Chapter 4 - Geocentric Models

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).

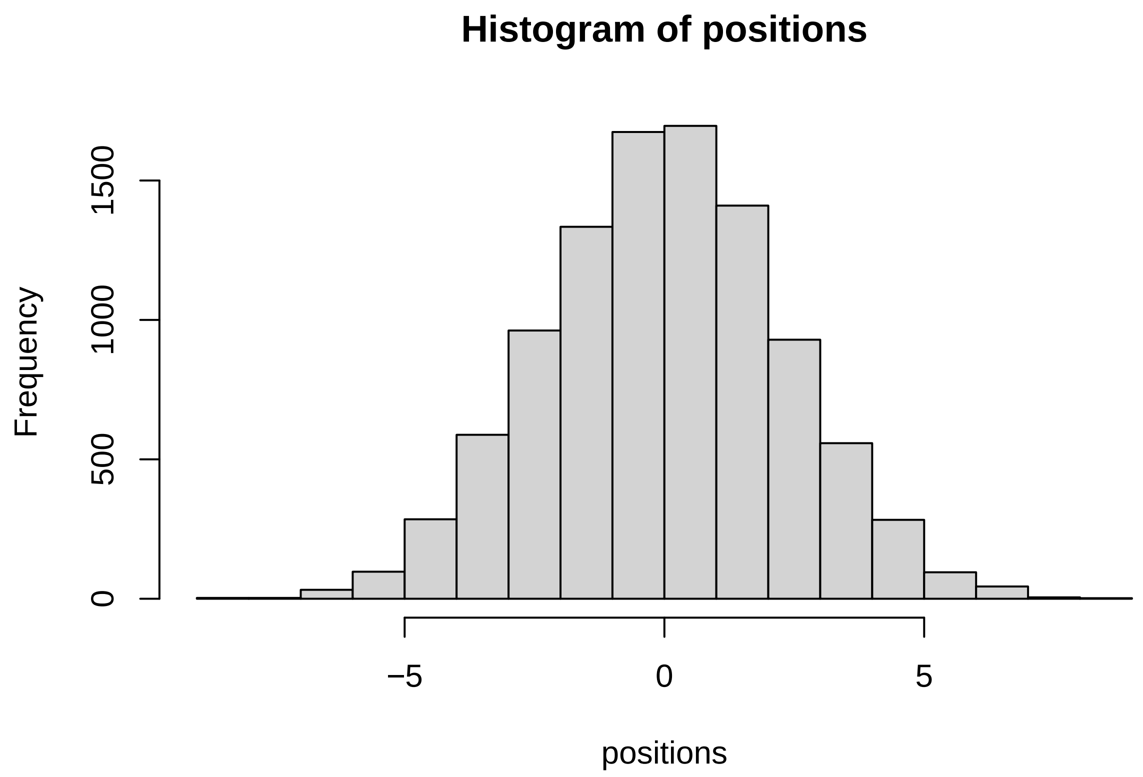
The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

Linear Models are the geocentric models for applied statistics, i.i. simple, mostly wrong but works if calibrated correctly and one knows the assumptions. It is a family of simple stats models which attempt to learn about the mean and variance of some measurement, using an additive combination of other measurements. One LM could correspond to several process models and that is fine as we don't care. LM make assumption of using gaussian distribution to quantify their uncertainity about the measurement of interest.

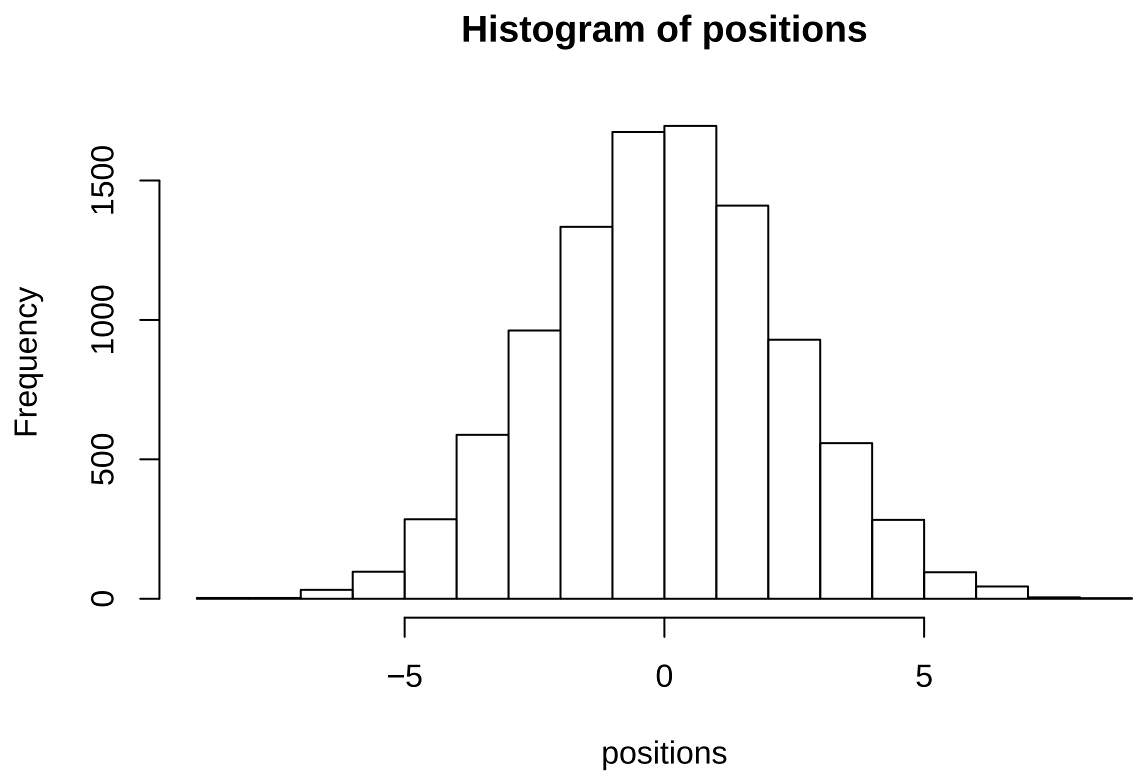
0.1 Normal Distribution

0.1.0.1 By sum Simulate 1e4 times a process where a person moves left or right with uniformly distributed steps 16 times and observe the resulting deviation

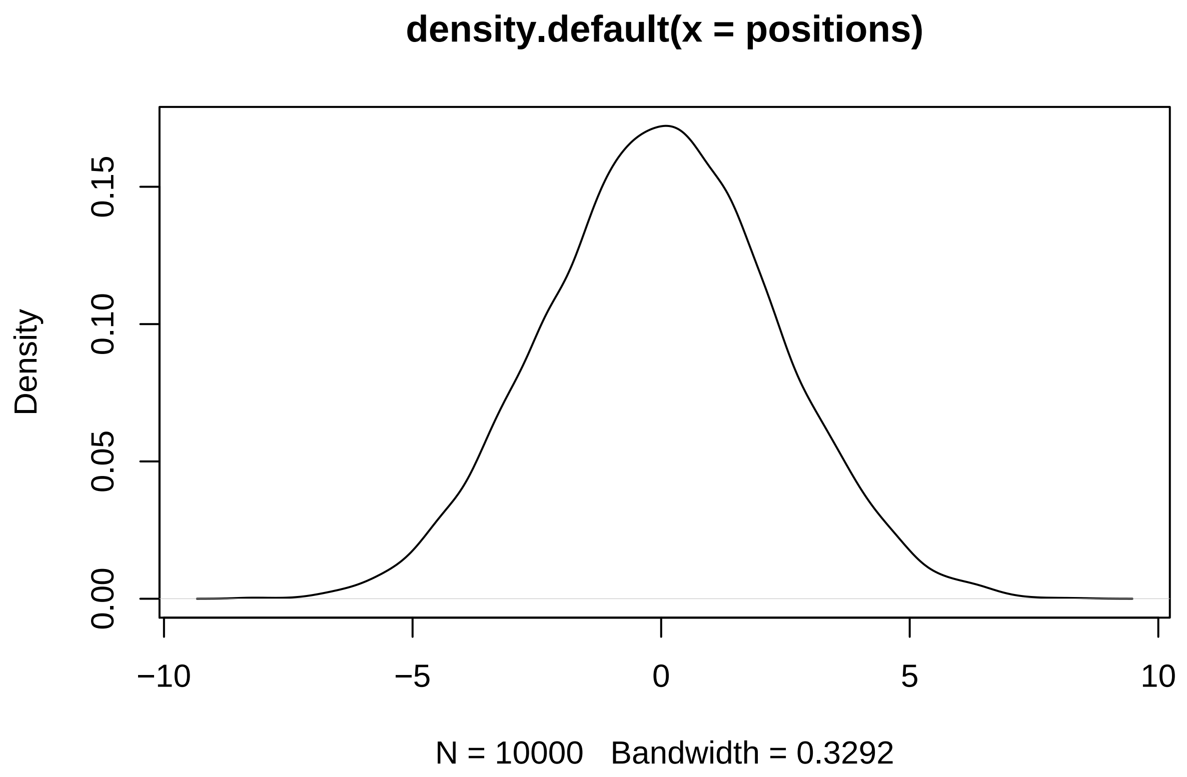
```
positions <- replicate(1e4, sum(runif(n=16, min = -1, max = 1)))
plot(hist(positions))
```



Histogram of positions
Frequency
0
500
1000
1500
−5
0
5
positions

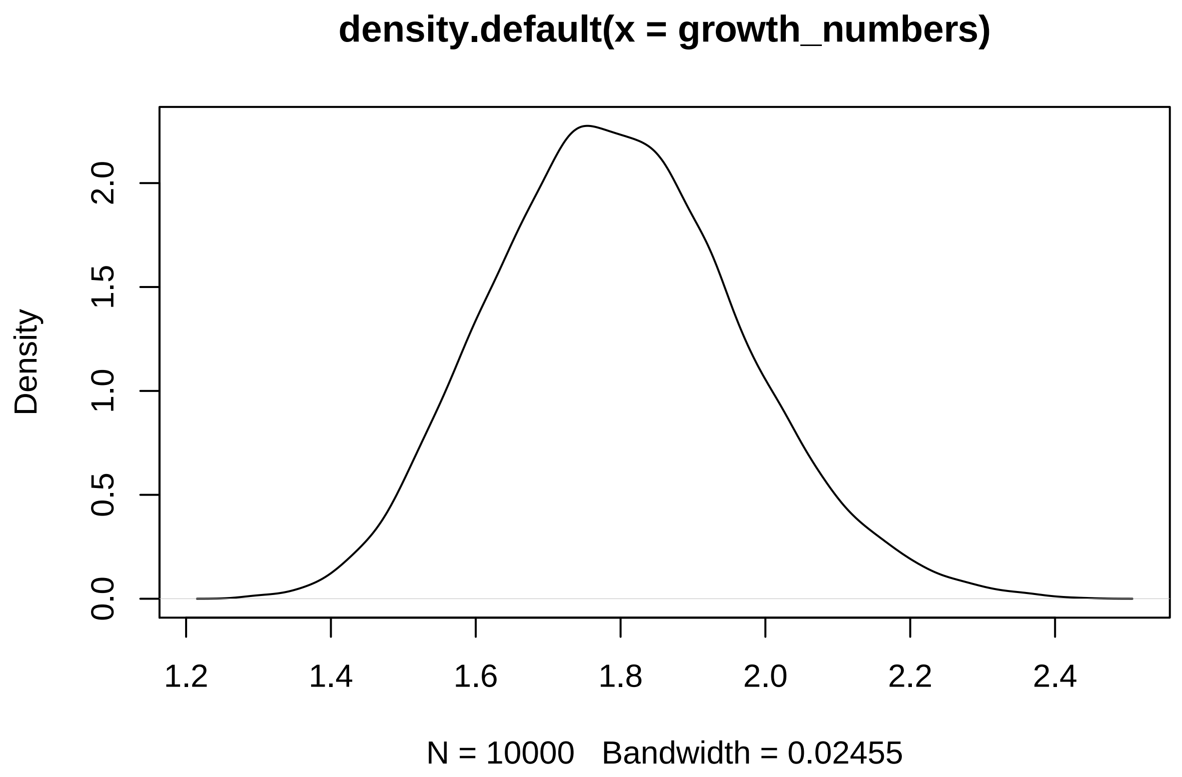


```r
plot(density(positions))
```



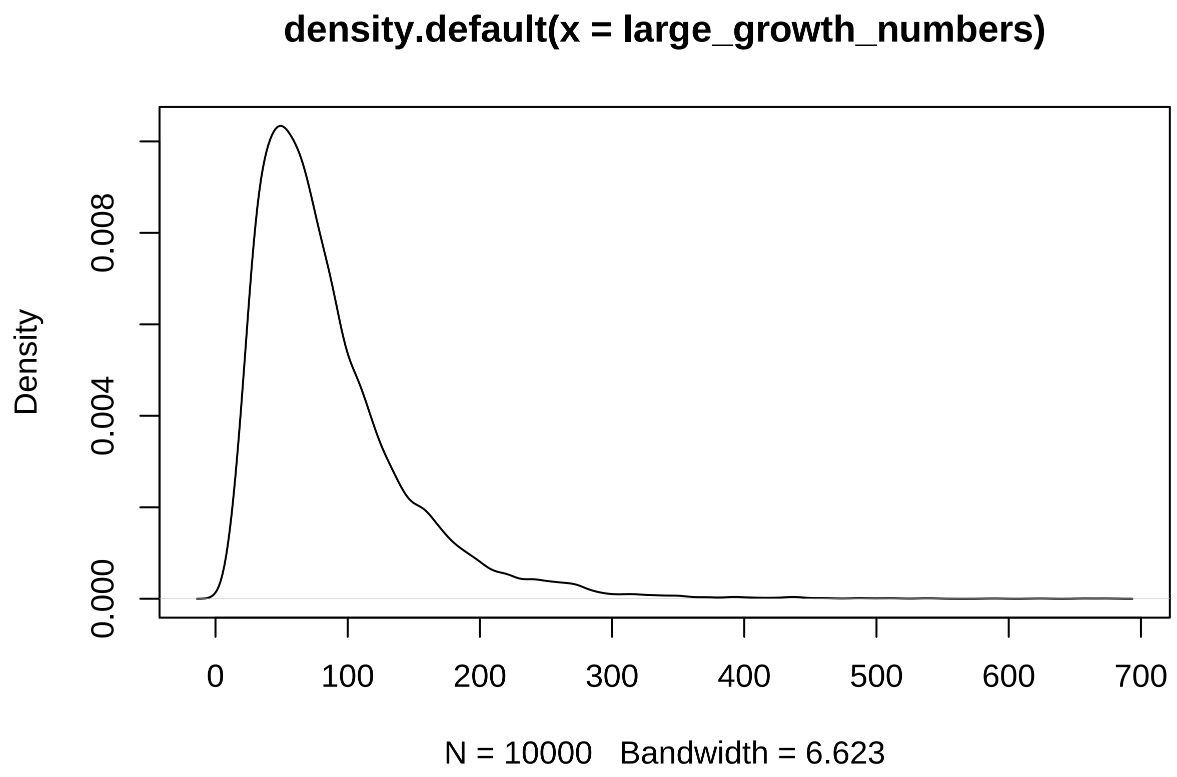
0.1.0.2 By Product Suppose an organism grows by a dozen loci such that they each contributes a percentage to final growth (by multiplication). So, if each lous has ~Unif(0, 0.1) then what is value of final growth rate?

```
growth_numbers = replicate(1e4, prod(1+runif(n=12, min=0, max=0.1)))
plot(density(growth_numbers))
```



Again, looks like a Gaussian *_*. Why's this? Coz product of 1+small_numbers is approx their sum, this won't hold with larger numbers

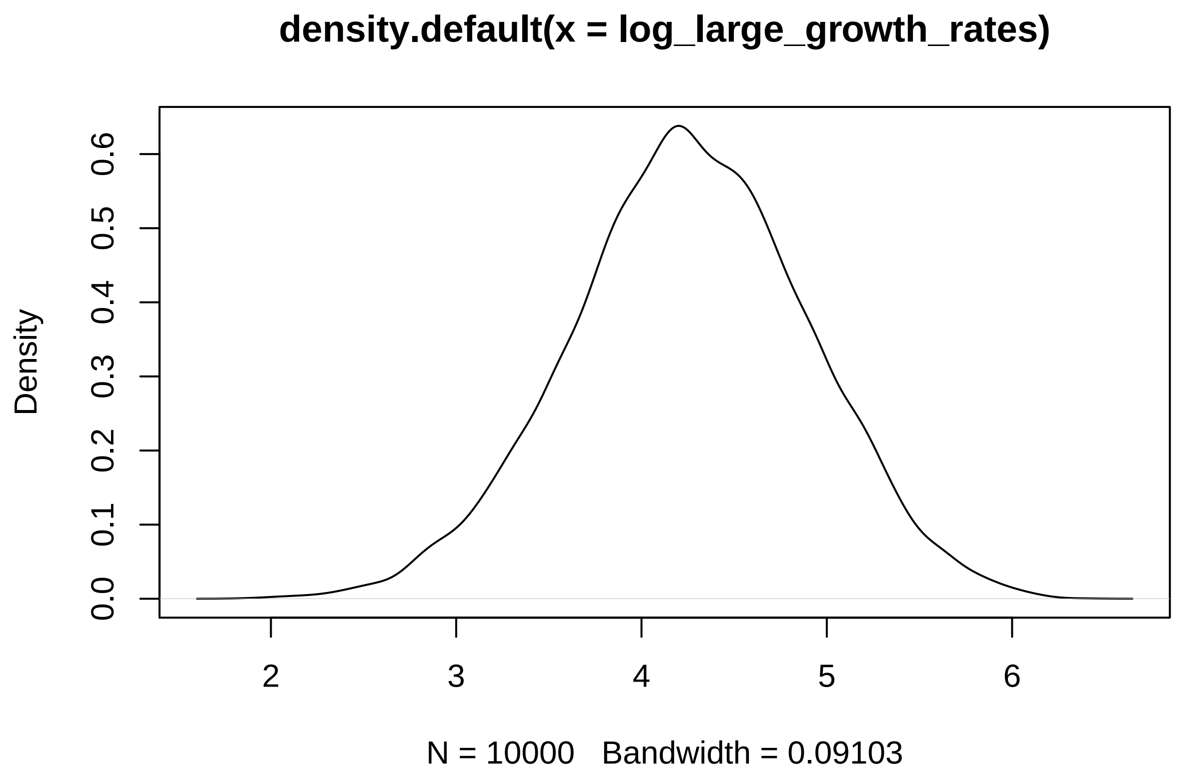
```
large_growth_numbers = replicate(1e4, prod(1+runif(n=12, min=0, max=0.9)))
plot(density(large_growth_numbers))
```



Nope, not looking like Gaussian at all.

0.1.0.3 Normal by log-multiplication However, large number multiplications on log scale? (coz, log of mults = sum of logs !)

```
log_large_growth_rates = replicate(1e4, log(prod(1+runif(12, min=0, max=0.9))))
plot(density(log_large_growth_rates))
```



And .. our Gaussians are back, note that there's nothing wrong with measuring output as log, it's just a transformation and can be,very rightly done. Many things are measure on this scale, e.g., earthquakes, information, Covd rise (!) etc.

0.1.1 Surely a few graphs don't prove anything? - Other justifications of using Normal

0.1.1.1 Ontological Justification (Gaussian is common in natural processes) Gaussians occur commonly in nature as a result of processes which involve averaging over large number of fluctuations. As this arises over large numbers, we cannot and should not expect Gaussian based models to identify micro processes. Also, Gaussians are a member of the exponential family of distributions which also includes Poissons, Gamma, Exponentials etc. All of these occur in nature, not just Gassian (e.g., Covid cases rise exponentially in initial stages)

0.1.1.2 Epistemological Justification (Sometimes, Gaussian is best we can do with our information) If all we know about a list of measures is their mean and std. then assuming Gaussian adds no further assumptions and hence, is more accurate of our understanding.

> Gaussian is the most natural expression of our state of ignorance, because if all we're willing to assume is that a measure has finite vqariance, the Gaussian distribution is the shape that can be realized in the largets number of ways and does not introduce and new assumptions.

Also, don't randomly apply Gaussians everywhere as there are thick tailed distributions which assign decent probability mass to high deviation outcomes which will completely surprise you if you use Gaussian. E.g., Stock MArket returns over a short time are Gaussian, but over medium and long term are defintiely not!

0.2 Language of Models

So how can we define models such that a few lines can tell everything about our assumptions and all our variables?

$$
\begin{aligned}
y_i &\sim Normal(\mu_i, \sigma) \\
\mu_i &= \beta x_i \\
\beta &\sim Normal(0, 10) \\
\sigma &\sim Exponential(1) \\
x_i &\sim Normal(0, 1)
\end{aligned}
$$

1. Include set of variables we wish to understand - observable (data) and unobservable (parameters)
2. Each variable is either a function of others (like μ_i) or has a distribution (σ). Stochastic variables are ones which don't have deterministic values, like σ
3. Combined as above, all the variables define a join generative model which can be used both to simulate hypothetical observations and analyze real ones.

0.3 Show a man a model and he will learn better

Let's model the height of a particular tribe !Kung San using what we've learnt till now

```r
library(rethinking)
```

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

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

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
## rstan (Version 2.21.2, 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)
```

```
## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
## Loading required package: parallel
```

```
## rethinking (Version 2.13)
```

```
##
## Attaching package: 'rethinking'
## The following object is masked from 'package:stats':
##
##     rstudent
```

```
data(Howell1)
d <- Howell1
# Get the heights of people with age>18
d2 <- d[d$age>=18, ]
```

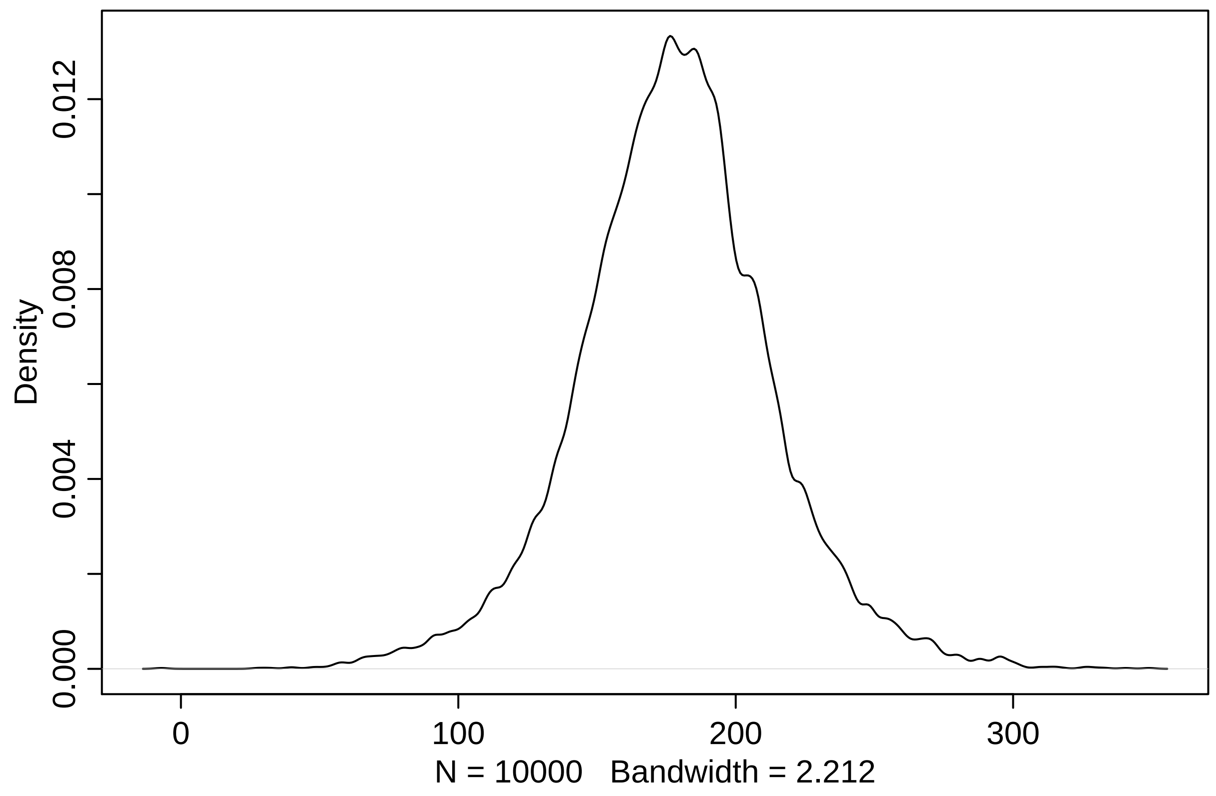
Model

$$
\begin{aligned}
h_i &\sim Normal(\mu, \sigma) \\
\mu &\sim Normal(178, 20) \\
\sigma &\sim Uniform(0, 50)
\end{aligned}
$$

Explaination - We assume height is Normally distributed with some mean and sigma which we have to find. The priors are defined (this is BAyesian analysis after all!)

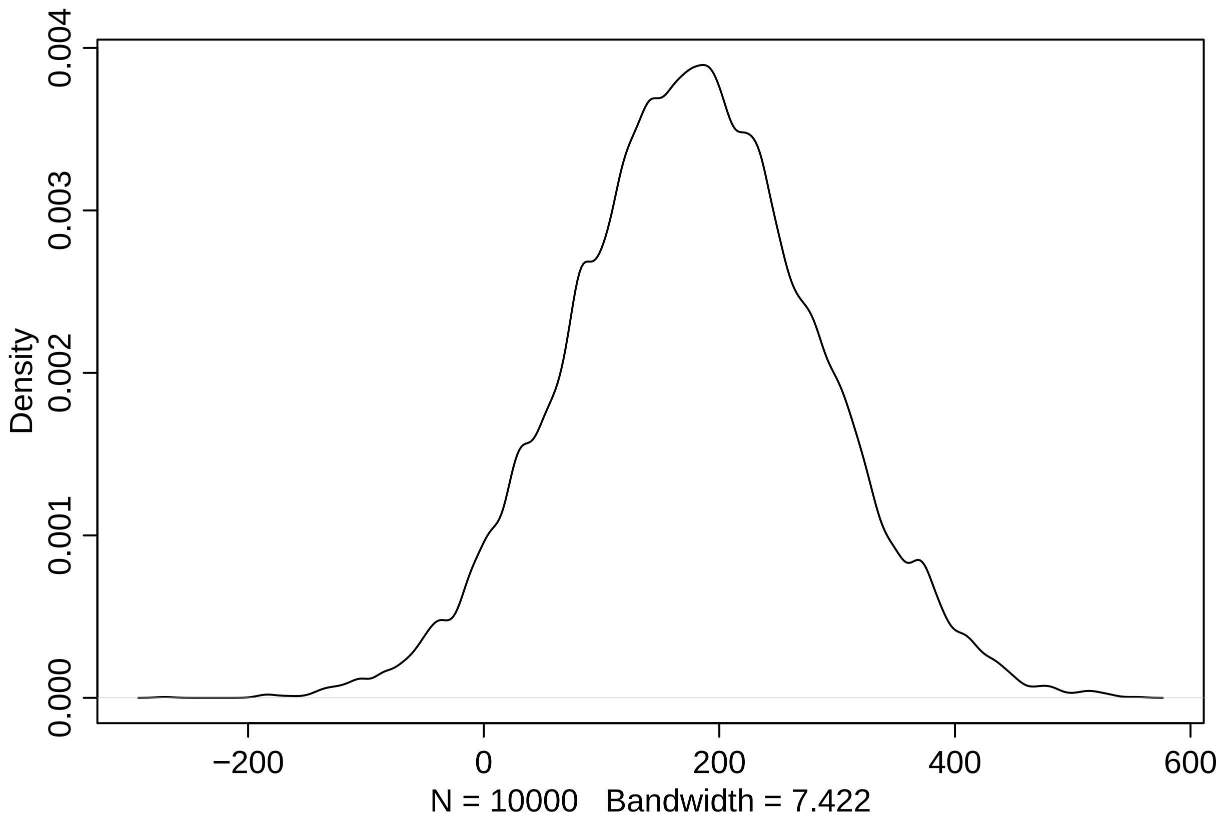
0.3.1 Sanity Checks - Generating data from our generative model

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



Looks pretty ain't it? What if we assign a different, flatter prior for sigma?

```r
sample_mu <- rnorm(1e4, 178, 100)
sample_sigma<-runif(1e4, 0, 50)
prior_h<- rnorm(1e4, sample_mu, sample_sigma)
dens(prior_h)
```



```
print("Density of heights above 2.5 m")
```

```
## [1] "Density of heights above 2.5 m"
```

```
sum(prior_h>250)/length(prior_h)
```

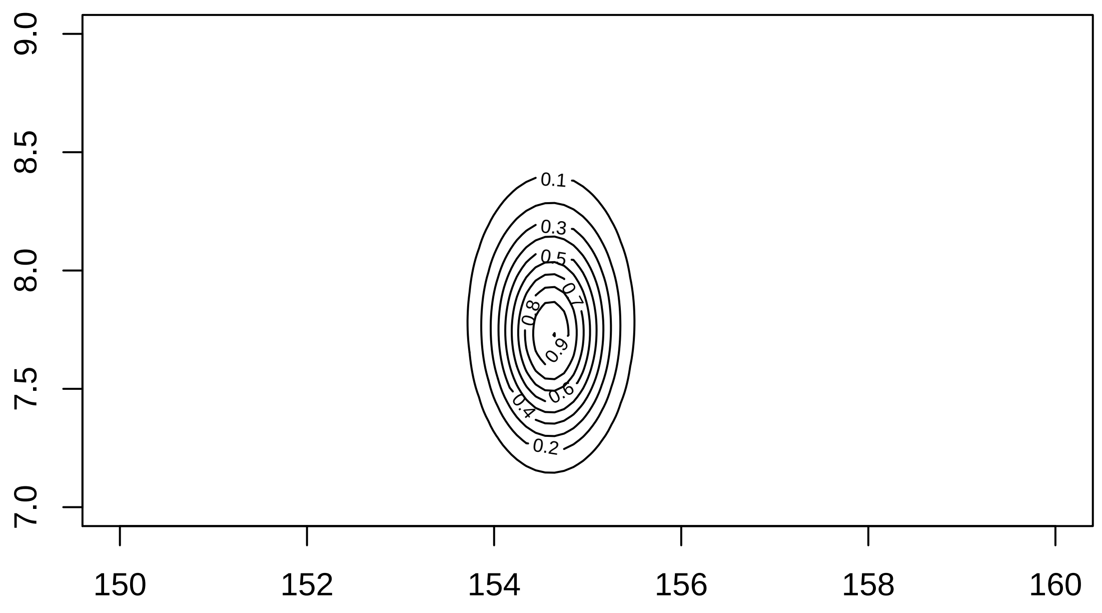
```
## [1] 0.2422
```

Clearly, 24.9 % of population can't have height more than 2.5 meters, hence we reject the choice of 100 as variance of mean prior. See, that's why this is useful.

0.3.2 Model Solving 2: Grid approximation of posterior distribution

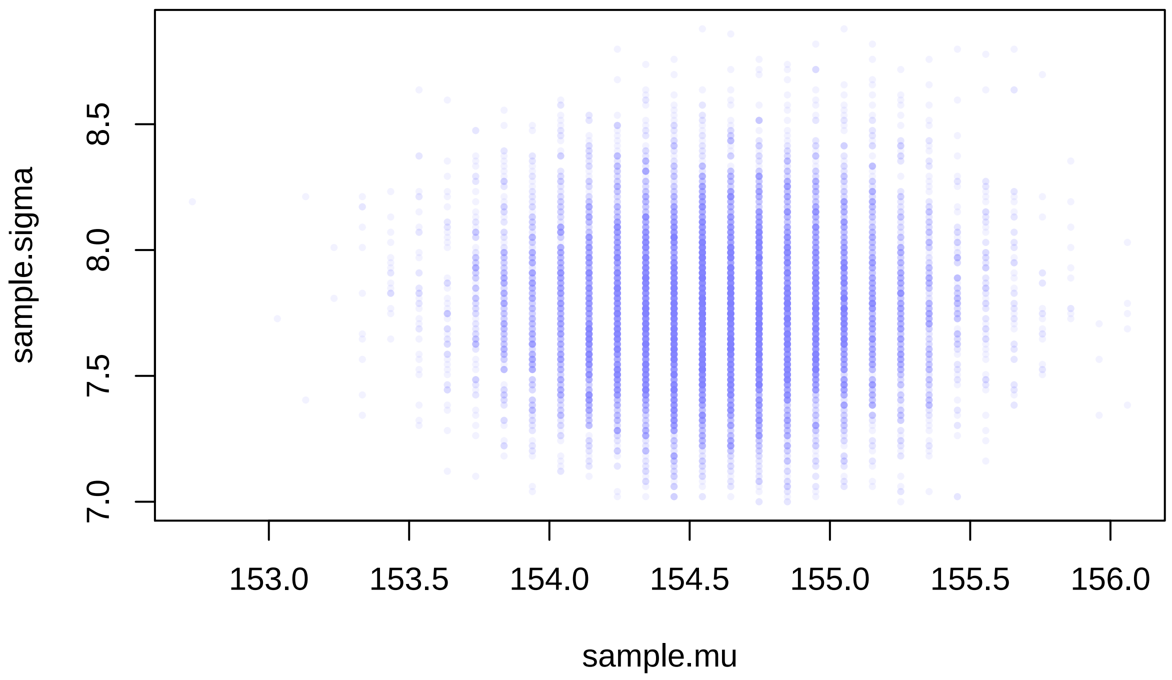
As few parameters, let's try to use Grid approximation to get likelihood (LL) and the posterior distribution

```
mu.list <- seq(from=150, to=160, length.out = 100) # We know mean is between these
sigma.list <- seq(from=7, to=9, length.out=100)
post <- expand.grid(mu=mu.list, sigma=sigma.list) # create a grid of params
post$LL <- sapply(1:nrow(post),
                  function(i) sum(dnorm(d2$height, post$mu[i], post$sigma[i], log=TRUE))) # We're using
post$prod <- post$LL + dnorm(post$mu, 178, 20, log=TRUE) + dunif(post$sigma, 0, 50, log=TRUE) # Multipl
post$prob <- exp(post$prod - max(post$prod)) # convert back from log, but honestly not sure why normali
contour_xyz(x=post$mu, y=post$sigma, z=post$prob)
```

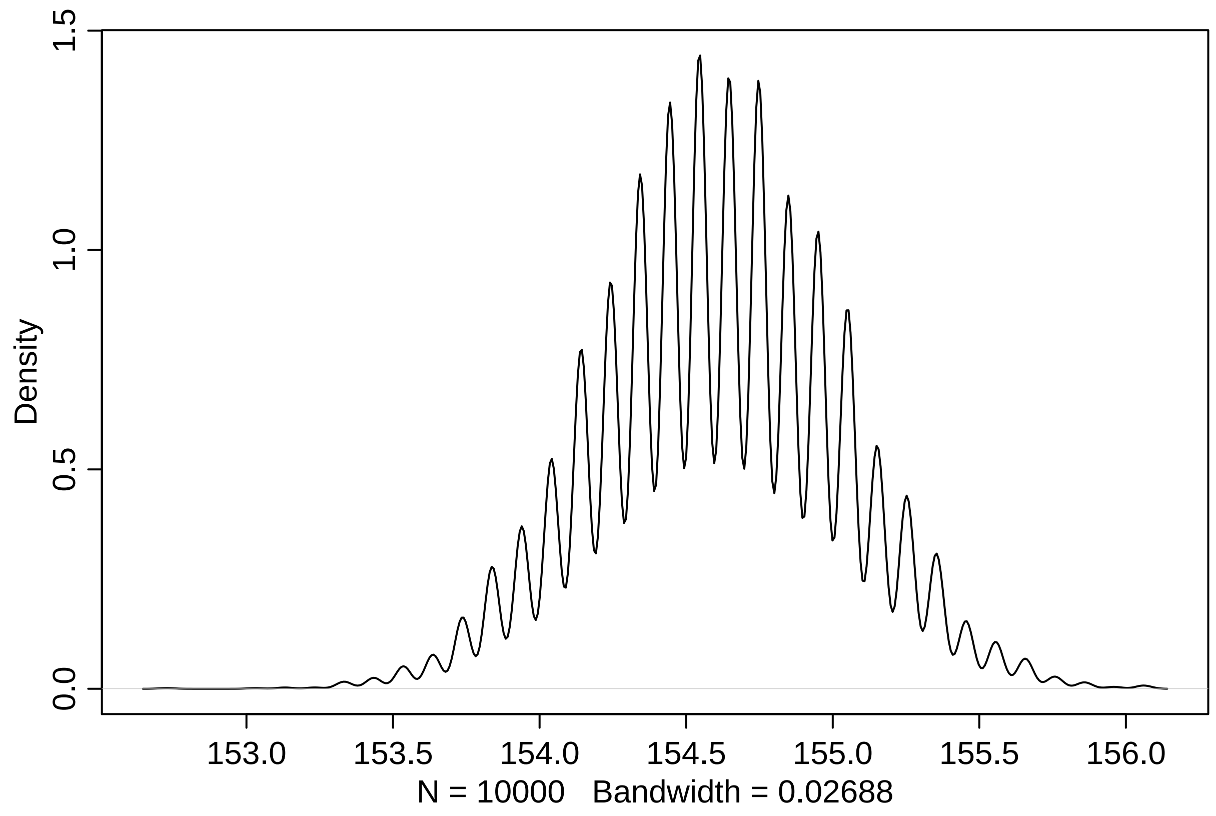


Yes graphs are pretty but can we get more scientific numbers about the μ and σ ? - Well we ahve a joint distribution (kinda, only a discrete approximation of it) so let's sample from it and get the confidence intervals.

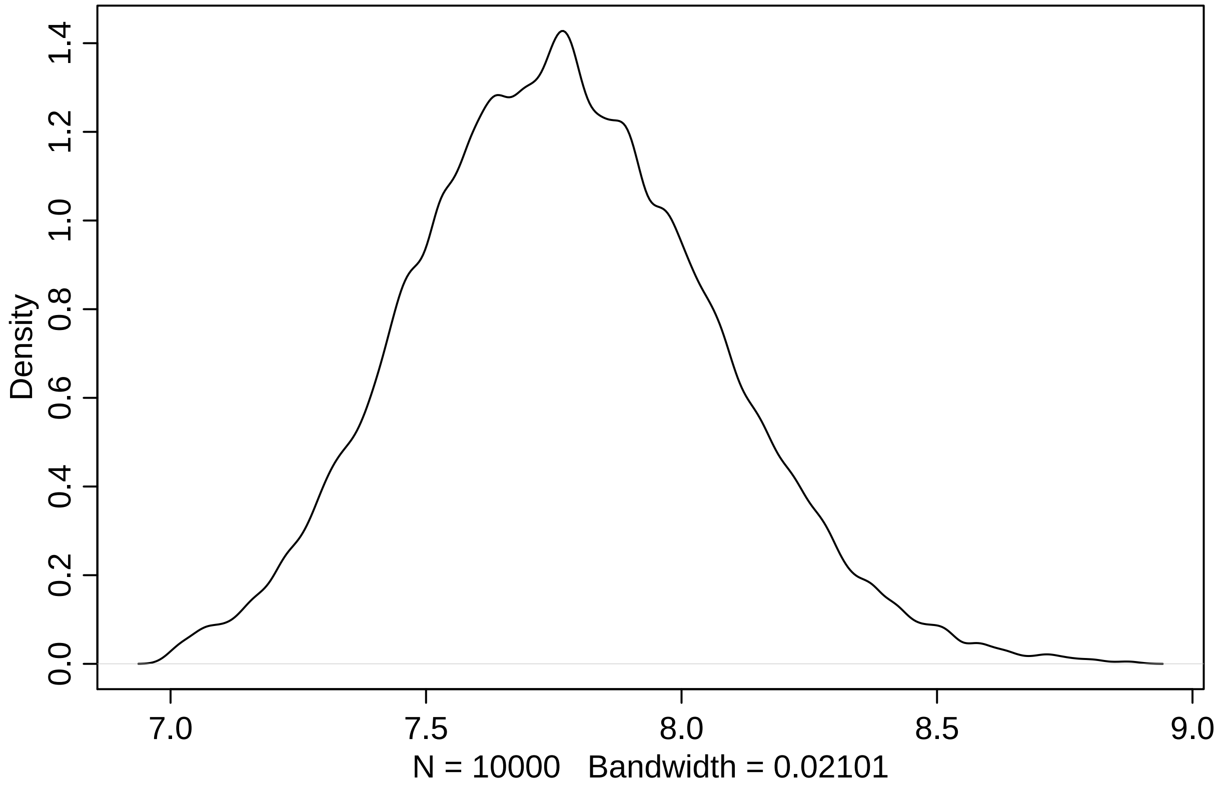
```
sample.rows <- sample(1:nrow(post), size=1e4, replace=TRUE, prob = post$prob)
sample.mu <- post$mu[sample.rows]
sample.sigma <- post$sigma[sample.rows]
plot(sample.mu, sample.sigma, cex=0.5, pch=16, col=col.alpha(rangi2, 0.1))
```



```
dens(sample.mu)
```



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



```r
PI(sample.mu)
```

```
##       5%      94%
## 153.9394 155.2525
```

```r
PI(sample.sigma)
```

```
##       5%      94%
## 7.323232 8.252525
```

Beautiful ain't it? We acheieved, in a few lines, identifying the complete distribution of the mean and variance for the height of adult tribesmen, including confidence intervals, all the while using proper statistical procedures! Keep it up! A few points to note -
1. Mean plot is not very smooth because we sampled a discrete range, it's a shortcoming of this Grid Approximation, one which can be resolved by having finer grid and/ or increasing bandwidth of the plot 2. Mean pretty much looks like Gaussian, they always are if the prior is Gaussian. But the Standard Deviation? It has a bit of fat tail to the right, reason is complex and will come in future but one way to understand is there's more way for σ to be larger than smaller because it has to stay positive (as implied by our Prior ?)

0.3.3 Model Solving 1: Quadratic Approximation of Posterior Distribution

```
data(Howell1)
d<- Howell1
d2 <- d[d$age>=18, ]
flist <- alist(height ~ dnorm(mu, sigma), mu ~ dnorm(156, 10), sigma ~ dunif(0, 50))
m4.1 <- quap(flist, data=d2)
```

This is how quap provided by rethinking package is used, it takes a list of functions which define the model (much like how we started) and the data. The names of parameters like height have, of course, to match the one in dataframe for it to be able to find it. Let's look at the results.

```
precis(m4.1)
```

```
##             mean        sd      5.5%     94.5%
## mu    154.599472 0.4117338 153.941442 155.257503
## sigma   7.731367 0.2913891   7.265671   8.197063
```

Above, we can see the estimated values with confidence intervals for the mu and sigma of height, much easier but under the hood!

How does quap work anuways? It starts off somewhere, finds the MAP (Maximum a Posteriori), fancy way of saying mu and sigma with max probability and, once on the peak, assumes they have a Gaussian distribution and goes about finding the mean and std. of that distribution.
So where does the Quap start climbing from? I think it randomly samples the Prior, but in this case, we do have a nice way to figure good starting points

```
start <- list(
  mu=mean(d2$height),
  sigma = sd(d2$height)
)
m4.1 <- quap(flist, data = d2, start = start)
precis(m4.1)
```

```
##             mean        sd      5.5%     94.5%
## mu    154.599468 0.4117316 153.941442 155.257495
## sigma   7.731325 0.2913851   7.265635   8.197014
```

PRetty much similar results as expected, what if we provide a much more concentrated prior for mu? Why ? - Just for fun sake!

```
m4.2 <- quap(
  alist(
    height ~ dnorm(mu, sigma),
    mu ~ dnorm(178, 0.1), # A very sharp prior, implying we're very very certain mean height is near 17
    sigma ~ dunif(0, 50) # No change here
  ),
  data = d2
)
precis(m4.2)
```

```
##            mean        sd      5.5%     94.5%
## mu    177.86375 0.1002354 177.70356 178.02395
## sigma  24.51757 0.9289236  23.03297  26.00216
```

We see, estimated mu is very close to our prior but sigma has gone from 7 to 24 - coz it had to explain most observations far away from mean by assuming a much larger sigma. Nice!

So, to replicate our analysis of sampling from the posterior, we need to understand that for a Multi-dimensional Gaussian we need to work with the variance-covariance (vcov) matrix - this is the glue that binds the dimensions together.
Confused? You should be .. A vcov matrix can be better understood by factoring in two -
1. Vector of variances for each dim (that's the usual one) 2. correlation (cor) matrix which tells how change in one variable impacts others

Let's see …

```
vcov(m4.1)
```

```
##                 mu        sigma
## mu    1.695229e-01 5.209366e-05
## sigma 5.209366e-05 8.490529e-02
```

```
diag(vcov(m4.1))
```

```
##         mu      sigma
## 0.16952290 0.08490529
```

```
cov2cor(vcov(m4.1))
```

```
##                 mu        sigma
## mu    1.0000000000 0.0004342135
## sigma 0.0004342135 1.0000000000
```

Note that diag(vcov(m4.1)) shows variance, which is $sigma^2$.

So, how do we then sample from the posterior?

```
post <- extract.samples(m4.1, n=1e4)
head(post)
```

```
##         mu    sigma
## 1 154.5136 7.446365
## 2 154.4123 7.588867
## 3 154.8896 8.265074
## 4 154.4832 8.179085
## 5 153.8293 7.541748
## 6 154.6215 7.942426
```

Beautiful! Let's check in detail

```
precis(post)
```

```
##             mean        sd       5.5%      94.5%
## mu    154.593807 0.4101677 153.931903 155.250521
## sigma   7.731611 0.2906313   7.269944   8.191174
##                                                                                                 histogra
## mu                                    <U+2581><U+2581><U+2581><U+2585><U+2587><U+2582><U+2581><U+258
## sigma <U+2581><U+2581><U+2581><U+2582><U+2585><U+2587><U+2587><U+2583><U+2581><U+2581><U+2581><U+258
```

Similar results as expected. But I'm not comfortable depending too much on this extract.sample function, what's happening under the hood?

```r
library(MASS)
post <- mvrnorm(n=1e4, mu=coef(m4.1), Sigma = vcov(m4.1))
head(post)
```

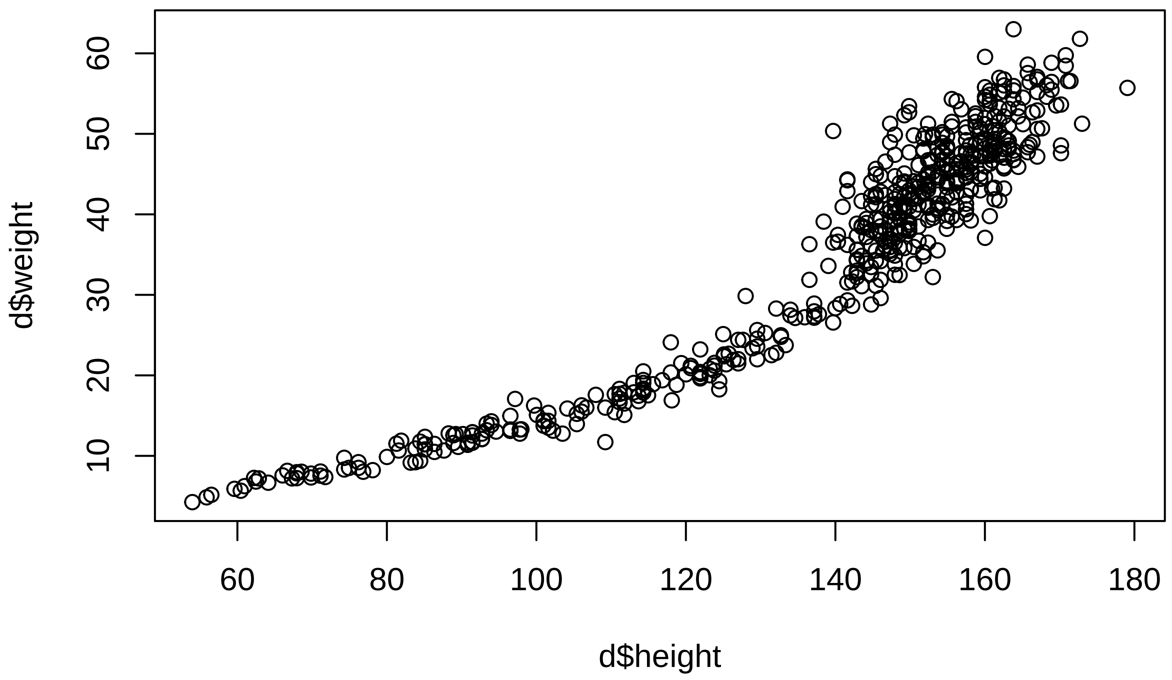
```
##             mu    sigma
## [1,] 154.1028 7.442315
## [2,] 154.1043 7.897337
## [3,] 154.3577 7.566989
## [4,] 153.6804 7.681441
## [5,] 155.4552 8.131808
## [6,] 154.6663 7.971271
```

Same same .. but different .. but still same !

0.4 Linear Prediction

How does weight of person correlate with height?

```r
plot(d$height, d$weight)
```



Clearly, there is a relation.

0.4.1 Building a regression model

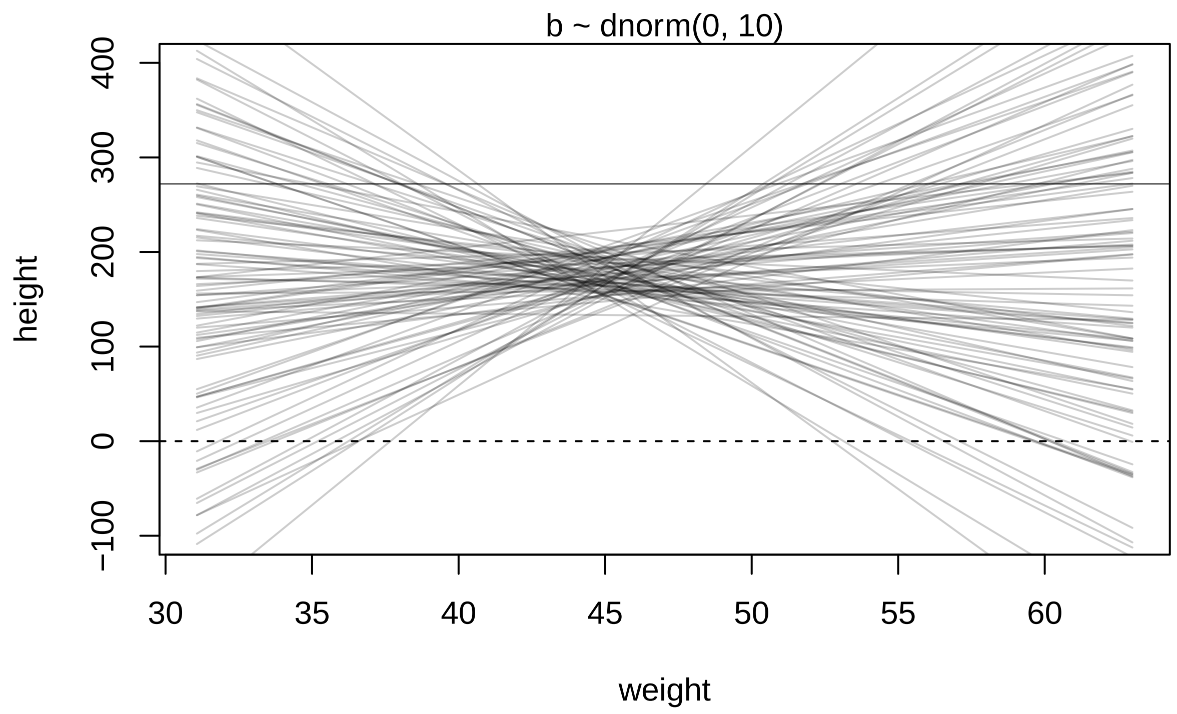
Regression is just a word, the meaning we will concretize as you follow the notes, for now think of it as a model which includes at elast one predictor variable, weight in above case. Also regression does not have to be linear although we'll start with one. Our first task in this is to formalize definition of the model. Let's give it a try.

$$\begin{aligned} h_i &\sim Normal(\mu_i, \sigma) \\ \mu_i &= \alpha + \beta * (x_i - \bar{x}) \\ \alpha &\sim Normal(178, 20) \\ \beta &\sim Normal(0, 10) \\ \sigma &\sim Uniform(0, 50) \end{aligned}$$

1. This time, we model mean of h_i, μ_i to be dependent on the weight of the person
2. This is new, instead of mu being a Gaussian, we now imply a deterministic relationship between mu and weight, given α and β where :
    1. [likelihood] α represents the mu for a person with average height(cm)
    2. [linear model] β represents how much height of a person would change for one unit of weight change (cm/kg) Note that we chose a linear relationship but it doesn't have to be so, it could very well be $\gamma * \exp(-(x_i - \bar{x}))$ for some other kind of problem where intuition tells us so, e.g., some sort of Covid decline. It would still be regression analysis
3. [Prior] α is still a Gaussian as before
4. [Prior] Should β be Gaussian centered at 0? Let's assume so for now, implying the weight height relationship could be postivie or negative equally likely
5. [Prior] We assume σ has to be independent of height, this doesn't have to be so but is usually a workable assumption for most Regression Models

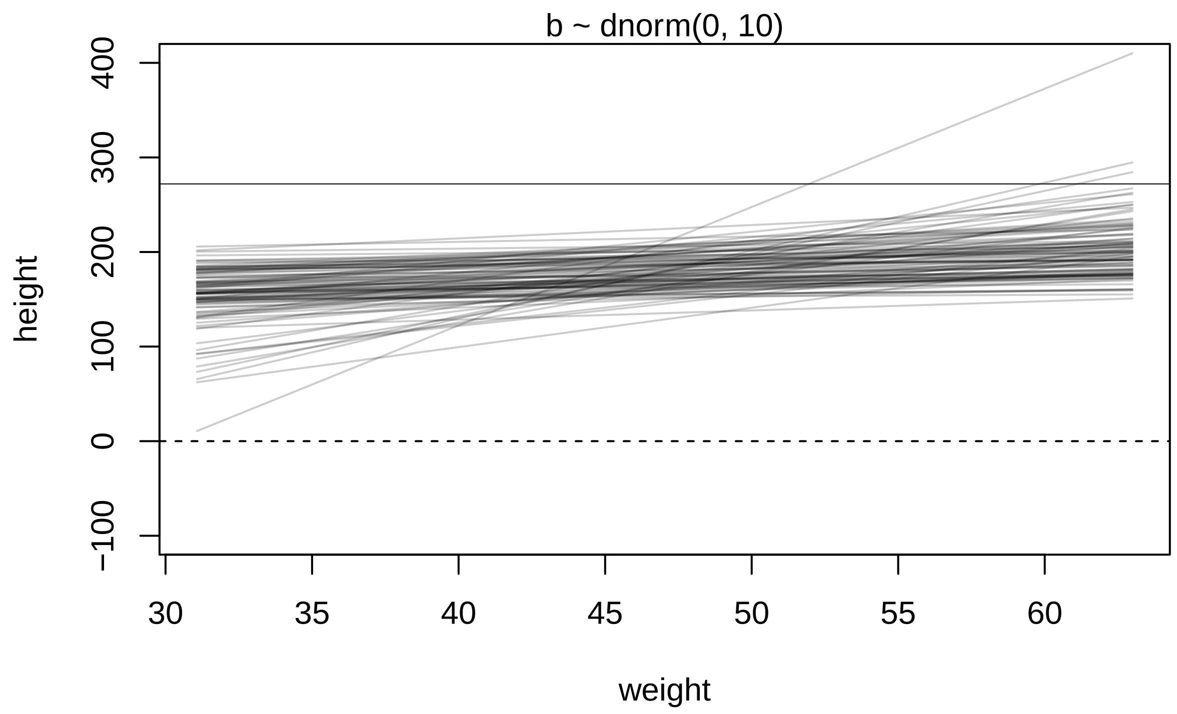
As a sanity check - Simulate Prior Predictive Distribution for μ_i - we could construct our own weights data to that (we haven't added the model for weights!), but for now let's use the existing weights.

```r
set.seed(2971)
N <- 100 # Number of simulations
a <- rnorm(N, 178, 20) # alpha prior
b <- rnorm(N, 0, 10) # beta prior
# Prepare plot
plot(NULL, xlim=range(d2$weight), ylim=c(-100, 400),
     xlab="weight", ylab="height")
# Add lines for tallest and shortest possible human
abline(h=0, lty=2)
abline(h=272, lty=1, lwd=0.5)
mtext("b ~ dnorm(0, 10)")
# define mean weight
xbar <- mean(d2$weight)
for (i in 1:N)
  curve(a[i] + b[i]*(x-xbar), from = min(d2$weight), to = max(d2$weight), add=TRUE, col=col.alpha("blac
```



In the simulation, a lot of heights tend to overshoot tallest possible or are negative, we kind of knew this would happen due to our β prior but sometimes these things are not clear and hence important to know how it's done.
Clearly, we need to change β prior, a log-normal would be good idea. $\beta \sim Log-Normal(0,1)$ This basically implies that $\log(\beta)$ would have normal distribution. Among other places, it's use to model returns on stock portfolios. Let's redo the above analysis now.

```
set.seed(2971)
N <- 100 # Number of simulations
a <- rnorm(N, 178, 20) # alpha prior
b <- rlnorm(N, 0, 1) # beta prior
# Prepare plot
plot(NULL, xlim=range(d2$weight), ylim=c(-100, 400),
     xlab="weight", ylab="height")
# Add lines for tallest and shortest possible human
abline(h=0, lty=2)
abline(h=272, lty=1, lwd=0.5)
mtext("b ~ dnorm(0, 10)")
# define mean weight
xbar <- mean(d2$weight)
for (i in 1:N)
  curve(a[i] + b[i]*(x-xbar), from = min(d2$weight), to = max(d2$weight), add=TRUE, col=col.alpha("blac
```



Much better now! There's an odd one for sure but most are well within human reason.

Note that the above analysis must be performed before we feed the data to model as otherwise we risk biasing our models.

Now on to finding the posterior distribution

```
d <- Howell1
d2 <- d[d$age >= 18, ]

xbar<- mean(d2$weight)

#fit model using quadratic

m4.3 <- quap(flist = alist(
  height ~ dnorm(mu, sigma),
  mu <- alpha + beta*(weight - xbar),
  alpha ~ dnorm(178, 20),
  beta ~ dlnorm(0, 1),
  sigma ~ dunif(0, 50)
),
data=d2)
```

0.4.2 Understanding our Models - Tables vs Ploting Simulations

With a Bayesian model such as above, we can (once skilled enough) answer almost any question with a confidence bound, all we gotta do is sample the parameters from their assigned probability distribution, in above case, we can draw large sample from joint distribution of α, β and *gamma* and hence, answer any question about remaining variables.

```
precis(m4.3)
```

```
##              mean         sd        5.5%       94.5%
## alpha 154.6013671 0.27030766 154.1693633 155.0333710
## beta    0.9032807 0.04192363   0.8362787   0.9702828
## sigma   5.0718809 0.19115478   4.7663786   5.3773831
```

```
round(vcov(m4.3) , 3 )
```

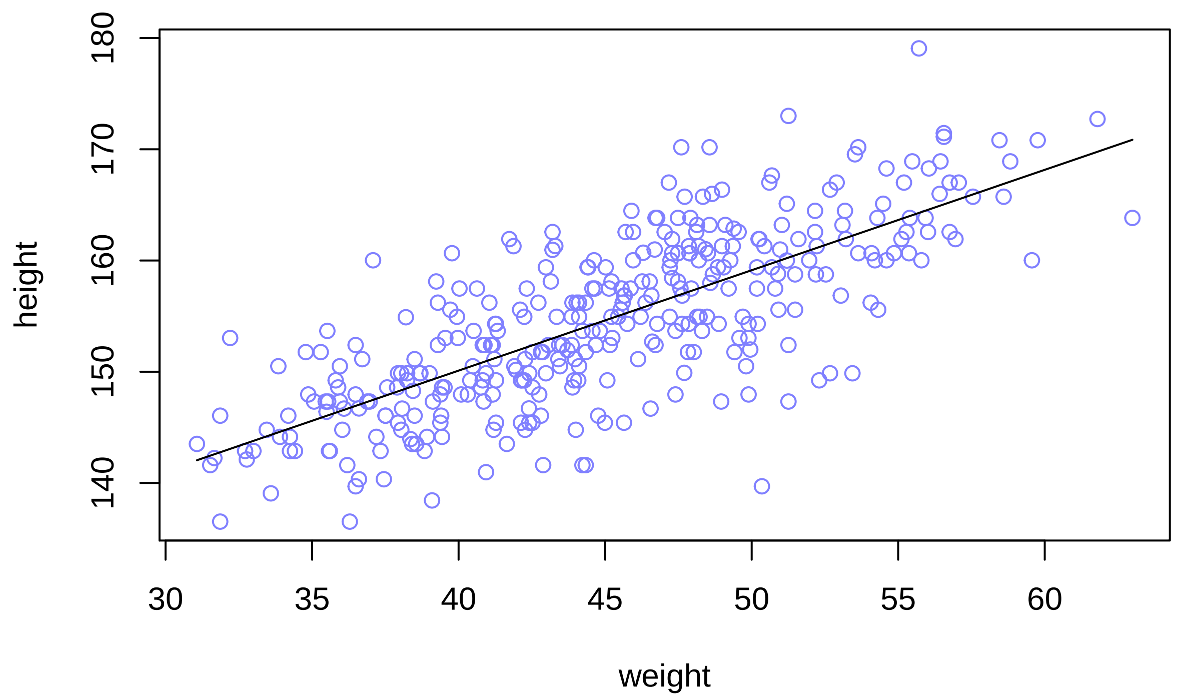
```
##       alpha  beta sigma
## alpha 0.073 0.000 0.000
## beta  0.000 0.002 0.000
## sigma 0.000 0.000 0.037
```

So, our mean height for a person with avergae weight is estimated to have mean of 154.6 with sd 0.27, hence a pretty confident prediction. On average, the height increases by 0.9 cm per 1 kg rise in weight. Now, this most certainly does not say that the relationship is a line, but rather, if you're committed to a line then 0.90 is the most appropriate slope for it.

The vcov matrix suggest there's almost no correlation between these parameters.

Now let's plot posterior inference against the data

```
plot(height ~ weight, data=d2, col=rangi2)
post <- extract.samples(m4.3)
a_map <- mean(post$a) # map estimate can be gotten from mean
b_map <- mean(post$b)

curve(a_map + b_map*(x-xbar), add=TRUE) # Plots mean inference
```

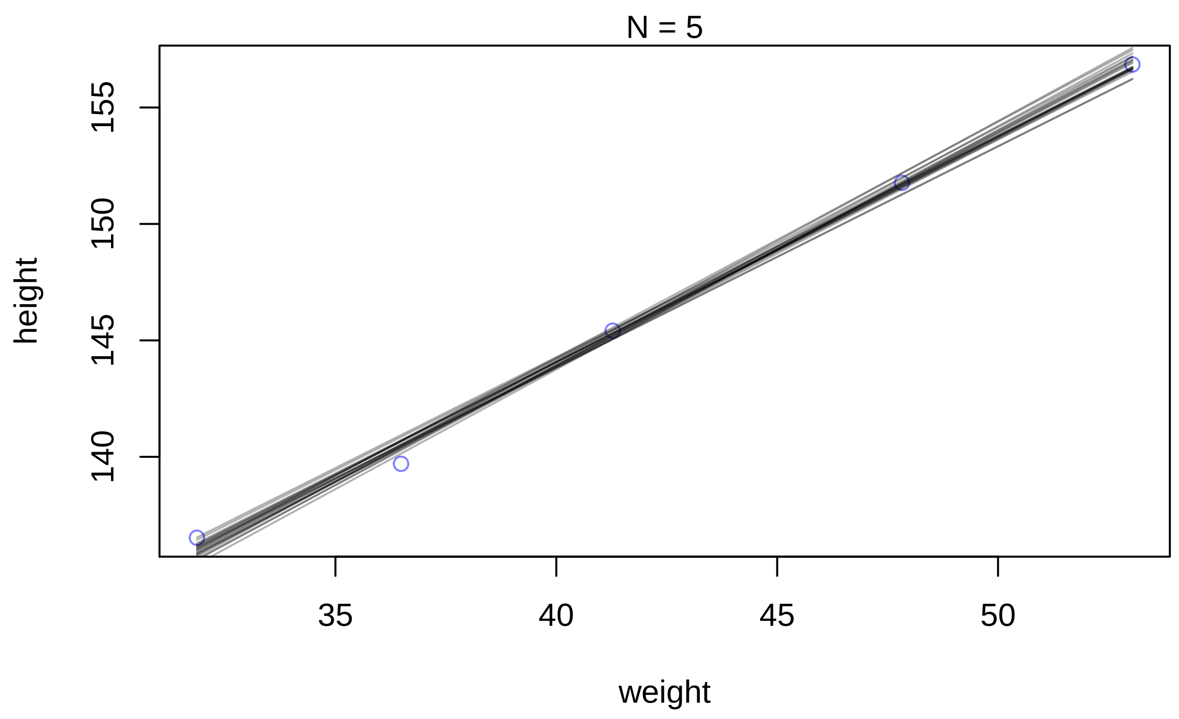


Look's alright! Now, note that this is the most probable line, model actually assigns probability to each possible line and by putting some of that in the plot can tell us how confident the model is.

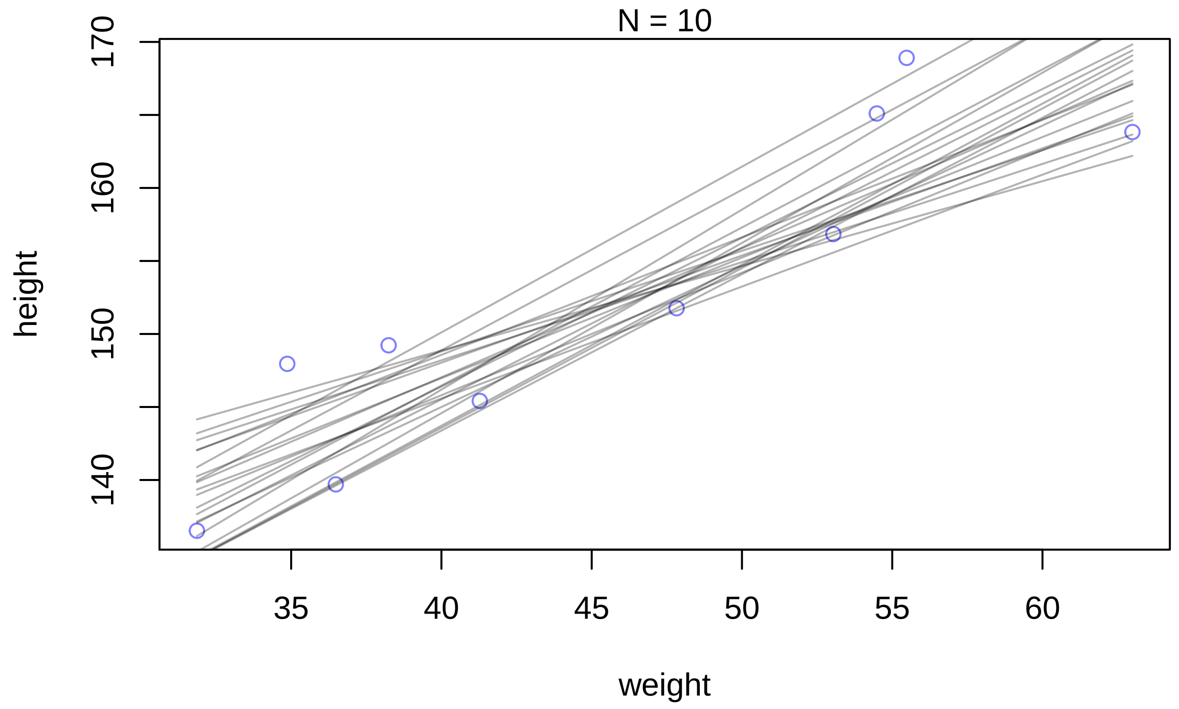
I know it's overloading by now but, let's also take this opportunity to see how the confidence changes as data increases as well.

```
plot_for_N <- function(N)
{
  dN <- d2[1:N, ]

  mN <- quap(
  alist(height ~ dnorm(mu, sigma),
        mu <- alpha + beta*(weight - mean(weight)),
        alpha ~ dnorm(178, 20),
        beta ~ dlnorm(0, 1),
        sigma ~ dunif(0, 50)),
  data=dN,
  start = list(alpha=178, beta=0.5, sigma=10)
  )

  # Let's plot 20 of the lines from above data to see what the uncertainty looks like
  post <- extract.samples(mN, n=20) # Samples 20 points from the joint distribution of alpha, beta, sig

  # display raw data and the lines
  plot(dN$weight, dN$height,
     xlim=range(dN$weight), ylim=range(dN$height),
     xlab="weight", ylab="height", col=rangi2)
  mtext(concat("N = ", N))
  mean_weight = mean(dN$weight)
  # plot the lines
  for (i in 1:20)
  curve(post$alpha[i] + post$beta[i]*(x-mean_weight),
        col=col.alpha("black", 0.3), add=TRUE)
}
```

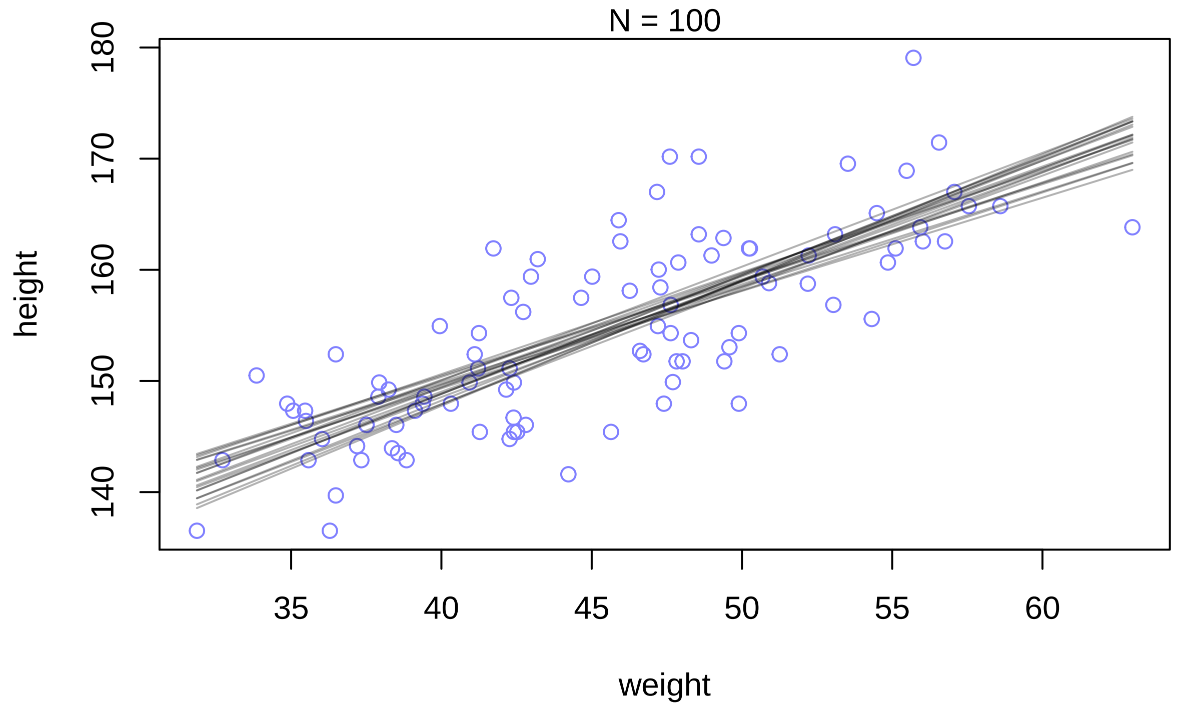
```
plot_for_N(5)
```



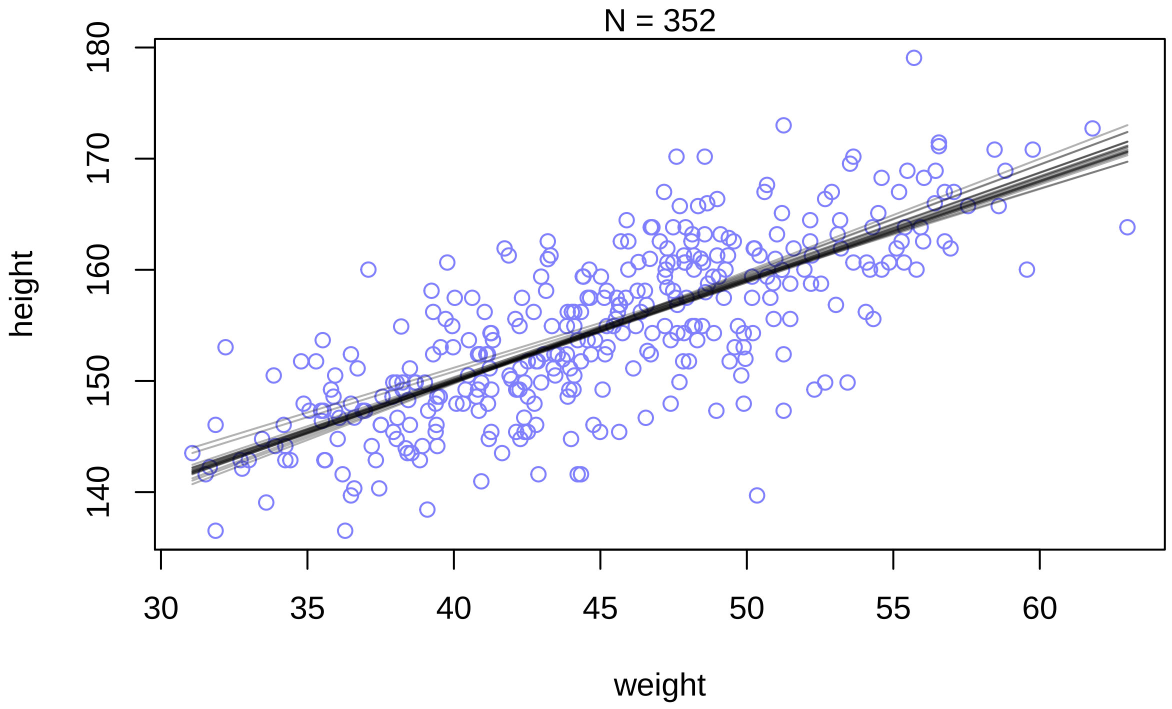
```
plot_for_N(10)
```



```r
plot_for_N(100)
```



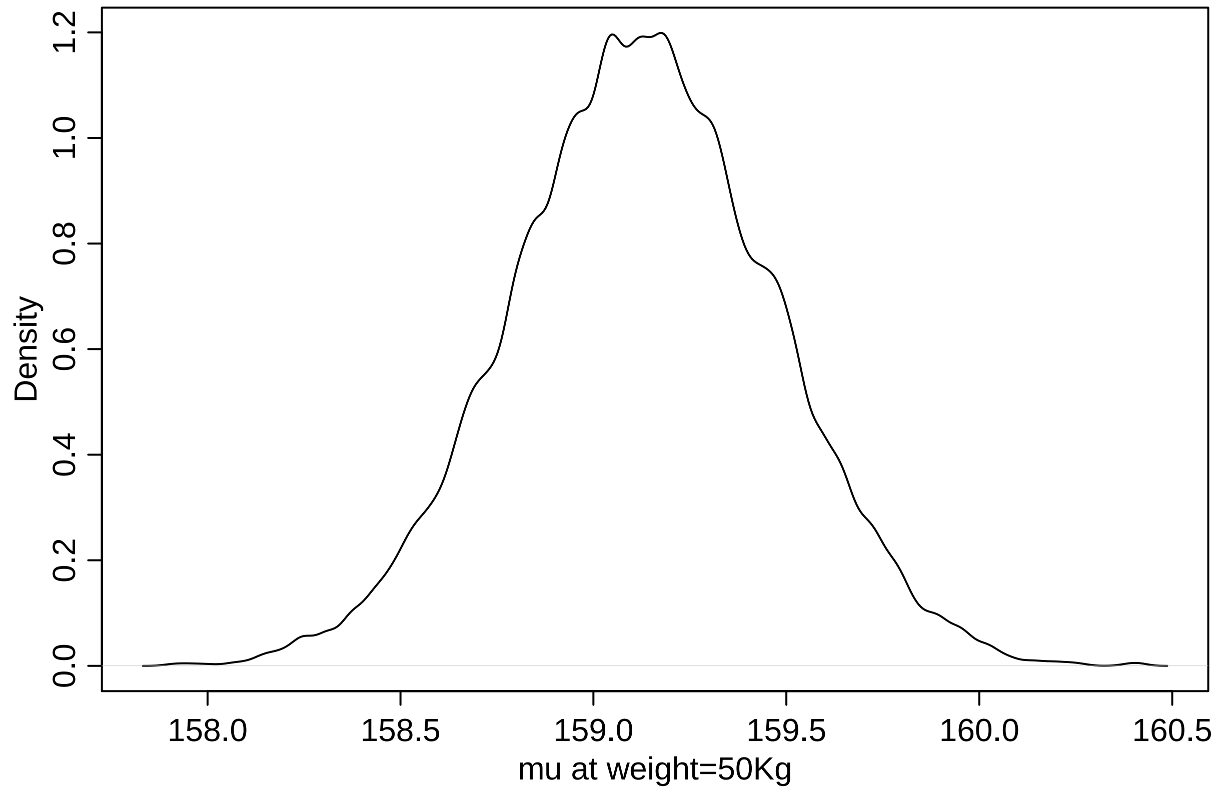
```
plot_for_N(length(d2$weight))
```



As we can see, more points to learn from => more confident (hence closer) lines. Also note the close lines for N=5, that is weird, not sure why it happens. But this does tell us not to read too much in this distribution of lines as high model confidence is only in the "small world", not in "large world".

Now, let's see distribution of μ (average height) for a person who's weight is 50 Kg.

```r
post <- extract.samples(m4.3)
mu_at_50 <- post$alpha + post$beta*(50 - mean(d2$weight))
dens(mu_at_50, xlab="mu at weight=50Kg")
```



```
PI(mu_at_50, prob=0.89)
```

```
##       5%      94%
## 158.5797 159.6737
```

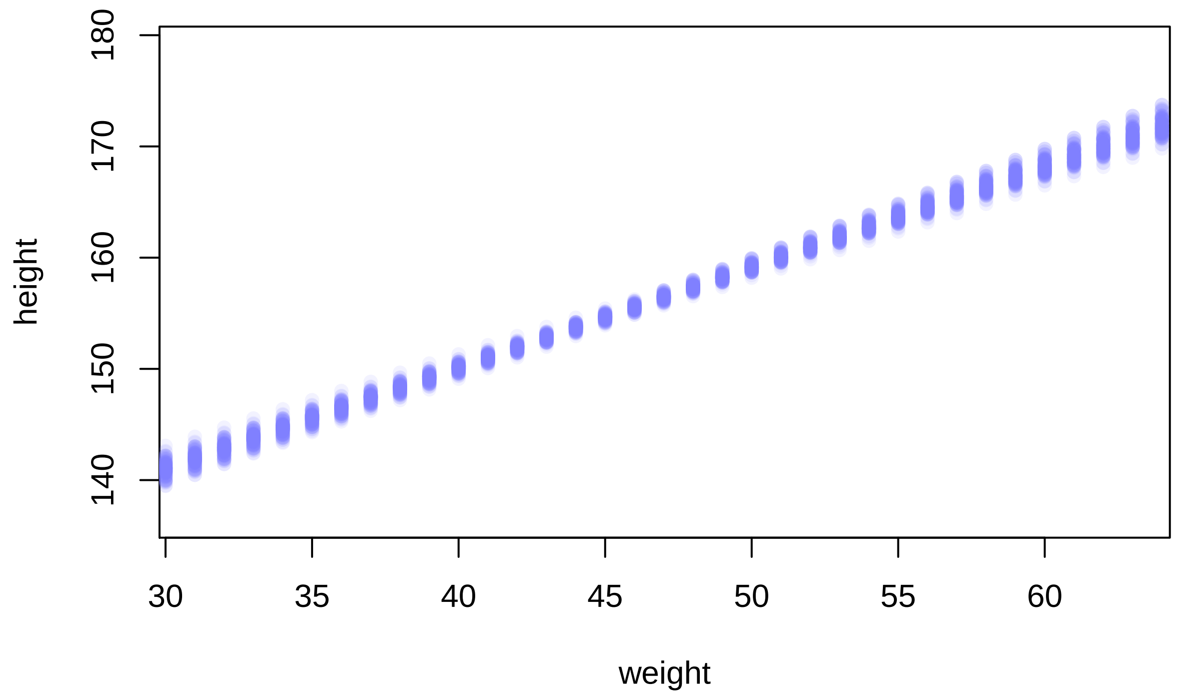
So our model is 89% confident that μ for this person lies between 158.59 and 159.67 cm.
Let's repeat above calculation for every person in our dataset and plot this 89% confidence interval to see the spread over our data. Why? Because we can :) Also for practice ... Also makes us feel like real statitsician.

```
post <- extract.samples(m4.3, 100)
xbar <- mean(d2$weight)
mu.link <- function(weight)
{post$alpha + post$beta*(weight-xbar)}
weight.seq <- seq(from=1, to=200, by=1) # LArge interval for fun sake
mu <- sapply(weight.seq, mu.link) # Shape is number of simulations x number of weights
mu.mean <- apply(mu, 2, mean)
mu.PI <- apply(mu, 2, PI, prob=0.89)
str(mu)
```

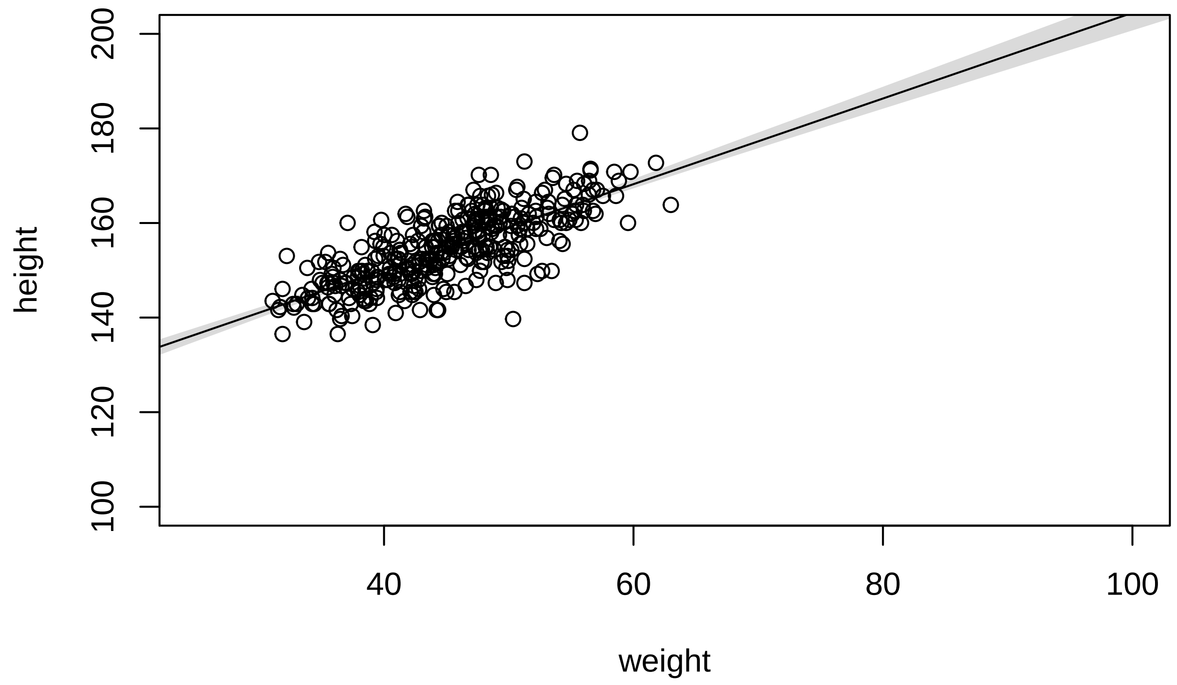
```
##  num [1:100, 1:200] 119 115 114 115 113 ...
```

```
plot(height ~ weight, d2, type="n")

# loop over weights and plot each mu value
```

```r
for (i in 1:100)
  points(weight.seq, mu[i, ], pch=16, col=col.alpha(rangi2, 0.1)) # everytime you plot one simulation
```



```r
# Now for a cooler "shade" plot!
plot(height ~ weight, d2, xlim=c(25, 100), ylim=c(100, 200))
lines(weight.seq, mu.mean) # This is the line joining mean prediction of mu, I know a bit confusing but
shade(mu.PI, weight.seq) # Provided by the Author of book, shades the area capturing the confidence int
```



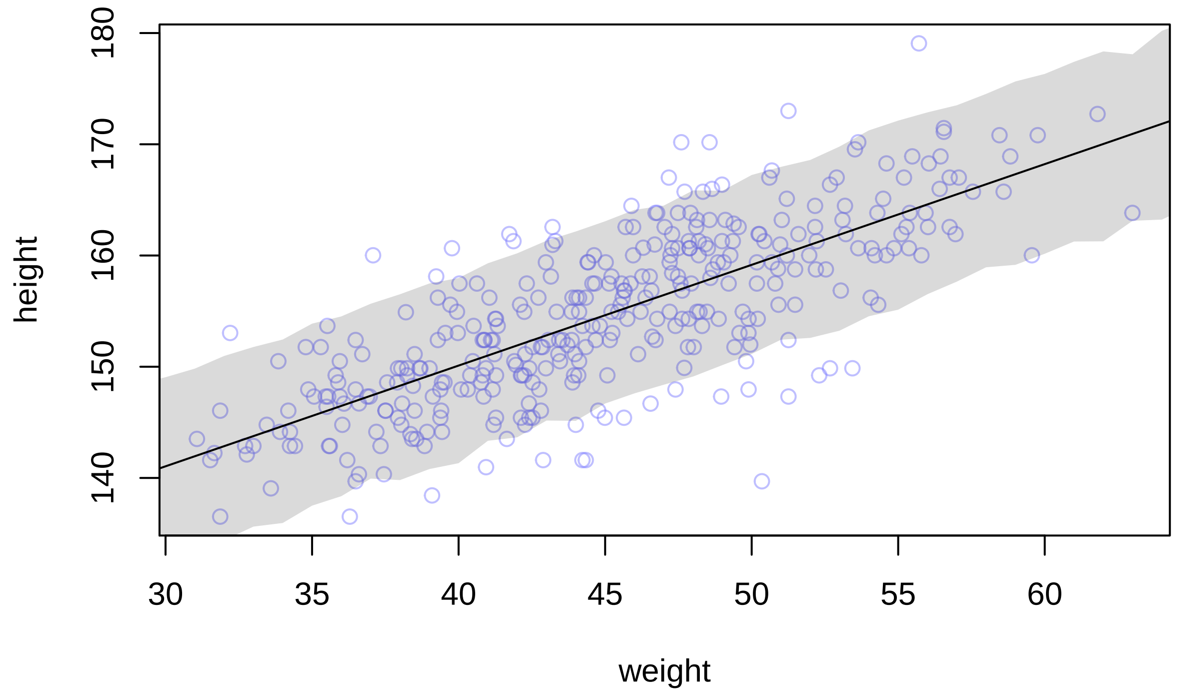
Looks cool eh? Basically, our model is more uncertain about what μ to assign to weights that are far outside the middle range of weights, hence the confidence interval get's broader. Also as you can see, the model doesn't make physical sense for weights near 20 kg (140 cm? That's one lanky boy) or the other end (100 kg, 200 cm? not possible unless ofcourse these are Dinka tribesmen, which they are not) clearly, linear has it's limitations.

So far, we've been working with mu not the actual height the model is predicting, that will have more uncertainty.
Basically, for a given weight, sample the posterior to α, β, γ, then for each such sample, calculate μ, (it's deterministic, rememebr?) then calculate simulated height using this μ and earlier σ.

```
sim.height <- sim(m4.3, data=list(weight=weight.seq))
str(sim.height) # Number of simulations x Number of weight samples
```

```
##  num [1:1000, 1:200] 103 107 115 107 109 ...
```

```
height.PI <- apply(sim.height,2,  PI, prob=0.89)
#plot raw data
plot(height~weight, d2, col=col.alpha(rangi2, 0.5))

#plot MAP line
lines(weight.seq, mu.mean)

#draw PI region based on PI intervals above
shade(height.PI, weight.seq)
```



There it is! Our models 89% confidence interval for each prediction as shaded. It's interesting to note that there are two uncertainties at play which make the above plot -

1. Uncertainty assigned by the model to the height of a person in line with the Gaussian distribution assumed
2. Uncertainty by virtue of generating samples to get the above plot. This can be reduced by sampling more and more points, but ultimately, this is part of the model assumptions as well. For e.g., some distributions are inherently difficult to sample from

Now we've been using sim function to simulate but just to show that simulations are simple, let's write our own code to get them

```
# generate weights for which to simulte
weights <- seq(from=25, to=70)
# Simulate parameters - okay we're still using a custom function
posterior_params <- extract.samples(m4.3)
mean_weight <- mean(weights)
# Define function to simulate height from params
simulate_height <- function(weight, params, mean_weight)
  rnorm(n=nrow(params), # One simulation per each parameter set
        mean = params$alpha + params$beta*(weight - mean(mean_weight)),
        sd = params$sigma)


simulated_heights <- sapply(weights, function(x) simulate_height(x, posterior_params, mean_weight))
```

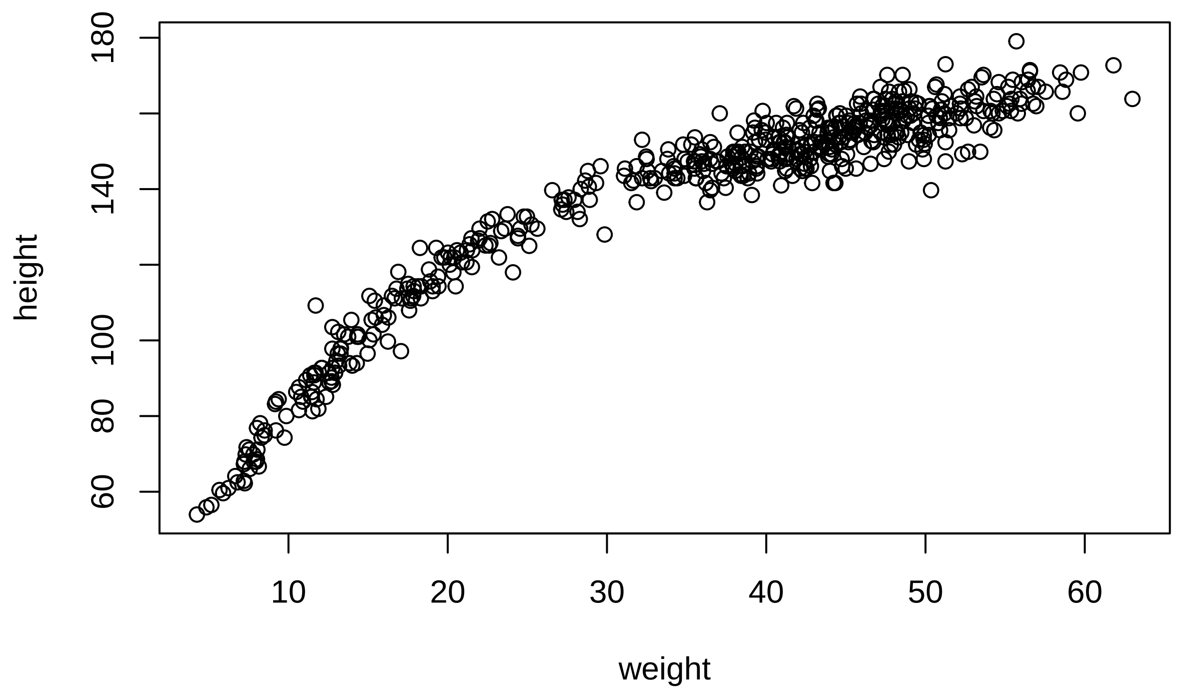
0.4.3 But I like Curves, not lines!

There's nothing special about fitting a line, we could as well fit a curve if the relationship is complicated.

Two primary methods exist, both use multiple versions of the predictor (weight above) and combine them with parameters which are then estimated.

Let's plot the full data including children.

```
plot(height~weight, d)
```



Let's add a $weight^2$ term to our model

$$
\begin{aligned}
h_i &\sim Normal(\mu_i, \sigma)\mu_i &= \alpha_i + \beta_1 * x_i + \beta 2 * x_i^2 \\
\alpha &\sim Normal(178, 20) \\
\beta_1 &\sim Log - Normal(0, 1) \\
\beta_2 &\sim Normal(0, 1) \\
\sigma &\sim Uniform(0, 50)
\end{aligned}
$$

- β_2 prior - It's tricky to assign it, but we do know it could be negative and so, can't be a log-normal
- It's important to normalize your target variable, especially in models with polynomial terms, else there may be (and often are) numerical issues. Always normalize target before feeding it, I know we didn't do that before but will do from now on

Let's build the model and infer!

```r
# Add normalized column and other features
d$weight_s <- (d$weight - mean(d$weight))/ sd(d$weight)
d$weight_s2 <- d$weight_s^2
d$weight_s3 <- d$weight_s^3

# A quadratic model
m4.5 <- quap(
  alist(
    height ~ dnorm(mu, sigma),
    mu <- a + b1*weight_s + b2*weight_s2,
    a ~ dnorm(178, 20),
    b1 ~ dlnorm(0, 1),
    b2 ~ dnorm(0, 1),
    sigma ~ dunif(0, 50)
  ),
  data=d
)


# A cubic model
m4.6 <- quap(
  alist(
    height ~ dnorm(mu, sigma),
    mu <- a + b1*weight_s + b2*weight_s2 + b3*weight_s3,
    a ~ dnorm(178, 20),
    b1 ~ dlnorm(0, 1),
    b2 ~ dnorm(0, 1),
    b3 ~ dnorm(0, 1),
    sigma ~ dunif(0, 50)
  ),
  data=d
)

precis(m4.5)
```

```
##              mean        sd       5.5%      94.5%
## a      146.057422 0.3689754 145.467728 146.647116
## b1      21.733062 0.2888890  21.271361  22.194762
## b2      -7.803274 0.2741837  -8.241473  -7.365075
## sigma    5.774473 0.1764650   5.492448   6.056498
```

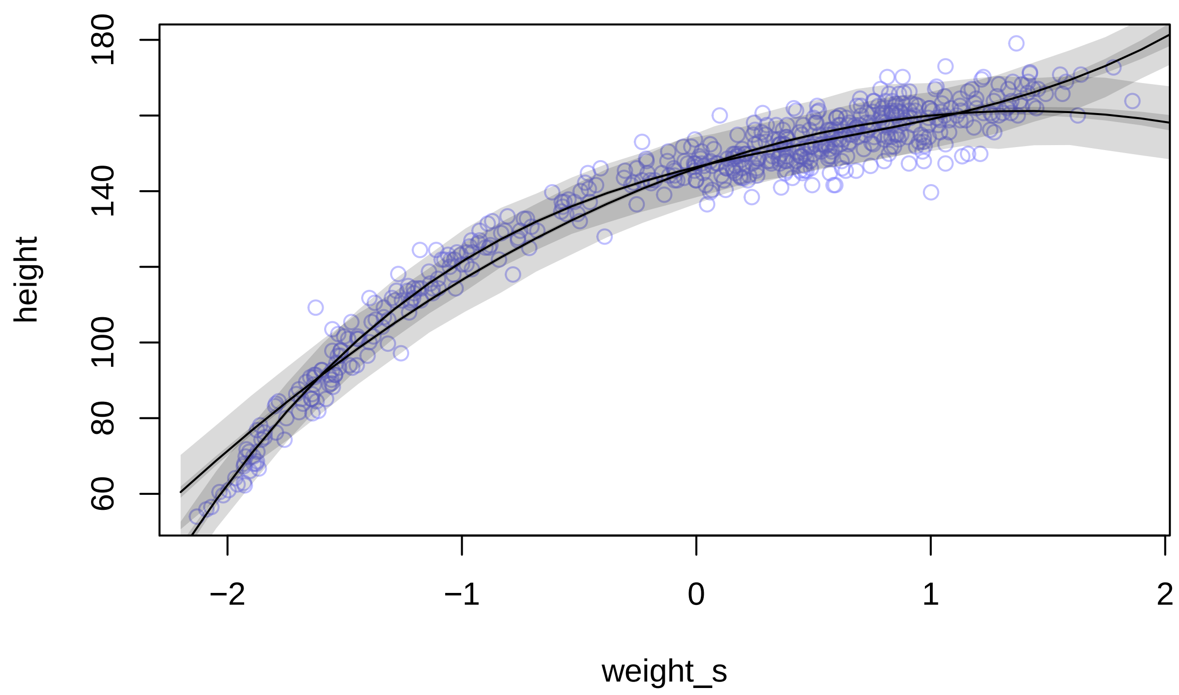
```r
precis(m4.6)
```

```
##              mean        sd       5.5%      94.5%
## a      146.394528 0.3099867 145.899110 146.889947
## b1      15.219726 0.4762645  14.458563  15.980888
## b2      -6.202627 0.2571578  -6.613615  -5.791639
## b3       3.583376 0.2287731   3.217753   3.949000
## sigma    4.829881 0.1469420   4.595039   5.064723
```

- b2 is indeed negative for the quadratic and cubic so our Gaussian prior was good choice

Let's plot the results

```r
weight.seq <- seq(from=-2.2, to=2.2, length.out=30)
pred_dat <- list(weight_s = weight.seq, weight_s2=weight.seq^2, weight_s3=weight.seq^3)

# Simulate for 2nd degree
mu <- link(m4.5, data=pred_dat)
mu.mean <- apply(mu, 2, mean)
mu.PI <- apply(mu, 2, PI,prob=0.89)
sim.height <- sim(m4.5, data=pred_dat)
height.PI <-apply(sim.height, 2, PI, prob=0.89)

# Simulate for 3rd degree
mu3 <- link(m4.6, data=pred_dat)
mu3.mean <- apply(mu3, 2, mean)
mu3.PI <- apply(mu3, 2, PI,prob=0.89)
sim.height3 <- sim(m4.6, data=pred_dat)
height3.PI <-apply(sim.height3, 2, PI, prob=0.89)

plot(height~weight_s, d, col=col.alpha(rangi2, 0.5)) # Original but standardized weights


lines(weight.seq, mu.mean)
shade(mu.PI, weight.seq)
shade(height.PI, weight.seq)


lines(weight.seq, mu3.mean)
shade(mu3.PI, weight.seq)
shade(height3.PI, weight.seq)
```



Now, the third power fits even better than the 2nd power. But which one is correct? Note that none give any explaination of any underlying connection between height and weight, they are just fitting the data, don't help biology much. Our purpose here was just to show that Linear Model (LM) don't just mean linear curves.

> The parabolic and cubic model of μ_i is still a "linear model" of the mean, even though the equation clearly is not a straight line. ... These models are geocentric engines, devices for describing partial correlations among variables. We should feel embarassed to use them, just so we don't become satisfied with the phenomenological explainations they provide.

0.4.3.1 Splines Spline originally referred to a long, thin piece of wood or metal that could be anchored in a few places in order to aid drafters or designers in drawing curves, In Stats, a spline is a smooth function built out of smaller, component functions.

We will primarily be focussing on B-Splines where B stands for basis functions (fancy term for component functions).

Enough measuring heights, let's measure something beautiful .. Temperature at peak Sakura blossom in Japan :) (thanks Prof Aono)

```
library(rethinking)
data("cherry_blossoms")
d<-cherry_blossoms
str(d)
```

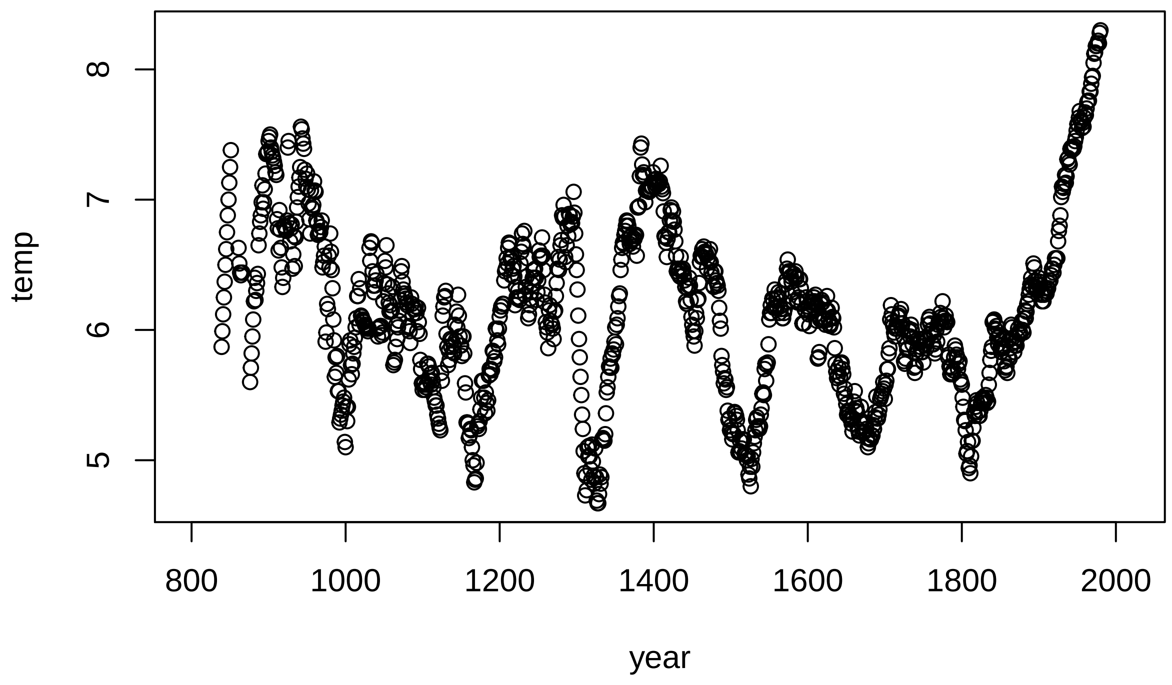
```
## 'data.frame':    1215 obs. of  5 variables:
```

```
##  $ year      : int  801 802 803 804 805 806 807 808 809 810 ...
##  $ doy       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ temp      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ temp_upper: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ temp_lower: num  NA NA NA NA NA NA NA NA NA NA ...
```

```r
precis(d)
```

```
##                    mean          sd      5.5%      94.5%
## year       1408.000000 350.8845964 867.77000 1948.23000
## doy         104.540508   6.4070362  94.43000  115.00000
## temp          6.141886   0.6636479   5.15000    7.29470
## temp_upper    7.185151   0.9929206   5.89765    8.90235
## temp_lower    5.098941   0.8503496   3.78765    6.37000
##
## year                           <U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U+2587><U
## doy                                                                        <U+2581><U+2582><U+2585><U+2587><U
## temp                                                                       <U+2581><U+2583><U+2585><U+2587><U
## temp_upper <U+2581><U+2582><U+2585><U+2587><U+2587><U+2585><U+2582><U+2582><U+2581><U+2581><U+2581><U
## temp_lower <U+2581><U+2581><U+2581><U+2581><U+2581><U+2581><U+2581><U+2583><U+2585><U+2587><U+2583><U
```

```r
plot(temp~year, data=d)
```

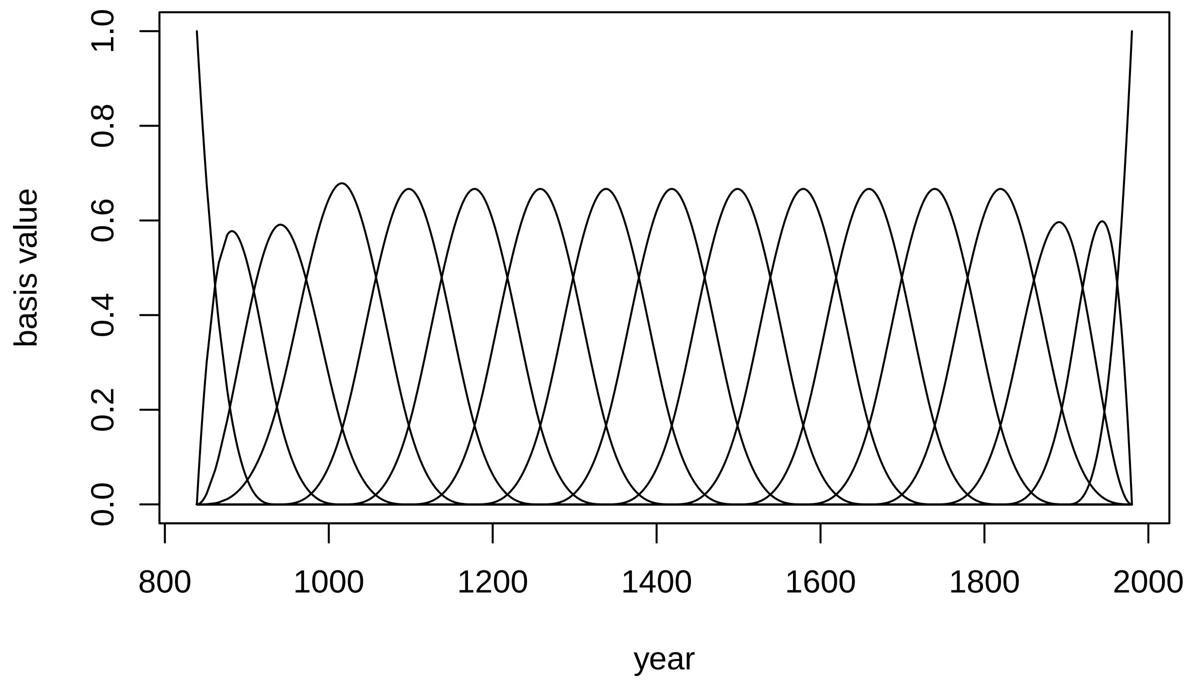


This ones more complex, let’s discuss theoty of B-Splines after going through it’s construction.

```r
d2<- d[complete.cases(d$temp), ] # Drop points where tmp not available
num_knots <- 15
knot_list <- quantile(d2$year, probs = seq(0, 1, length.out=num_knots)) # Get evenly spaced years, tota
str(knot_list)
```

```
##  Named num [1:15] 839 937 1017 1098 1178 ...
##  - attr(*, "names")= chr [1:15] "0%" "7.142857%" "14.28571%" "21.42857%" ...
```

```r
library(splines) # library to build splines
B <- bs(d2$year, # On what?
        knots = knot_list[-c(1, num_knots)], # Which knots to use
        degree = 3, # What degree polynomial should be used as basis function ?
        intercept=TRUE)

# B now contains the years and the variable values which will be used as Basis functions

plot(NULL, xlim=range(d2$year), ylim=c(0, 1), xlab="year", ylab="basis value") # create placeholder plo

for (i in 1:ncol(B) )
  lines(d2$year, B[,i])
```



The above plot shows the values of basis functions over time, at any given year, only few of the num_knots basis functions are active and contribute to the prediction, this helps keep predictions simple.
Once we get our basis functions as above, to get rest of the model is just basic linear regression with each column of the matrix B as a variable.

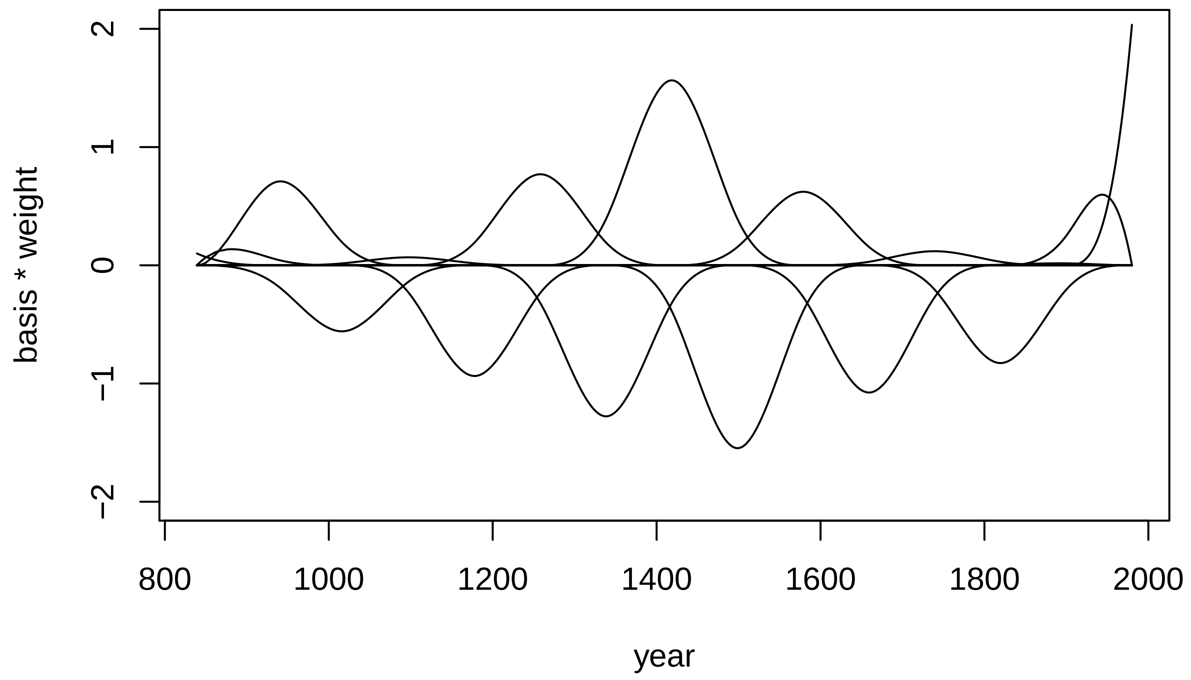
Now let's run quap on our model

```r
m4.7 <- quap(
  alist(T ~ dnorm(mu, sigma), # How does temperature depend on mu and sigma that's what we're trying to
        mu <- a + B %*% w, # This is just another way to restate a linear function by using a matrix re
        a  ~ dnorm(6, 10), # Just a relatively flat prior
        w ~ dnorm(0, 1), # All spline weights are normally distributed
        sigma ~ dexp(1) # Why is sigma exponential?
        ),
  data=list(T=d2$temp, B=B), # Provides T and B as input
  start=list(w=rep(0, ncol(B))) # Provide a starting point as 0 for all the spline weights
)

precis(m4.7, depth = 2) # Without depth it doesn't show the spline weights
```

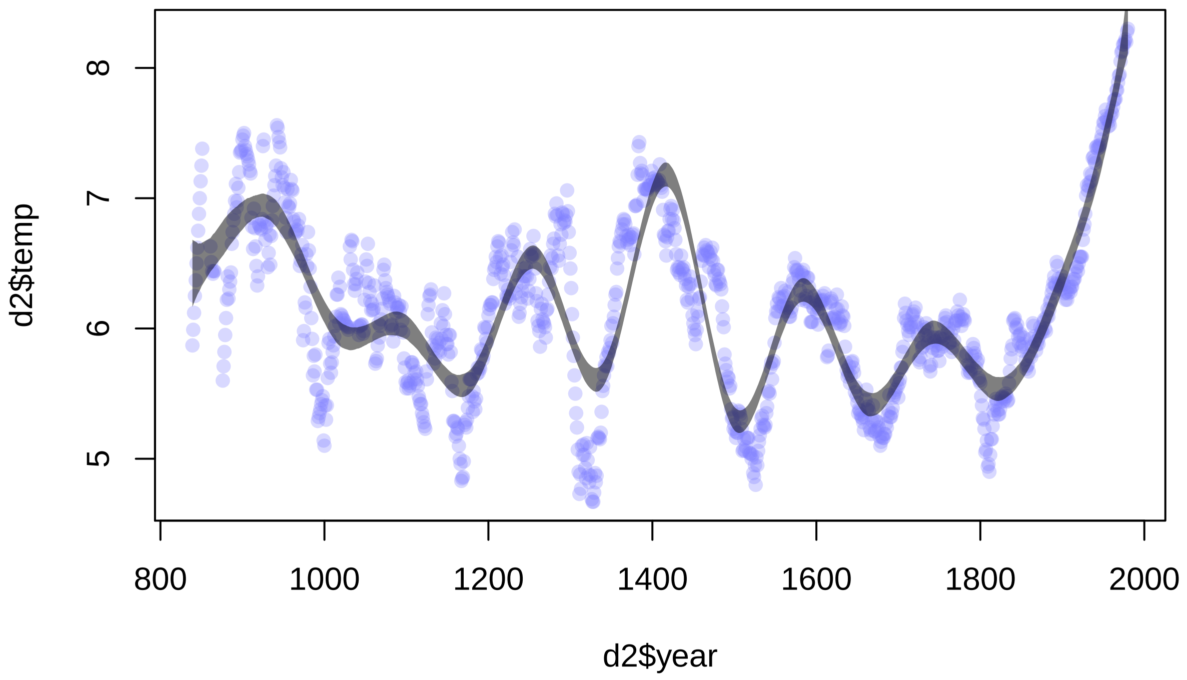
```
##              mean          sd       5.5%      94.5%
## w[1]   0.09749500 0.267674287 -0.3303002  0.5252902
## w[2]   0.23888549 0.279071546 -0.2071247  0.6848957
## w[3]   1.20069439 0.271685354  0.7664887  1.6349001
## w[4]  -0.82196742 0.259287425 -1.2363588 -0.4075760
## w[5]   0.10175704 0.257448405 -0.3096952  0.5132093
## w[6]  -1.40469662 0.257093170 -1.8155812 -0.9938121
## w[7]   1.15651338 0.256907959  0.7459248  1.5671019
## w[8]  -1.91586638 0.256912577 -2.3264623 -1.5052705
## w[9]   2.34741077 0.256876508  1.9368725  2.7579490
## w[10] -2.32072820 0.256924763 -2.7313436 -1.9101128
## w[11]  0.93550259 0.256914986  0.5249028  1.3461024
## w[12] -1.61490018 0.257200743 -2.0259566 -1.2038437
## w[13]  0.18054189 0.257668845 -0.2312627  0.5923465
## w[14] -1.24147653 0.260466610 -1.6577525 -0.8252006
## w[15]  0.03145816 0.270382834 -0.4006658  0.4635822
## w[16]  0.99683493 0.274757249  0.5577198  1.4359501
## w[17]  2.03576825 0.268397351  1.6068174  2.4647191
## a      6.32227482 0.242765645  5.9342884  6.7102612
## sigma  0.34423998 0.007262463  0.3326332  0.3558468
```

NOw, as usual, let's extract samples from the posterior.

```r
post <- extract.samples(m4.7)
w<-apply(post$w, 2, mean) # get mean weights for the spline
plot(NULL, xlim=range(d2$year), ylim=c(-2, 2),
     xlab="year", ylab="basis * weight") # Why yrange is -2 to 2? It's because our prior was -1 to 1 (r


# Below then plots what do all the Basis multiplied by weights contribute to our mean temperature of bl
for (i in 1:ncol(B))
  lines(d2$year, w[i]*B[, i])
```



```r
mu <- link(m4.7) # Link samples 1e4 predictions from our model vs weights
mu_PI <- apply(mu, 2, PI, prob=0.97)
plot(d2$year, d2$temp, col=col.alpha(rangi2, 0.3), pch=16)
shade(mu_PI, d2$year, col=col.alpha("black", 0.5))
```



Now, it looks like our final spline did a great job of associating temp with yearm abd, if you increase the number of knots and/or increase the degree of polynomcal, you'll do even better, but what should be their values? Do we aim for a perfect fit? We'll answer these question a bit later down the line, but those who have experience in ML recognize this as the overfitting problem.