ASTA

The ASTA team

Contents

1 Resampling techniques

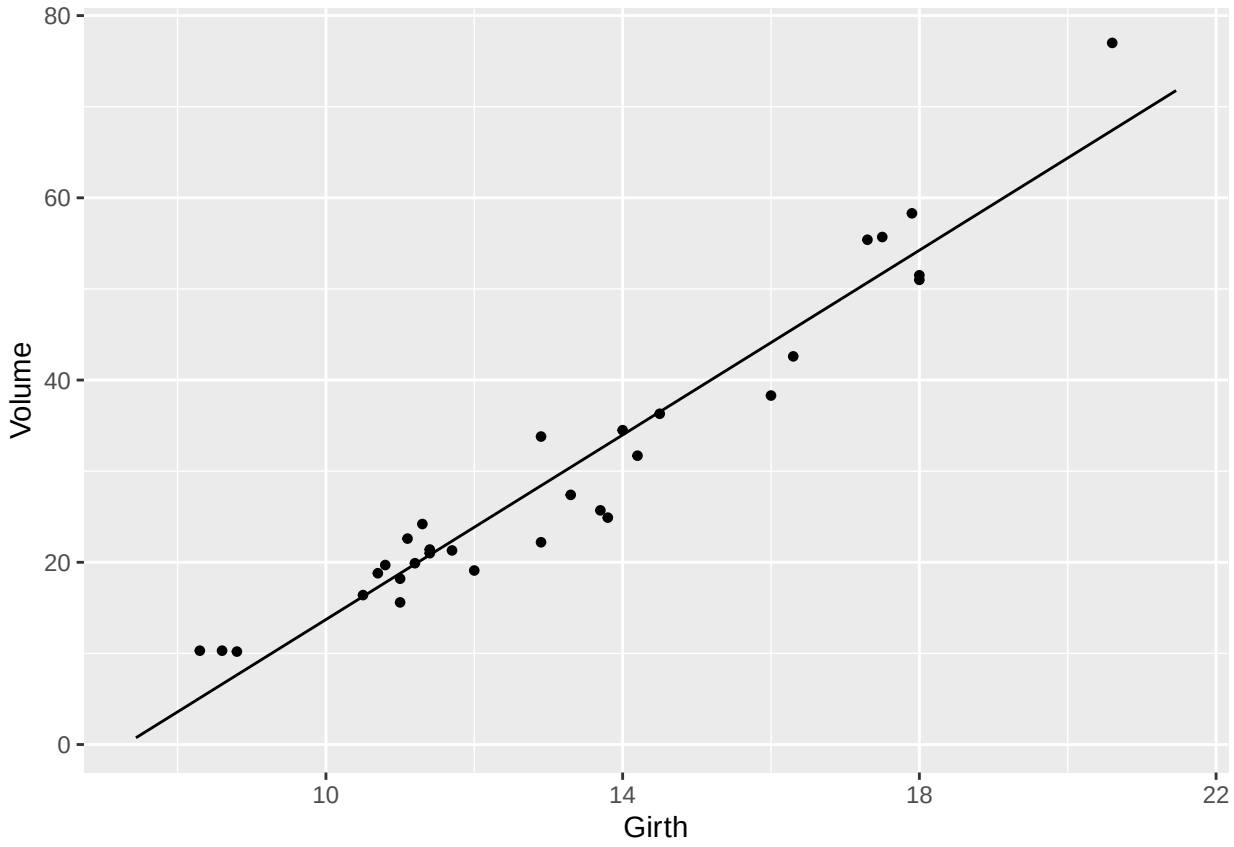
- Generalisation - how well a model performs on a new sample
  - Avoid overfitting
- Cross-validation (estimate out-of-sample prediction error)
- Bootstrap (estimate standard errors)

2 Signal or noise?

```
library(mosaic)
trees <- read.delim("https://asta.math.aau.dk/datasets?file=trees.txt")
head(trees)
```

```
##   Girth Height Volume
## 1   8.3     70     10
## 2   8.6     65     10
## 3   8.8     63     10
## 4  10.5     72     16
## 5  10.7     81     19
## 6  10.8     83     20
```

```
m0 <- lm(Volume ~ Girth, data = trees)
plotModel(m0)
```




```
summary(m0)$r.squared
```

```
## [1] 0.94
```

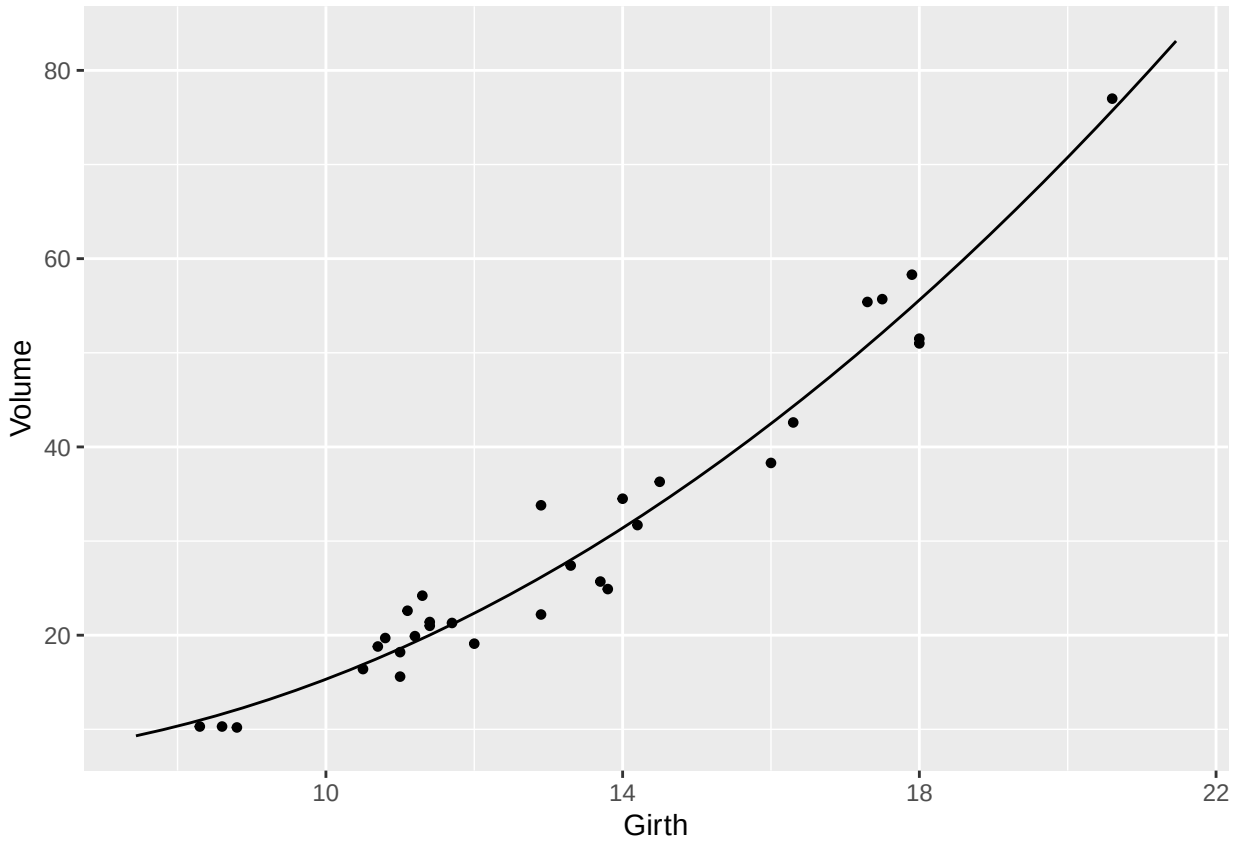
```
library(equatiomatic)
extract_eq(m0)
```

$$\text{Volume} = \alpha + \beta_1(\text{Girth}) + \epsilon \tag{1}$$

```
extract_eq(m0, use_coefs = TRUE)
```

$$\widehat{\text{Volume}} = -36.94 + 5.07(\text{Girth}) \tag{2}$$

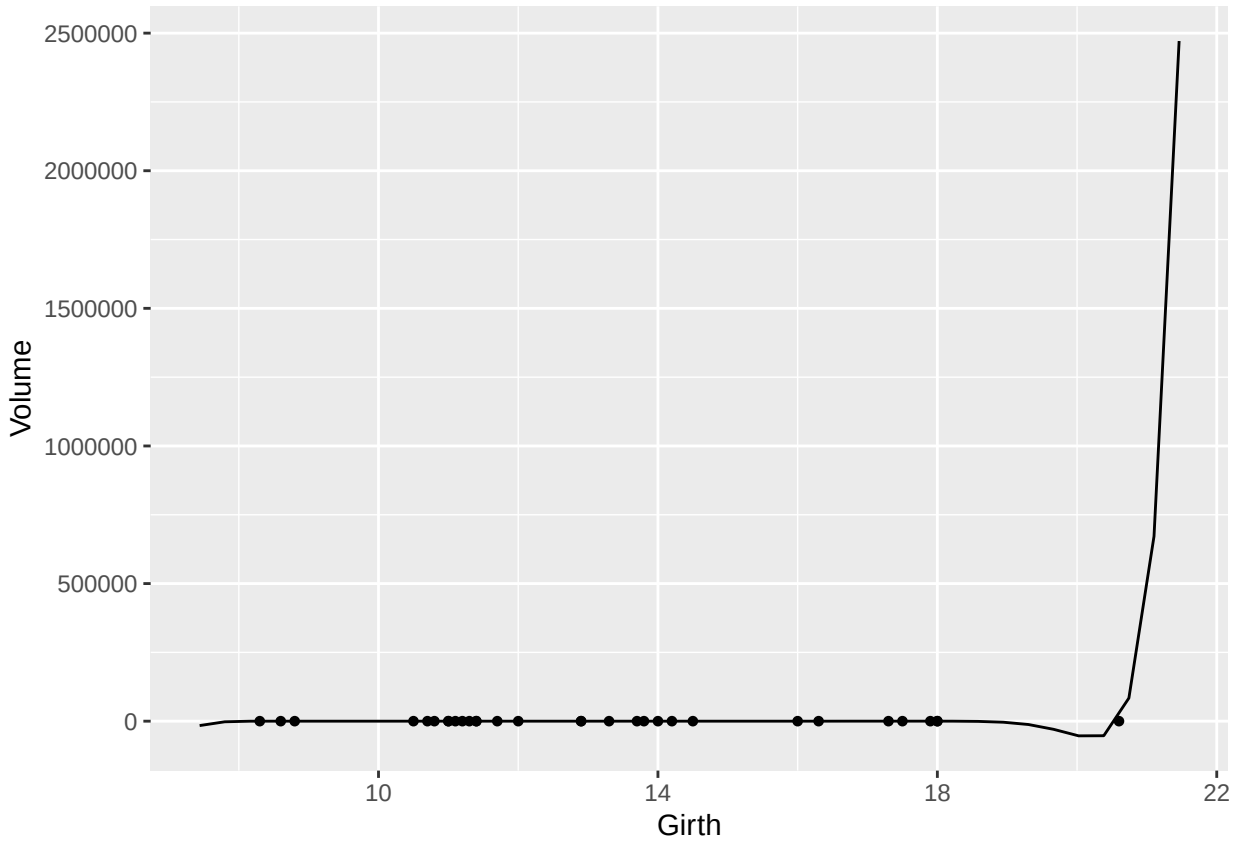
```
m1 <- lm(Volume ~ poly(Girth, 2), data = trees)
plotModel(m1)
```



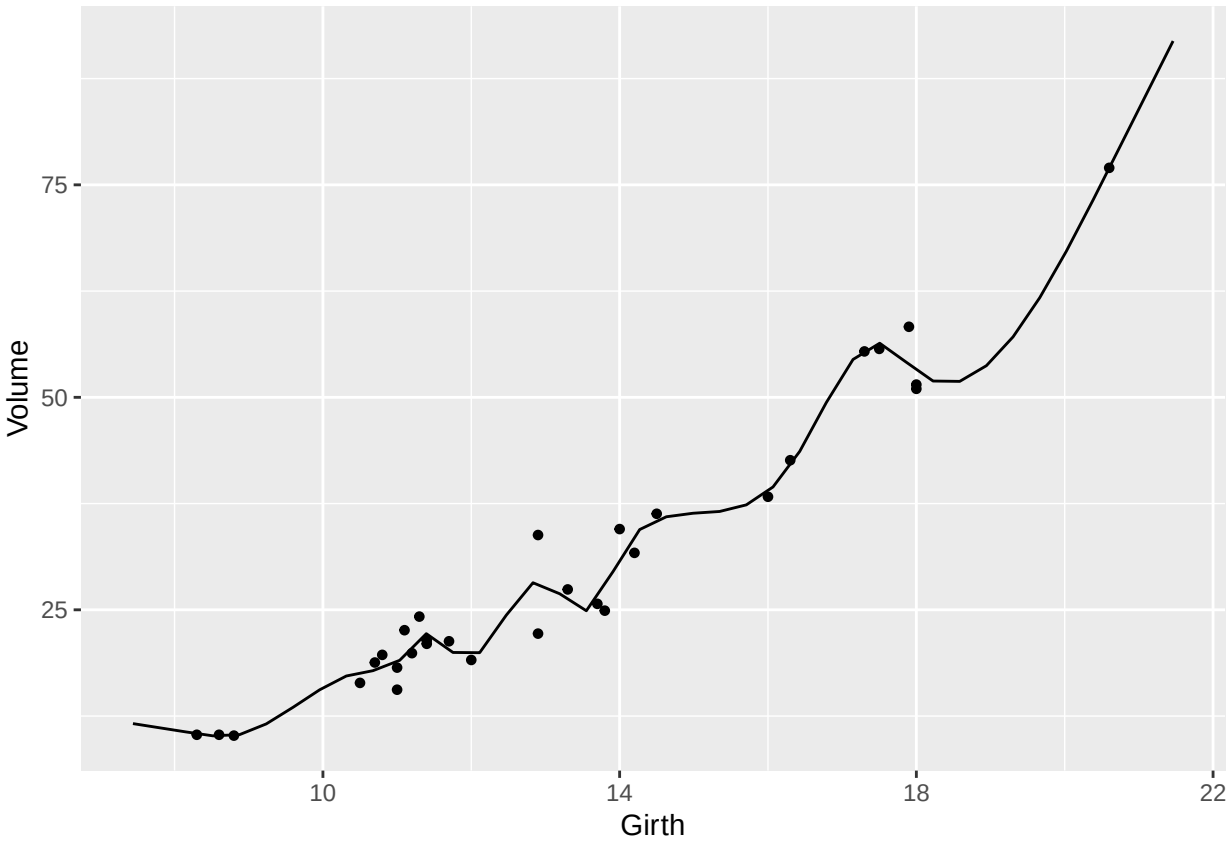

```
summary(m1)$r.squared
```

```
## [1] 0.96
```

```
m1_bad <- lm(Volume ~ poly(Girth, 15), data = trees)
plotModel(m1_bad)
```




```
library(splines)
m2 <- lm(Volume ~ ns(Girth, 15), data = trees)
plotModel(m2)
```



R^2 and correlation:

```
summary(m0)$r.squared
```

```
## [1] 0.94
```

```
summary(m1)$r.squared
```

```
## [1] 0.96
```

```
summary(m2)$r.squared
```

```
## [1] 0.98
```

```
# or: m0$fitted
cor(predict(m0, newdata = trees), trees$Volume)^2
```

```
## [1] 0.94
```

```
cor(predict(m1, newdata = trees), trees$Volume)^2
```

```
## [1] 0.96
```

```
cor(predict(m2, newdata = trees), trees$Volume)^2
```

```
## [1] 0.98
```

Mean squared error (MSE):

```r
mean((predict(m0, newdata = trees) - trees$Volume)^2)
```

```
## [1] 17
```

```r
mean((predict(m1, newdata = trees) - trees$Volume)^2)
```

```
## [1] 10
```

```r
mean((predict(m2, newdata = trees) - trees$Volume)^2)
```

```
## [1] 4.9
```

2.1 Reproducibility and random number generation

```r
rnorm(3)
```

```
## [1] -1.95  0.79 -1.11
```

```r
rnorm(3)
```

```
## [1]  0.13 -1.02 -2.01
```

```r
rnorm(3)
```

```
## [1] -0.637 -0.082  0.671
```

```r
set.seed(1)
rnorm(3)
```

```
## [1] -0.63  0.18 -0.84
```

```r
set.seed(1)
rnorm(3)
```

```
## [1] -0.63  0.18 -0.84
```

2.2 Out-of-sample error

```r
nrow(trees)
```

```
## [1] 31
```

```r
set.seed(1)
train_idx <- sample(x = seq_len(nrow(trees)), size = 20, replace = FALSE)
trees_train <- trees[train_idx, ]
nrow(trees_train)
```

```
## [1] 20
```

```r
trees_test <- trees[-train_idx, ]
nrow(trees_test)
```

```
## [1] 11
```

Using `trees_train` for training:

```r
m0_train <- lm(Volume ~ Girth, data = trees_train)
m1_train <- lm(Volume ~ poly(Girth, 2), data = trees_train)
m2_train <- lm(Volume ~ ns(Girth, 15), data = trees_train)
```

Using `trees_test` for testing ("out-of-sample" error):

```r
cor(predict(m0_train, newdata = trees_test), trees_test$Volume)^2
```

```
## [1] 0.98
```

```r
cor(predict(m1_train, newdata = trees_test), trees_test$Volume)^2
```

```
## [1] 0.97
```

```r
cor(predict(m2_train, newdata = trees_test), trees_test$Volume)^2
```

```
## [1] 0.016
```

```r
mean((predict(m0_train, newdata = trees_test) - trees_test$Volume)^2)
```

```
## [1] 40
```

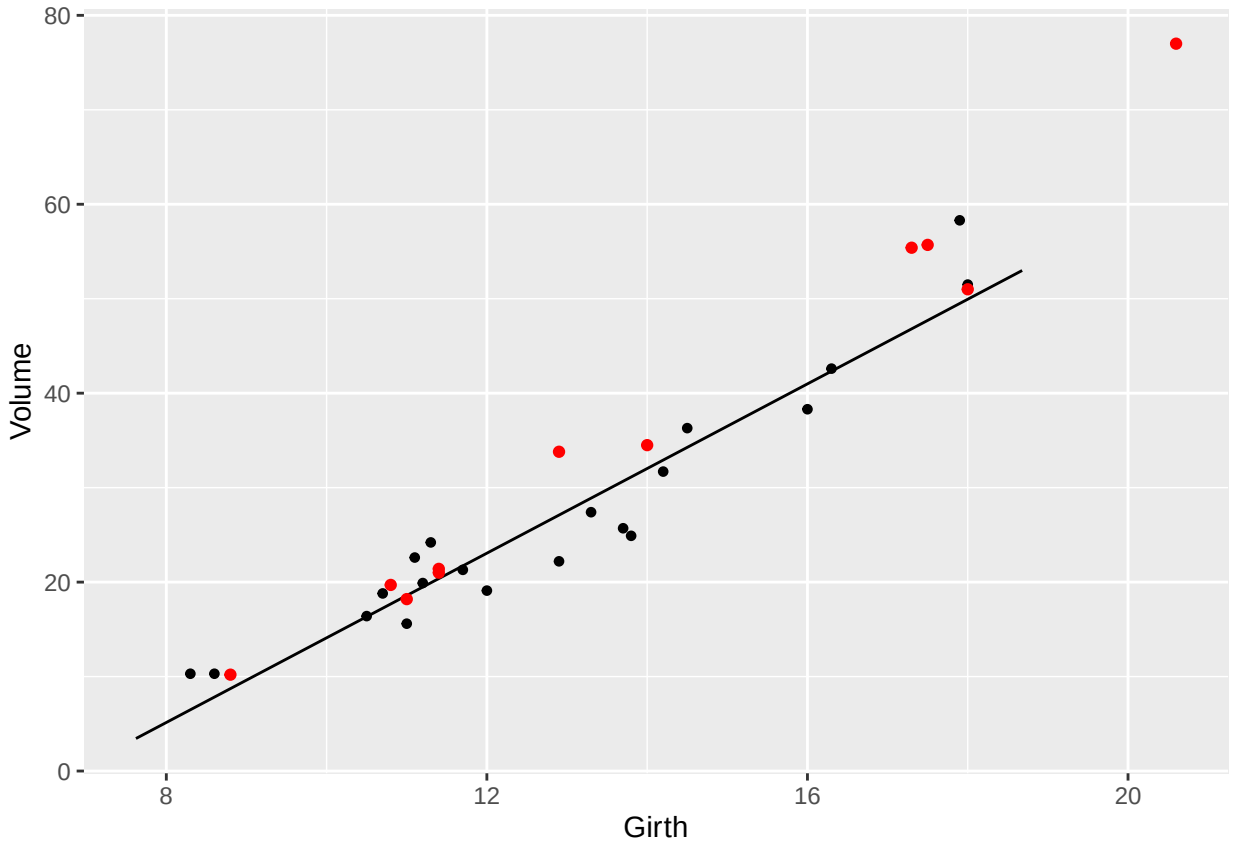
```r
mean((predict(m1_train, newdata = trees_test) - trees_test$Volume)^2)
```

```
## [1] 17
```

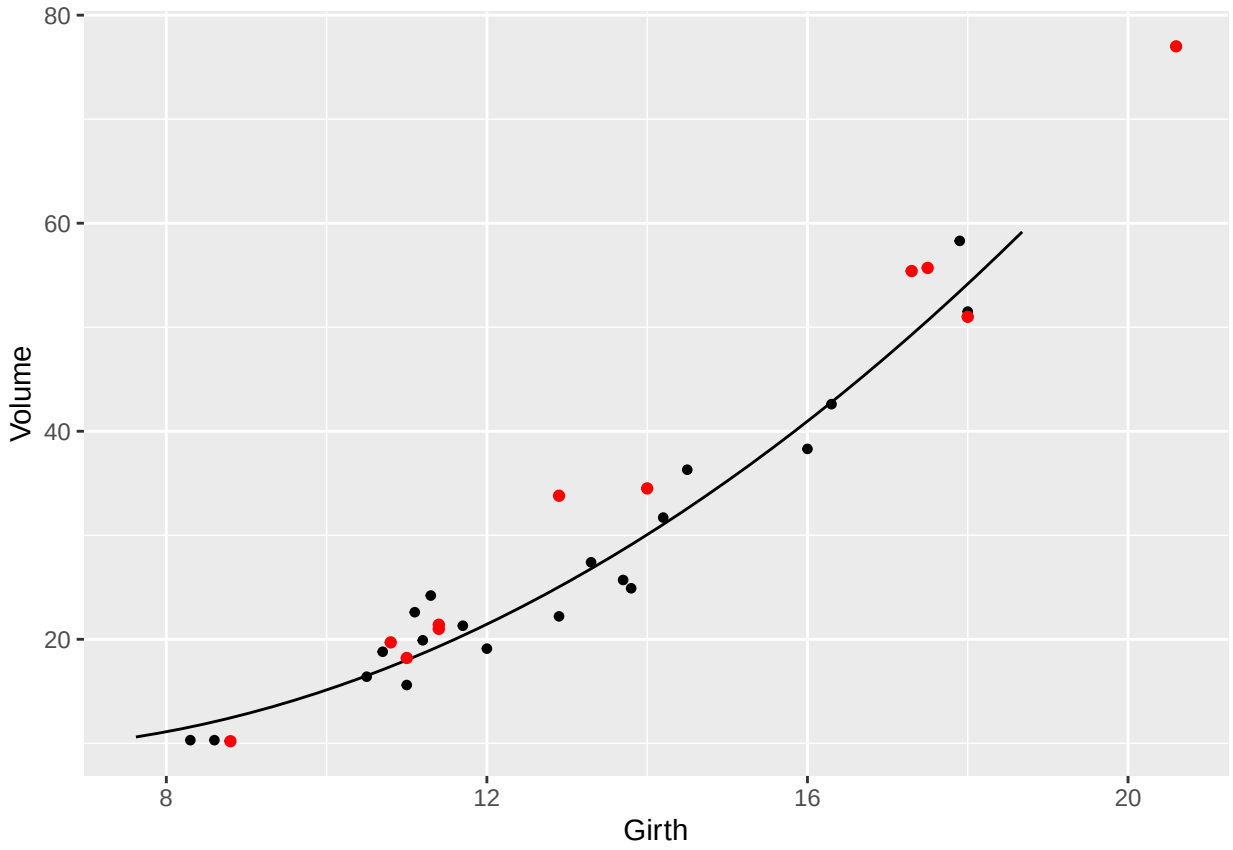
```r
mean((predict(m2_train, newdata = trees_test) - trees_test$Volume)^2)
```

```
## [1] 2074
```

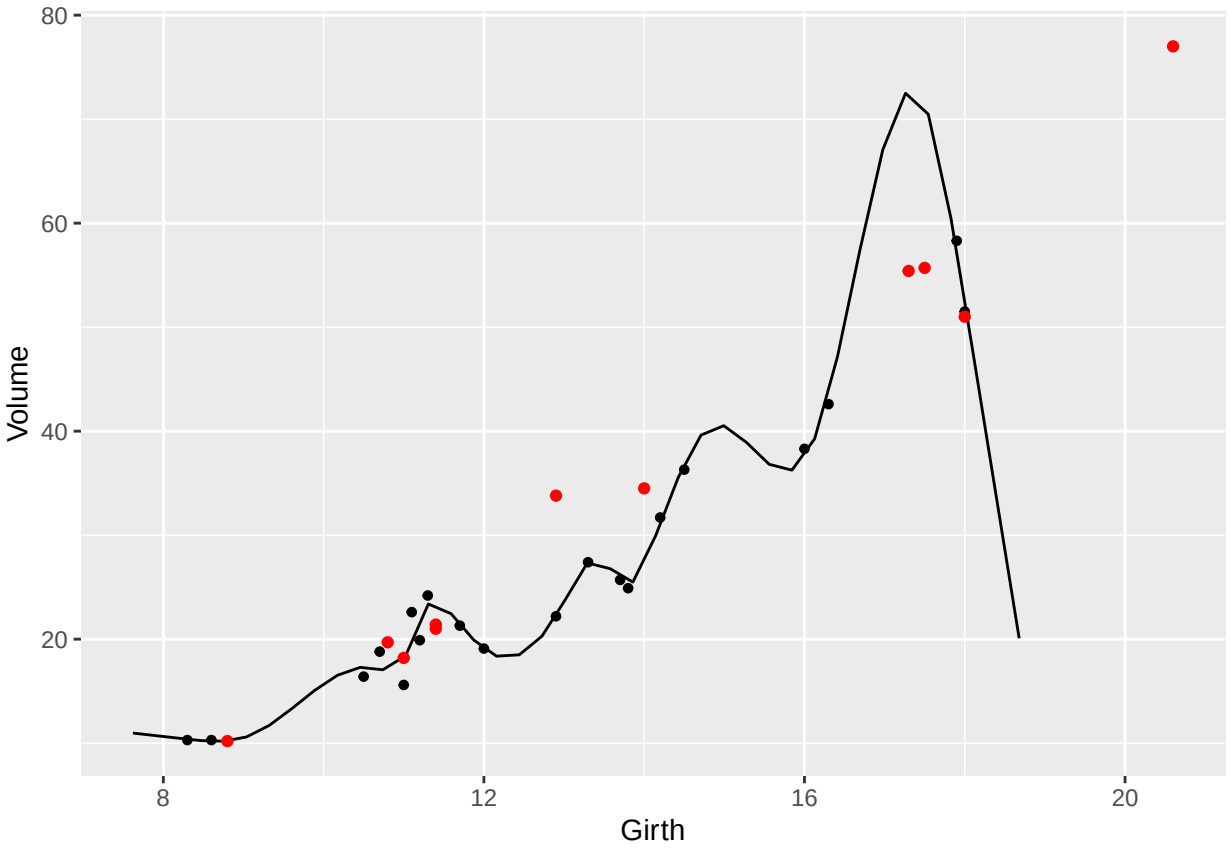
```r
plotModel(m0_train) +
  geom_point(aes(Girth, Volume), data = trees_test, color = "red")
```




```
plotModel(m1_train) +
  geom_point(aes(Girth, Volume), data = trees_test, color = "red")
```

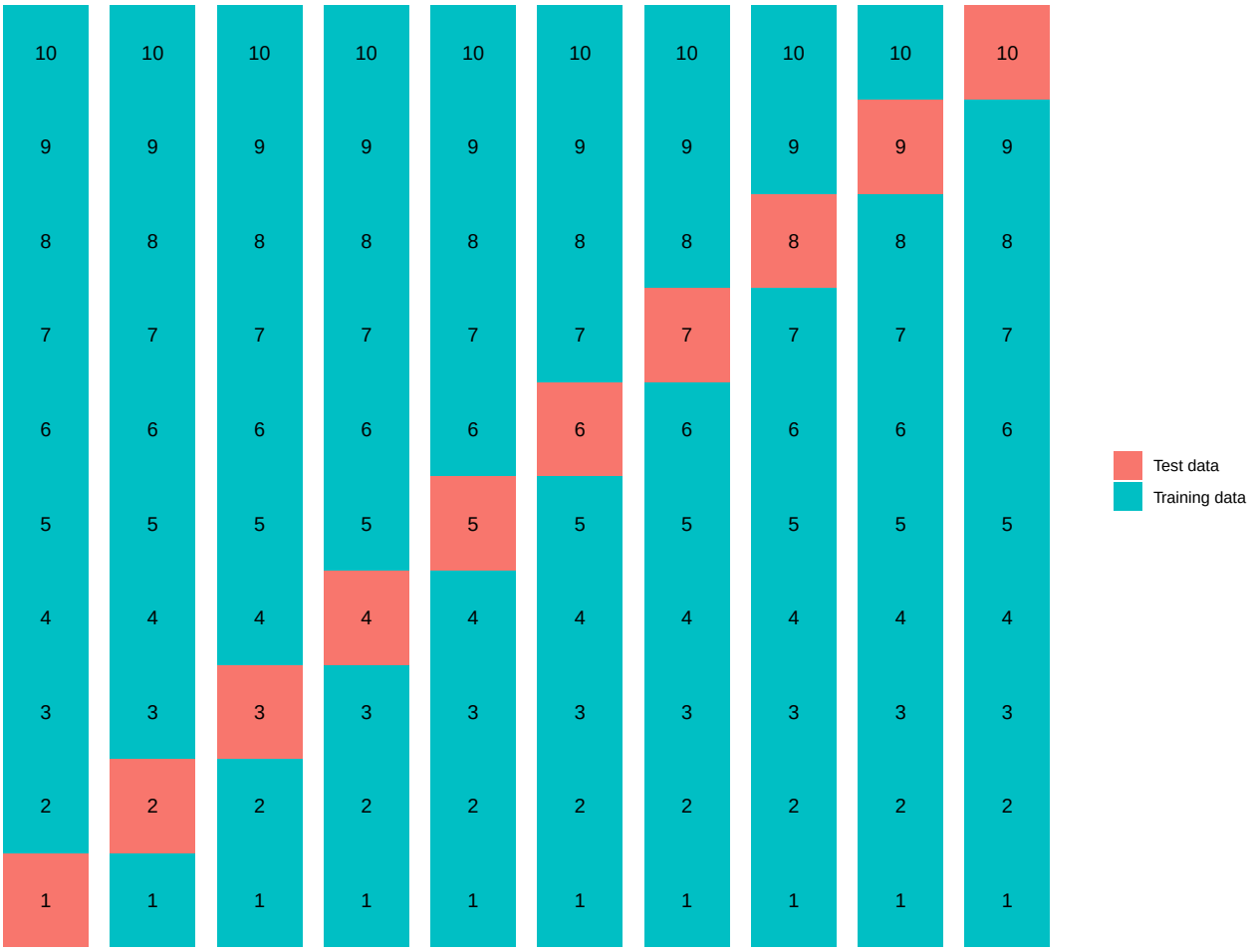



```
plotModel(m2_train) +
  geom_point(aes(Girth, Volume), data = trees_test, color = "red")
```



3 Cross-validation

- Repeat out-of-sample error estimation multiple times
- Resampling without replacement (partitioning of data)
- k-fold cross validation for $k = 10$:



```
nrow(trees)
```

```
## [1] 31
```

```
seq_len(nrow(trees))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31
```

- Number of folds depends on size of dataset
- 4-fold cross validation.

Divide data into folds:

2,11,13,14,20,21,25
3,6,7,10,17,24,29,30
5,8,9,12,19,23,26,28
1,4,15,16,18,22,27,31

"Rotate" which folds are training data and which one is test data:


2,11,13,14,20,21,25
3,6,7,10,17,24,29,30
5,8,9,12,19,23,26,28
1,4,15,16,18,22,27,31
2,11,13,14,20,21,25
3,6,7,10,17,24,29,30
5,8,9,12,19,23,26,28
1,4,15,16,18,22,27,31
2,11,13,14,20,21,25
3,6,7,10,17,24,29,30
5,8,9,12,19,23,26,28
1,4,15,16,18,22,27,31
2,11,13,14,20,21,25
3,6,7,10,17,24,29,30
5,8,9,12,19,23,26,28
1,4,15,16,18,22,27,31
Test data
Training data

3.1 Repeated CV

2,11,13,14,20,21,25	7,9,11,15,21,27,28	1,3,5,6,9,12,24	3,11,12,14,19,23,24	4,8,14,15,17,20,27
3,6,7,10,17,24,29,30	1,6,10,12,14,16,22,24	7,10,13,17,22,23,28,30	1,2,4,8,9,16,29,30	2,3,10,12,13,23,25,28
5,8,9,12,19,23,26,28	2,3,5,13,18,19,25,29	8,11,18,19,21,25,26,27	5,6,10,13,17,21,27,28	1,7,9,16,18,22,24,30
1,4,15,16,18,22,27,31	4,8,17,20,23,26,30,31	2,4,14,15,16,20,29,31	7,15,18,20,22,25,26,31	5,6,11,19,21,26,29,31

```
library(caret)
```

https://cran.r-project.org/package=caret https://topepo.github.io/caret/

Setting up cross validation:

```
train_control <- trainControl(method = "repeatedcv",
                              # k-fold CV
                              number = 4,
                              # repeated ten times
                              repeats = 10)
```

```
set.seed(1)
m0_cv <- train(Volume ~ Girth, data = trees, trControl = train_control, method = "lm")
m1_cv <- train(Volume ~ poly(Girth, 2), data = trees, trControl = train_control, method = "lm")
m2_cv <- train(Volume ~ ns(Girth, 15), data = trees, trControl = train_control, method = "lm")
```

```
m0_cv
```

```
## Linear Regression
##
## 31 samples
##  1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (4 fold, repeated 10 times)
## Summary of sample sizes: 24, 23, 23, 23, 23, 23, ...
## Resampling results:
##
##   RMSE  Rsquared  MAE
##   4.5   0.95      3.7
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

- **RMSE** is root mean squared error

```
m0_cv$resample
```

```
##    RMSE Rsquared MAE    Resample
## 1   4.7     0.91 4.2 Fold1.Rep01
## 2   4.2     0.91 3.6 Fold2.Rep01
## 3   5.3     0.99 3.3 Fold3.Rep01
## 4   4.7     0.96 4.2 Fold4.Rep01
## 5   3.8     0.93 3.2 Fold1.Rep02
## 6   5.6     0.93 4.6 Fold2.Rep02
## 7   2.7     0.97 2.0 Fold3.Rep02
## 8   5.0     0.95 4.7 Fold4.Rep02
## 9   3.5     0.97 2.7 Fold1.Rep03
## 10  4.7     0.93 4.2 Fold2.Rep03
## 11  4.6     0.96 3.6 Fold3.Rep03
## 12  5.2     0.96 4.2 Fold4.Rep03
## 13  4.4     0.93 3.6 Fold1.Rep04
## 14  5.3     0.95 4.2 Fold2.Rep04
## 15  3.5     0.96 2.7 Fold3.Rep04
## 16  4.1     0.95 3.4 Fold4.Rep04
## 17  3.7     0.98 3.0 Fold1.Rep05
## 18  4.4     0.92 3.7 Fold2.Rep05
## 19  6.2     0.98 4.6 Fold3.Rep05
## 20  4.8     0.92 4.2 Fold4.Rep05
## 21  5.8     0.96 4.4 Fold1.Rep06
## 22  4.8     0.92 4.2 Fold2.Rep06
## 23  3.7     0.93 3.1 Fold3.Rep06
## 24  3.0     0.96 2.5 Fold4.Rep06
## 25  5.8     0.95 4.8 Fold1.Rep07
## 26  3.5     0.94 2.8 Fold2.Rep07
## 27  3.8     0.92 3.0 Fold3.Rep07
## 28  4.3     0.97 3.9 Fold4.Rep07
## 29  5.6     0.98 3.8 Fold1.Rep08
## 30  4.5     0.93 4.4 Fold2.Rep08
## 31  5.0     0.94 4.2 Fold3.Rep08
## 32  3.8     0.97 2.8 Fold4.Rep08
## 33  3.8     0.97 3.3 Fold1.Rep09
## 34  4.7     0.94 4.2 Fold2.Rep09
```

```
## 35  4.2     0.96 3.7 Fold3.Rep09
## 36  5.3     0.95 3.8 Fold4.Rep09
## 37  3.5     0.97 3.1 Fold1.Rep10
## 38  4.7     0.90 4.1 Fold2.Rep10
## 39  4.2     0.90 3.3 Fold3.Rep10
## 40  4.9     0.96 3.5 Fold4.Rep10
```

```
mean(m0_cv$resample$RMSE)
```

```
## [1] 4.5
```

```
m0_cv$results$RMSE
```

```
## [1] 4.5
```

```
m0_cv$results$RMSE
```

```
## [1] 4.5
```

```
m1_cv$results$RMSE
```

```
## [1] 3.4
```

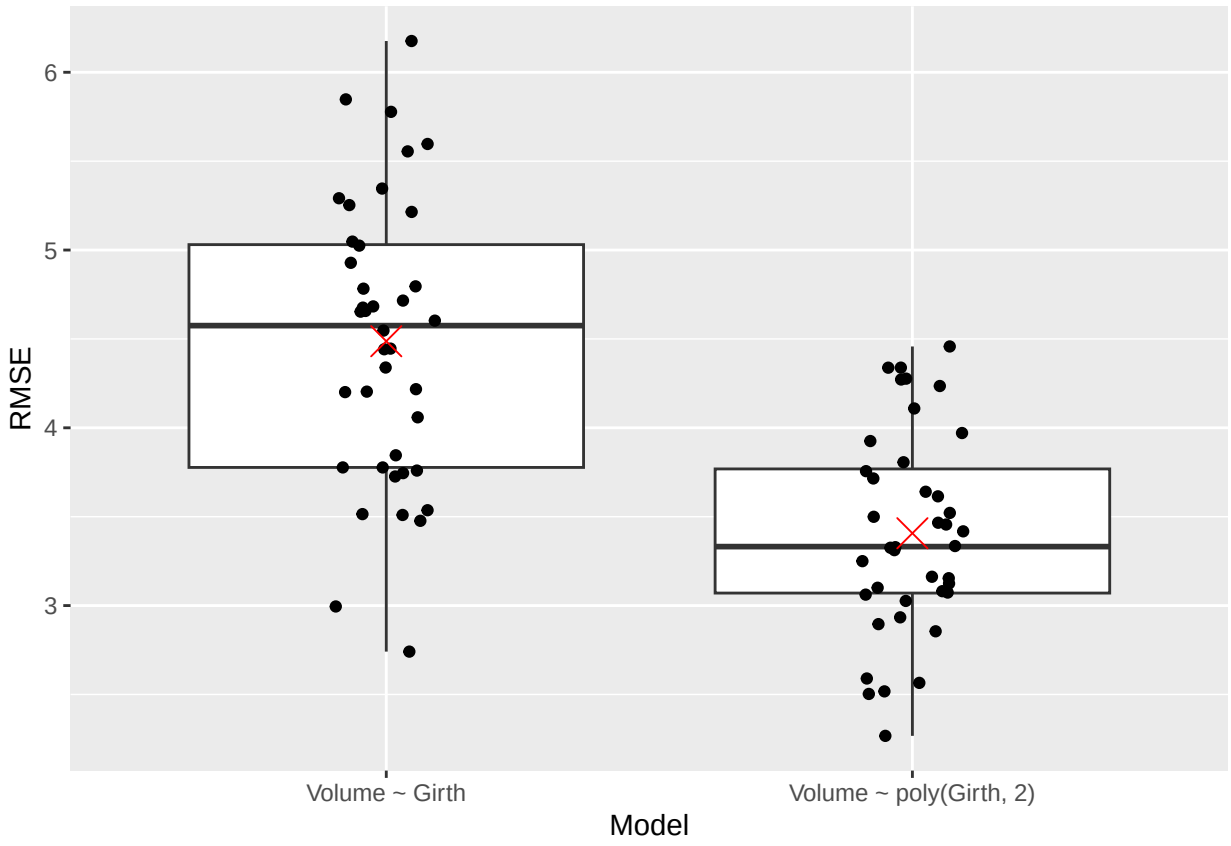
```
m2_cv$results$RMSE
```

```
## [1] 112
```

Extra (with `tidyverse`, remember `library(tidyverse)`):

```
d <- bind_rows(
  m0_cv$resample |> mutate(Model = "Volume ~ Girth"),
  m1_cv$resample |> mutate(Model = "Volume ~ poly(Girth, 2)")
)
d_mean <- d |>
  group_by(Model) |>
  summarise(RMSE = mean(RMSE), .groups = "drop")

ggplot(d, aes(Model, RMSE)) +
  geom_boxplot() +
  geom_jitter(height = 0, width = 0.1) +
  geom_point(data = d_mean, color = "red", shape = 4, size = 5)
```



4 Non-parametric bootstrap

- Resampling data with replacement (some data is used, some not)
- Mimic a new sample

```
m0 <- lm(Volume ~ Girth, data = trees)
coef(m0)
```

```
## (Intercept)       Girth
##       -36.9         5.1
```

Preparing bootstrap:

```
model_coef <- function(index){
  coef(lm(Volume ~ Girth, data = trees, subset = index))
}
model_coef(1:nrow(trees))
```

```
## (Intercept)       Girth
##       -36.9         5.1
```

```
model_coef(c(rep(1, 10), 11:nrow(trees)))
```

```
## (Intercept)       Girth
##       -30.6         4.7
```

```
set.seed(1)
model_coef(sample(1:nrow(trees), replace = TRUE))
```

```
## (Intercept)       Girth
```

```
##       -29.8         4.5
```

```
model_coef(sample(1:nrow(trees), replace = TRUE))
```

```
## (Intercept)       Girth
##       -34.4         4.8
```

```
set.seed(1)
bootstrap_coefs <- replicate(1000, {
  model_coef(sample(1:nrow(trees), replace = TRUE))
})
bootstrap_coefs[, 1:10]
```

```
##              [,1]  [,2]  [,3]  [,4] [,5]  [,6]  [,7]  [,8]  [,9] [,10]
## (Intercept) -29.8 -34.4 -34.3 -42.6  -36 -35.0 -36.5 -39.4 -34.0 -32.1
## Girth         4.5   4.8   4.8   5.5    5   4.9   5.1   5.2   4.8   4.7
```

```
dim(bootstrap_coefs)
```

```
## [1]    2 1000
```

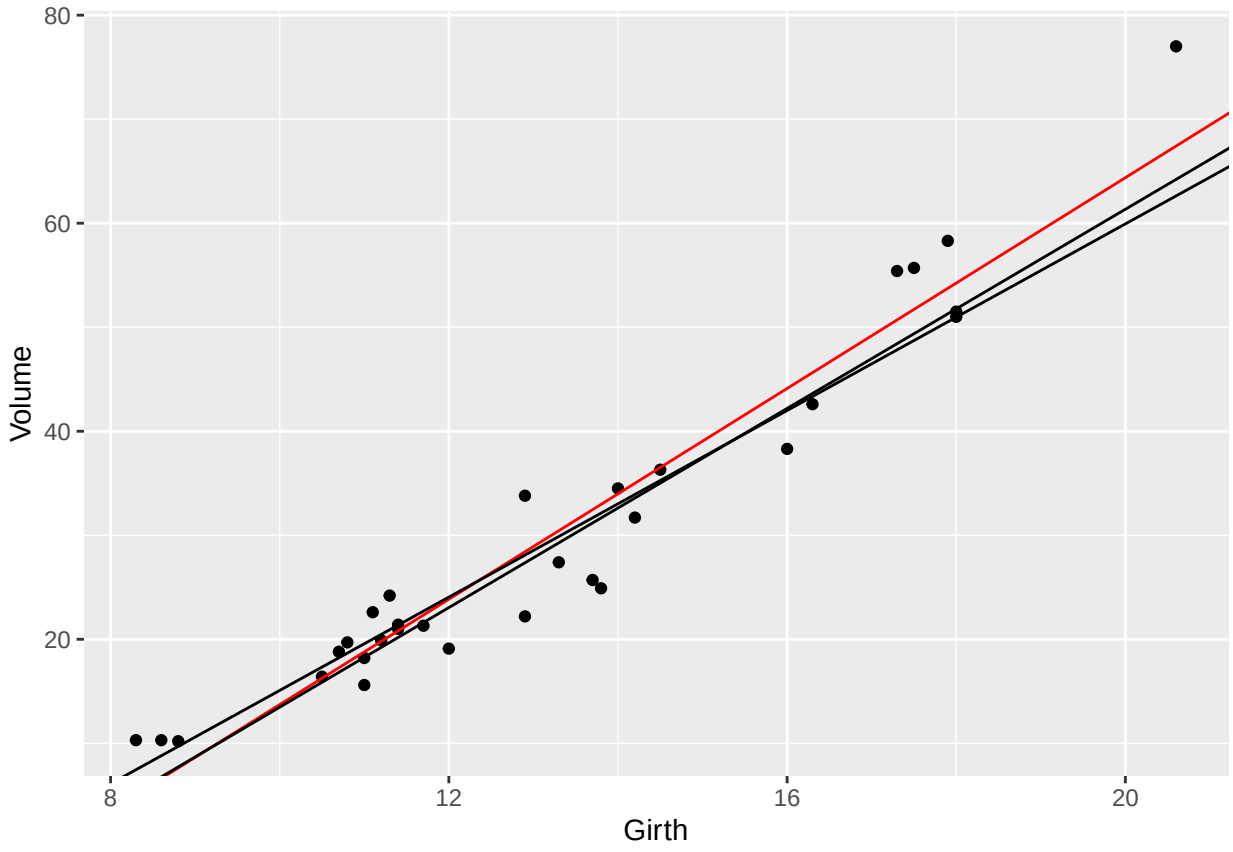
```
apply(bootstrap_coefs, 1, sd)
```

```
## (Intercept)       Girth
##        3.98         0.32
```

```
coef(summary(m0))
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -36.9       3.37     -11  7.6e-12
## Girth            5.1       0.25      20  8.6e-19
```

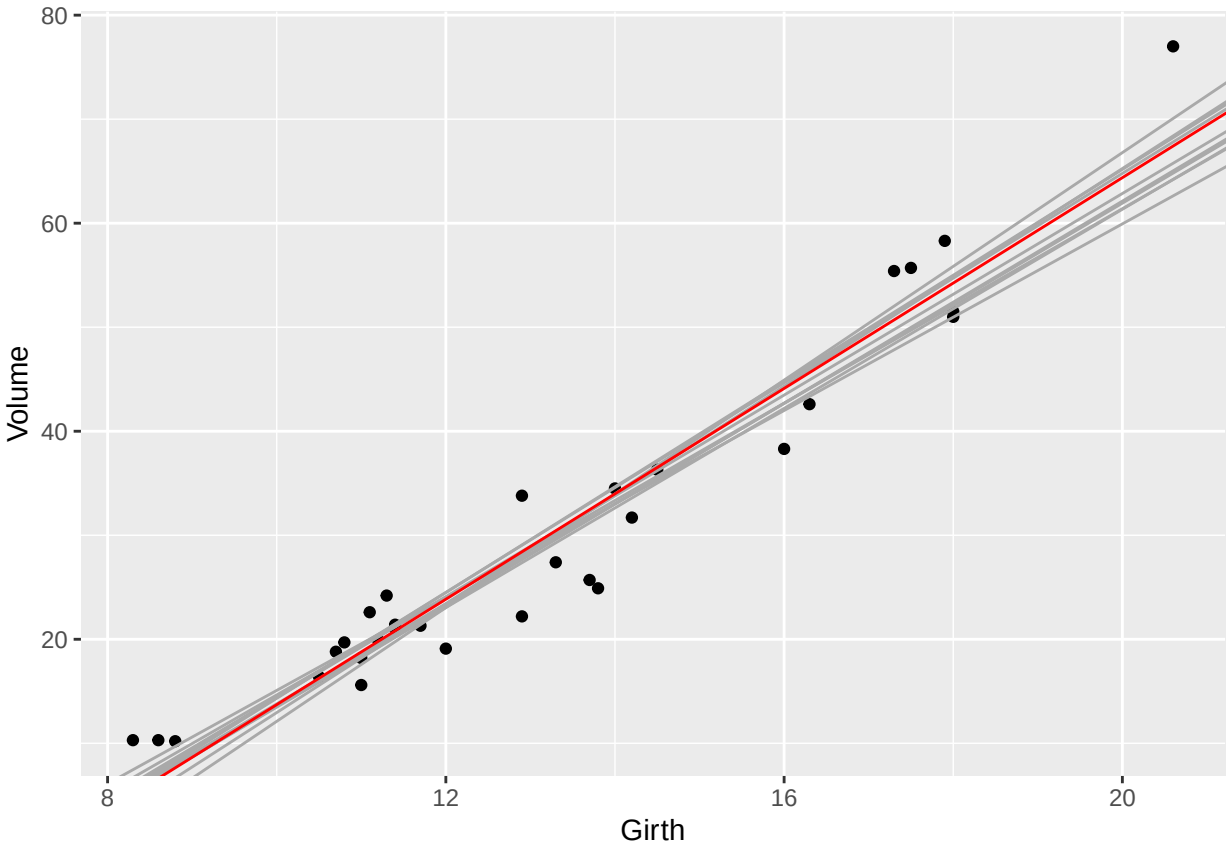
```
gf_point(Volume ~ Girth, data = trees) %>%
  gf_abline(intercept = coef(m0)[1], slope = coef(m0)[2], color = "red") %>%
  gf_abline(intercept = bootstrap_coefs[1, 1], slope = bootstrap_coefs[2, 1], color = "black") %>%
  gf_abline(intercept = bootstrap_coefs[1, 2], slope = bootstrap_coefs[2, 2], color = "black")
```




```
p <- gf_point(Volume ~ Girth, data = trees)

#for (i in 1:ncol(bootstrap_coefs)) {
for (i in 1:10) {
  p <- p %>%
    gf_abline(intercept = bootstrap_coefs[1, i], slope = bootstrap_coefs[2, i], color = "darkgrey")
}

p %>% gf_abline(intercept = coef(m0)[1], slope = coef(m0)[2], color = "red")
```



Also see the `boot` package: https://cran.r-project.org/package=boot

```
library(boot)
```

```
model_coef_boot <- function(data, index){
  coef(lm(Volume ~ Girth, data = data, subset = index))
}
```

```
set.seed(1)
b <- boot(trees, model_coef_boot, R = 1000)
b
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = trees, statistic = model_coef_boot, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    -36.9   0.372        4.05
## t2*      5.1  -0.038        0.33
```

```
coef(summary(m0))
```

```
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)    -36.9       3.37     -11  7.6e-12
## Girth            5.1       0.25      20  8.6e-19
```

5 Parametric bootstrap by resampling residuals

Resampling residuals with replacement:

```
library(boot)
```

```
m0 <- lm(Volume ~ Girth, data = trees)
```

```
set.seed(1)
parbootstrap_coefs <- replicate(1000, {
  new_y <- m0$fitted.values + sample(m0$residuals, replace = TRUE)
  coef(lm(new_y ~ trees$Girth))
})
parbootstrap_coefs[, 1:10]
```

```
##              [,1]  [,2]  [,3] [,4] [,5]  [,6]  [,7]  [,8]  [,9] [,10]
## (Intercept) -38.4 -38.5 -34.0  -36  -35 -31.7 -38.6 -40.8 -30.5 -38.8
## trees$Girth   5.2   5.1   4.8    5    5   4.6   5.2   5.3   4.6   5.2
```

```
apply(parbootstrap_coefs, 1, sd)
```

```
## (Intercept) trees$Girth
##        3.31        0.24
```

```
apply(bootstrap_coefs, 1, sd)
```

```
## (Intercept)       Girth
##        3.98        0.32
```

```
coef(summary(m0))
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -36.9       3.37     -11  7.6e-12
## Girth            5.1       0.25      20  8.6e-19
```

6 The probability density/mass function

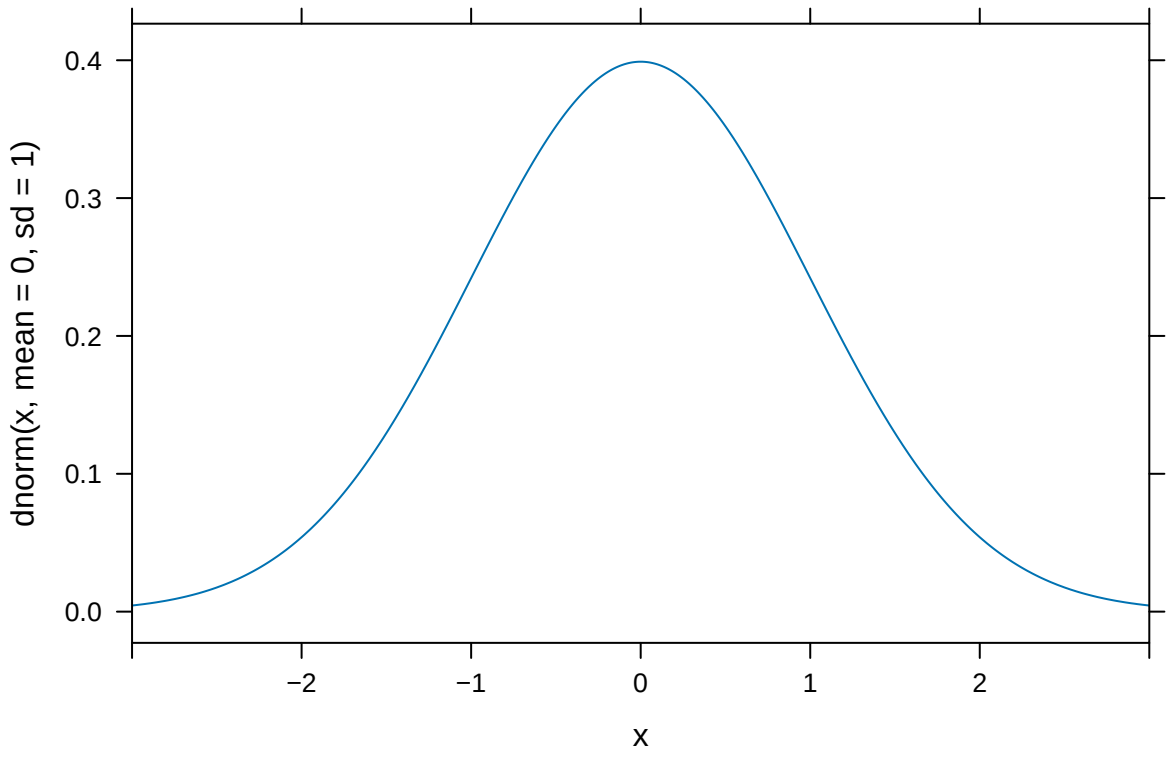
$$X \sim N(\mu, \sigma)$$

Assumes μ and σ are some fixed values:

$$f(x) = f(X = x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

E.g. for $\mu = 0$ and $\sigma = 1$:

```
plotFun(dnorm(x, mean = 0, sd = 1) ~ x, xlim = c(-3, 3))
```



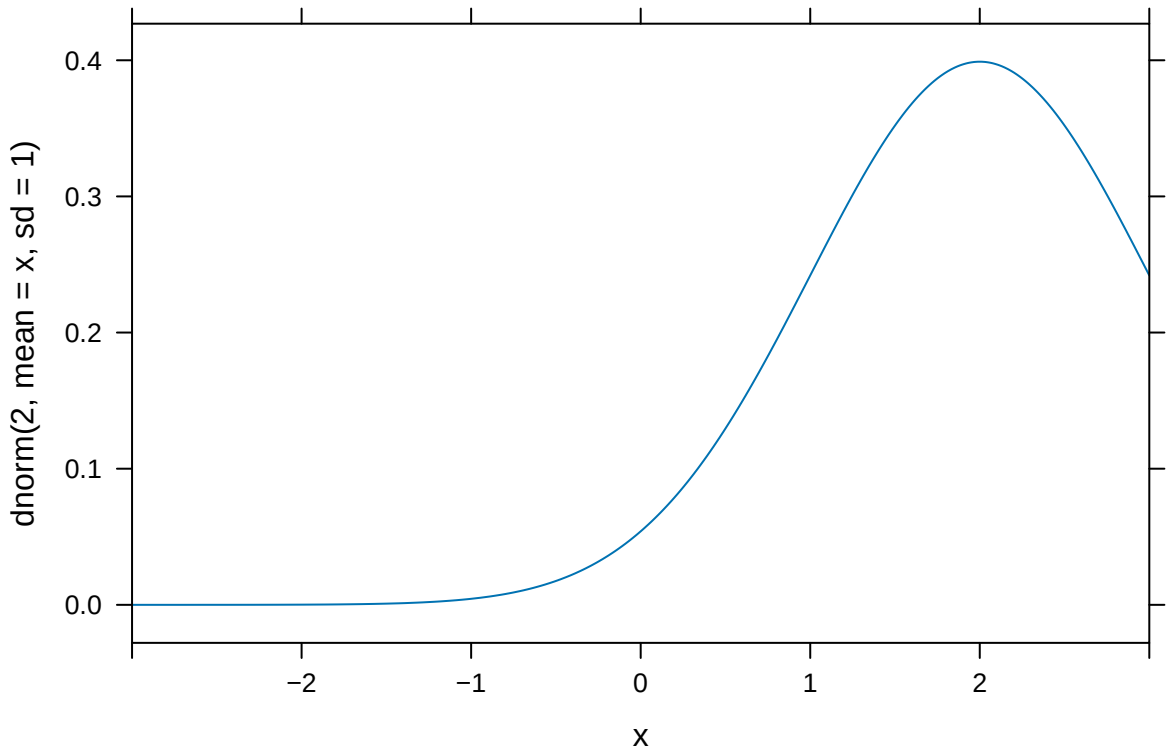
7 The likelihood function for a single observation

What if instead the data is fixed and we let μ and σ vary?

$$L(\mu, \sigma) = L(\mu, \sigma; X = x) = f(X = x; \mu, \sigma)$$

E.g. assume that $X = 2$ was observed:

```
plotFun(dnorm(2, mean = x, sd = 1) ~ x, xlim = c(-3, 3))
```



The value of μ that gives the highest probability (density) of observing $X = 2$ is $\mu = 2$.

8 The likelihood function for n observations

Assume independence for n observations, then

$$L(\mu, \sigma) = L(\mu, \sigma; X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} L(\mu, \sigma; X_i) = \prod_{i=1}^{n} L(\mu, \sigma; X_i). \tag{3}$$

- Maximum likelihood estimation (MLE):
  - Find the values of μ and σ that gives the largest value of L for fixed X's.
- Would you rather differentiate products or sums?

Do you know a function that turns products into sums?

$$\log(a \cdot b) = \log(a) + \log(b)$$

Assume independence for n observations, then

$$L(\mu, \sigma) = L(\mu, \sigma; X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} L(\mu, \sigma; X_i) = \prod_{i=1}^{n} L(\mu, \sigma; X_i) \tag{4}$$

$$l(\mu, \sigma) = \log L(\mu, \sigma) = \log L(\mu, \sigma; X_1, X_2, \ldots, X_n) \tag{5}$$

$$= \log \left(\prod_{i=1}^{n} L(\mu, \sigma; X_i) \right) \tag{6}$$

$$= \sum_{i=1}^{n} \log L(\mu, \sigma; X_i) \tag{7}$$

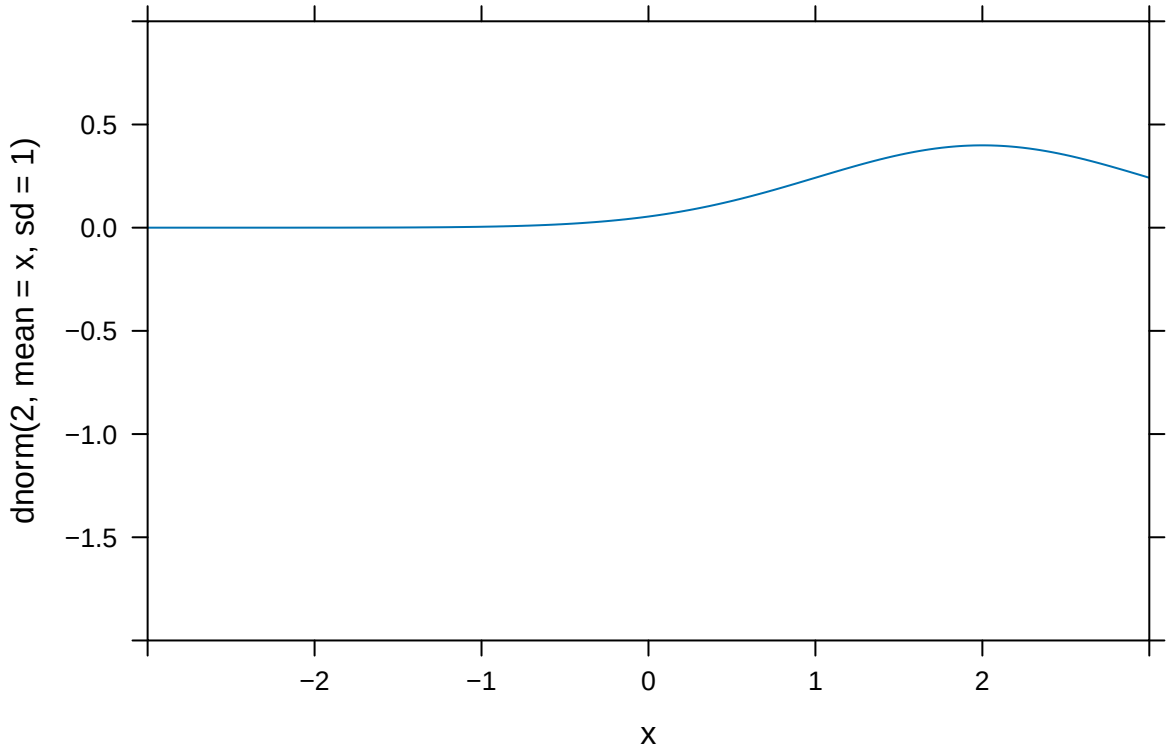
$$= \sum_{i=1}^{n} l(\mu, \sigma; X_i). \tag{8}$$

Note that $\log$ is a monotonic (increasing) function, so maximum of $l = \log L$ is the same as that of L.

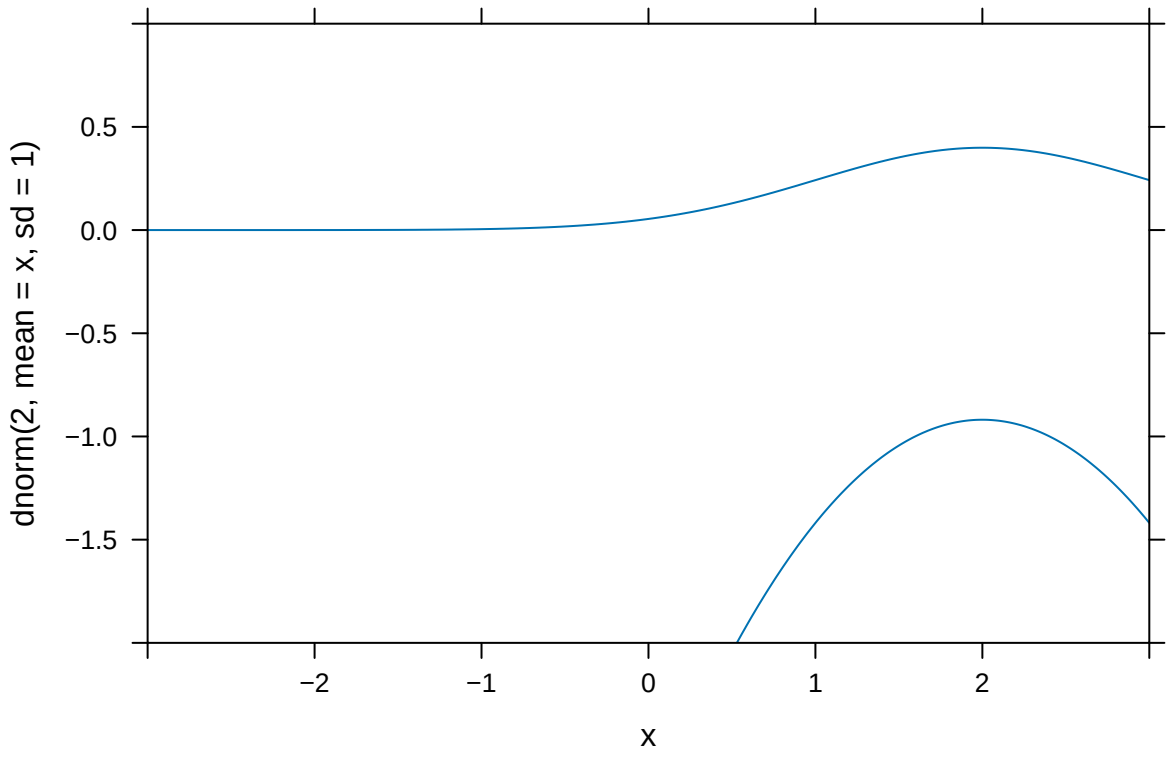
$$L(\mu, \sigma) = L(\mu, \sigma; X = x) = f(X = x; \mu, \sigma) \tag{9}$$

$$l(\mu, \sigma; X) = \log L(\mu, \sigma; X) = \log f(X; \mu, \sigma) \tag{10}$$

```
plotFun(dnorm(2, mean = x, sd = 1) ~ x, xlim = c(-3, 3), ylim = c(-2, 1));
```




```
plotFun(log(dnorm(2, mean = x, sd = 1)) ~ x, xlim = c(-3, 3), add = TRUE)
```

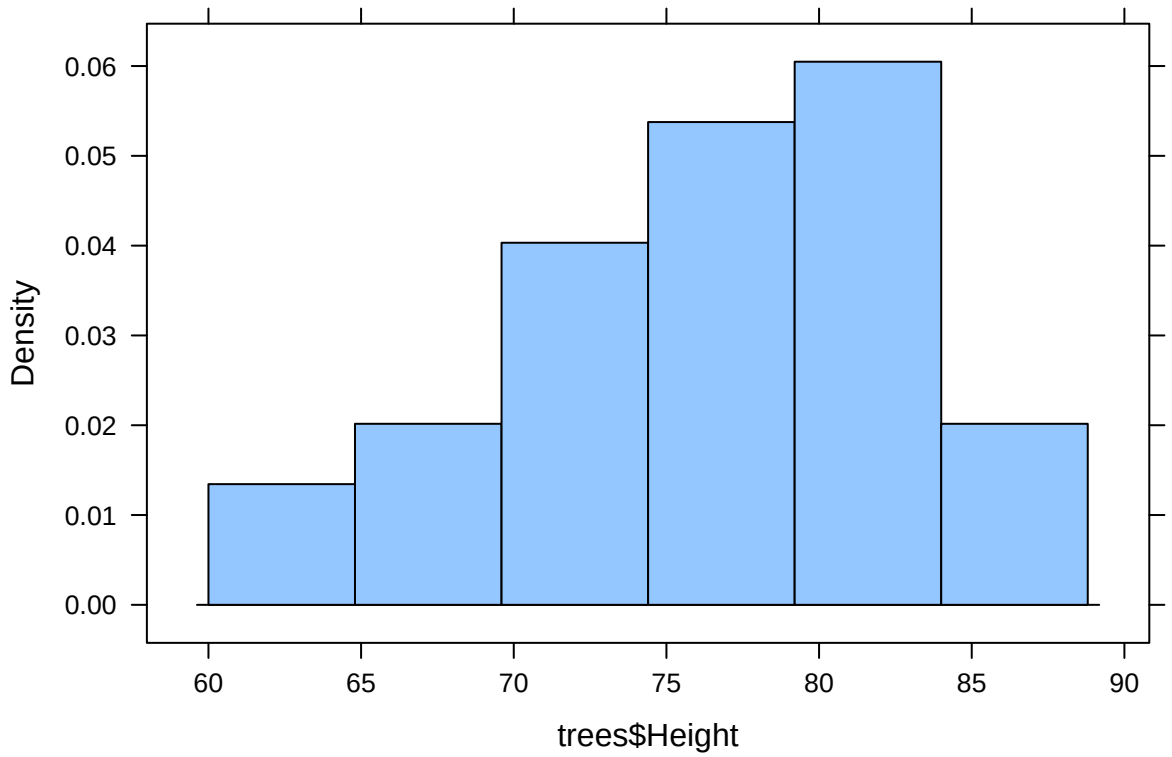


9 Example

```
trees <- read.delim("https://asta.math.aau.dk/datasets?file=trees.txt")
head(trees)
```

```
##   Girth Height Volume
## 1   8.3     70     10
## 2   8.6     65     10
## 3   8.8     63     10
## 4  10.5     72     16
## 5  10.7     81     19
## 6  10.8     83     20
```

```
histogram(trees$Height)
```



If Height is normally distributed, what are the parameters (mean and standard deviation)?

```
sd(trees$Height)
```

```
## [1] 6.4
```

```
single_loglik <- function(mu) {
  sum(log(dnorm(trees$Height, mean = mu, sd = 6)))
}
single_loglik(65)
```

```
## [1] -153
```

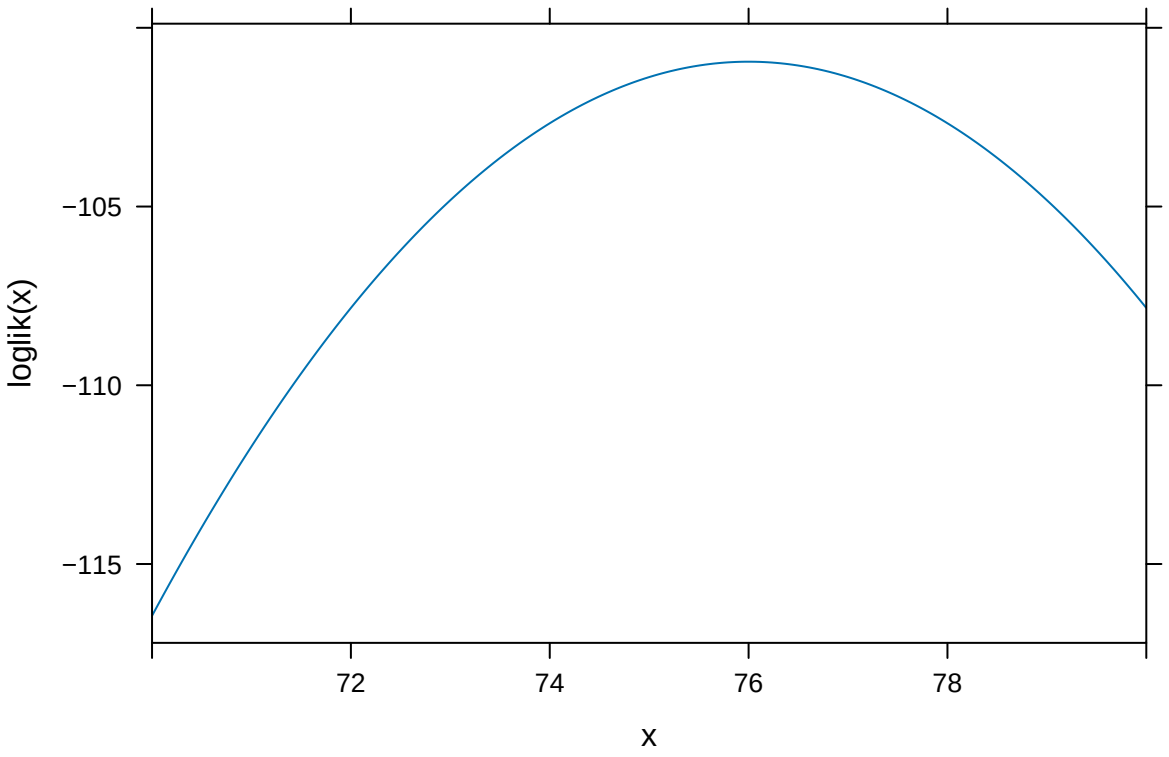
```
single_loglik(c(60, 65, 70))
```

```
## [1] -159
```

```
loglik <- Vectorize(single_loglik)
loglik(c(60, 65, 70))
```

```
## [1] -211 -153 -116
```

```
plotFun(loglik(x) ~ x, xlim = c(70, 80))
```



Maximum around 76.

```
optimise(single_loglik, interval = c(0, 100))
```

```
## $minimum
## [1] 4.6e-05
##
## $objective
## [1] -2588
```

```
optimise(single_loglik, interval = c(0, 100), maximum = TRUE)
```

```
## $maximum
## [1] 76
##
## $objective
## [1] -101
```

Maximum of l is minimum of $-l$.

```
single_loglik <- function(mu) {
  -sum(log(dnorm(trees$Height, mean = mu, sd = 6)))
}
optimise(single_loglik, interval = c(0, 100))
```

```
## $minimum
## [1] 76
##
## $objective
## [1] 101
```

Both parameters:

```
single_loglik <- function(pars) {
  mu <- pars[1]
  sigma <- pars[2]
  -sum(log(dnorm(trees$Height, mean = mu, sd = sigma)))
}
optim(c(1, 50), single_loglik)
```

```
## Warning in dnorm(trees$Height, mean = mu, sd = sigma): NaNs produced

## Warning in dnorm(trees$Height, mean = mu, sd = sigma): NaNs produced

## Warning in dnorm(trees$Height, mean = mu, sd = sigma): NaNs produced
```

```
## $par
## [1] 76.0  6.3
##
## $value
## [1] 101
##
## $counts
## function gradient
##       87       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
single_loglik <- function(pars) {
  mu <- pars[1]
  sigma <- exp(pars[2])
  -sum(log(dnorm(trees$Height, mean = mu, sd = sigma)))
}
mle <- optim(c(1, 50), single_loglik)
mle
```

```
## $par
## [1] 76.0  1.8
##
## $value
## [1] 101
##
## $counts
## function gradient
##      113       NA
##
## $convergence
## [1] 0
##
## $message
```

```
## NULL
```

```
exp(mle$par[2])
```

```
## [1] 6.3
```

```
mean(trees$Height)
```

```
## [1] 76
```

```
sd(trees$Height)
```

```
## [1] 6.4
```

```
n <- length(trees$Height)
sd(trees$Height)
```

```
## [1] 6.4
```