Q4 - Bayesian Anscombe

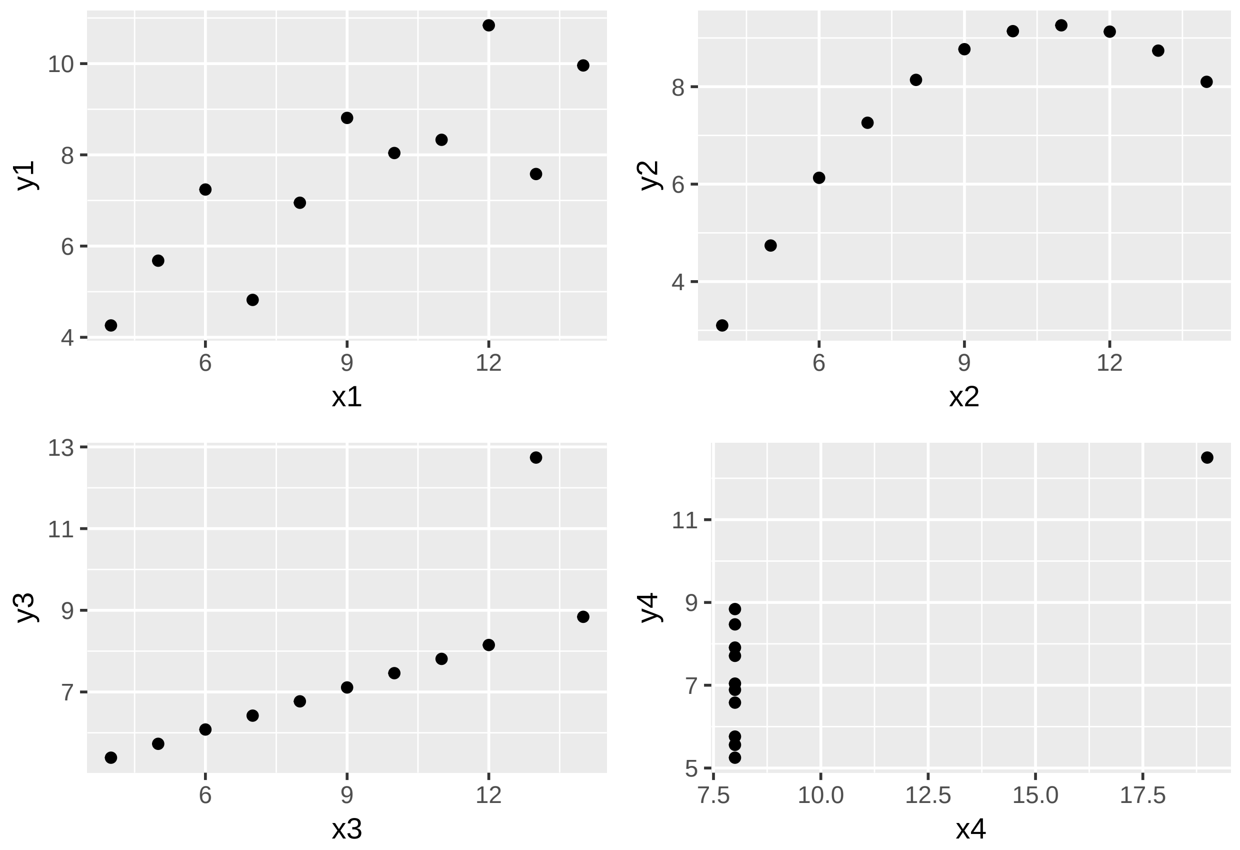
Varun Nayyar

30/04/2019

Fit linear regression's to anscombe's quartet to get good fits (type 'anscombe' into R). - Can you choose good prior's and models to get good fits?

Let's plot the anscombe quartet

```r
par(mfrow=c(2,2))
p1 = ggplot(anscombe, aes(x1, y1)) + geom_point()
p2 = ggplot(anscombe, aes(x2, y2)) + geom_point()
p3 = ggplot(anscombe, aes(x3, y3)) + geom_point()
p4 = ggplot(anscombe, aes(x4, y4)) + geom_point()
grid.arrange(p1, p2, p3, p4)
```



This is famous for having the exact same linear regression despite having very different true underlying data.

I'm going to do an example on quartet2. I recommend trying different versions and seeing the effect on the posteriors

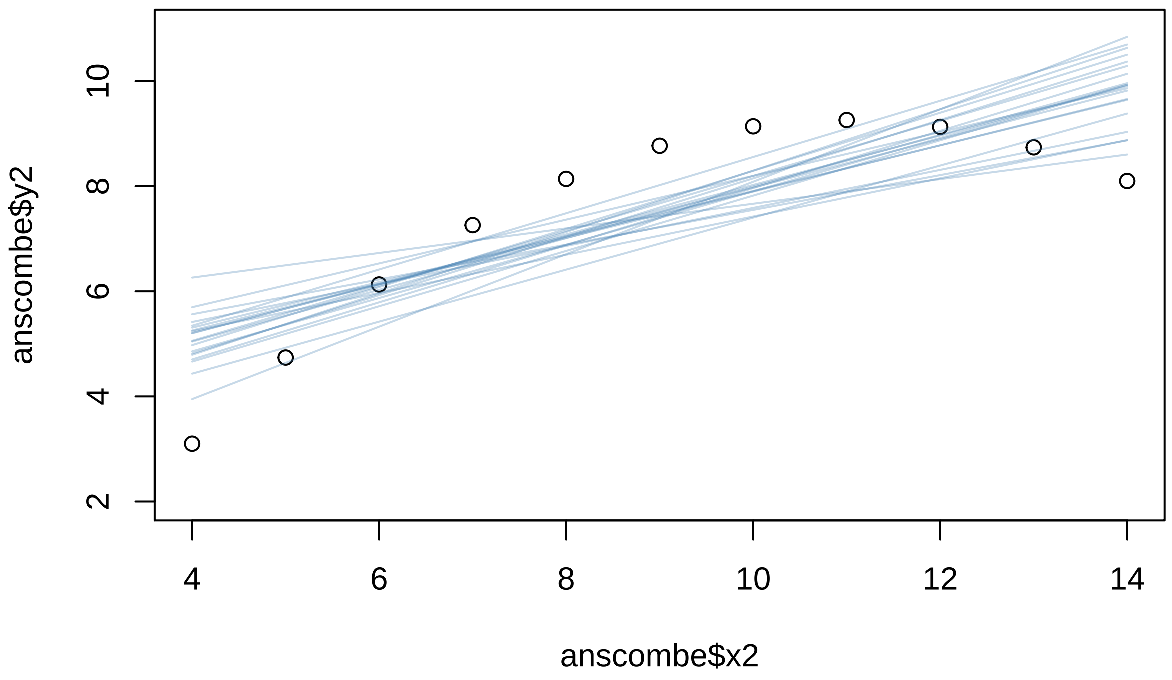
```r
# copied from McElreath
library(rethinking)
```

```
## Loading required package: rstan
## Loading required package: StanHeaders

## rstan (Version 2.18.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

## Loading required package: parallel

## rethinking (Version 1.80)
```

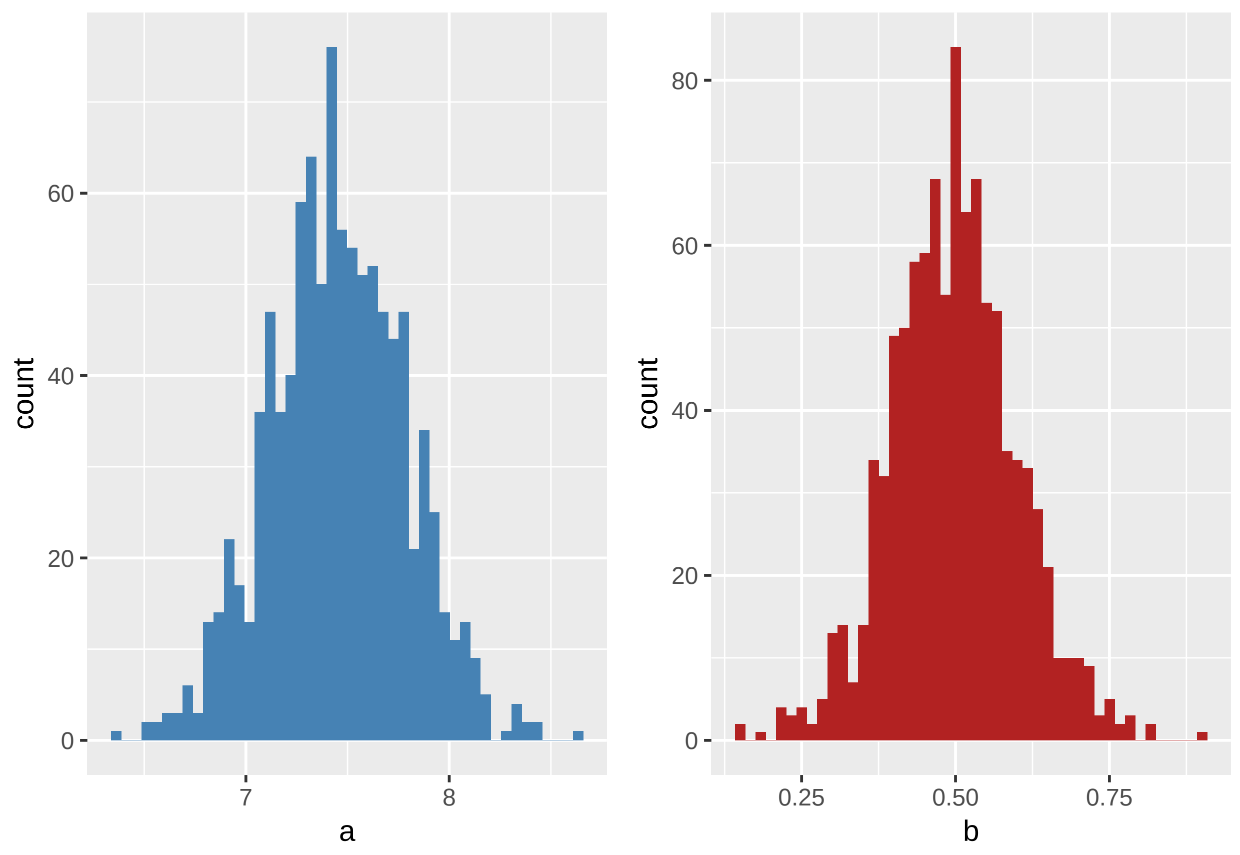
```
mN <- quap(
  alist(
    y2 ~ dnorm( mu , sigma ) ,
    mu <-  a + b* normx,
    normx <-  x2 - mean(x2),
    a ~ dnorm( 3 ,  3) ,
    b ~ dnorm( 0.5 , 3 ) ,
    sigma ~ dunif( 0 , 3 )
  ), data=anscombe )

post = extract.samples( mN , n=20 )
precis(mN)
```

```
##            mean        sd      5.5%    94.5%
## a     7.4445409 0.3365698 6.9066373 7.982444
## b     0.5000002 0.1067699 0.3293614 0.670639
## sigma 1.1205215 0.2394945 0.7377630 1.503280
```

```
plot(anscombe$x2, anscombe$y2,
     ylim=c(2, 11))
#plot(p2)
for (i in 1:20){
  curve( post$a[i] + post$b[i]*(x-mean(anscombe$x1)) ,add=TRUE, col=col.alpha("steelblue",0.3))
}
```



```r
post2 = extract.samples( mN , n=1000 )
ap = ggplot(post2) + geom_histogram(aes(a), bins=46, fill="steelblue")
bp = ggplot(post2) + geom_histogram(aes(b), bins=46, fill="firebrick")
grid.arrange(ap, bp, nrow=1)
```

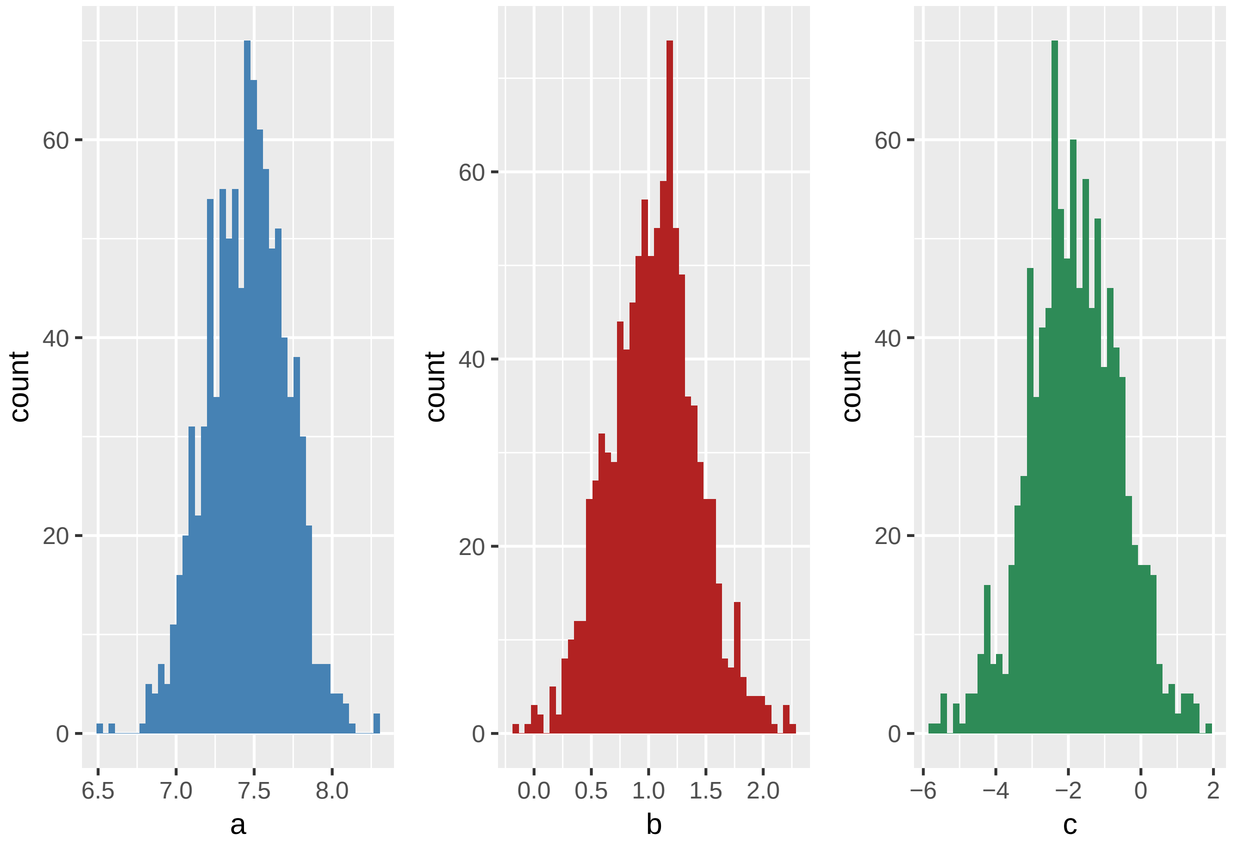


- Show what happens when the quadratic term is added to the 2nd quartet on the posterior of the other terms.

```
ansfilter = anscombe %>% transmute(y2, normx2 = x2-mean(x2), x2sq=x2^2) %>% mutate(normx2sq = (x2sq-mea

mQ <- quap(
  alist(
    y2 ~ dnorm( mu , sigma ) ,
    mu <-  a + b * normx2 + c * normx2sq ,

    a ~ dnorm( 3, 3) ,
    b ~ dnorm( 0 , 3 ) ,
    c ~ dnorm(0, 1),
    sigma ~ dunif( 0 , 3 )
  ), data=ansfilter,
  #start = list(a=7.5, b=2.78, c=-0.1267),
  #start = list(a=7.5, b=0.5, c=-0.1267),
  debug=FALSE)

precis(mQ)
```

```
##             mean        sd       5.5%      94.5%
## a      7.4684708 0.2554018  7.0602894 7.87665226
## b      1.0536122 0.3698273  0.4625567 1.64466775
## c     -1.8606623 1.2110735 -3.7961916 0.07486698
## sigma  0.8477207 0.2534593  0.4426438 1.25279761
```

```
postQ = extract.samples( mQ , n=1000 )
ap = ggplot(postQ) + geom_histogram(aes(a), bins=46, fill="steelblue")
bp = ggplot(postQ) + geom_histogram(aes(b), bins=46, fill="firebrick")
cp = ggplot(postQ) + geom_histogram(aes(c), bins=46, fill="seagreen")
grid.arrange(ap, bp, cp, nrow=1)
```



- Use a T distribution on the error model for the 3rd quartet, one with a low df and another with high df.

Left as an exercise for the reader Use a `dt` instead of a `dnorm`. Note the `ncp` parameter and ordering.

- Invert y and x for Anscombe's 4th and model. Can you make the prior strong enough to overcome the outlier?

Left as an exercise for the reader. Play with the various priors on the `slope` and see what gives more robustness. How much information are you adding to this system?

- (Bonus) Without using quap, can you create a naive linear regression using something like rejection sampling?

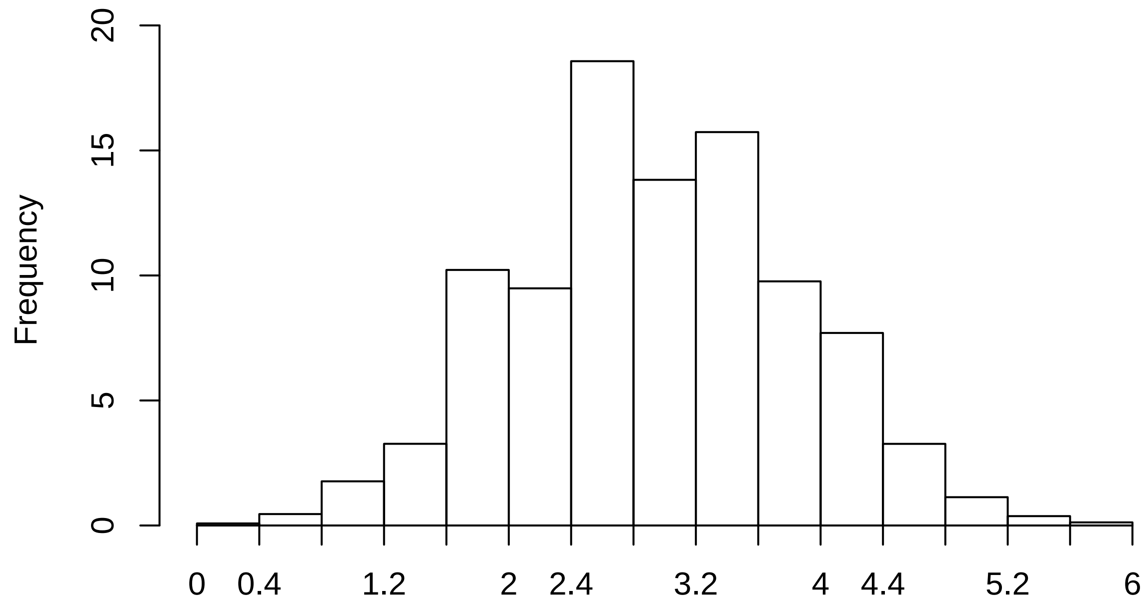
```
x = anscombe$x1
y = anscombe$y1

Nsample = 10000
a = rnorm(Nsample, 3, 3)
b = rnorm(Nsample, 0.5, 1)

outval = rep(0, Nsample)
# we assume that the datalen is much smaller
```

```r
# than the number of parameters to try
for (i in 1:length(x)){
  pred = a + b * x[i]
  # we use the log likelihood since
  # prod(probabilities) = sum(logprobs)
  # and we have less underflow this way
  outval = outval + dnorm(y[i], mean = pred, sd = 1, log = T)
}
# normalise and turn into probabilities
probs = (outval - max(outval)) %>% exp
```

Using a technique called importance sampling, I draw a weighted histogram. Effectively each sample is proportional to it's probability. Hence things closer to the centre are worth more

```r
library(plotrix)
weighted.hist(a, probs, seq(0,6,0.4))
```

```
## Warning in weighted.hist(a, probs, seq(0, 6, 0.4)): Not all values will be
## included in the histogram
```



```r
sum(probs)
```

```
## [1] 95.80634
```

For a 1000 sample, the above is a measure of how many samples worth of inference I've done