```r
set.seed(1234)
n = 50
x = runif(n, 0, 1)
noise = rnorm(n, 0, 0.2)
y = rep(1,n) + x^2 +noise
```

```r
data = as.data.frame(cbind(x, y))
data$x2 = data$x^2
model = lm(y ~ x2 + x, data =data)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x2 + x, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42676 -0.15765 -0.04785  0.11788  0.57410
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.8487     0.1054   8.051 2.14e-10 ***
## x2            0.5004     0.4840   1.034    0.306
## x             0.5330     0.4907   1.086    0.283
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2198 on 47 degrees of freedom
## Multiple R-squared:  0.6156, Adjusted R-squared:  0.5993
## F-statistic: 37.64 on 2 and 47 DF,  p-value: 1.747e-10
```

Write a function for taking a sample and delivering an estimate for $\hat{}$x0.

```r
bootstrap = function(data){
  choices = sample(1:n, size=n, replace=T)
  bsdata = as.data.frame(cbind(data$x[choices], data$x2[choices], data$y[choices]))
  colnames(bsdata) = c("x", "x2", "y")
  bs_model = lm(y ~ x2 + x, data=bsdata)
  w0 = bs_model$coefficients[1]
  w2 = bs_model$coefficients[2]
  w1 = bs_model$coefficients[3]
  x0_hat = (-w1 + sqrt(w1^2 - 4*w2*(w0-1.3))) / (2*w2)
  return(x0_hat)
}
```

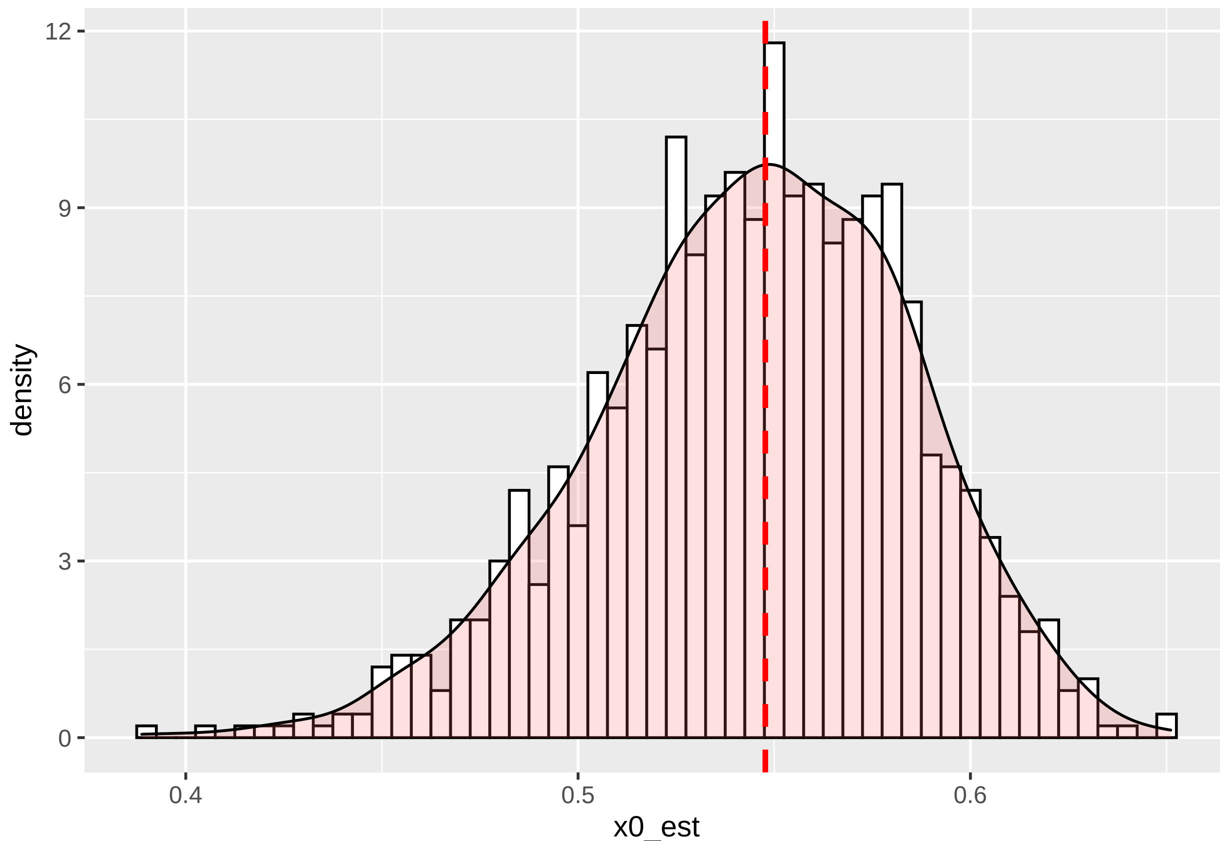
Generate a sample from this model, and compute $\hat{}$x0 for this sample.

Simulate a 1000 realizations from this model, to approximate the sampling distribution of $\hat{}$x0.

```r
x0_est = vector()
for (i in 1:1000){
  n = 50
  x = runif(n, 0, 1)
  noise = rnorm(n, 0, 0.2)
  y = rep(1,n) + x^2 +noise
  data = as.data.frame(cbind(x, y))
```

```
  data$x2 = data$x^2
  model = lm(y ~ x2 + x, data = data)
  w0 = model$coefficients[1]
  w2 = model$coefficients[2]
  w1 = model$coefficients[3]
  x0_hat = (-w1 + sqrt(w1^2 - 4*w2*(w0-1.3))) / (2*w2)
  x0_est[i] = x0_hat
}

library(ggplot2)
x0_est = as.data.frame(x0_est)
ggplot(x0_est, aes(x=x0_est)) +
    geom_histogram(aes(y=..density..),
                   binwidth=0.005,
                   colour="black", fill="white") +
    geom_density(alpha=.2, fill="#FF6666")+
  geom_vline(aes(xintercept=sqrt(0.3)),   # Ignore NA values for meany
             color="red", linetype="dashed", size=1)# Overlay with transparent density plot
```



Using just your original sample, compute the bootstrap distribution of ^x0 .

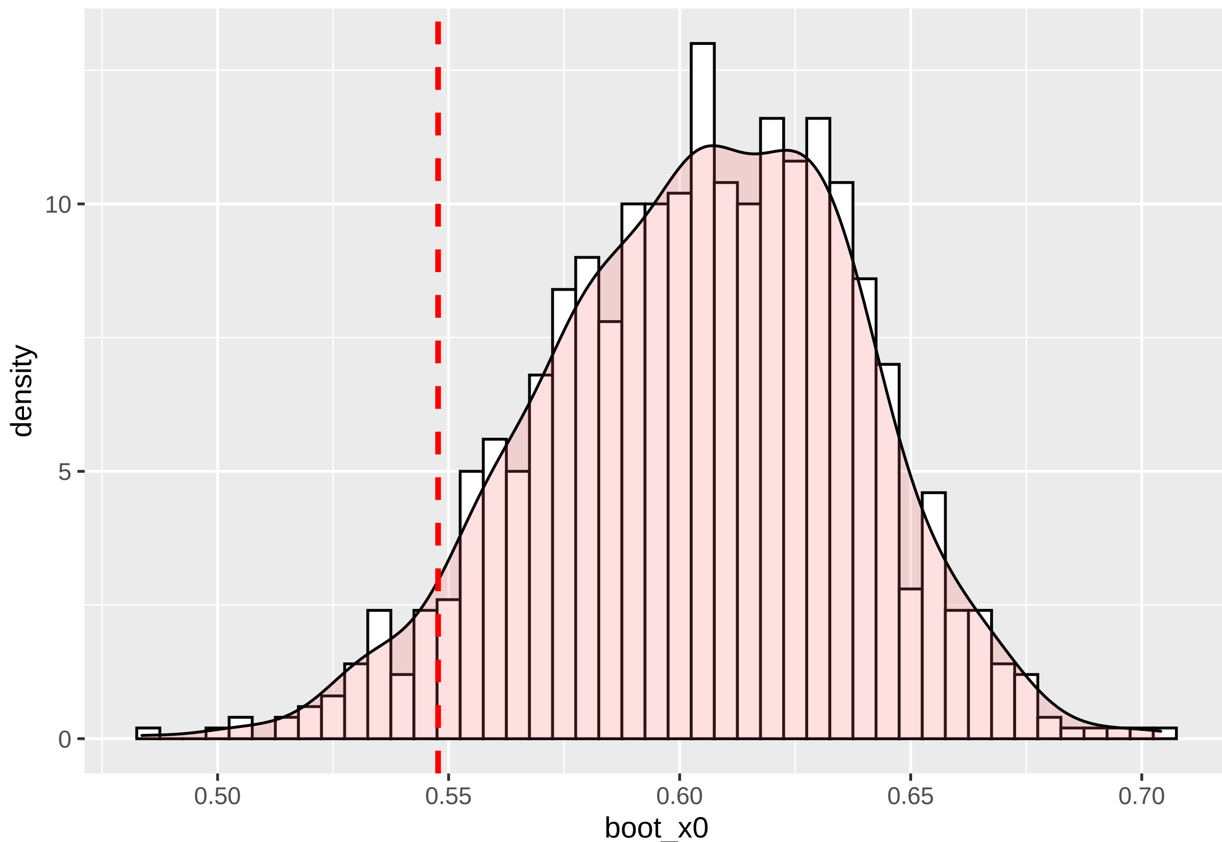
```
boot_x0 = vector()
print(sqrt(0.3))
```

```
## [1] 0.5477226
```

```
for (i in 1:1000){
  boot_x0[i] = bootstrap(data)
```

```
}
```

```
library(ggplot2)
boot_x0 = as.data.frame(boot_x0)
ggplot(boot_x0, aes(x=boot_x0)) +
    geom_histogram(aes(y=..density..),      # Histogram with density instead of count on y-axis
                   binwidth=0.005,
                   colour="black", fill="white") +
    geom_density(alpha=.2, fill="#FF6666")+
  geom_vline(aes(xintercept=sqrt(0.3)),   # Ignore NA values for mean
               color="red", linetype="dashed", size=1)# Overlay with transparent density plot
```



Compare these two distributions, and in particular their standard deviations.

```
print(sqrt(var(as.vector(x0_est))))
```

```
##            x0_est
## x0_est 0.04009306
```

```
print(sqrt(var(as.vector(boot_x0))))
```

```
##             boot_x0
## boot_x0 0.03355649
```

Here, we can see that the bootstrap result has less variance than the simulation result does, though with larger bias.