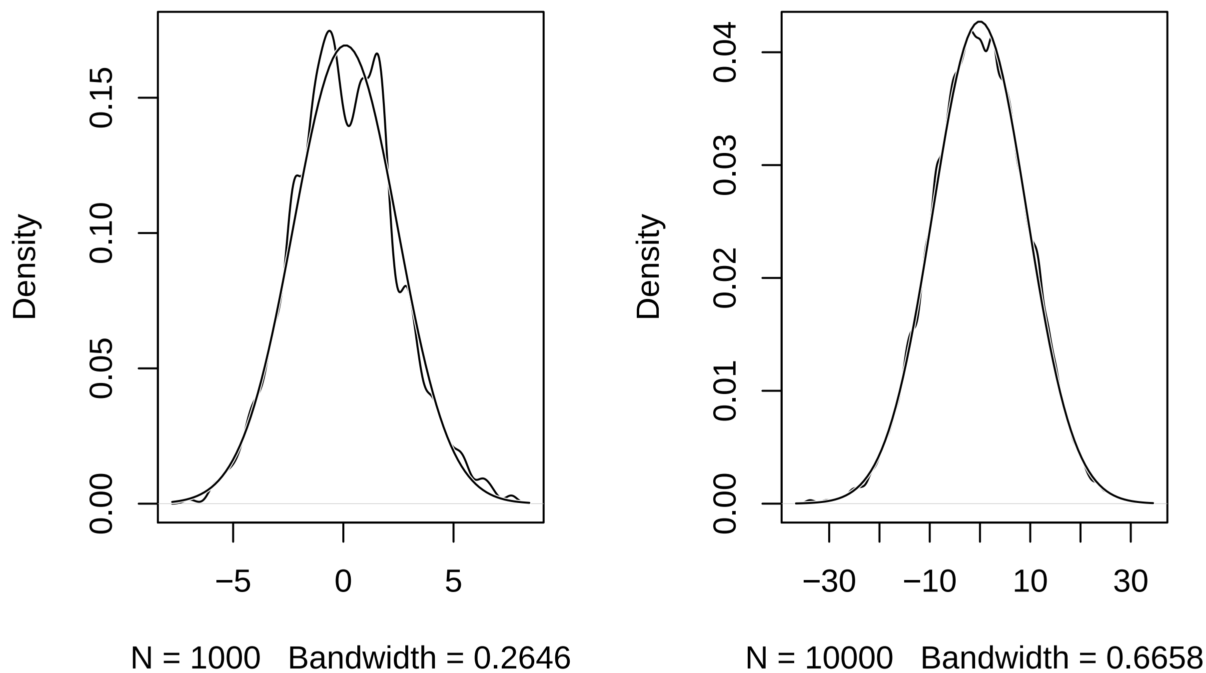
4 - Linear Models

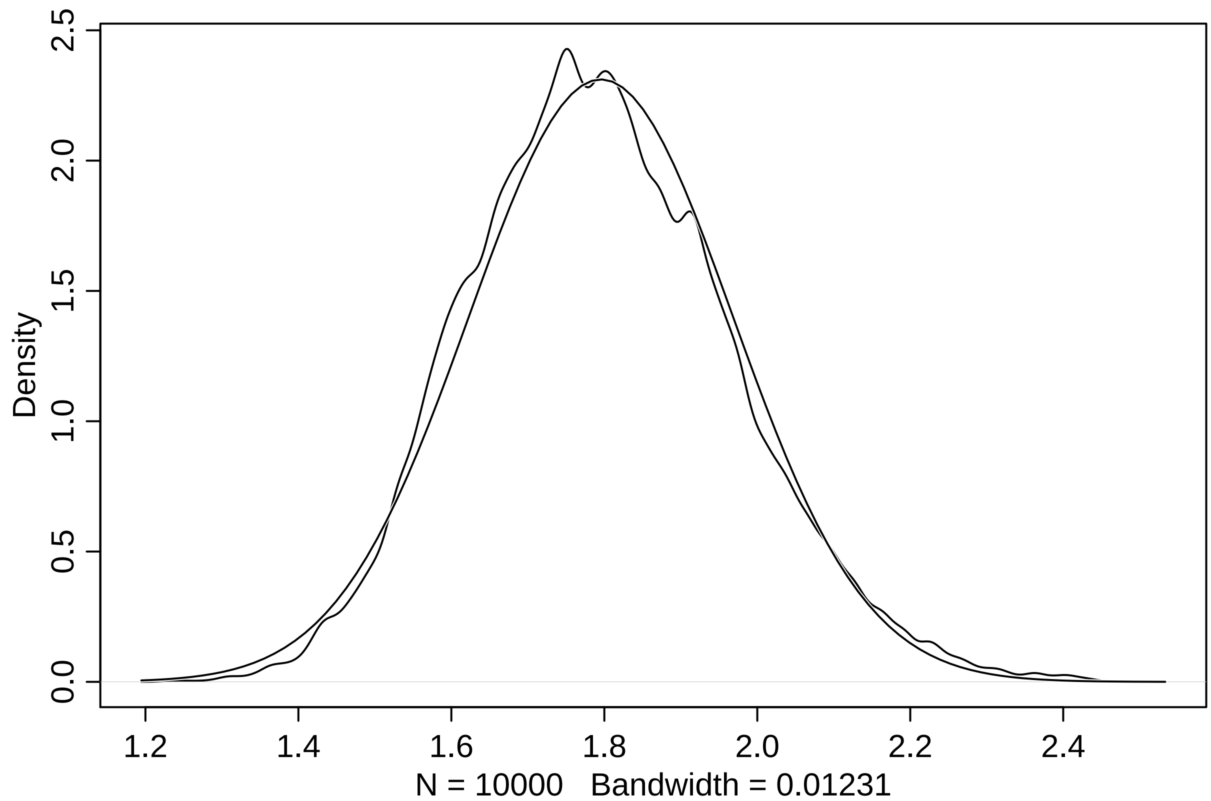
4.1.1. Normal by addition

```
# 4.1
pos <- replicate(1000, sum(runif(16, -1, 1)))
par(mfrow=c(1, 2))
dens(pos, norm.comp = T)
dens(replicate(10000, sum(runif(256, -1, 1))), norm.comp = T)
```

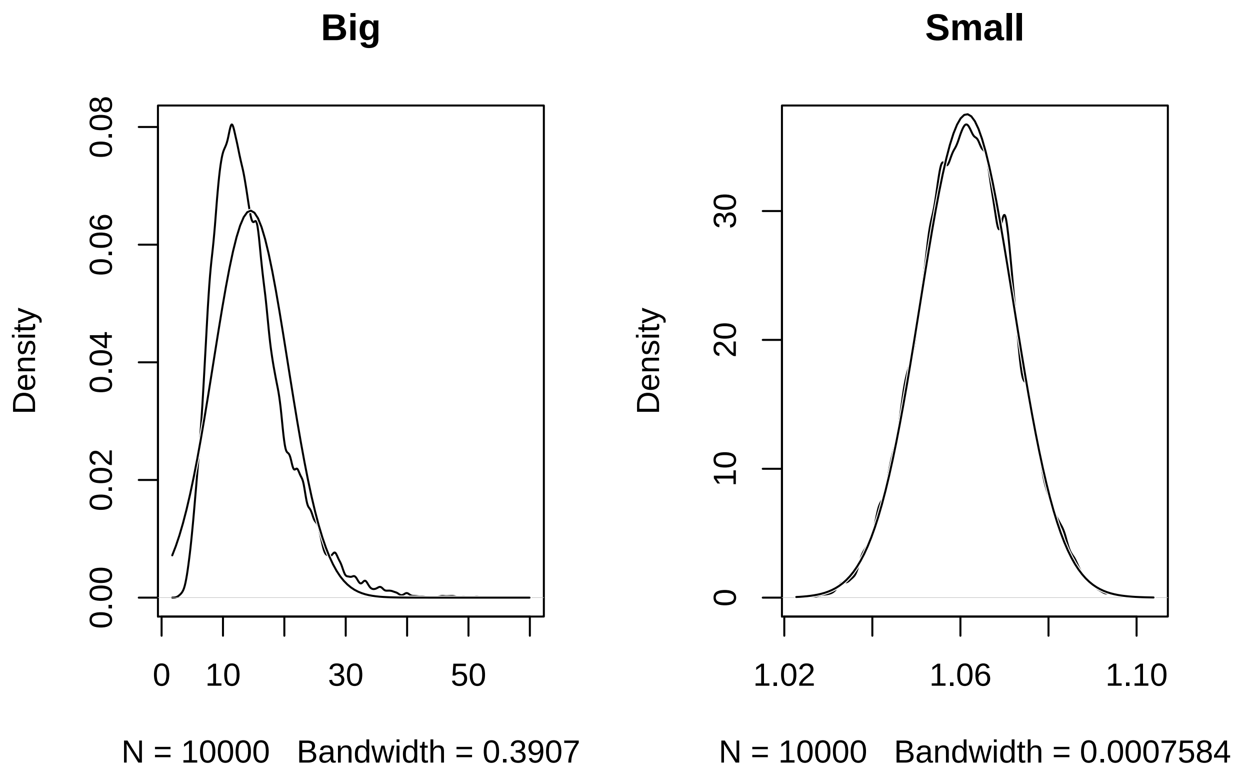


4.1.2. Normal by multiplication

```
# 4.2
dens(replicate(1e4, prod(1 + runif(12, 0, 0.1))), norm.comp = T)
```

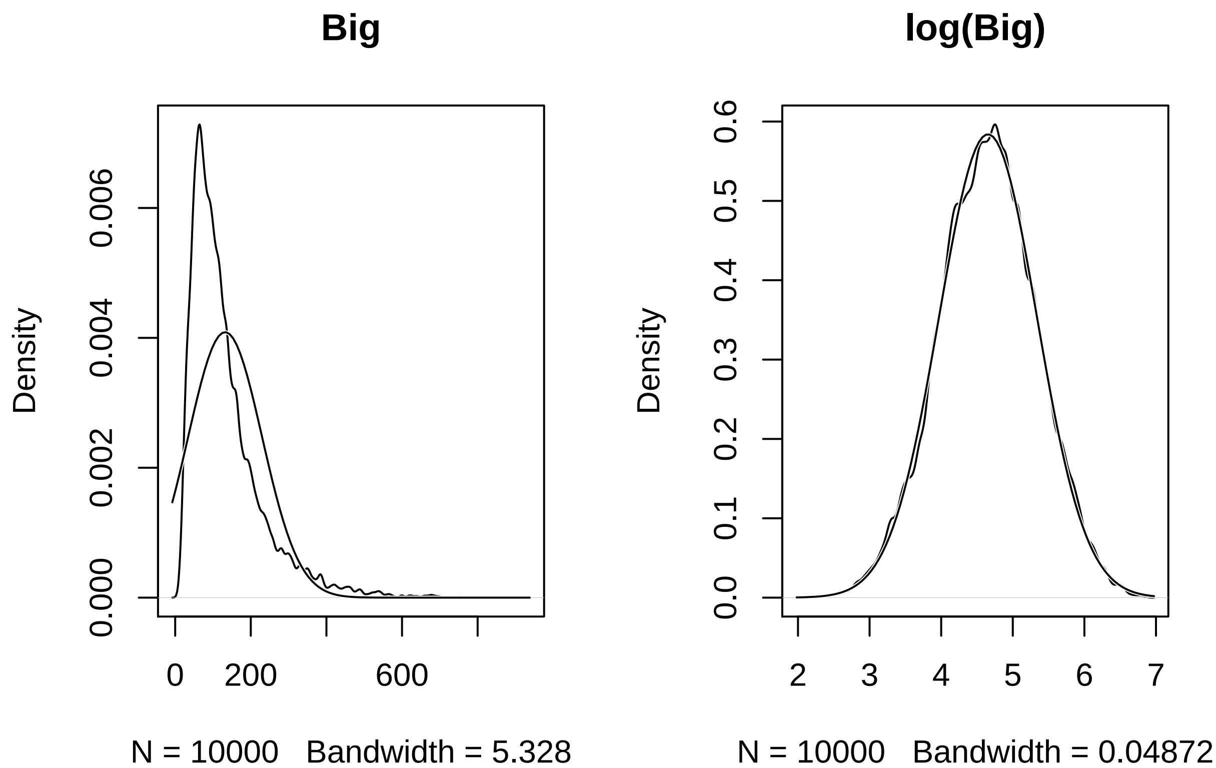



```r
# 4.4
big <- replicate(1e4, prod(1 + runif(12, 0, 0.5)))
small <- replicate(1e4, prod(1 + runif(12, 0, 0.01)))
par(mfrow=c(1, 2))
dens(big, norm.comp = T, main = "Big")
dens(small, norm.comp = T, main = "Small")
```



Normal by log-multiplication

```r
# 4.5
big <- replicate(1e4, prod(1 + runif(12, 0, 1)))
log_big <- log(big)
par(mfrow=c(1, 2))
dens(big, norm.comp = T, main = "Big")
dens(log_big, norm.comp = T, main = "log(Big)")
```



4.3 A Gaussian model of height

```r
# 4.7
library(rethinking)
data(Howell1)
d <- Howell1
```

```r
# 4.8
str(d)
```

```
## 'data.frame':    544 obs. of  4 variables:
##  $ height: num  152 140 137 157 145 ...
##  $ weight: num  47.8 36.5 31.9 53 41.3 ...
##  $ age   : num  63 63 65 41 51 35 32 27 19 54 ...
##  $ male  : int  1 0 0 1 0 1 0 1 0 1 ...
```

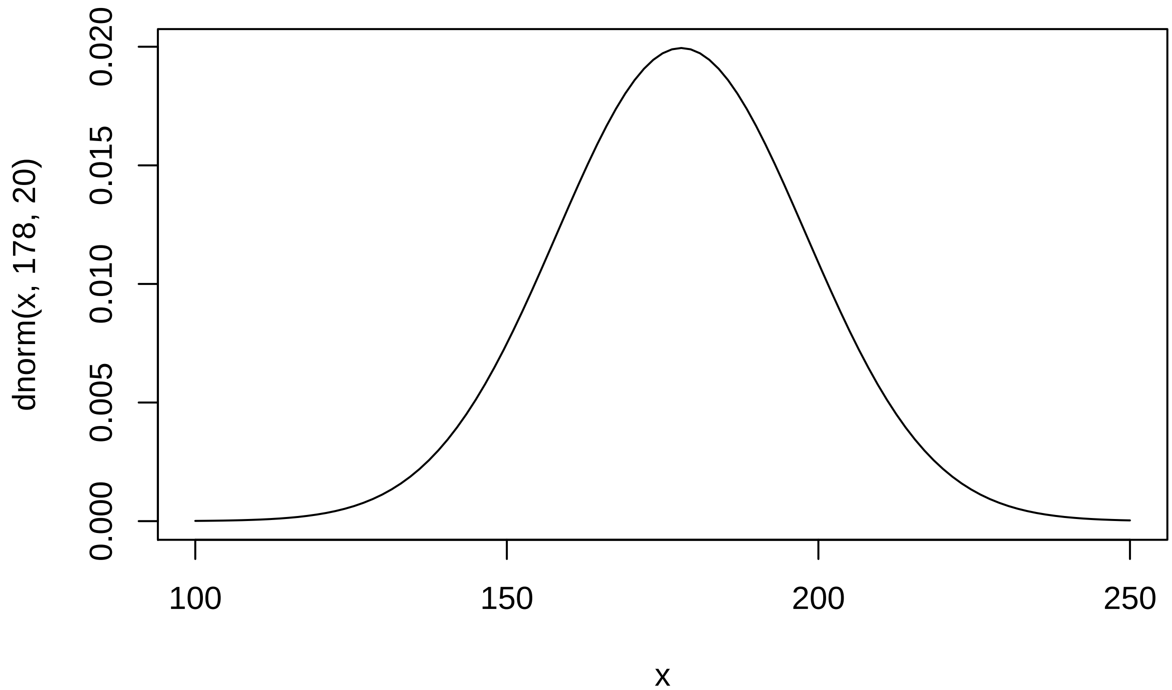
We want heights of adults only (352 rows):

```r
# 4.10
d2 <- d[d$age >= 18, ]
```

4.3.2 The model

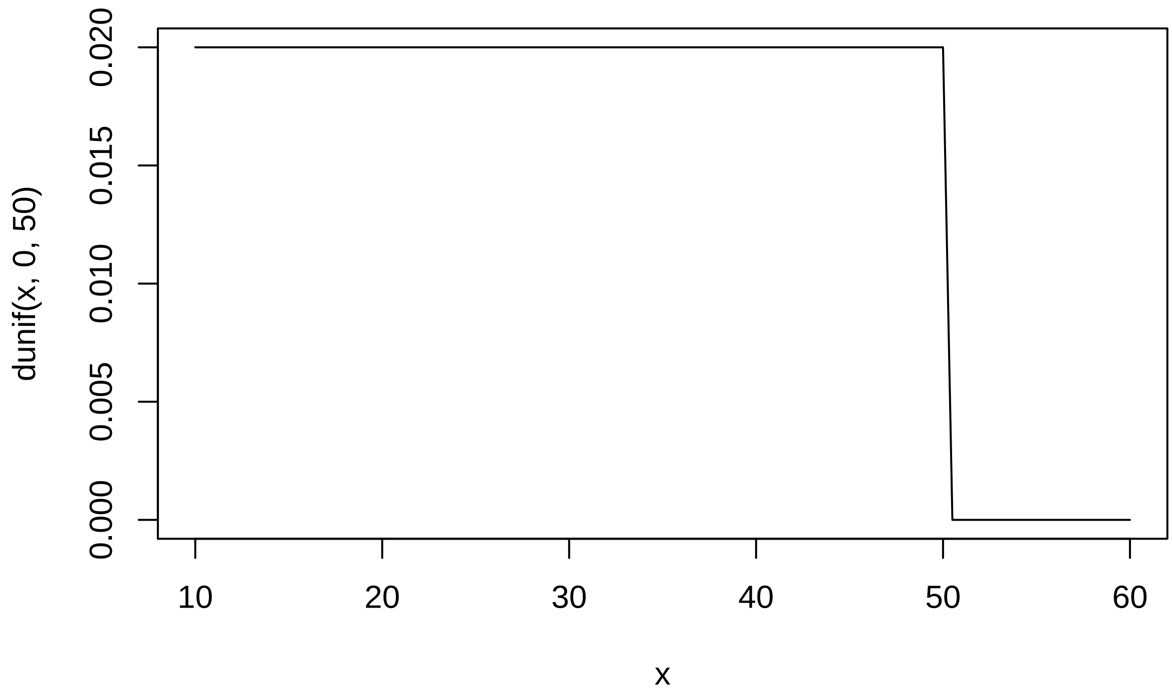
Height mean:

```r
# 4.11
curve(dnorm(x, 178, 20), from=100, to=250)
```

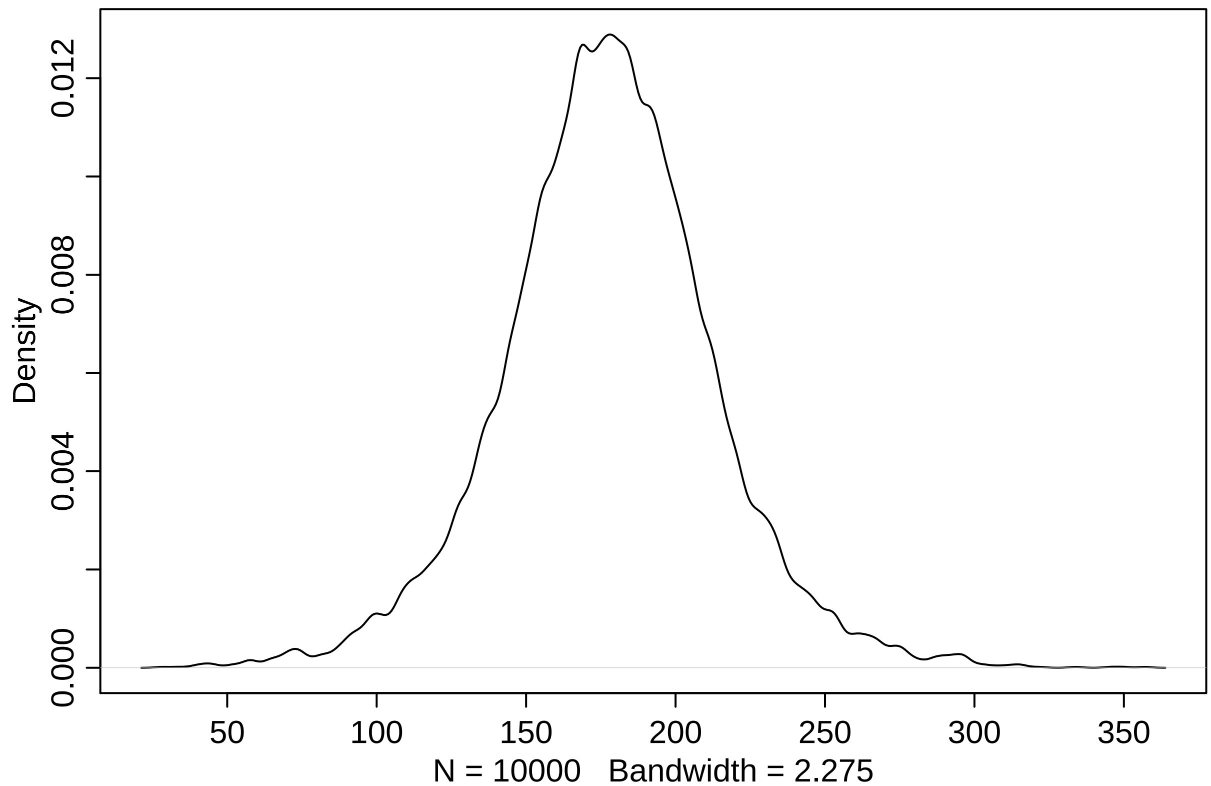


Height standard deviation:

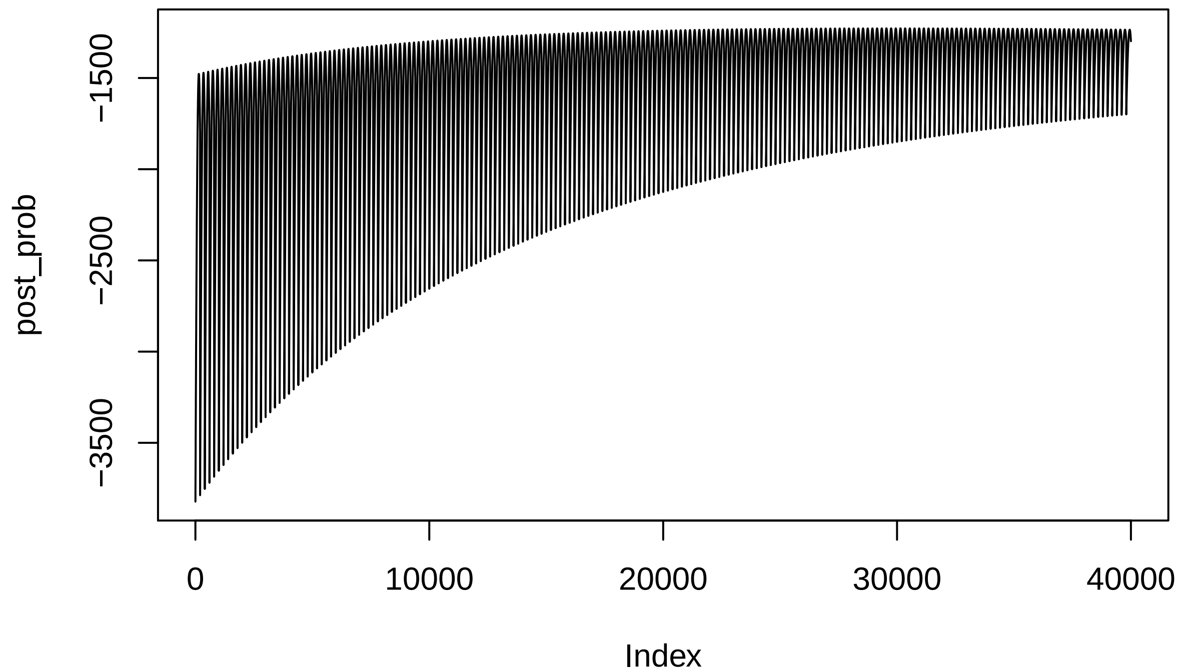
```r
# 4.12
curve(dunif(x, 0, 50), from=10, to=60)
```




```r
# 4.13
sample_mu <- rnorm(1e4, 178, 20)
sample_sigma <- runif(1e4, 0, 50)
prior_h <- rnorm(1e4, sample_mu, sample_sigma)
dens(prior_h)
```

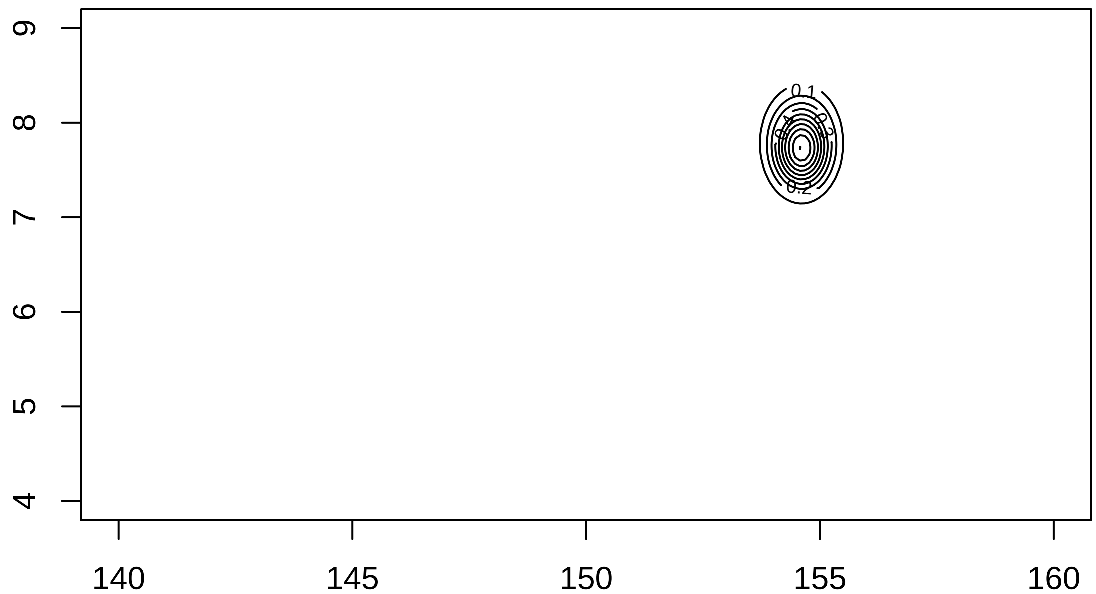



```
# 4.14
mu_list <- seq(from=140, to=160, length.out=200)
sigma_list <- seq(from=4, to=9, length.out=200)
post <- expand.grid(mu=mu_list, sigma=sigma_list)
post_ll <- sapply(1:nrow(post), function(i) sum(dnorm(
  d2$height,
  mean=post$mu[i],
  sd=post$sigma[i],
  log=T
)))
post_prob <- post_ll + dnorm(post$mu, 178, 20, T) + dunif(post$sigma, 0, 50, T)
plot(post_prob, type="l")
```

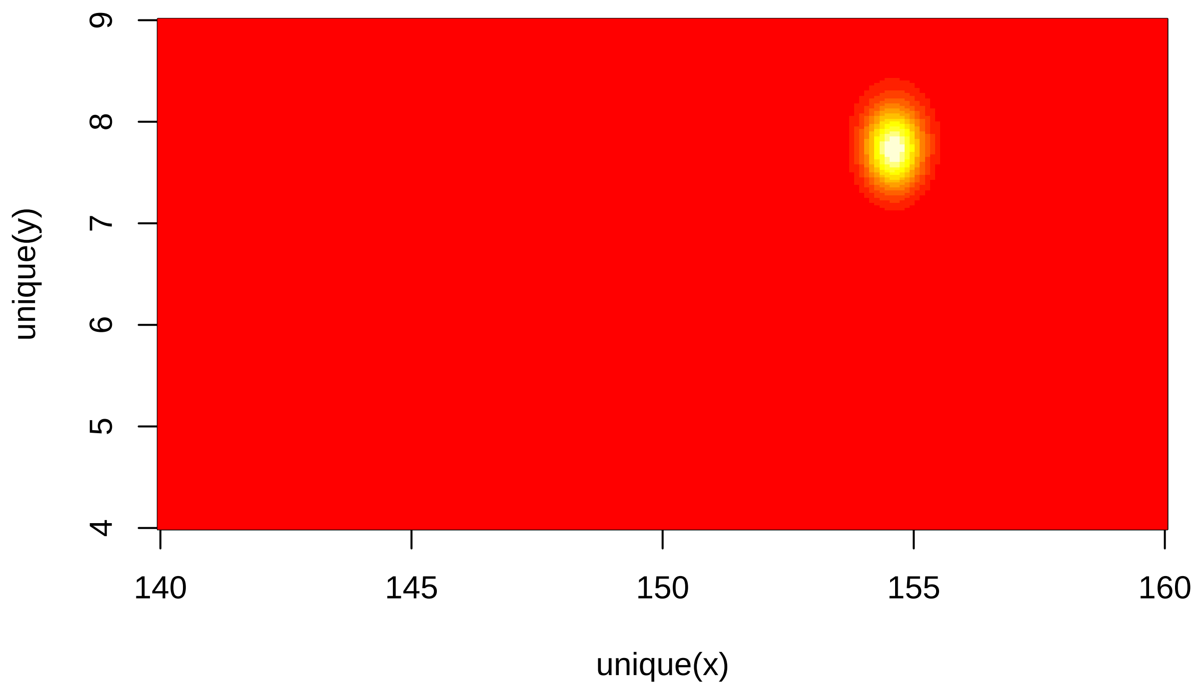



```r
post_prob <- exp(post_prob - max(post_prob))
```

```r
# 4.15
contour_xyz(post$mu, post$sigma, post_prob)
```



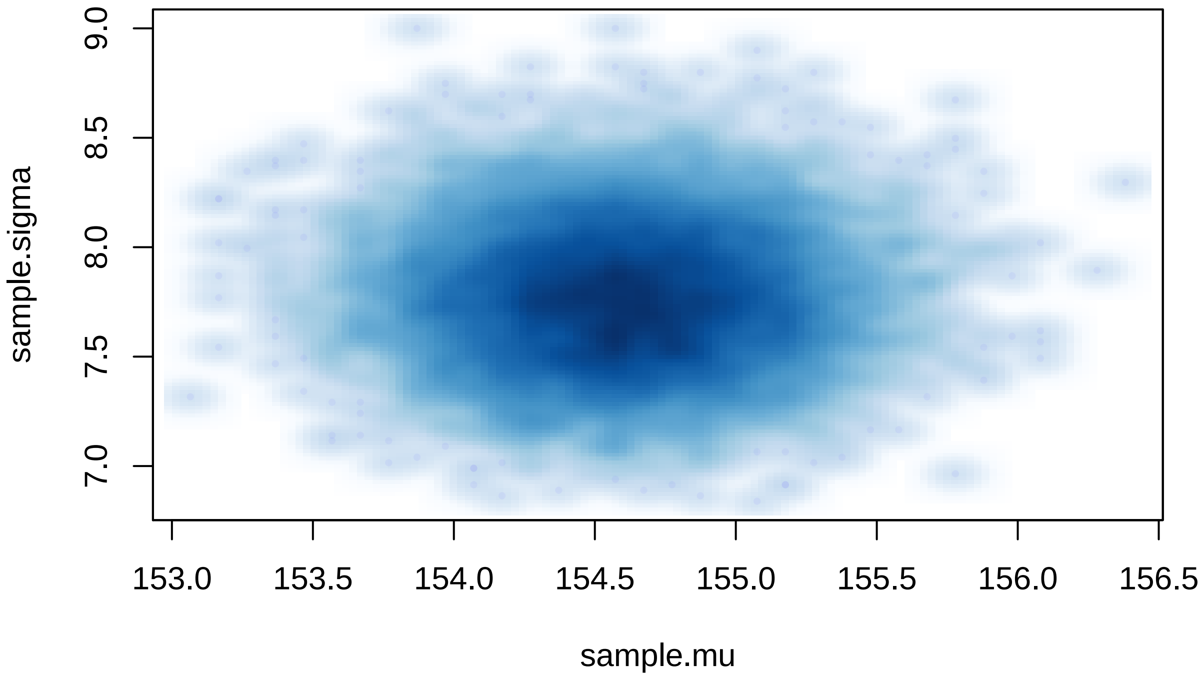

```r
# 4.16
image_xyz(post$mu, post$sigma, post_prob)
```



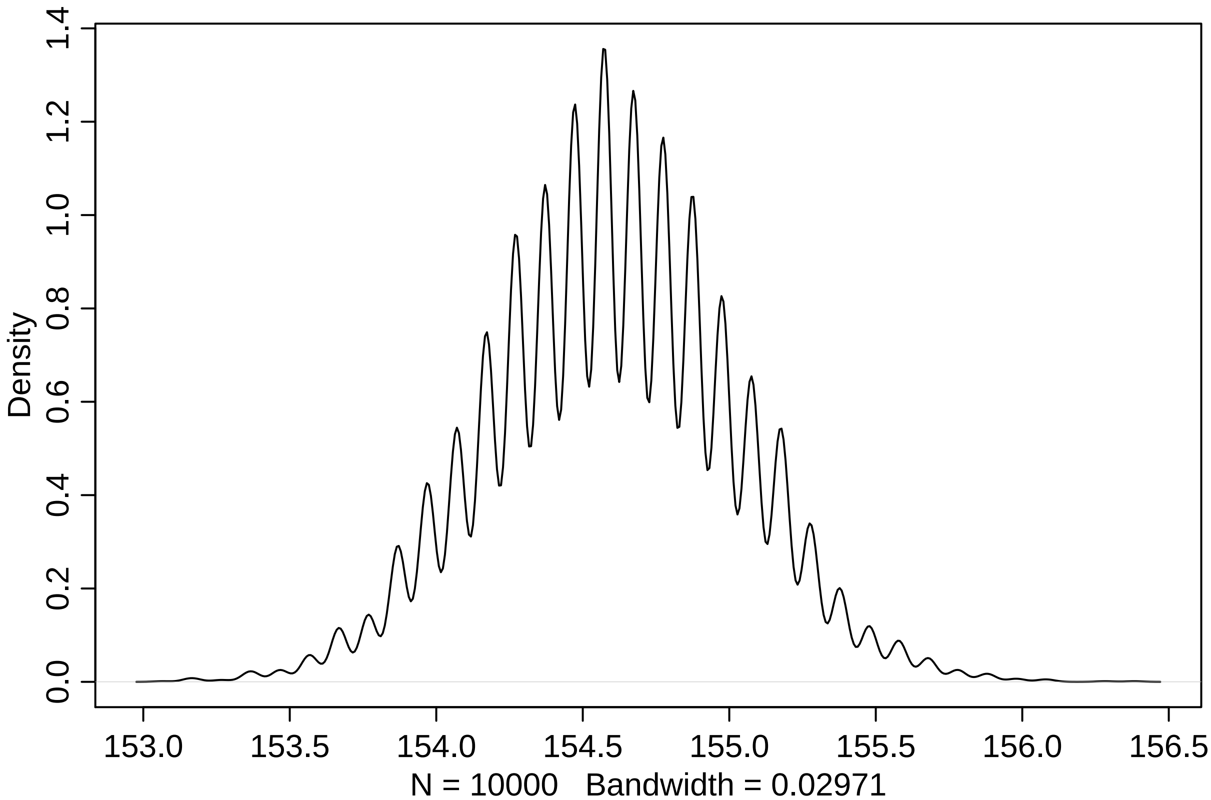
4.3.4 Sampling from the posterior

```
# 4.17
sample.rows <- sample(1:nrow(post), size=1e4, replace = T, prob = post_prob)
sample.mu <- post$mu[sample.rows]
sample.sigma <- post$sigma[sample.rows]
```

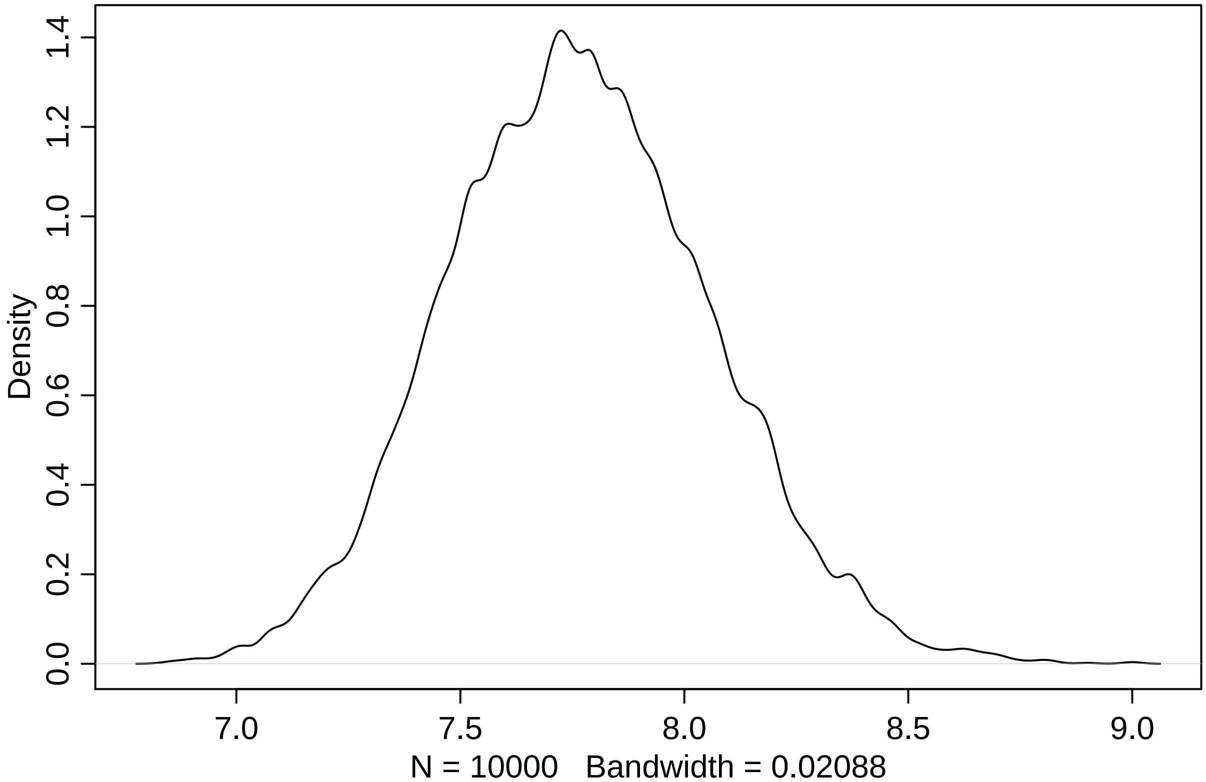
```
# 4.18
smoothScatter(sample.mu, sample.sigma, cex=0.5, pch=16, col=col.alpha(rangi2, 0.1))
```




```r
# 4.19
dens(sample.mu)
```




```
dens(sample.sigma)
```




```
# 4.20
HPDI(sample.mu)
```

```
##    |0.89    0.89|
## 153.8693 155.1759
```

```
HPDI(sample.sigma)
```

```
##    |0.89    0.89|
## 7.291457 8.195980
```

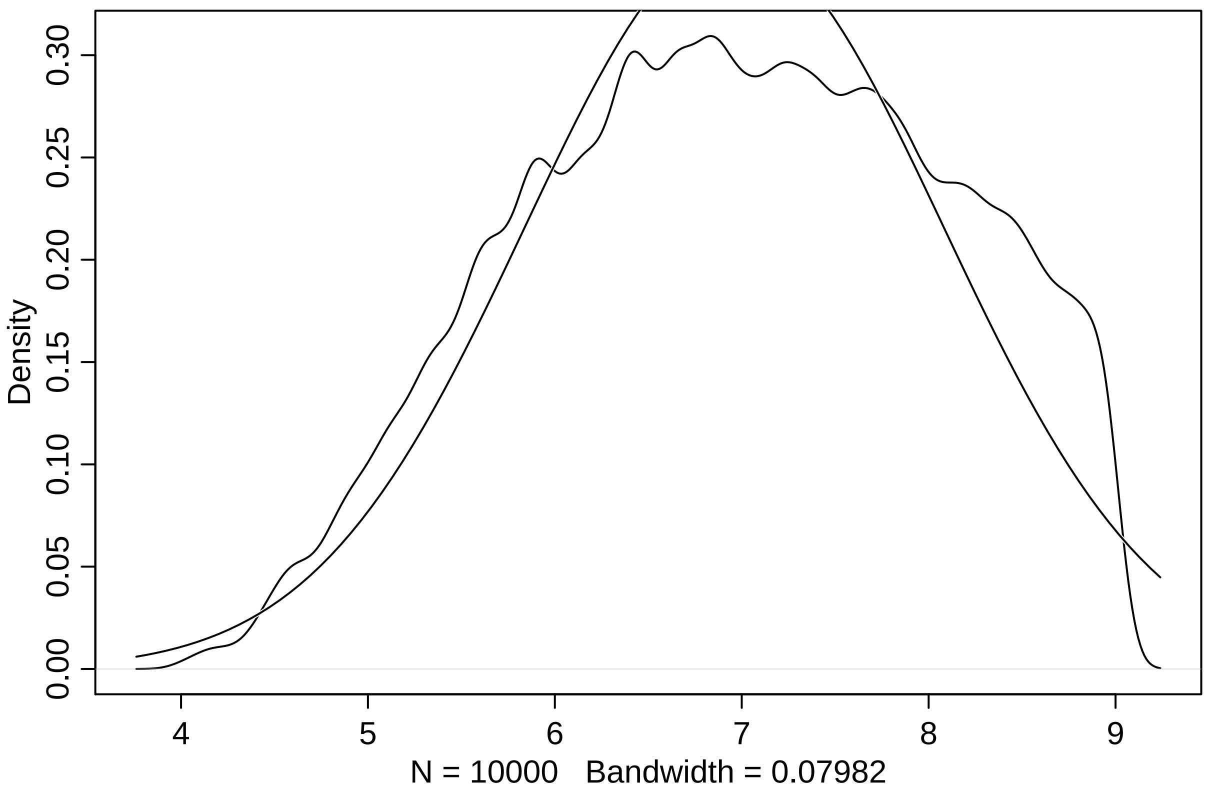
Smaller Sample

To illustrate the posterior is not always Guassian in shape.

```
# 4.22
d3 <- sample(d2$height, size=10)

small.post_ll <- sapply(1:nrow(post), function(i) sum(dnorm(d3, mean=post$mu[i], sd=post$sigma[i], log=

small.post_product <- small.post_ll + dnorm(post$mu, 178, 20, T) + dunif(post$sigma, 0, 50, T)

small.post_proba <- exp(small.post_product - max(small.post_product))

small.sample.rows <- sample(1:nrow(post), size=1e4, replace = T, prob=small.post_proba)

small.sample.mu <- post$mu[small.sample.rows]
```

```r
small.sample.sigma <- post$sigma[small.sample.rows]
```

```r
# 4.23
dens(small.sample.sigma, norm.comp = T)
```



4.3.5. Fitting the model with *map*

map finds the values of μ and σ that maximize the posterior probability.

```r
# 4.25
model.list <- alist(
  height ~ dnorm(mu, sigma),
  mu ~ dnorm(178, 20),
  sigma ~ dunif(0, 50)
)
```

```r
# 4.26
model.solved <- map(model.list, data=d2)
```

```r
# 4.27
precis(model.solved)
```

```
##          Mean StdDev   5.5%  94.5%
## mu     154.61   0.41 153.95 155.27
## sigma    7.73   0.29   7.27   8.20
```

Compare to HPDI intervals from above.

```
HPDI(sample.mu)
```

```
##    |0.89    0.89|
## 153.8693 155.1759
```

```
HPDI(sample.sigma)
```

```
##    |0.89    0.89|
## 7.291457 8.195980
```

We've calculated the HPDI intervals using the grid approximation. The model is solved via a quadratic approximation. The quadratic approximation does a very good in identifying the 89% intervals.

It works because the posterior is approximately Gaussian.

The priors we used so far are very weak. We'll splice in a more informative prior for μ.

```
#4.29
model.solved_narrow_mu <- map (
  alist(
    height ~ dnorm(mu, sigma),
    mu ~ dnorm(178, 0.1),
    sigma ~ dunif(0, 50)
  ),
  data=d2)
precis(model.solved_narrow_mu)
```

```
##         Mean StdDev   5.5%  94.5%
## mu    177.86   0.10 177.70 178.02
## sigma  24.52   0.93  23.03  26.00
```

The estimate for μ has hardly moved off the prior. The estimate for σ has changed a lot, even though we didn't change the prior at all. Our machine had to make μ and σ fit out data. Since μ is very concerntrated around 178, the machine had to change σ to accomodate the data.

4.3.6. Sampling from a *map* fit.

Variance-covariance matrix:

```
# 4.30
vcov(model.solved)
```

```
##                  mu        sigma
## mu    0.1697374534 0.0002175861
## sigma 0.0002175861 0.0849031274
```

We can split it into (1) vector of variances, and (2) the correlation matrix:

```
# 4.31
diag(vcov(model.solved))
```

```
##         mu      sigma
## 0.16973745 0.08490313
```

```
cov2cor(vcov(model.solved))
```

```
##               mu      sigma
## mu    1.00000000 0.00181251
## sigma 0.00181251 1.00000000
```

Sampling from the posterior:

```r
# 4.34
coef(model.solved)
```

```
##          mu      sigma
## 154.607004   7.731284
```

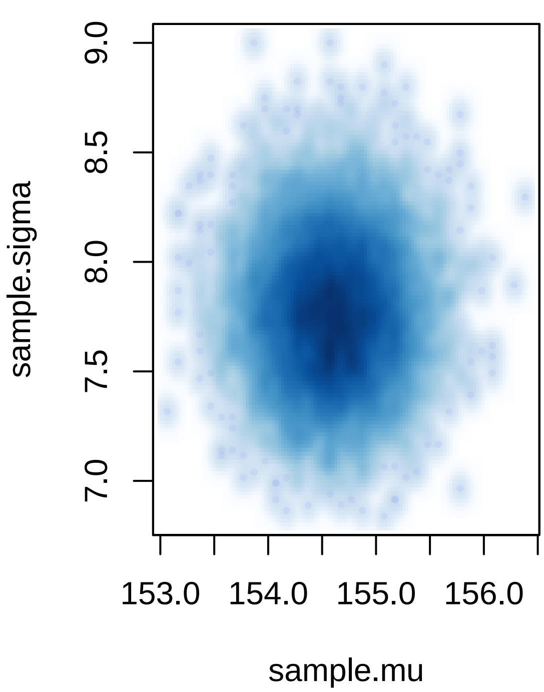
```r
library(MASS)
post <- mvrnorm(n=1e4, mu=coef(model.solved), Sigma=vcov(model.solved))
post = data.frame(post)
head(post)
```

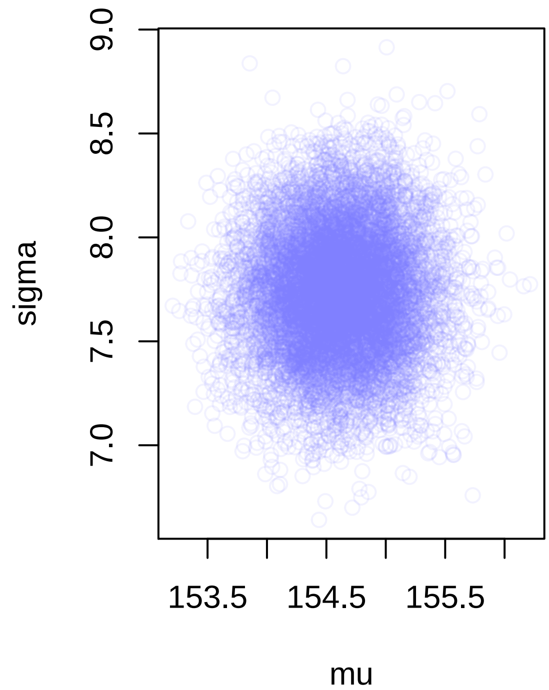
```
##         mu    sigma
## 1 154.2442 7.604488
## 2 155.0787 7.913638
## 3 154.5483 7.974797
## 4 154.8471 7.821662
## 5 153.6019 7.651472
## 6 154.7389 7.679386
```

```r
# 4.33
precis(post)
```

```
##         Mean StdDev  |0.89  0.89|
## mu    154.61   0.41 153.95 155.25
## sigma   7.73   0.29   7.28   8.22
```

```r
par(mfrow=c(1, 2))
smoothScatter(sample.mu, sample.sigma, cex=0.5, pch=16, col=col.alpha(rangi2, 0.1))
plot(post, col=col.alpha(rangi2, 0.1))
```



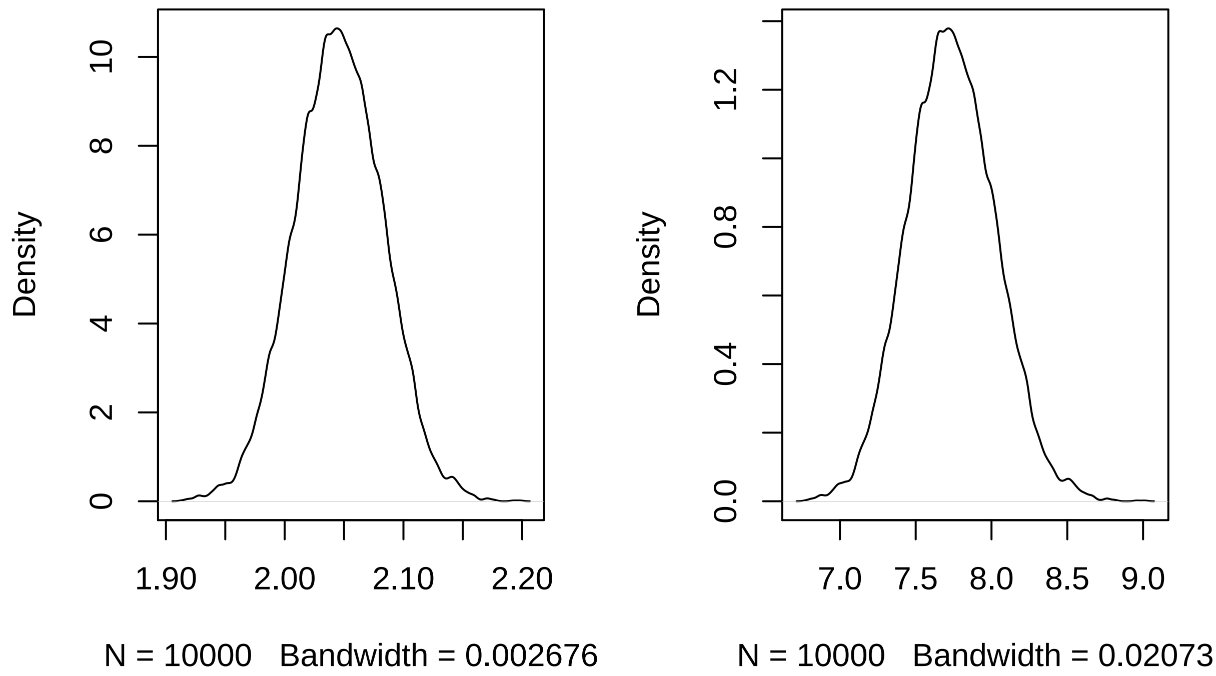


Getting sigma right

The quadratic assumption for σ may be not correct. In this case it's better to estimate $\log(\sigma)$ instead, because the distriubtion of *log* will be much closer to Guassian.

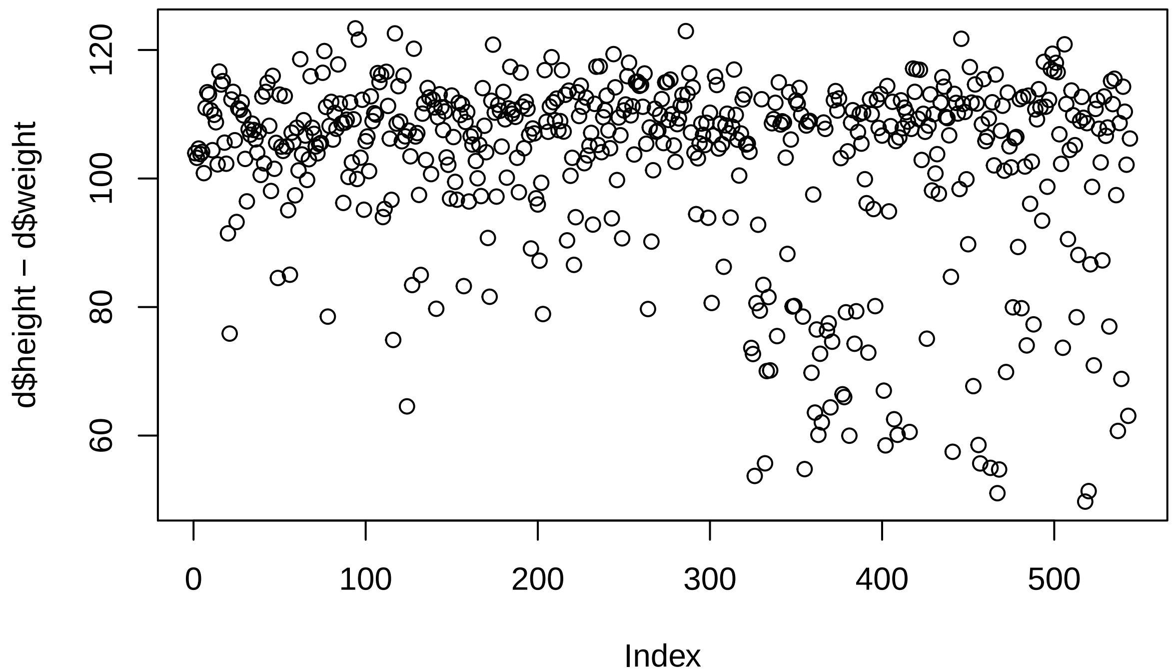
```
# 4.35
model.solved_log_sigma <- map(
  alist(
    height ~ dnorm(mu, exp(log_sigma)),
    mu ~ dnorm(178, 20),
    log_sigma ~ dnorm(2, 10)
  ),
  data = d2
)
```

```
# 4.36
post <- mvrnorm(n=1e4, mu=coef(model.solved_log_sigma), Sigma=vcov(model.solved_log_sigma))
post <- data.frame(post)
par(mfrow=c(1, 2))
dens(post$log_sigma)
dens(exp(post$log_sigma))
```



4.4. Adding a predictor

```
#4.37
plot(d$height ~ d$weight)
```



4.4.2. Fitting the model

```
# 4.38
model.linear_m43 <- map(
  alist(
    height ~ dnorm(mu, sigma),
    mu <- a + b * weight,
    a ~ dnorm(178, 100),
    b ~ dnorm (0, 10),
    sigma ~ dunif(0, 50)
  ),
  data=d2
)
```

```
# 4.40
precis(model.linear_m43, corr=T)
```

```
##           Mean StdDev   5.5%  94.5%     a     b sigma
## a       113.90   1.91 110.86 116.95  1.00 -0.99     0
## b         0.90   0.04   0.84   0.97 -0.99  1.00     0
## sigma     5.07   0.19   4.77   5.38  0.00  0.00     1
```

Centering

```r
# 4.42
d2$weight_centered <- d2$weight - mean(d2$weight)
```

```r
# 4.43
model.linear_m44 <- map(
  alist(
    height ~ dnorm(a + b * weight_centered, sigma),
    a ~ dnorm(178, 100),
    b ~ dnorm(0, 10),
    sigma ~ dunif(0, 50)
  )
  , data=d2
)
```

```r
# 4.44
precis(model.linear_m44, corr=T)
```

```
##         Mean StdDev   5.5%  94.5% a b sigma
## a     154.60   0.27 154.17 155.03 1 0     0
## b       0.91   0.04   0.84   0.97 0 1     0
## sigma   5.07   0.19   4.77   5.38 0 0     1
```

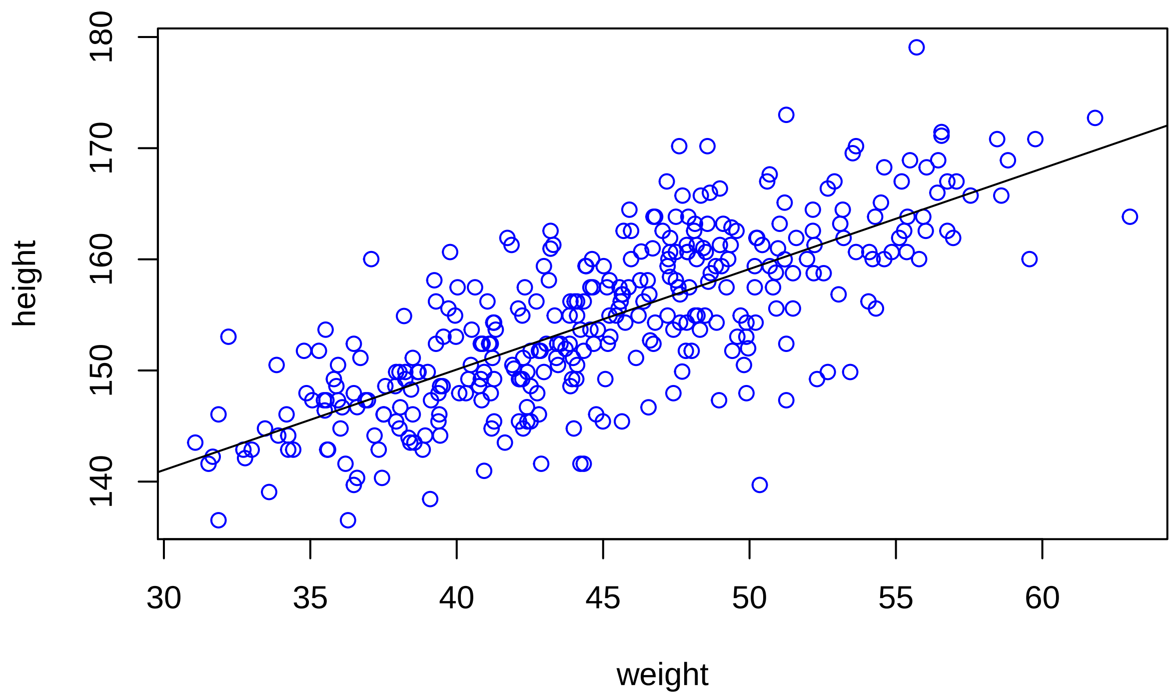
The new estimate for α is now the same as mean:

```r
mean(d2$height)
```

```
## [1] 154.5971
```

Let's plot the posterior against the data:

```r
# 4.45
plot(height ~ weight, data=d2, col="blue")
abline(a=coef(model.linear_m43)["a"], b = coef(model.linear_m43)["b"])
```



This line is just the posterior mean, the most plausible line. There are infinite regression lines from the posterior.

Let's extract some examples from the model:

```
# 4.46
post <- mvrnorm(n=1e4, mu=coef(model.linear_m43), Sigma=vcov(model.linear_m43))
post <- data.frame(post)
post[1:6, ]
```

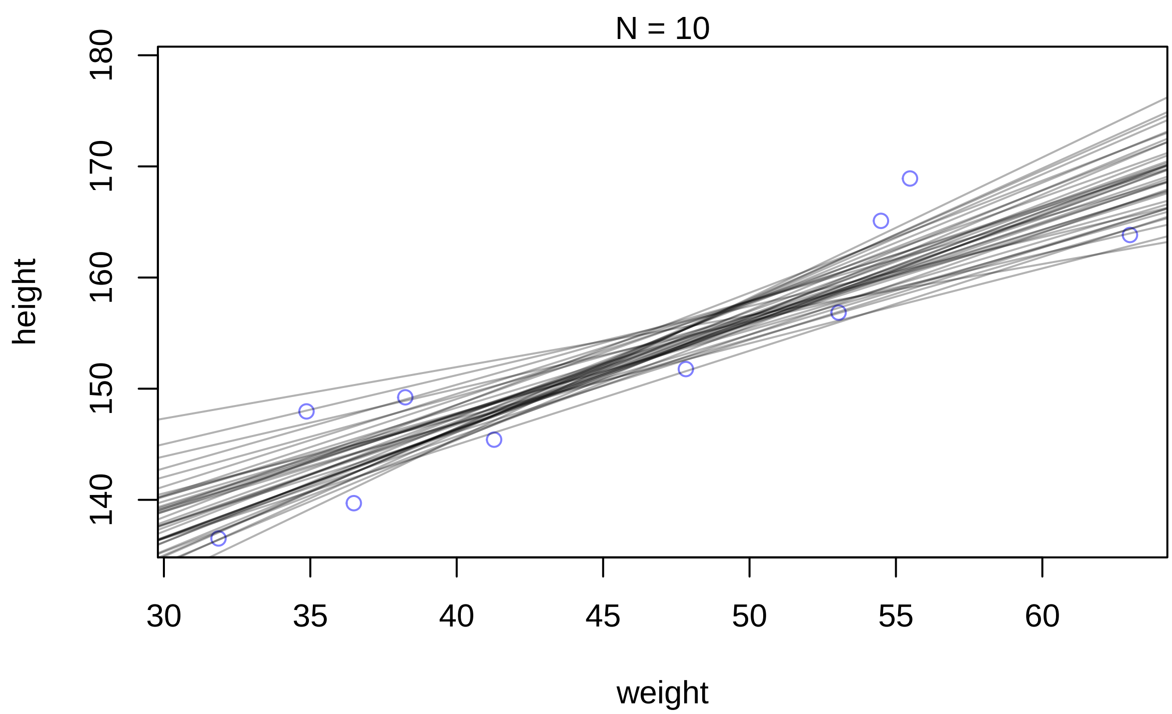
```
##          a         b    sigma
## 1 115.2694 0.8703205 5.046725
## 2 111.4837 0.9569139 4.929714
## 3 116.5549 0.8383781 5.187861
## 4 113.0038 0.9260824 5.299574
## 5 115.1940 0.8760237 4.846943
## 6 113.8806 0.9052025 4.521383
```

Let's try on the small data set first to see how the regression lines vary:

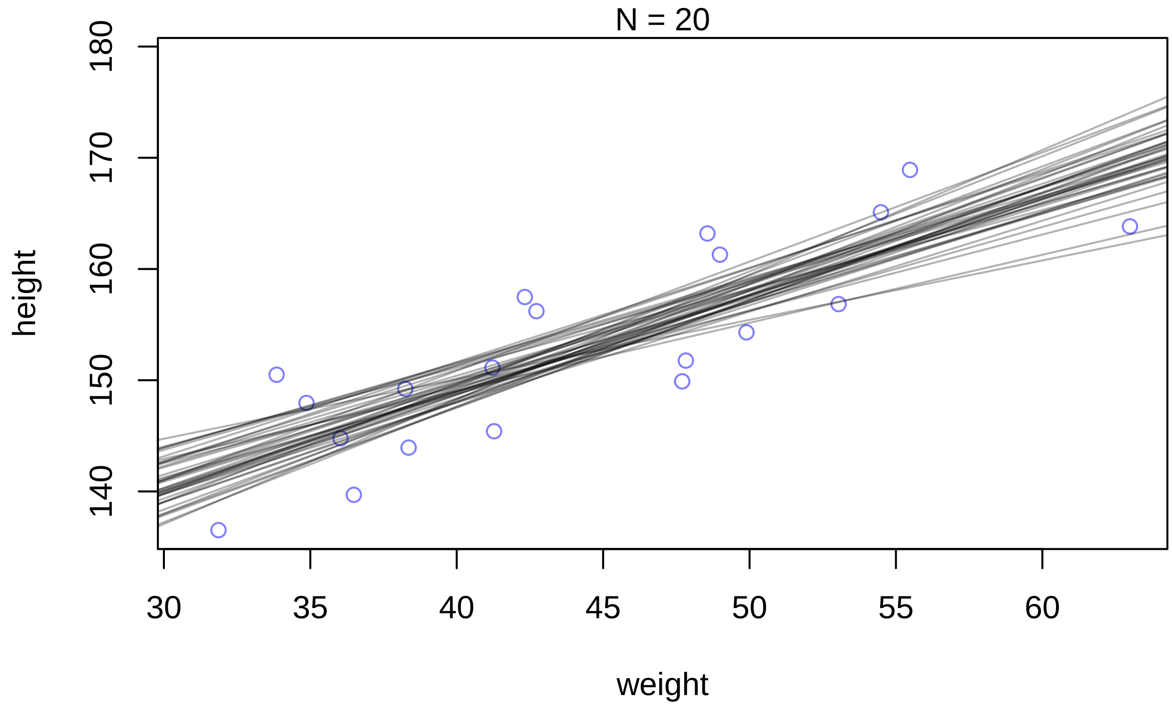
```
# 4.48

ablines_N = function (N_) {
  library(rethinking)
  data(Howell1)
  d <- Howell1
  d2 <- d[d$age >= 18, ]

  dN <- d2[1:N_,]
```

```
  mN <- map(
    alist(
      height ~ dnorm(a + b * weight, sigma),
      a ~ dnorm(178, 100),
      b ~ dnorm(0, 10),
      sigma ~ dunif(0, 50)
    ),
    data = dN
  )
  post <- mvrnorm(n=40, mu=coef(mN), Sigma=vcov(mN))
  post <- data.frame(post)

  plot(dN$weight, dN$height, xlim=range(d2$weight), ylim=range(d2$height),
       col=rangi2, xlab="weight", ylab="height")
  mtext(concat("N = ", N_))

  for (i in 1:nrow(post))
    abline(a=post$a[i], b=post$b[i], col=col.alpha("black", 0.3))
}
```

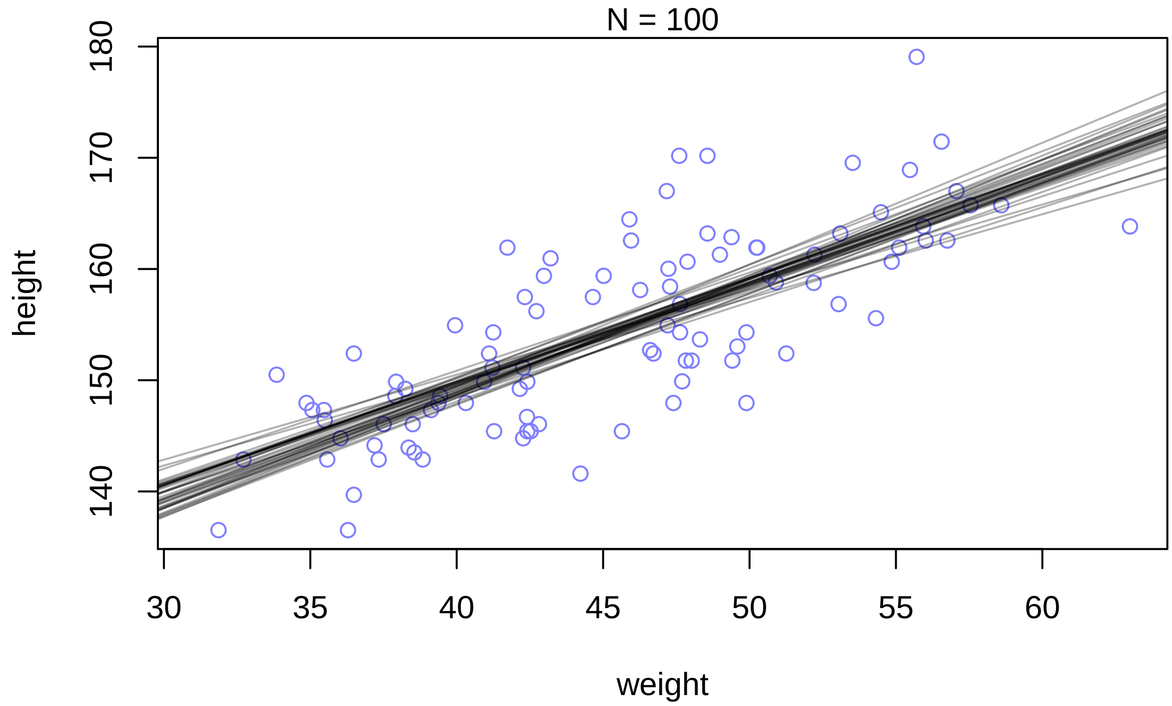
```
ablines_N(10)
```



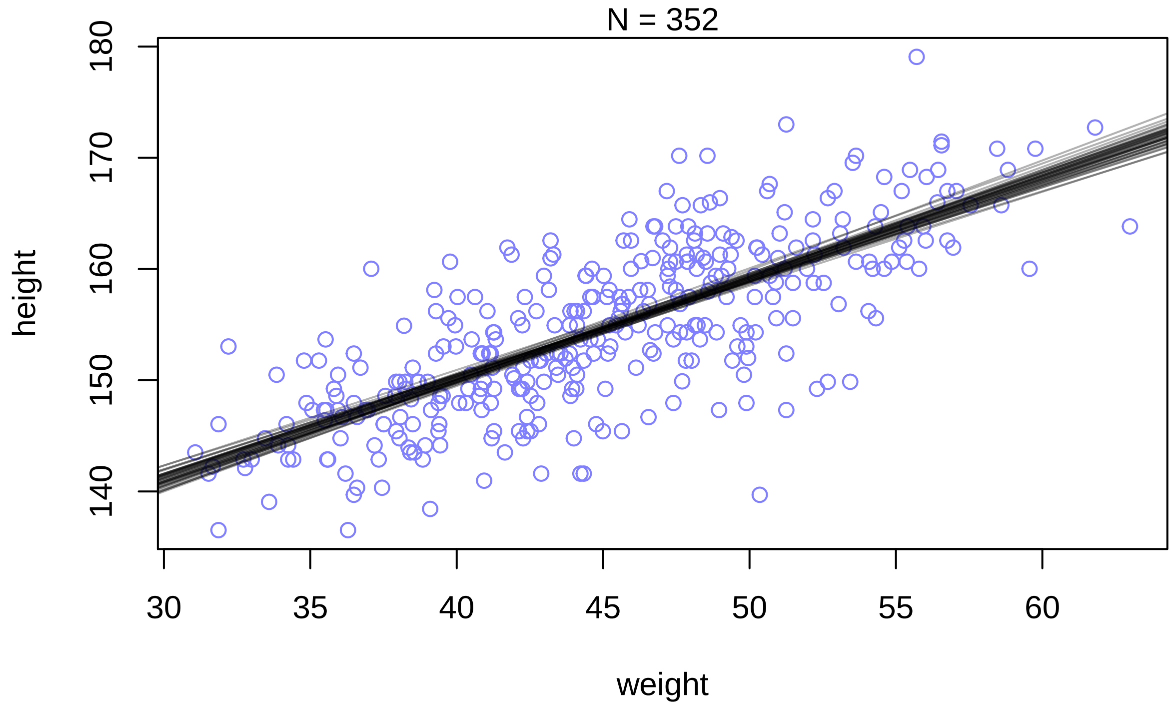

```
ablines_N(20)
```




```
ablines_N(100)
```

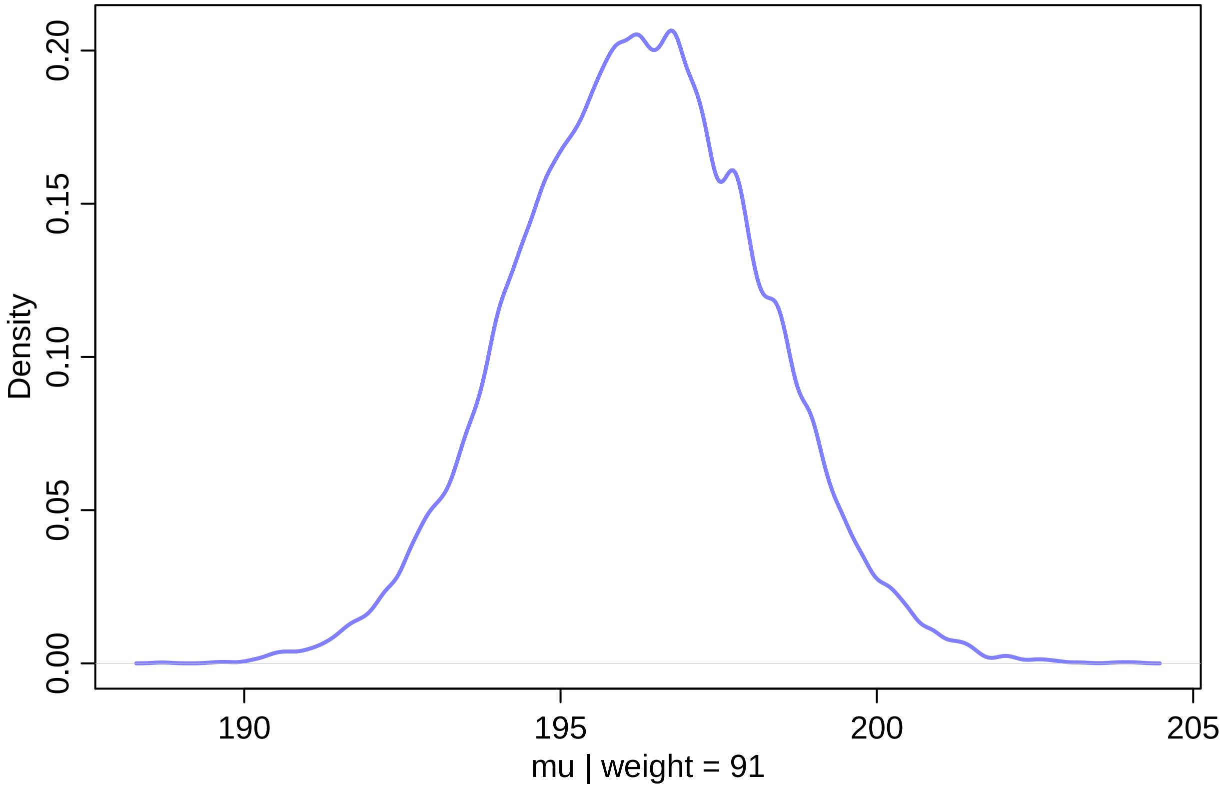



```
ablines_N(352)
```



Let's predict value for individual who weighs 91 kg:

```
# 4.50
mu_at_50 <- post$a + post$b * 91
dens(mu_at_50, col=rangi2, lwd=2, xlab = "mu | weight = 91")
```




```
# 4.52
HPDI(mu_at_50, prob=0.89)
```

```
##     |0.89    0.89|
## 193.2407 199.3245
```

```
# 4.53
mu <- link(model.linear_m43)
```

```
## [ 100 / 1000 ]
[ 200 / 1000 ]
[ 300 / 1000 ]
[ 400 / 1000 ]
[ 500 / 1000 ]
[ 600 / 1000 ]
[ 700 / 1000 ]
[ 800 / 1000 ]
[ 900 / 1000 ]
[ 1000 / 1000 ]
```

```
str(mu)
```

```
##  num [1:1000, 1:352] 158 157 157 157 157 ...
```

Compute the distribution for each weight:

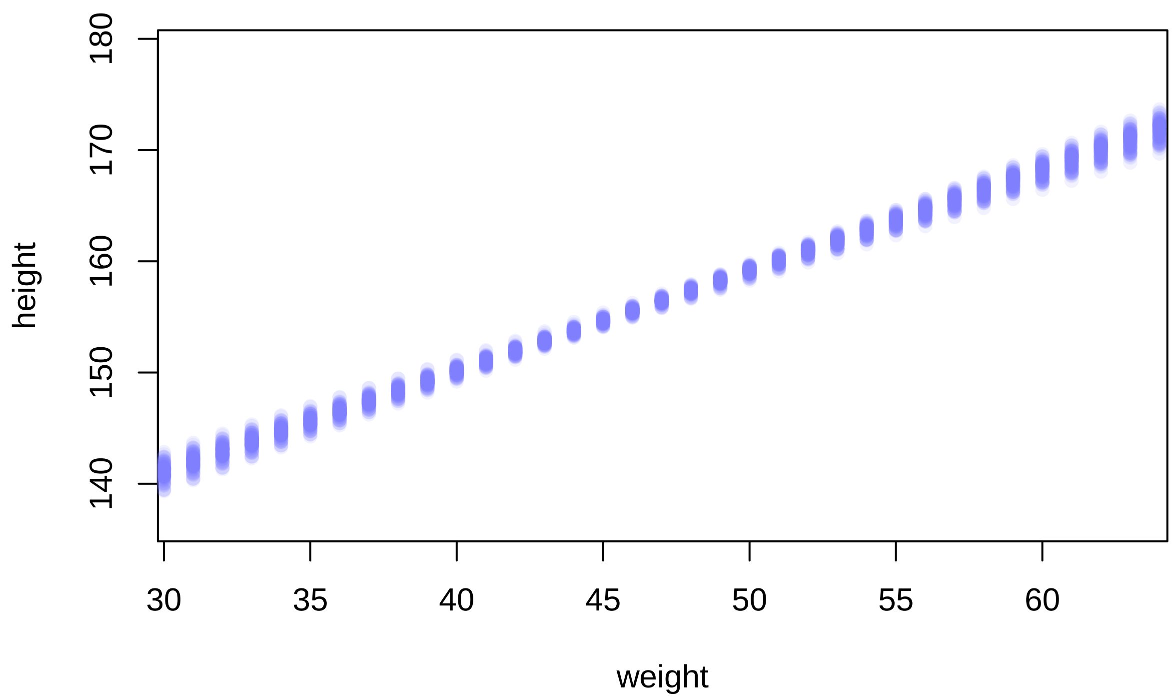
```
# 4.54
weight_seq <- seq(from=25, to=100, by=1)
mu <- link(model.linear_m43, data=data.frame(weight=weight_seq))
```

```
## [ 100 / 1000 ]
[ 200 / 1000 ]
[ 300 / 1000 ]
[ 400 / 1000 ]
[ 500 / 1000 ]
[ 600 / 1000 ]
[ 700 / 1000 ]
[ 800 / 1000 ]
[ 900 / 1000 ]
[ 1000 / 1000 ]
```

```r
str(mu)
```

```
##  num [1:1000, 1:76] 139 136 136 137 137 ...
```

```r
# 4.55
plot(height ~ weight, d2, type="n")
for (i in 1:100)
  points(weight_seq, mu[i,], pch=16, col=col.alpha(rangi2, 0.1))
```




```r
# 4.56
mu.mean <- apply(mu, 2, mean)
mu.hpdi <- apply(mu, 2, HPDI, prob=0.89)
mu.mean
```

```
##  [1] 136.4935 137.3995 138.3056 139.2116 140.1177 141.0237 141.9297
##  [8] 142.8358 143.7418 144.6479 145.5539 146.4600 147.3660 148.2720
## [15] 149.1781 150.0841 150.9902 151.8962 152.8023 153.7083 154.6143
```

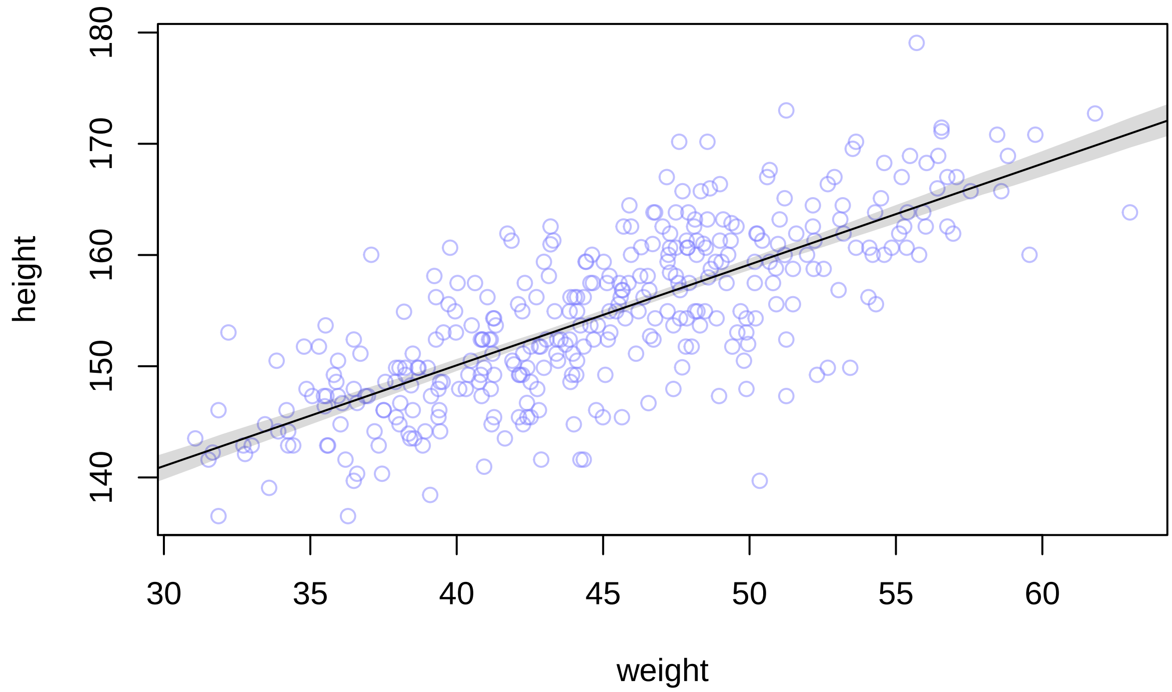
```
## [22] 155.5204 156.4264 157.3325 158.2385 159.1446 160.0506 160.9566
## [29] 161.8627 162.7687 163.6748 164.5808 165.4869 166.3929 167.2989
## [36] 168.2050 169.1110 170.0171 170.9231 171.8292 172.7352 173.6412
## [43] 174.5473 175.4533 176.3594 177.2654 178.1715 179.0775 179.9835
## [50] 180.8896 181.7956 182.7017 183.6077 184.5138 185.4198 186.3259
## [57] 187.2319 188.1379 189.0440 189.9500 190.8561 191.7621 192.6682
## [64] 193.5742 194.4802 195.3863 196.2923 197.1984 198.1044 199.0105
## [71] 199.9165 200.8225 201.7286 202.6346 203.5407 204.4467
```

```
mu.hpdi
```

```
##             [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## |0.89 134.9920 135.8288 136.8081 137.7800 138.8990 139.8334 140.8062
## 0.89| 138.0119 138.7152 139.5514 140.3814 141.3657 142.1631 142.9899
##             [,8]     [,9]    [,10]    [,11]    [,12]    [,13]    [,14]
## |0.89 141.8498 142.8171 143.7804 144.7537 145.7088 146.7071 147.6378
## 0.89| 143.9036 144.7396 145.5715 146.3994 147.2319 148.1069 148.9090
##            [,15]    [,16]    [,17]    [,18]    [,19]    [,20]    [,21]
## |0.89 148.5777 149.5714 150.4990 151.4655 152.3683 153.3032 154.2458
## 0.89| 149.7519 150.6592 151.4826 152.3709 153.2133 154.1261 155.0638
##            [,22]    [,23]    [,24]    [,25]   [,26]    [,27]    [,28]
## |0.89 155.1278 156.0229 156.8721 157.7779 158.643 159.5013 160.2709
## 0.89| 155.9435 156.8742 157.7754 158.7525 159.691 160.6641 161.5469
##            [,29]    [,30]    [,31]    [,32]    [,33]    [,34]    [,35]
## |0.89 161.1369 161.9782 162.8389 163.6827 164.5925 165.4602 166.2232
## 0.89| 162.5233 163.5075 164.4902 165.4543 166.4877 167.4653 168.3659
##            [,36]    [,37]    [,38]    [,39]    [,40]    [,41]    [,42]
## |0.89 167.0710 167.9273 168.7748 169.6618 170.4665 171.2979 172.2074
## 0.89| 169.3427 170.3329 171.3174 172.3347 173.2888 174.2645 175.3130
##            [,43]    [,44]    [,45]    [,46]    [,47]    [,48]    [,49]
## |0.89 173.0535 173.8690 174.7079 175.5351 176.3759 177.2142 178.0494
## 0.89| 176.2984 177.2495 178.2229 179.2045 180.1868 181.1690 182.1637
##            [,50]    [,51]    [,52]    [,53]    [,54]    [,55]    [,56]
## |0.89 178.4983 179.8062 180.6549 181.4911 182.3147 183.1492 183.9995
## 0.89| 182.7583 184.1957 185.1859 186.1699 187.1408 188.1241 189.1255
##            [,57]    [,58]    [,59]    [,60]    [,61]    [,62]    [,63]
## |0.89 184.3822 185.1786 186.0273 186.8546 187.6681 188.4942 189.3218
## 0.89| 189.6439 190.5791 191.5690 192.5431 193.4950 194.4567 195.4192
##            [,64]    [,65]    [,66]    [,67]    [,68]    [,69]    [,70]
## |0.89 190.1474 190.9706 191.7938 192.6038 193.4336 194.2459 195.0797
## 0.89| 196.3897 197.3629 198.3308 199.2968 200.2638 201.2091 202.1755
##            [,71]    [,72]    [,73]    [,74]    [,75]    [,76]
## |0.89 195.9098 196.7330 197.5553 198.3771 199.1989 200.0206
## 0.89| 203.1418 204.1082 205.0764 206.0469 207.0173 207.9878
```

Plot raw data, fading out points to make line and interval more visible:

```
# 4.57
plot(height ~ weight, data=d2, col=col.alpha(rangi2, 0.5))

# Plot the MAP line, i.e. the mean mu for each weight
lines(weight_seq, mu.mean)

# Plot a shaded region for 89% HPDI
shade(mu.hpdi, weight_seq)
```




```r
# 4.58
post <- extract.samples(model.linear_m43)
head(post)
```

```
##          a         b    sigma
## 1 117.6337 0.8271641 5.221055
## 2 113.3196 0.9247954 5.032240
## 3 112.1253 0.9392986 4.953837
## 4 115.3479 0.8726435 4.916701
## 5 116.8700 0.8492300 5.232097
## 6 114.2362 0.8898995 4.894965
```

```r
model.linear_m43
```

```
##
## Maximum a posteriori (MAP) model fit
##
## Formula:
## height ~ dnorm(mu, sigma)
## mu <- a + b * weight
## a ~ dnorm(178, 100)
## b ~ dnorm(0, 10)
## sigma ~ dunif(0, 50)
##
## MAP values:
##           a           b       sigma
## 113.9035411   0.9045029   5.0718696
```

```
##
## Log-likelihood: -1071.01
```

```r
mu.link <- function(weight) post$a + post$b * weight

weight.seq <- seq(from=27, to=70, by=1)

mu <- sapply(weight.seq, mu.link)
head(mu)
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## [1,] 139.9671 140.7943 141.6214 142.4486 143.2758 144.1029 144.9301
## [2,] 138.2891 139.2139 140.1387 141.0635 141.9883 142.9131 143.8379
## [3,] 137.4864 138.4257 139.3650 140.3043 141.2436 142.1829 143.1222
## [4,] 138.9093 139.7819 140.6545 141.5272 142.3998 143.2725 144.1451
## [5,] 139.7992 140.6484 141.4976 142.3469 143.1961 144.0453 144.8945
## [6,] 138.2635 139.1534 140.0433 140.9332 141.8231 142.7130 143.6029
##          [,8]     [,9]    [,10]    [,11]    [,12]    [,13]    [,14]
## [1,] 145.7573 146.5844 147.4116 148.2387 149.0659 149.8931 150.7202
## [2,] 144.7627 145.6875 146.6123 147.5370 148.4618 149.3866 150.3114
## [3,] 144.0615 145.0008 145.9401 146.8794 147.8187 148.7580 149.6973
## [4,] 145.0178 145.8904 146.7630 147.6357 148.5083 149.3810 150.2536
## [5,] 145.7438 146.5930 147.4422 148.2915 149.1407 149.9899 150.8392
## [6,] 144.4928 145.3827 146.2726 147.1625 148.0524 148.9423 149.8322
##         [,15]    [,16]    [,17]    [,18]    [,19]    [,20]    [,21]
## [1,] 151.5474 152.3746 153.2017 154.0289 154.8561 155.6832 156.5104
## [2,] 151.2362 152.1610 153.0858 154.0106 154.9354 155.8602 156.7850
## [3,] 150.6366 151.5759 152.5152 153.4545 154.3938 155.3331 156.2724
## [4,] 151.1263 151.9989 152.8716 153.7442 154.6168 155.4895 156.3621
## [5,] 151.6884 152.5376 153.3868 154.2361 155.0853 155.9345 156.7838
## [6,] 150.7221 151.6120 152.5019 153.3918 154.2817 155.1716 156.0615
##         [,22]    [,23]    [,24]    [,25]    [,26]    [,27]    [,28]
## [1,] 157.3376 158.1647 158.9919 159.8190 160.6462 161.4734 162.3005
## [2,] 157.7098 158.6346 159.5594 160.4842 161.4090 162.3338 163.2586
## [3,] 157.2117 158.1510 159.0903 160.0296 160.9689 161.9081 162.8474
## [4,] 157.2348 158.1074 158.9801 159.8527 160.7253 161.5980 162.4706
## [5,] 157.6330 158.4822 159.3315 160.1807 161.0299 161.8791 162.7284
## [6,] 156.9514 157.8413 158.7312 159.6211 160.5110 161.4009 162.2908
##         [,29]    [,30]    [,31]    [,32]    [,33]    [,34]    [,35]
## [1,] 163.1277 163.9549 164.7820 165.6092 166.4364 167.2635 168.0907
## [2,] 164.1834 165.1082 166.0330 166.9578 167.8825 168.8073 169.7321
## [3,] 163.7867 164.7260 165.6653 166.6046 167.5439 168.4832 169.4225
## [4,] 163.3433 164.2159 165.0886 165.9612 166.8338 167.7065 168.5791
## [5,] 163.5776 164.4268 165.2761 166.1253 166.9745 167.8238 168.6730
## [6,] 163.1807 164.0706 164.9605 165.8504 166.7403 167.6302 168.5201
##         [,36]    [,37]    [,38]    [,39]    [,40]    [,41]    [,42]
## [1,] 168.9179 169.7450 170.5722 171.3993 172.2265 173.0537 173.8808
## [2,] 170.6569 171.5817 172.5065 173.4313 174.3561 175.2809 176.2057
## [3,] 170.3618 171.3011 172.2404 173.1797 174.1190 175.0583 175.9976
## [4,] 169.4518 170.3244 171.1971 172.0697 172.9424 173.8150 174.6876
## [5,] 169.5222 170.3714 171.2207 172.0699 172.9191 173.7684 174.6176
## [6,] 169.4100 170.2999 171.1898 172.0797 172.9696 173.8595 174.7494
##         [,43]    [,44]
## [1,] 174.7080 175.5352
## [2,] 177.1305 178.0553
```

```
## [3,] 176.9369 177.8762
## [4,] 175.5603 176.4329
## [5,] 175.4668 176.3161
## [6,] 175.6393 176.5292
```

```r
mu.mean <- apply(mu, 2, mean)
mu.hpdi <- apply(mu, 2, HPDI, prob=0.89)
head(mu.mean)
```

```
## [1] 138.3252 139.2300 140.1348 141.0395 141.9443 142.8490
```

```r
head(mu.hpdi)
```

```
##           [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## |0.89 137.0826 138.0445 138.9931 139.9536 140.8856 141.9058 142.8673
## 0.89| 139.6083 140.4474 141.2751 142.1177 142.9280 143.8246 144.6727
##           [,8]     [,9]    [,10]    [,11]    [,12]    [,13]    [,14]
## |0.89 143.8412 144.7926 145.7338 146.6804 147.6374 148.6097 149.5446
## 0.89| 145.5299 146.3713 147.2009 148.0491 148.9087 149.7858 150.6381
##          [,15]    [,16]    [,17]    [,18]    [,19]    [,20]    [,21]
## |0.89 150.4554 151.4075 152.3445 153.2917 154.1941 155.0987 155.9897
## 0.89| 151.4743 152.3610 153.2504 154.1642 155.0583 155.9788 156.8970
##          [,22]    [,23]    [,24]    [,25]    [,26]    [,27]    [,28]
## |0.89 156.8587 157.7462 158.5684 159.4296 160.2766 161.1488 162.0108
## 0.89| 157.8166 158.7692 159.6686 160.6104 161.5545 162.5279 163.4917
##          [,29]    [,30]    [,31]    [,32]    [,33]    [,34]    [,35]
## |0.89 162.8777 163.7331 164.5731 165.4092 166.2828 167.1237 167.9689
## 0.89| 164.4666 165.4316 166.3867 167.3420 168.3335 169.2936 170.2611
##          [,36]    [,37]    [,38]    [,39]    [,40]    [,41]    [,42]
## |0.89 168.8003 169.6411 170.4308 171.2720 172.1037 172.9485 173.7901
## 0.89| 171.2162 172.1844 173.1031 174.0674 175.0209 175.9910 176.9600
##          [,43]    [,44]
## |0.89 174.6418 175.4786
## 0.89| 177.9415 178.9015
```