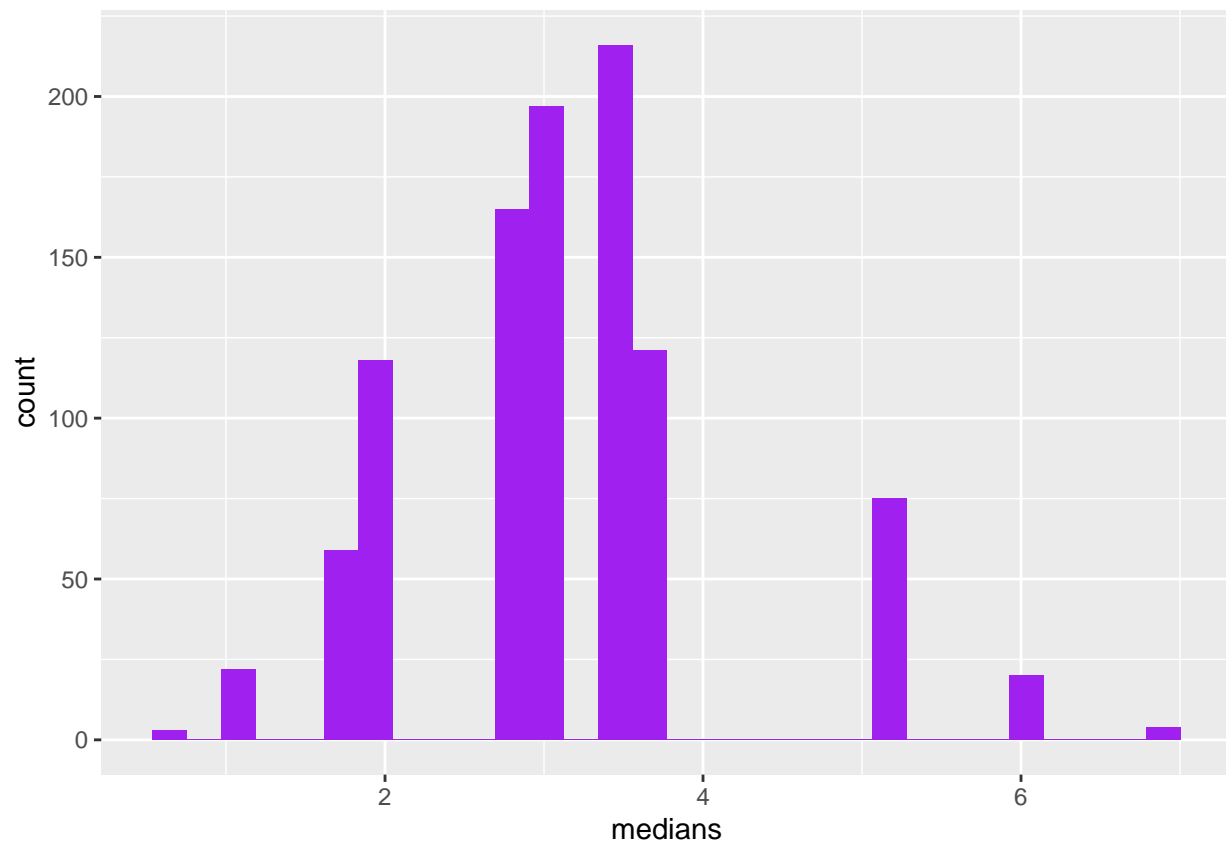
```
## [1] 1
```

Q 4.14

Bootstrap using the `bootstrap` package.

```
theta <- function(x){median(x)}
meds.bootstrap<-bootstrap(ZeaMays$diff, 1000, theta)$thetastar

#Plot medians
ggplot(tibble(medians = meds.bootstrap), aes(x = medians)) +
  geom_histogram(bins = 30, fill = "purple")
```

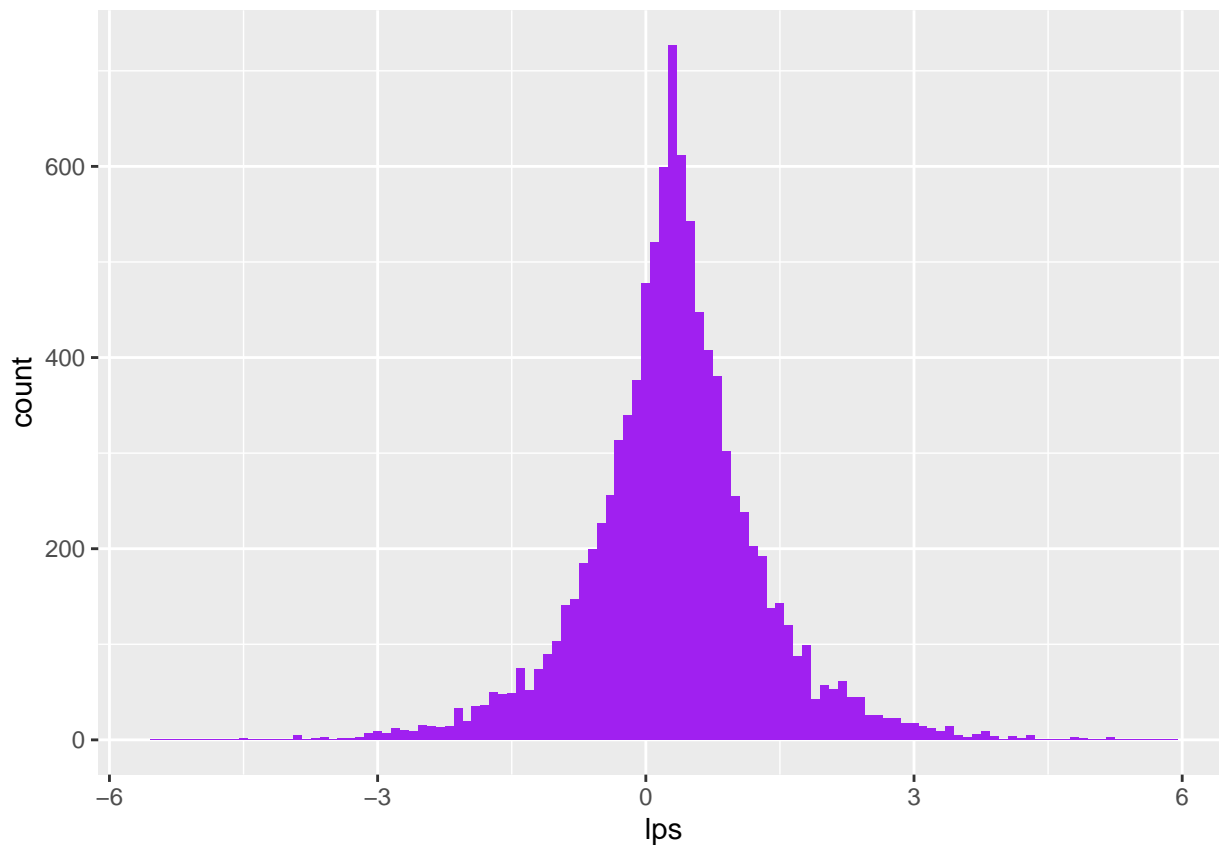


4.4 Infinite Mixtures

4.4.1 Infinite Mixture Models

```
# Sample 10000 exp(1) variables
w = rexp(10000, rate = 1)

# Generate 10000 N(0.3, sqrt(w)) random variables, i.e. w's are variances
mu = 0.3
lps = rnorm(length(w), mean = mu, sd = sqrt(w))
ggplot(data.frame(lps), aes(x = lps)) +
  geom_histogram(fill = "purple", binwidth = 0.1)
```



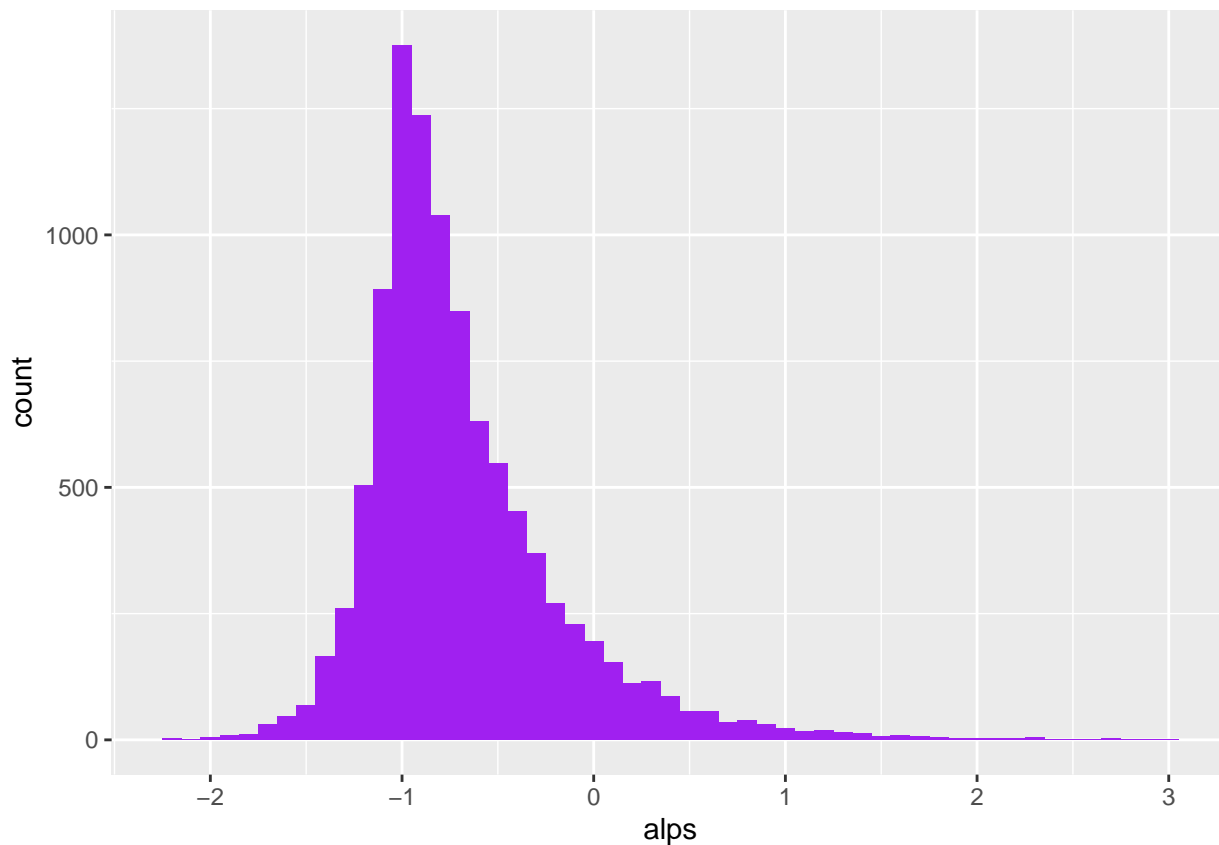
This has Laplace where the median is a good estimate of the location parameter θ and the median absolute deviation is used to estimate the scale parameter ϕ .

Q 4.17

Variable whose distribution is symmetric Laplace as a function of normal and exponential:

Let $w \sim \exp(1)$ and $X \sim N(\theta + w\mu, \sigma w)$:

```
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 = "purple", binwidth = 0.1)
```

Assymetric Laplace $AL(\theta, \mu, \sigma)$ have mean and variance:

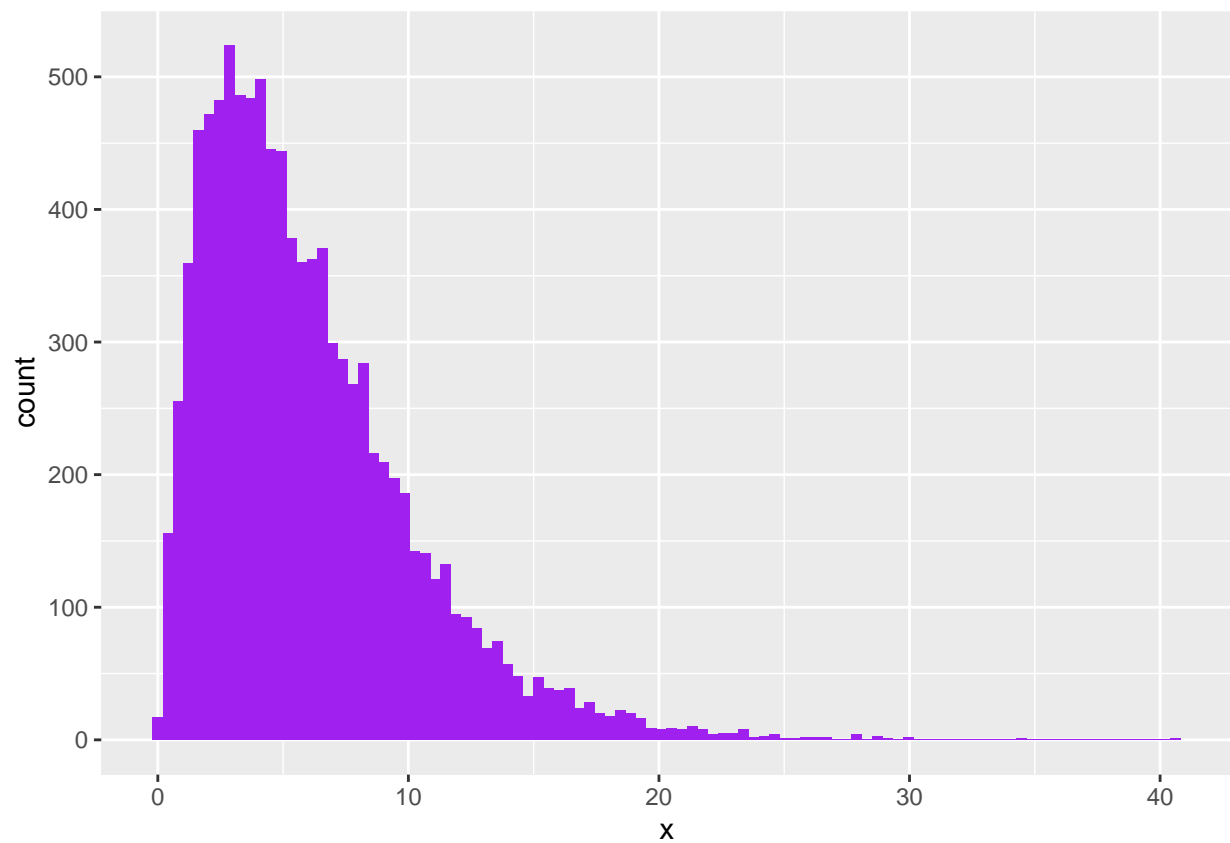
$$E(Y) = \theta + \mu, \quad \text{var}(y) = \theta^2 + \mu^2$$

Note that the variance is dependent on the mean.

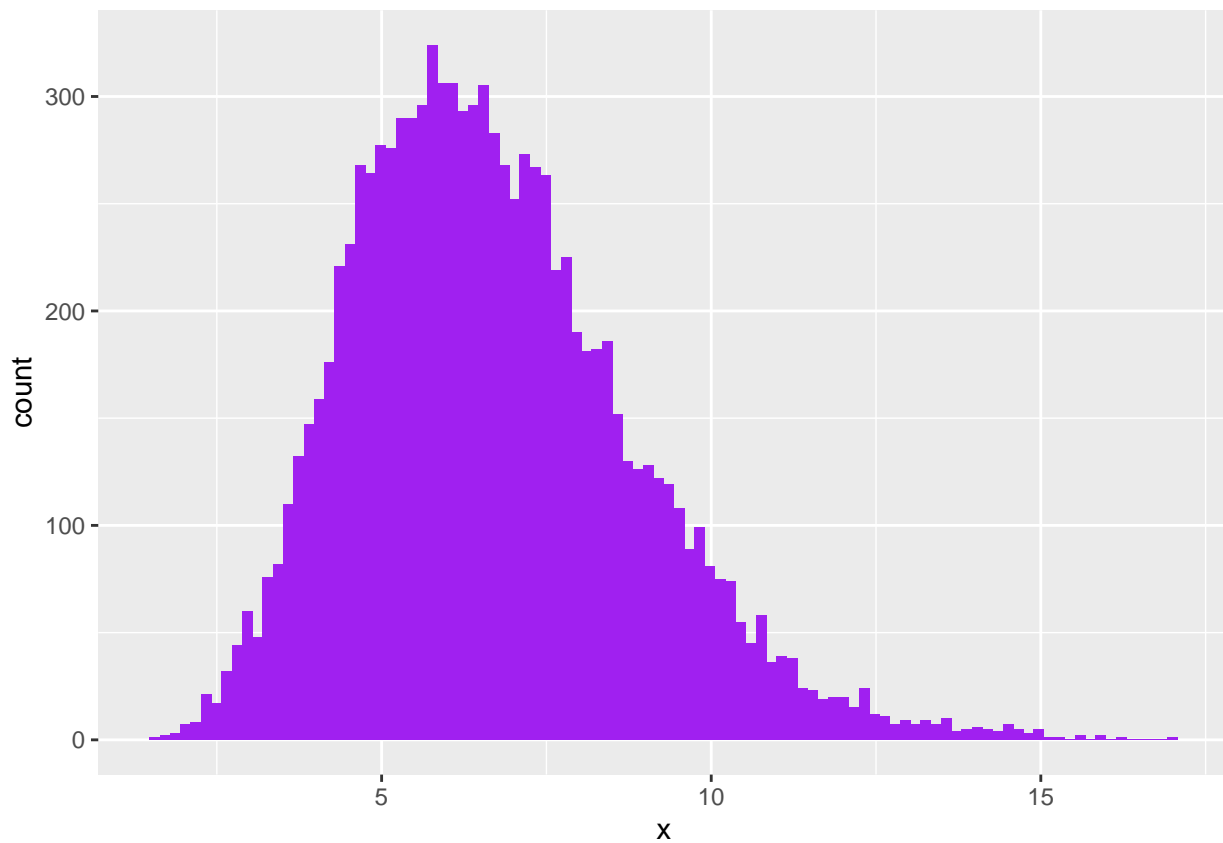
4.4.3 Gamma Distribution: two parameters (shape and scale)

Histograms of gammas:

```
ggplot(tibble(x = rgamma(10000, shape = 2, rate = 1/3)),
  aes(x = x)) + geom_histogram(bins = 100, fill= "purple")
```

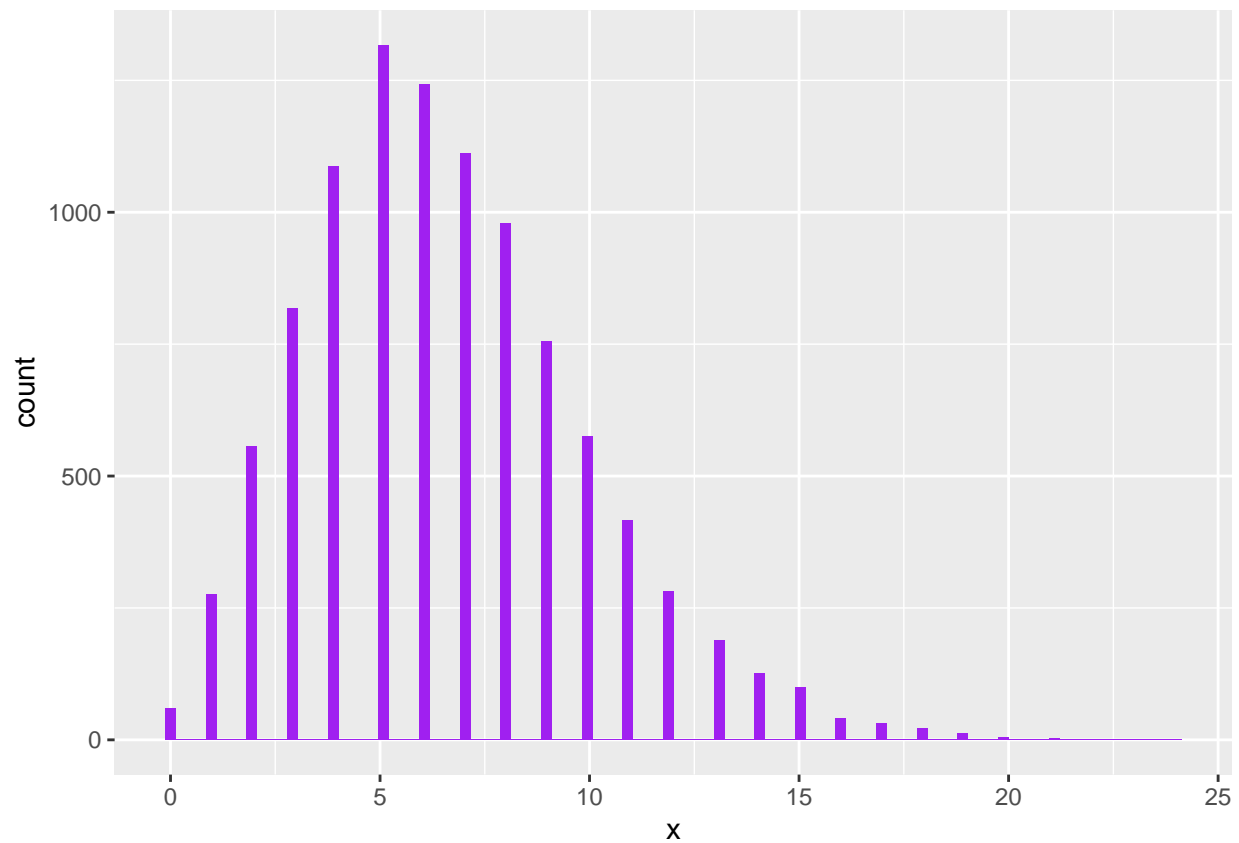


```
ggplot(tibble(x = rgamma(10000, shape = 10, rate = 3/2)),  
  aes(x = x)) + geom_histogram(bins = 100, fill= "purple")
```



Gamma-Poisson maixture:

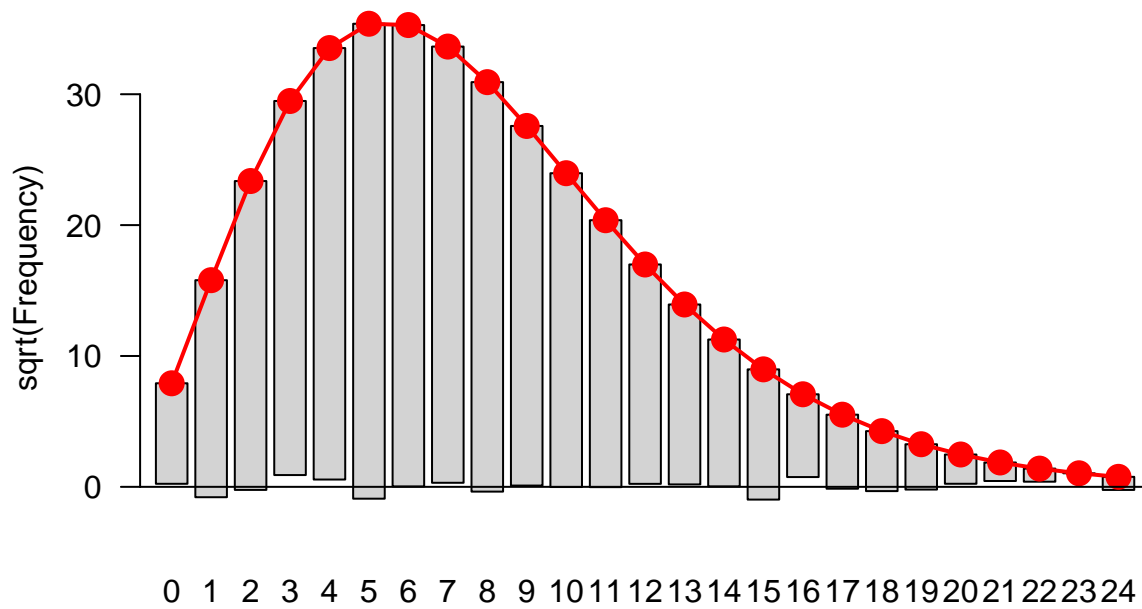
```
#Generate 10000 lambda from the gamma distribution  
lambda = rgamma(10000, shape = 10, rate = 3/2)  
#Generate random poissons using the generated lambdas  
gp = rpois(length(lambda), lambda = lambda)  
ggplot(tibble(x = gp), aes(x = x)) +  
  geom_histogram(bins = 100, fill= "purple")
```



Q 4.19

```
library("vcd")

## Loading required package: grid
##
## Attaching package: 'grid'
## The following object is masked from 'package:mixtools':
##
##   depth
ofit = goodfit(gp, "nbinomial")
plot(ofit, xlab = "")
```



```
ofit$par
```

```
## $size
## [1] 10.03836
##
## $prob
## [1] 0.6032705
```

ALERT!!: HOLY CRAP BALLS!!!! THE NEGATIVE BINOMIAL IS A GAMMA-POISSON DISTRIBUTION!!!!

Negative binomial:

$$P(K = k) = \binom{k + a - 1}{k} p^a (1 - p)^k$$

with mean $\mu = \frac{pa}{(1 - p)}$ and dispersion parameter $\alpha = 1/a$. The variance is $\mu_\alpha \mu^2$.

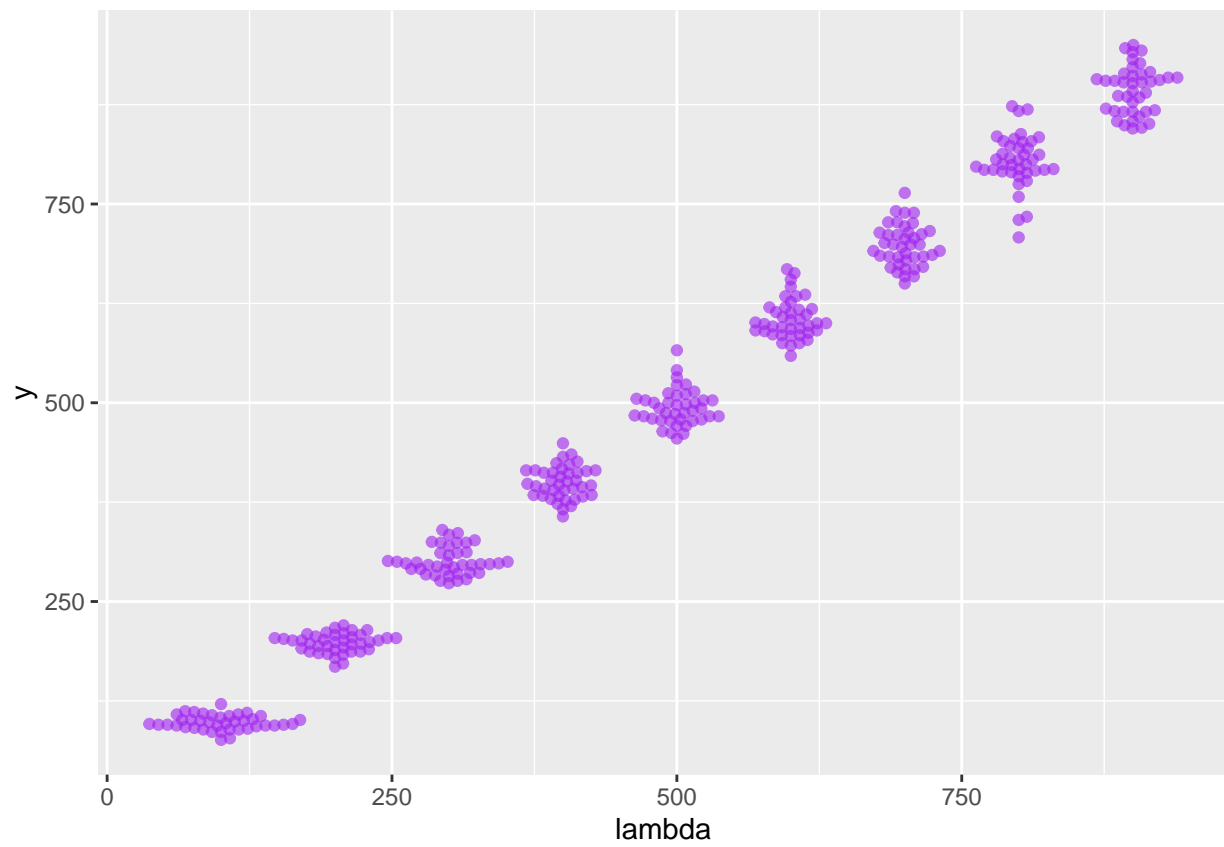
EXact parameters: use $X \sim \text{gamma}(a, b)$ to generate $K \sim \text{Pois}(x)$. Then

$$P(K = k) = \binom{k + a - 1}{k} \left(\frac{b}{b + 1} \right)^a \left(1 - \frac{b}{b + 1} \right)^k$$

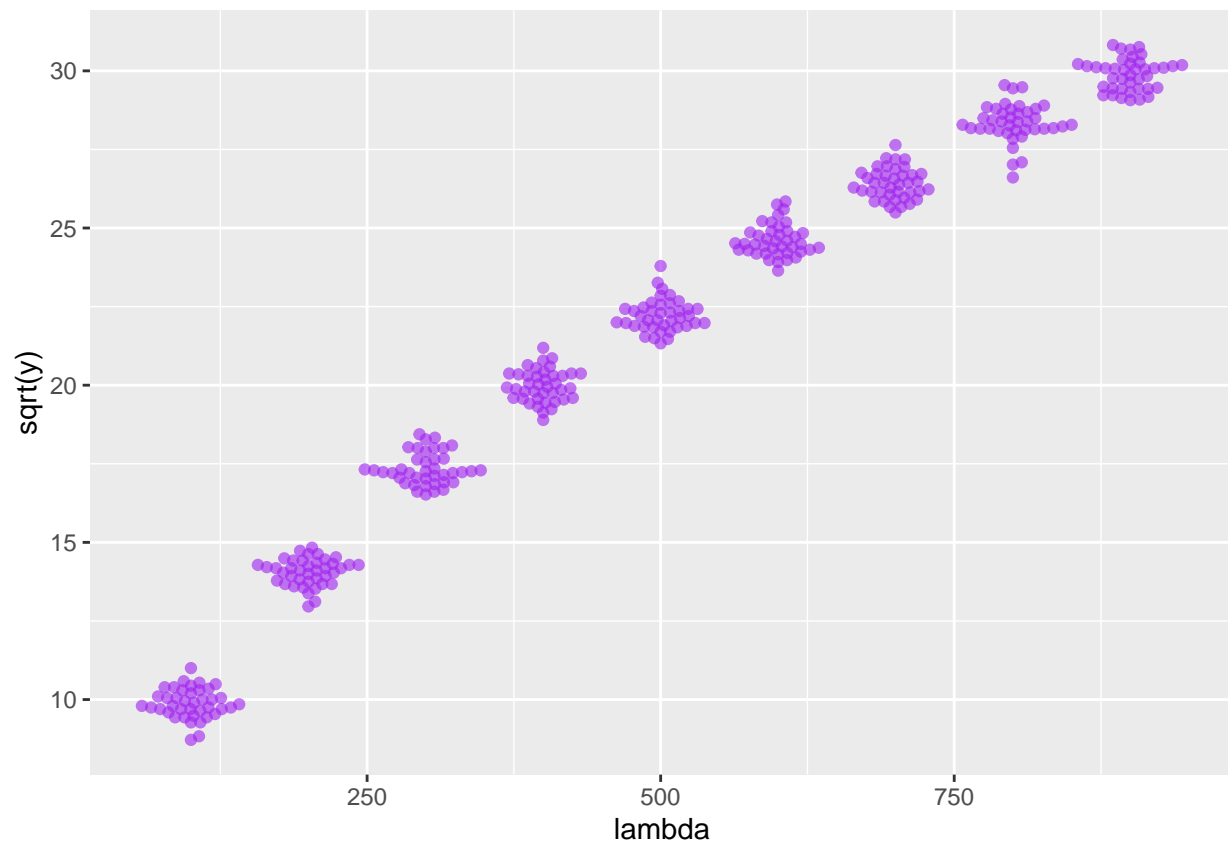
4.4.4 Variance Stabilizing Transformations

If we encounter heterogeneous variances, we may be unable to perform regressions and may need to do a transformation of variables.

```
lambdas = seq(100, 900, by = 100)
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 = "purple")
```

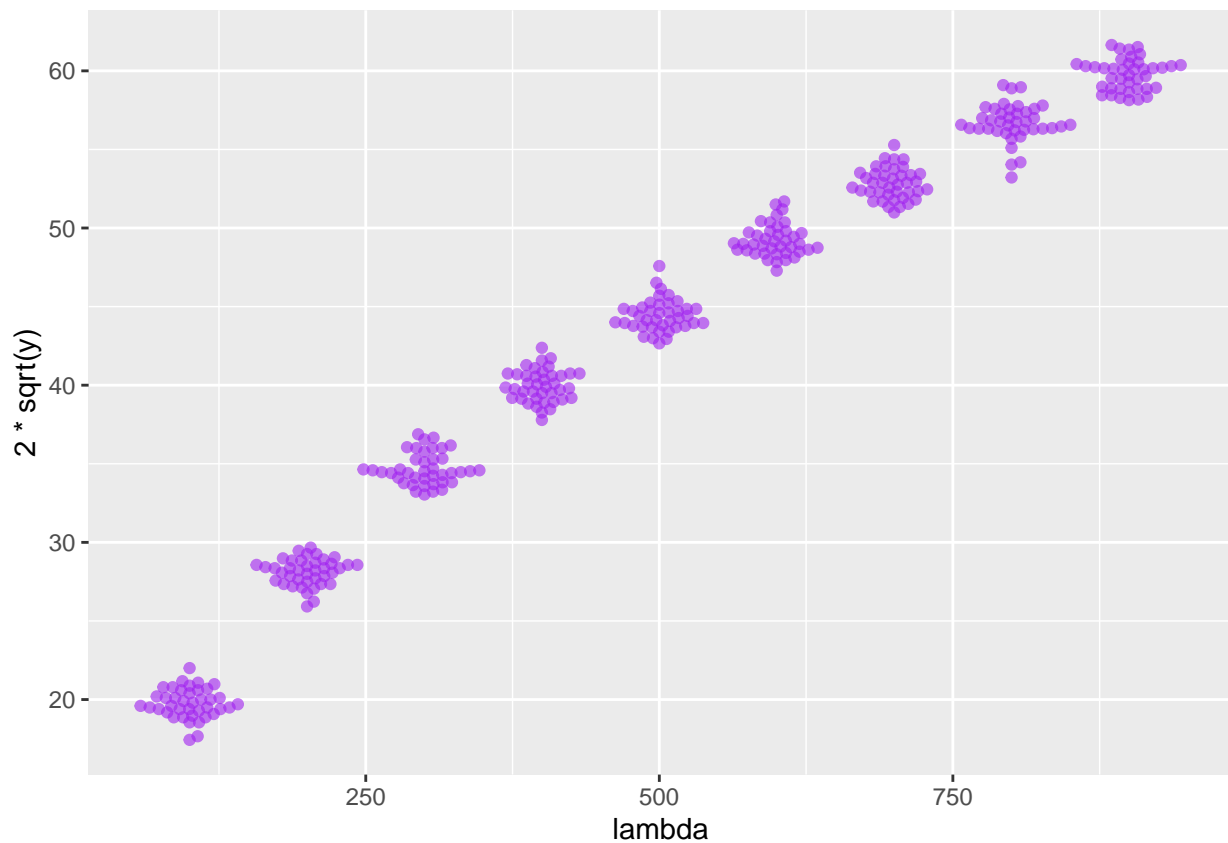


```
ggplot(simdat, aes(x = lambda, y = sqrt(y))) +  
  geom_beeswarm(alpha = 0.6, color = "purple")
```



Note that the square root transformation leads to variables with the same variance. And keep in mind that if variance is dependent upon the mean and we have randomly generated means... A transformation of $2\sqrt{y}$ will make variances approximately equal to 1.

```
ggplot(simdat, aes(x = lambda, y = 2*sqrt(y))) +  
  geom_beeswarm(alpha = 0.6, color = "purple")
```



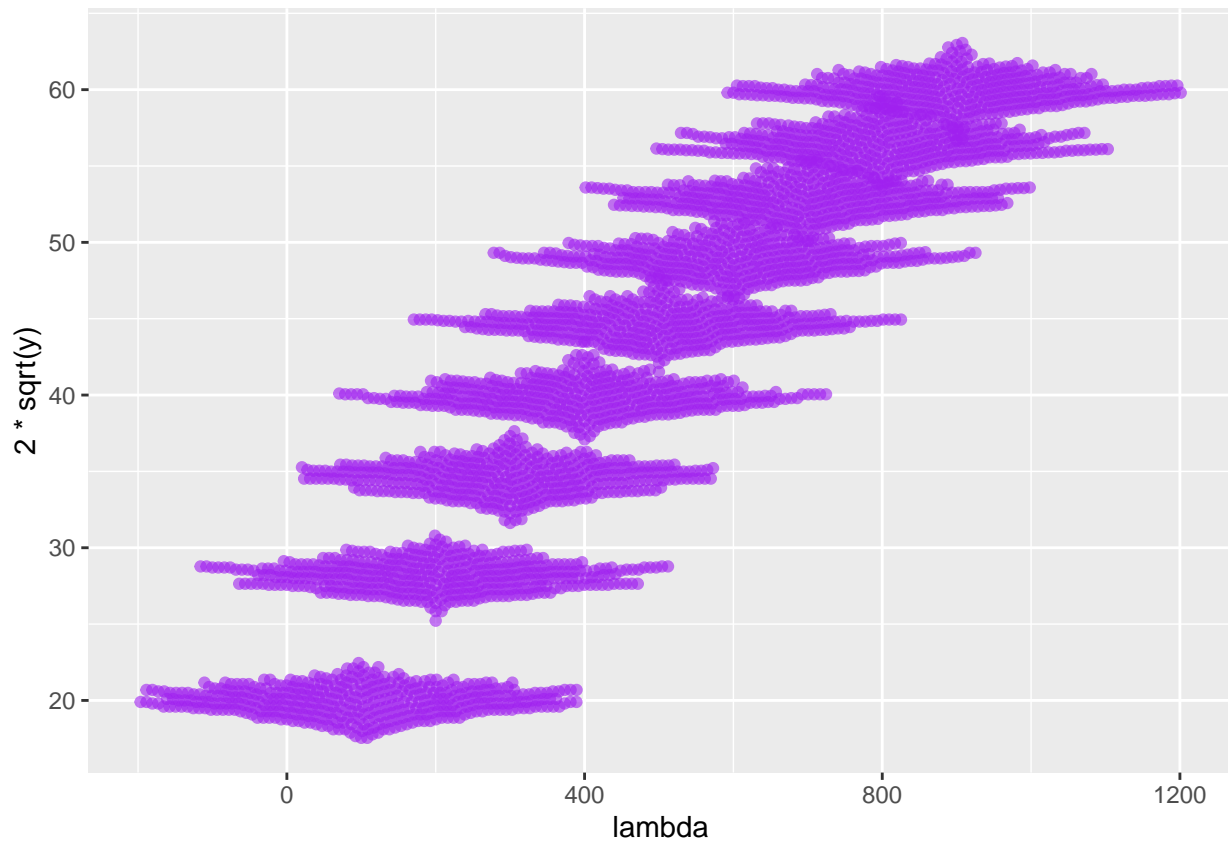
```
summarise(group_by(simdat, lambda), sd(y), sd(2*sqrt(y)))
```

```
## # A tibble: 9 x 3
##   lambda `sd(y)` `sd(2 * sqrt(y))`
##   <dbl>   <dbl>         <dbl>
## 1    100     9.26           0.940
## 2    200    11.8           0.847
## 3    300    17.7           1.01
## 4    400    20.3           1.01
## 5    500    22.6           1.01
## 6    600    25.0           1.01
## 7    700    25.9           0.979
## 8    800    33.8           1.20
## 9    900    29.7           0.994
```

Q 4.21

Using larger number of replicates:

```
simdat = lapply(lambdas, function(l)
  tibble(y = rpois(n = 400, lambda=l), lambda = l)
) %>% bind_rows
ggplot(simdat, aes(x = lambda, y = 2*sqrt(y))) +
  geom_beeswarm(alpha = 0.6, color = "purple")
```

```
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     9.25           0.928
## 2    200    13.4           0.950
## 3    300    18.1           1.05
## 4    400    19.7           0.987
## 5    500    22.6           1.01
## 6    600    24.2           0.990
## 7    700    25.9           0.980
## 8    800    29.0           1.03
## 9    900    30.3           1.01
```

Q 4.21

gamma-Poisson:

```
#Make mus a sequence of powers of 2's (length 10)
muvalues = 2^seq(0, 10, by = 1)
```

```
#Take those powers of 2, generate negbinom with those means and create 95% CIs
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
head(as.data.frame(simgp), 2)

```

```

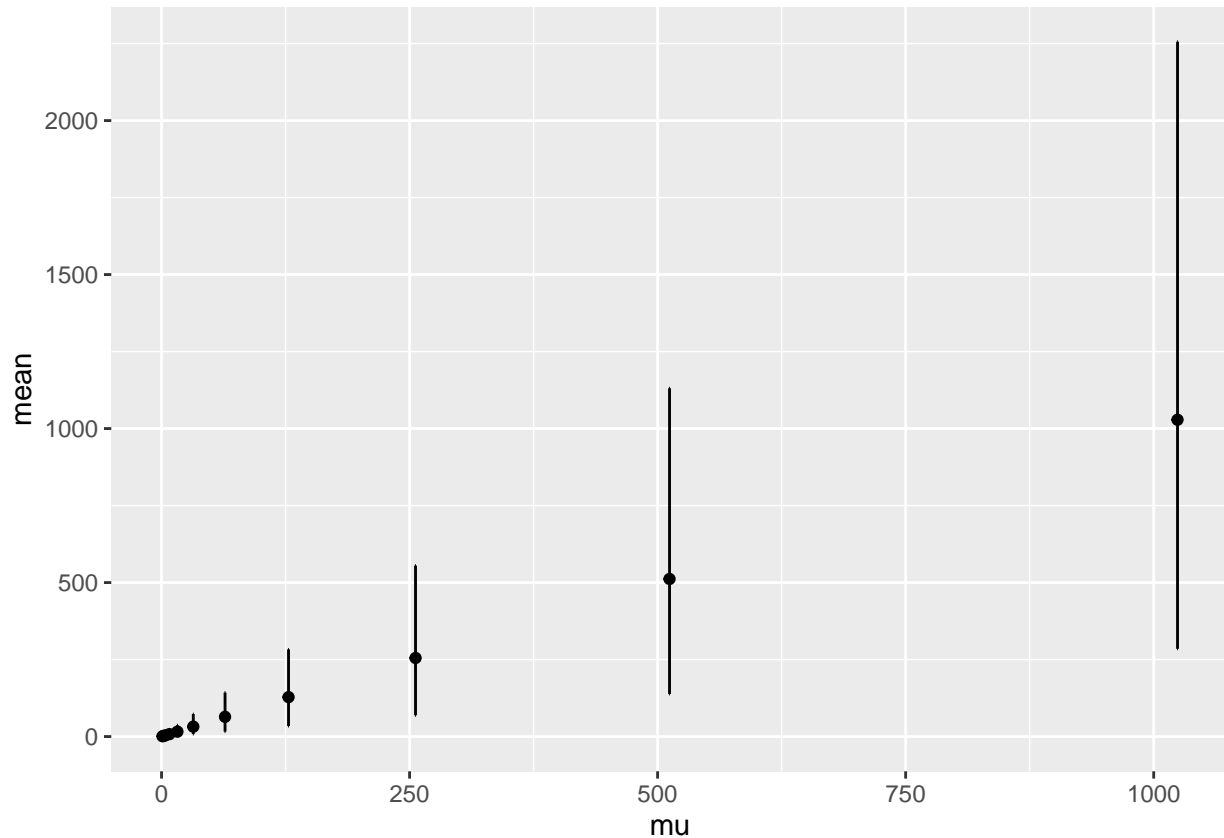
##      mean      sd lower upper mu
## 1 0.9955 1.130401    0     4   1
## 2 2.0144 1.739568    0     6   2

```

```

ggplot(simgp, aes(x = mu, y = mean, ymin = lower, ymax = upper)) +
  geom_point() + geom_errorbar()

```



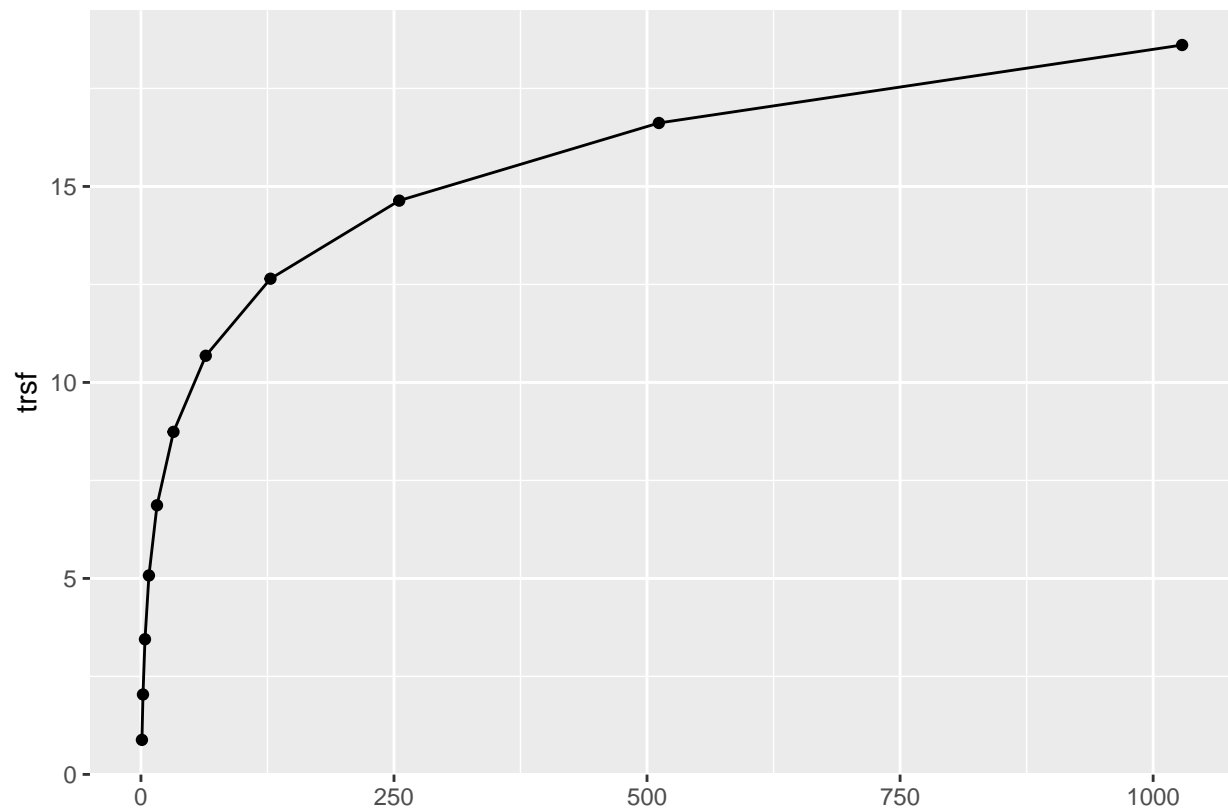
Now stabilize the variances

ALERT!!: What does this mutate function do?

```

simgp = mutate(simgp,
  slopes = 1 / sd,
  trsf = cumsum(slopes * mean))
ggplot(simgp, aes(x = mean, y = trsf)) +
  geom_point() + geom_line() + xlab("")

```

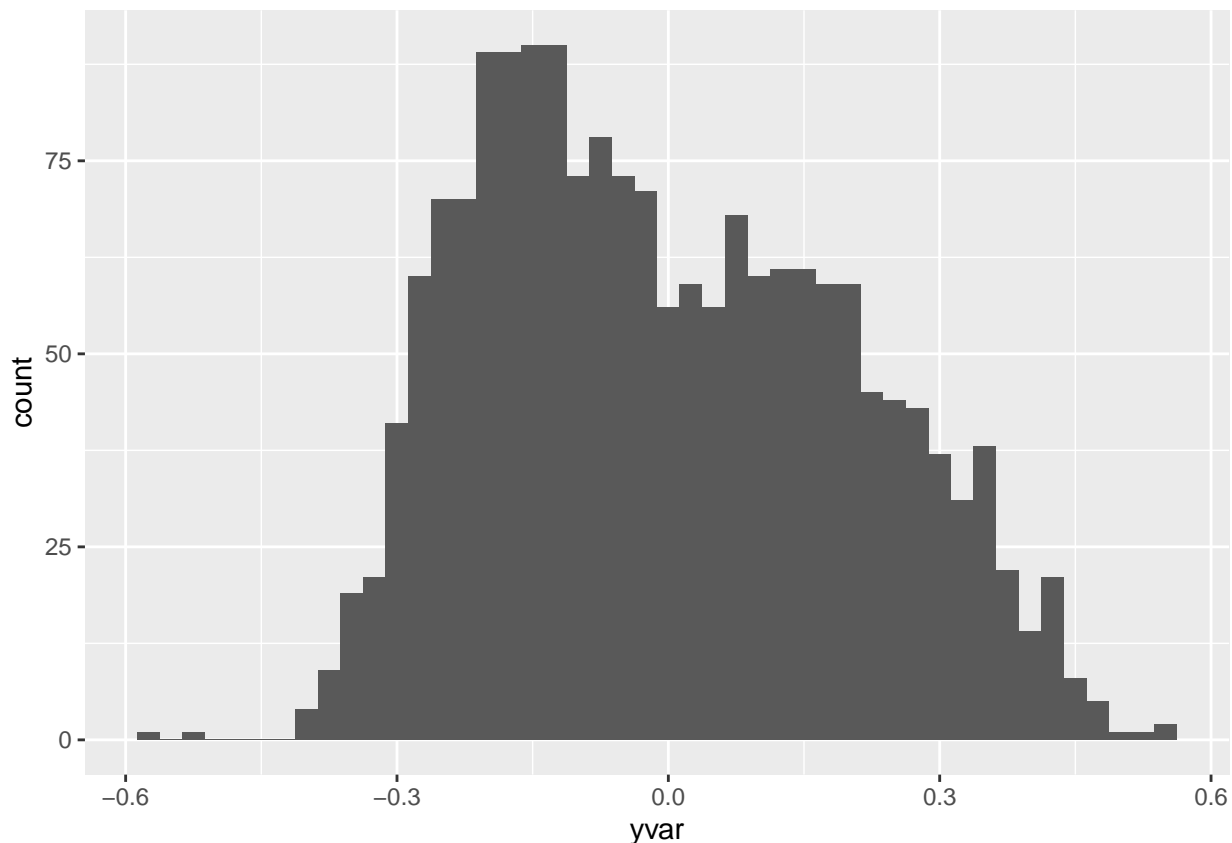


4.7 Exercises

4.1

Visualize the data that we'll model as a mixture of normals

```
yvar = readRDS(here("data", "Myst.rds"))$yvar  
ggplot(tibble(yvar), aes(x = yvar)) + geom_histogram(binwidth=0.025)
```



```
str(yvar)
```

```
##  num [1:1800] 0.3038 0.0596 -0.0204 0.1849 0.2842 ...
```

Generate a probability on 0 and 1 for coming from group *A* for each data point.

```
pA = runif(length(yvar))
pB = 1 - pA
```

Setup variables

```
#Track current iteration
iter = 0
#Current loglikelihood
loglik = -Inf
#Change in loglikelihood
delta = +Inf
#At what point we consider ceasing to improve
tolerance = 1e-3
#Minimum and max iteration
miniter = 50; maxiter = 1000
```

```
#As long as the change in loglikelihood is greater than the tolerance and we are below max iterations 0
while((delta > tolerance) && (iter <= maxiter) || (iter < miniter)) {
  #Set lambda to the average of the probabilities data belongs to group A
  lambda = mean(pA)
  #Set mean of A to weight average of data weighted by probability of being in A
  muA = weighted.mean(yvar, pA)
  #Set mean of B to weight average of data weighted by probability of being in B
```

```

muB = weighted.mean(yvar, pB)
#Ditto for standard deviations of A and B
sdA = sqrt(weighted.mean((yvar - muA)^2, pA))
sdB = sqrt(weighted.mean((yvar - muB)^2, pB))

#Likelihoods of each data being in A according to f(x) with current mean and sd of A
phiA = dnorm(yvar, mean = muA, sd = sdA)
#Likelihoods of each data being in B according to f(x) with current mean and sd of A
phiB = dnorm(yvar, mean = muB, sd = sdB)
#Update pA by multiplying likelihoods by average pA value
pA = lambda * phiA
#Update pB by multiplying likelihoods by 1- average pA value
pB = (1 - lambda) * phiB
#Find sum of new pA + pB
ptot = pA + pB
#Re-weight pA and pB so that they still sum to 1
pA = pA / ptot
pB = pB / ptot

## M - Step
#Assign current loglikelihood to a temp variable
loglikOld = loglik
#New loglikelihood is the sum of the logs of the probabilities each data point is from A
loglik = sum(log(pA))
#Find change in loglikelihood
delta = abs(loglikOld - loglik)
#Increase count of iterations
iter = iter + 1
}

#Print estimated values
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    -0.169 0.0983
## 2 B     0.147 0.150

```

Compare to normalmixEM

```
normalmix.output<-normalmixEM(yvar)
```

```
## number of iterations= 289
```

```
#lambdas
```

```
normalmix.output$lambda
```

```
## [1] 0.4756155 0.5243845
```

```
#mus
```

```
normalmix.output$mu
```

```
## [1] -0.1693591 0.1473216
```

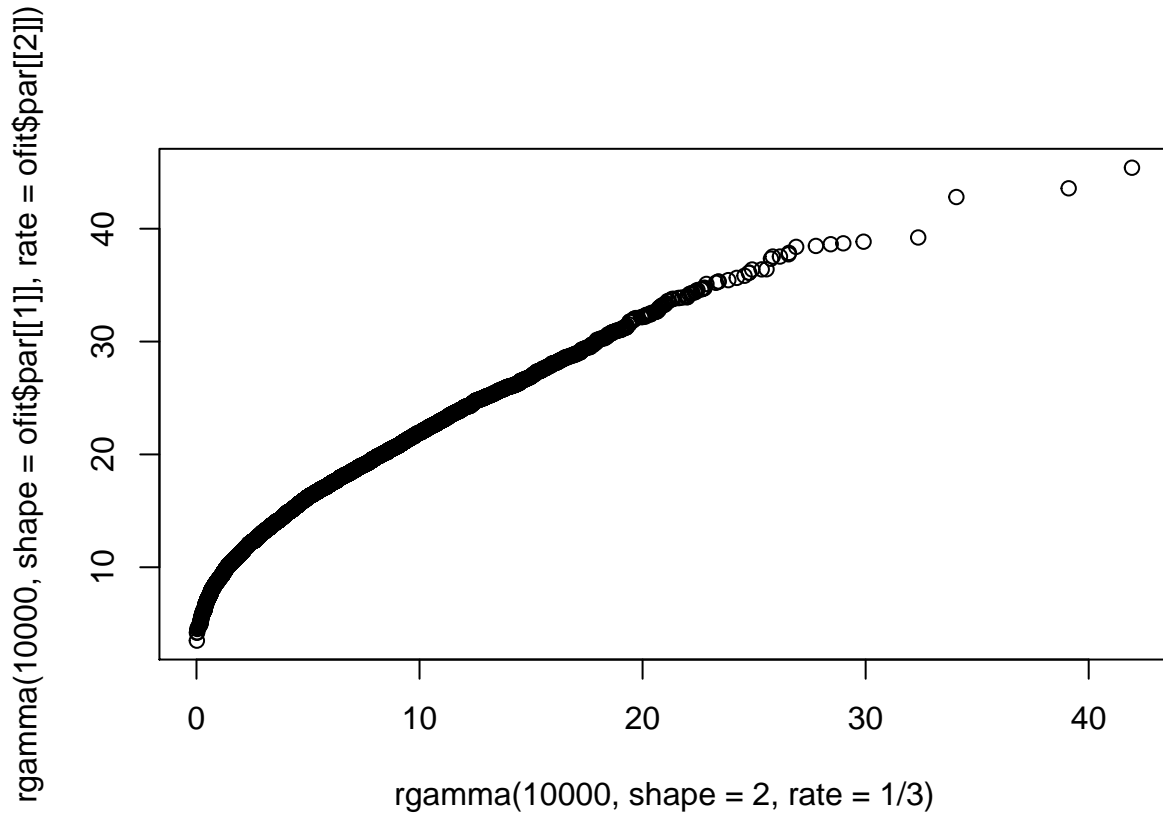
```
#sigmas
```

```
normalmix.output$sigma
```

```
## [1] 0.0982653 0.1497879
```

Q 4.2

```
qqplot(y=rgamma(10000, shape=ofit$par[[1]], rate=ofit$par[[2]]),  
       x=rgamma(10000, shape = 2, rate = 1/3))
```



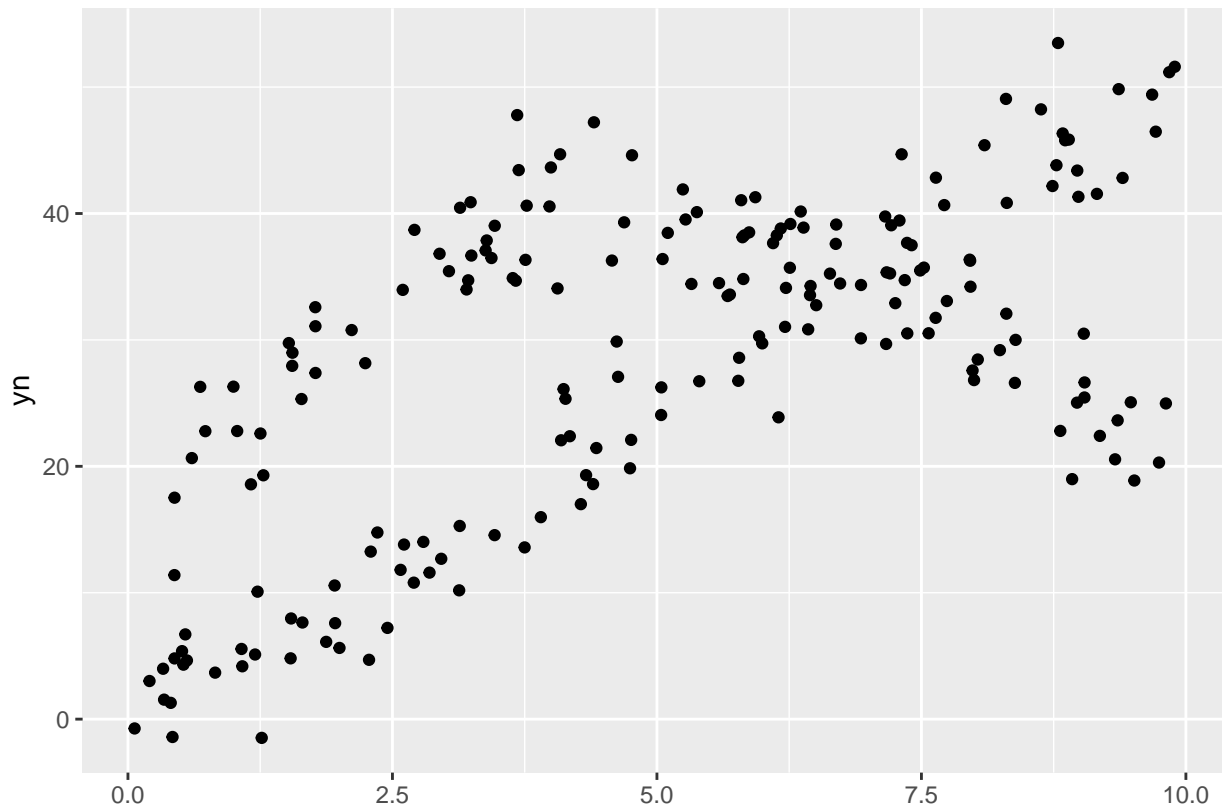
Q 4.3

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

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

```
data("NPreg")
```

```
ggplot(NPreg, aes(x = x, y = yn)) +  
  geom_point() + xlab("")
```



ALERT!!!: So first this was "misclassifying" nearly everything because it was giving opposite labels. I switched out `m1cluster` to `as.factor(m1cluster)` and it suddenly it working "correct".

```
#k is the number of clusters
m1 = flexmix(y_n ~ x + I(x^2), data = NPreg, k=2)
groups.pred = as.factor(m1@cluster)
groups.true = as.factor(NPreg$class)
truth.table = data.frame(True = groups.true, Predicted = groups.pred,
                          Correct = c(groups.true == groups.pred))
head(truth.table)
```

```
##   True Predicted Correct
## 1    1          2  FALSE
## 2    1          2  FALSE
## 3    1          2  FALSE
## 4    1          2  FALSE
## 5    1          2  FALSE
## 6    1          2  FALSE
```

Replotting with estimated classes:

```
ggplot(data.frame(NPreg, groups.pred), aes(x = x, y = y_n, color = groups.pred)) +
  geom_point() + xlab("")
```

