Variance, Covariance, Correlation

Variance, Covariance, Correlation

Variance

Variance is a simple way of quantifying the amount of variability around a mean in a dataset. It's calculated simply as the sum of squares of the data, after subtracting the mean of the data. Or in linear algebra:

$$Variance = \frac{(x-\bar{x})^T(x-\bar{x})}{df}$$

And df is usually N-1

Let's examine this with some data in R, will use the dataset "trees"

```
head(trees)
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
```

Let's calculate the variance of the tree girth.

```
girthAnom = trees[,1]-mean(trees[,1])
girthVar = t(girthAnom)%*%girthAnom / (length(girthAnom)-1)
print(girthVar)
```

```
##          [,1]
## [1,] 9.847914
```

```
#check it
var(trees[,1])
```

```
## [1] 9.847914
```

What are the units of this variance?

What if you wanted it in the units of the dataset?

Standard deviation.

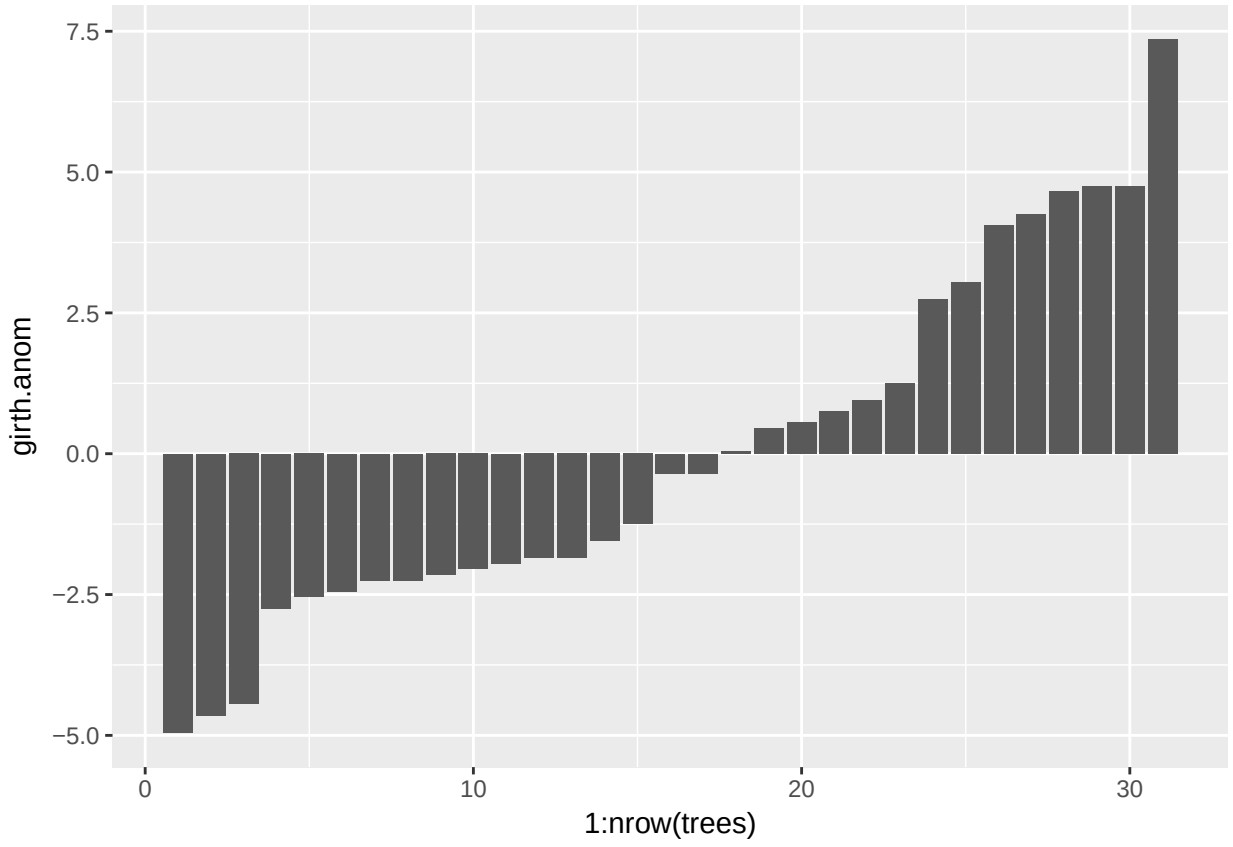
Yep, standard deviation is just the square root of the varaince

OK, let's visualize what we just did

```
library(ggplot2)
mean.girth = mean(trees$Girth)
trees = trees
trees$girth.anom = trees$Girth-mean.girth
head(trees)
```

```
##   Girth Height Volume girth.anom
## 1   8.3     70   10.3  -4.948387
## 2   8.6     65   10.3  -4.648387
## 3   8.8     63   10.2  -4.448387
## 4  10.5     72   16.4  -2.748387
## 5  10.7     81   18.8  -2.548387
## 6  10.8     83   19.7  -2.448387
```

```
ggplot(trees,aes(x=1:nrow(trees),y=girth.anom))+geom_bar(stat="identity")
```



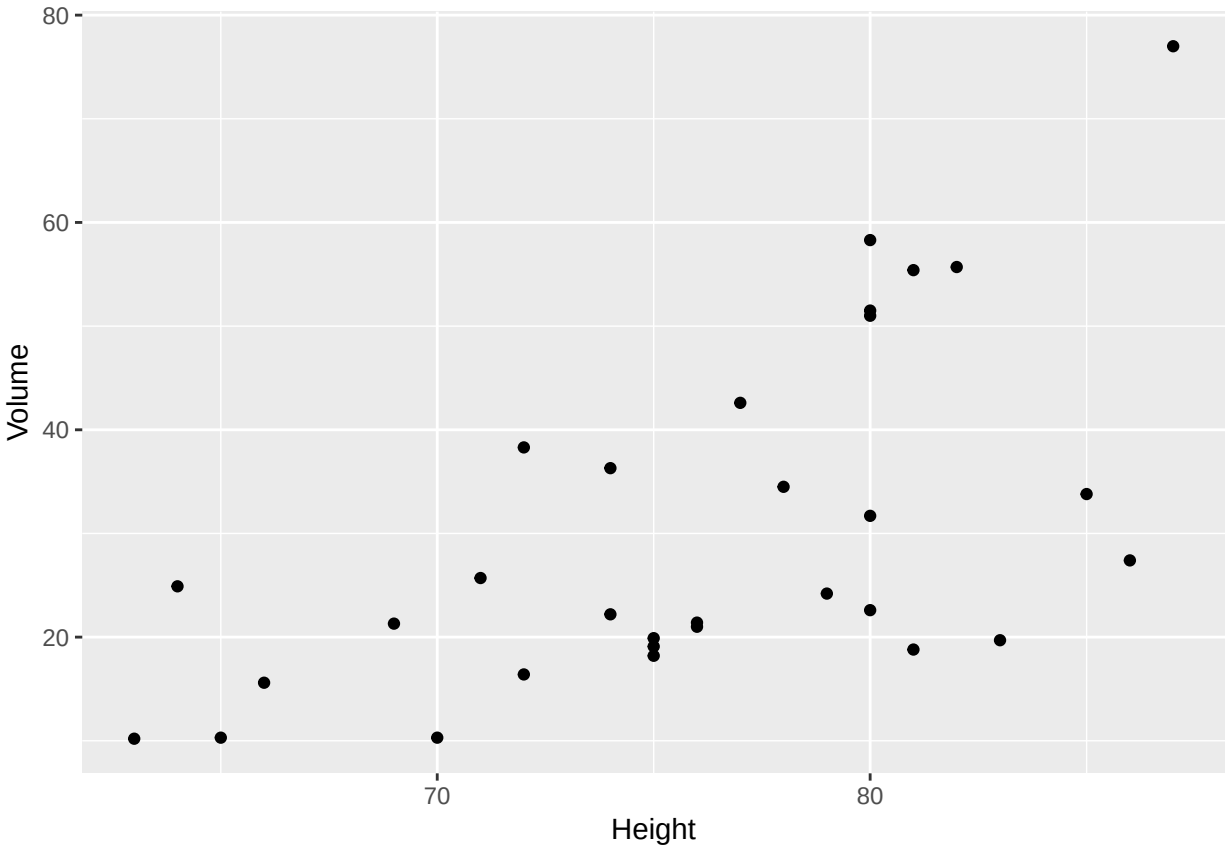
Standard deviation

Covariance

OK, now that you understand what variance is, what do you think covariance is?

We'll use trees again, and calculate the covariance of height and volume. First let's make a scatter plot.

```
ggplot(trees,aes(x=Height, y=Volume)) + geom_point()
```



OK, there's clearly a relationship, so let's calculate the covariance. The equation for covariance is just like variance:

$$Covariance = \frac{(x - \bar{x})^T (y - \bar{y})}{df}$$

and df is typically N-1

In R:

```
height.anom = trees$Height - mean(trees$Height)
volume.anom = trees$Volume - mean(trees$Volume)
covarianceHV = t(height.anom) %*% volume.anom / (length(trees$Volume)-1)
print(covarianceHV)
```

```
##       [,1]
## [1,] 62.66
```

```
#check it
cov(trees$Height,trees$Volume)
```

```
## [1] 62.66
```

What are the units on that number?

OK, let's compare that with covariance between Girth and Volume:

```
girth.anom = trees$Girth - mean(trees$Girth)
volume.anom = trees$Volume - mean(trees$Volume)
covarianceGV = t(girth.anom) %*% volume.anom / (length(trees$Volume)-1)
print(covarianceGV)
```

```
##          [,1]
## [1,] 49.88812
```

```
#check it
cov(girth.anom,volume.anom)
```

```
## [1] 49.88812
```

So which of the two datasets is more related to volume?

Correlation

Right. So covariance is a useful concept, but difficult to interpret on its own, and not particularly valuable for comparisons between unlike variables. Perhaps we should *normalize* that that number by something useful.

Suggestions?

Seems like if we normalize it by the highest possible amount of covariance, that would reveal the relative amount covariance.

The maximum possible covariance occurs when two vectors are linear transformations of themselves. If the identical, that's equivalent to variance. If not, the maximum covariance is

$$\sqrt{var_x * var_y}$$

So the relative amount of the highest covariance is

$$\rho = \frac{cov}{\sqrt{var_x * var_y}}$$

This is called *correlation*, and ρ is the correlation coefficient.

Let's calculate ρ for each of our covariances above and see which is higher.

```
maxCovHV = sqrt(t(height.anom)%*%height.anom / (length(height.anom)-1) * t(volume.anom)%*%(volume.anom)
maxCovGV =  sqrt(t(girth.anom)%*%girth.anom / (length(height.anom)-1) * t(volume.anom)%*%(volume.anom)

rhoHV = covarianceHV/maxCovHV
rhoGV = covarianceGV/maxCovGV

print(rhoHV)
```

```
##          [,1]
## [1,] 0.5982497
```

```
print(rhoGV)
```

```
##          [,1]
## [1,] 0.9671194
```

Let's check our answer using the built in cor() function:

```
print(cor(trees$Height,trees$Volume))
```

```
## [1] 0.5982497
```

```
print(cor(trees$Girth,trees$Volume))
```

```
## [1] 0.9671194
```

Phew. It worked.

Fraction of Variance.

So if we square ρ, get:

$$\rho^2 = \frac{cov^2}{var_x * var_y}$$

Which compares the covariance, to the total variance in both datasets. This is why it's reasonable to think of ρ^2 as the fraction of total variance explained by the covariance.

Significance testing

Now that we have a number that's comparable between multiple datasets, we'd like to have some idea if our correlation coefficients are something special. Is a value of 0.23 good? How about -0.5? Which means that there's a "real" correlation?

Let's think about his in terms of a null hypothesis. If we set our null hypothesis to be:

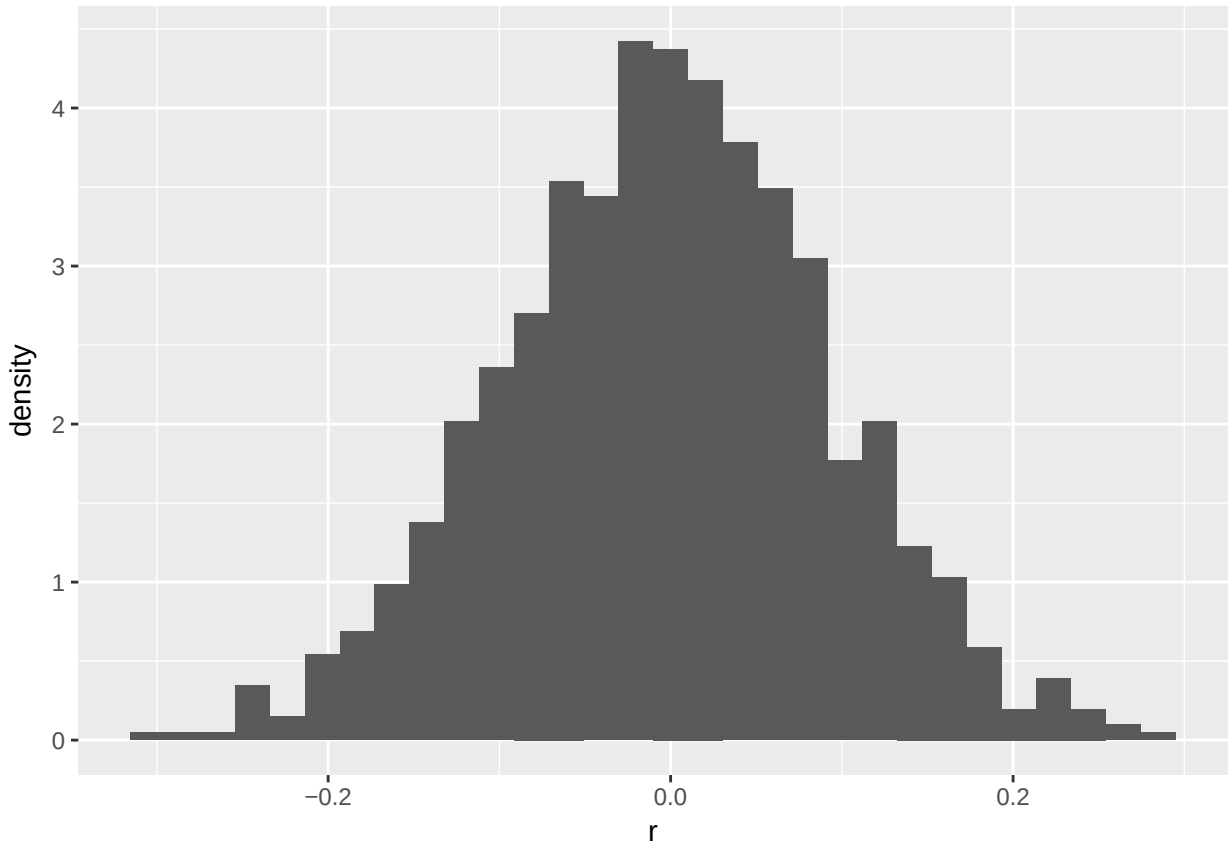
"The correlation between these two datasets is less extreme than what is produced by two random datasets 95% of the time"

If we can show that our value is higher than that 95% confidence level, then we reject the null hypothesis, and call it significant at the 95% level.

So let's generate a whole bunch of random data to see what that looks like.

```
r=c()#Initialize a spot for our random calculations to go
for(i in 1:1000){#lets do this a thousand times
  rd1 = rnorm(100)#simulate data each time through the loop
  rd2 = rnorm(100)#twice
  r[i] = cor(rd1,rd2)#correlate and store the values
}
#now let's plot it
ggplot()+geom_histogram(aes(x=r,y=..density..)) #Let's look at my favorite plot
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



And we see that this looks something like a normal distribution, but is sensitive to the number of observations. This is modeled well with a Student's T distribution, and you'll go through this in your lab. I'll do an example here with with a different test we want to use a t-distribution for.

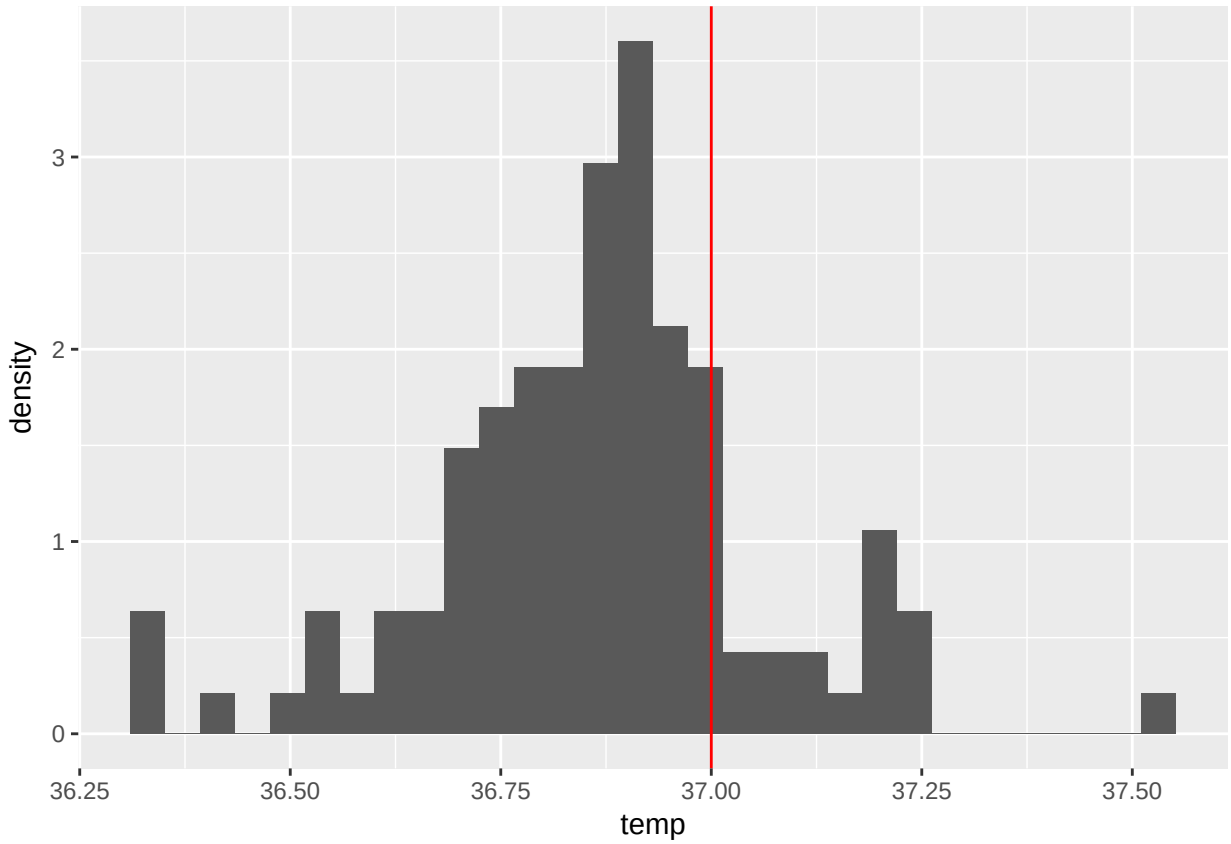
Probability testing example, difference of means testing.

To do this, we'll take a look at the Beavers dataset.

We're interested in the probability that Beaver1's mean body temperature is less than 37 degrees.

```
#Lets take a look at our observations
ggplot(beaver1)+geom_histogram(aes(x=temp,y=..density..))+geom_vline(xintercept = 37,color="red")
```

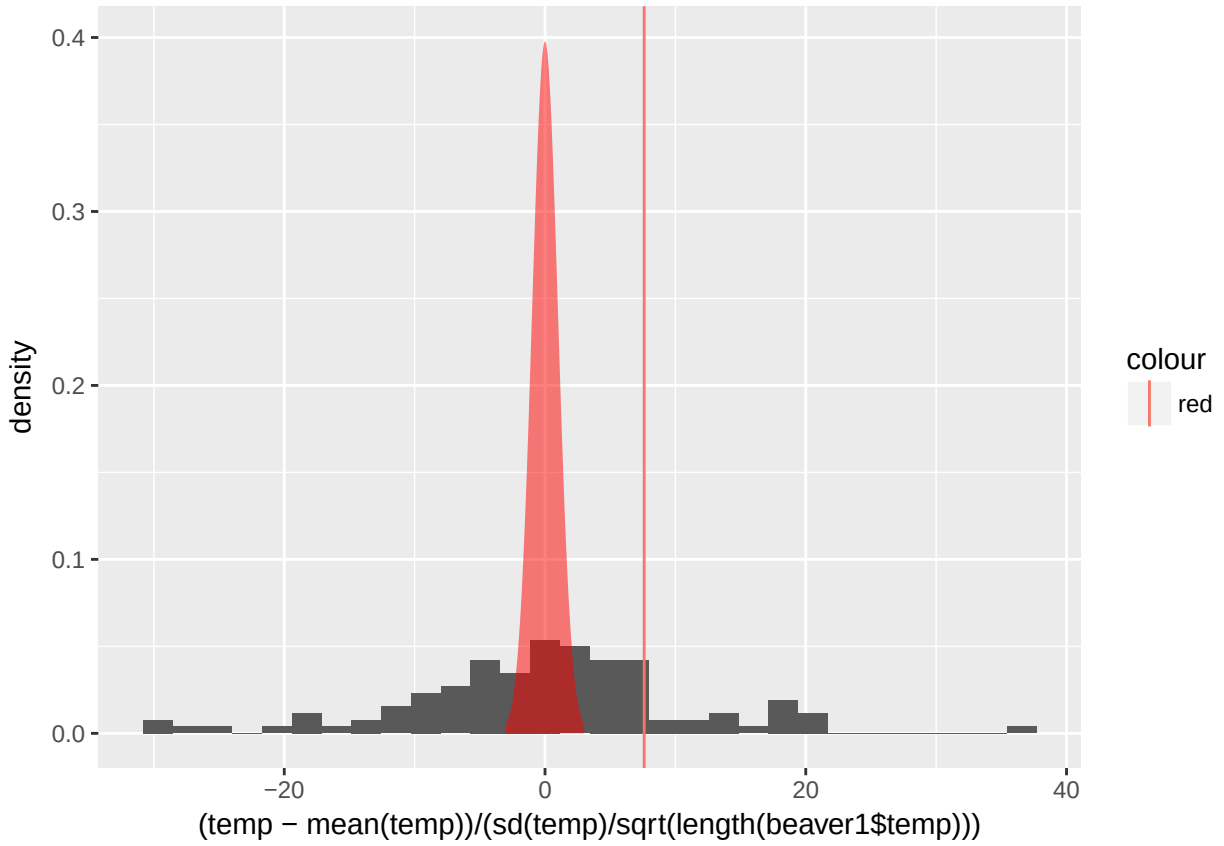
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Student T distributions, by definition, always have a mean of zero. Let's make our data have a mean of zero and a standard deviation of 1, and then we can compare it to a Students' T.

```
#Lets take a look at our observations
myplot = ggplot(beaver1)+geom_histogram(aes(x=(temp-mean(temp))/(sd(temp)/sqrt(length(beaver1$temp))),y=
#and let's add a tdistribution
td = dt(seq(-3,3,by=.1),df=length(beaver1$temp)-2)
ndf = data.frame(x=seq(-3,3,by=.1),y=td)
myplot+geom_area(data=ndf,aes(x=x,y=y),fill="red",alpha=0.5)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Again, we're no longer comparing to see if the mean temperature is less than 37, we've converted 37 into T-distribution space by subtracting the mean and dividing by the standard deviation.

```
Tstat = (37-mean(beaver1$temp))/(sd(beaver1$temp)/sqrt(length(beaver1$temp)))
print(Tstat)
```

```
## [1] 7.607089
```

So now we want to see how our T-distribution compares to our "T-stat" So we'll use the cumulative probability function, to see the probability that the T-dsitribution with the appropriate df is less than our T-stat

```
pval = pt(Tstat,df = length(beaver1$temp)-2)
print(pval)
```

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

OK, but what if we wanted to estimate how well we know our mean, or more specifically, the probability that our mean is more than 0.1 degrees different than our measured mean?

Now we're interested in the probability that the mean is:

less than 36.83 degrees or greater than 36.89 degrees

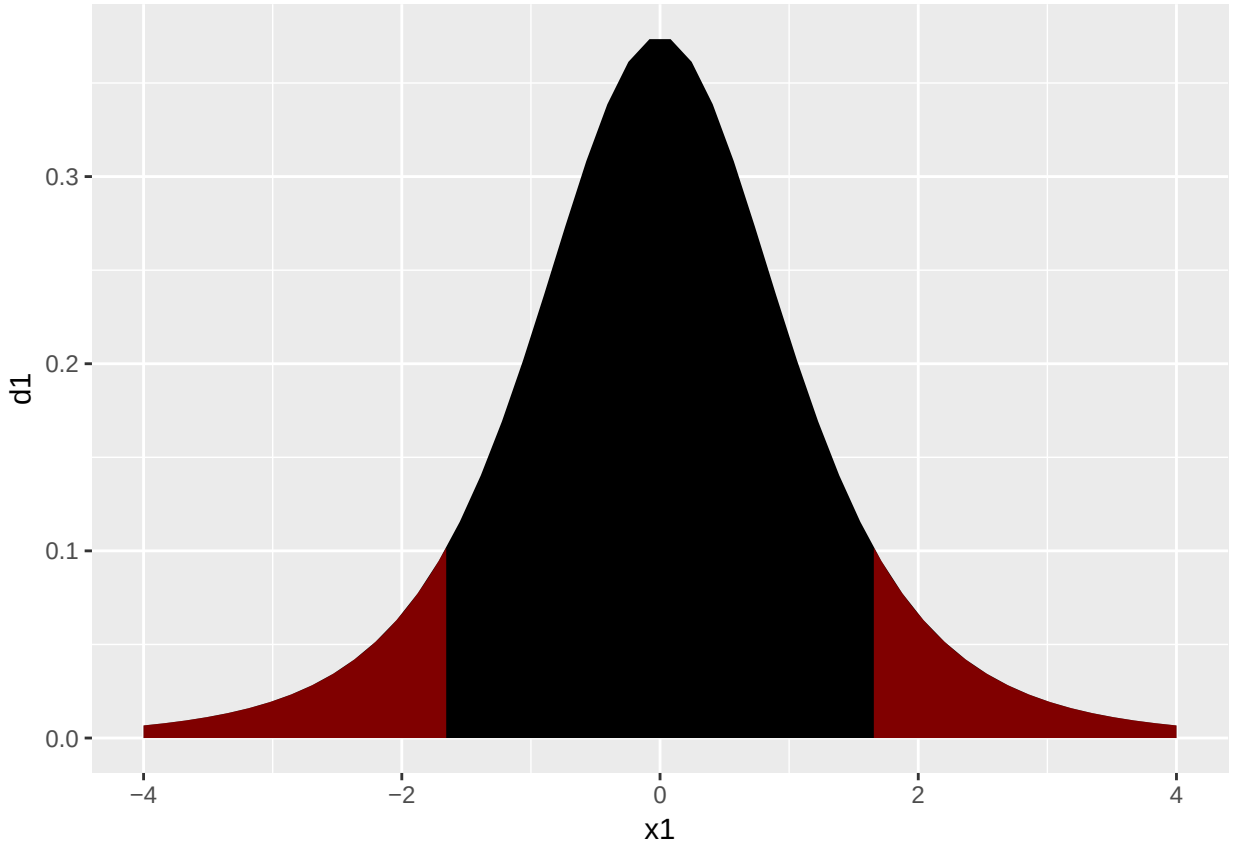
Graphically, this might look like this:

```
tstatHi = 0.03/(sd(beaver1$temp)/sqrt(length(beaver1$temp)))
tstatLo = -0.03/(sd(beaver1$temp)/sqrt(length(beaver1$temp)))

x1 = seq(-4,4,length.out = 50)
d1 = dt(x1,df=length(beaver1-2))
x2 = seq(tstatHi,4,length.out = 50)
```

```
d2 = dt(x2,df=length(beaver1-2))
x3 = seq(-4,tstatLo,length.out = 50)
d3 = dt(x3,df=length(beaver1-2))

plotdf = data.frame(x1,x2,x3,d1,d2,d3)
ggplot(plotdf)+geom_area(aes(x1,d1),fill = "black")+geom_area(aes(x2,d2),fill="red",alpha=0.5)+geom_area
```



We can test this with a T-test too, and because the T-distribution is symmetrical, we can just look at the cumulative probability on the left side, and double it:

```
pt(tstatLo,df=length(beaver1$temp))*2
```

```
## [1] 0.1004657
```

Or if we were generalizing, and about the probability that the value was x more extreme than the mean, we could say:

```
x=tstatHi#x can be positive or negative now!

pt(-abs(x),df=length(beaver1$temp))*2
```

```
## [1] 0.1004657
```

Monte-carlo methods

Let's imagine a different way to do this analysis. What if instead using a probability distribution, we wanted to just simulate the mean 1000 times, and see the probability that it's outside the range of the we defined above that way?

Using random number generators to make large simulations of data, and then looking at those results is a class of methods called "Monte Carlo" methods.

We'll use rnorm() to generate a our beaver temperature data 1000 times.

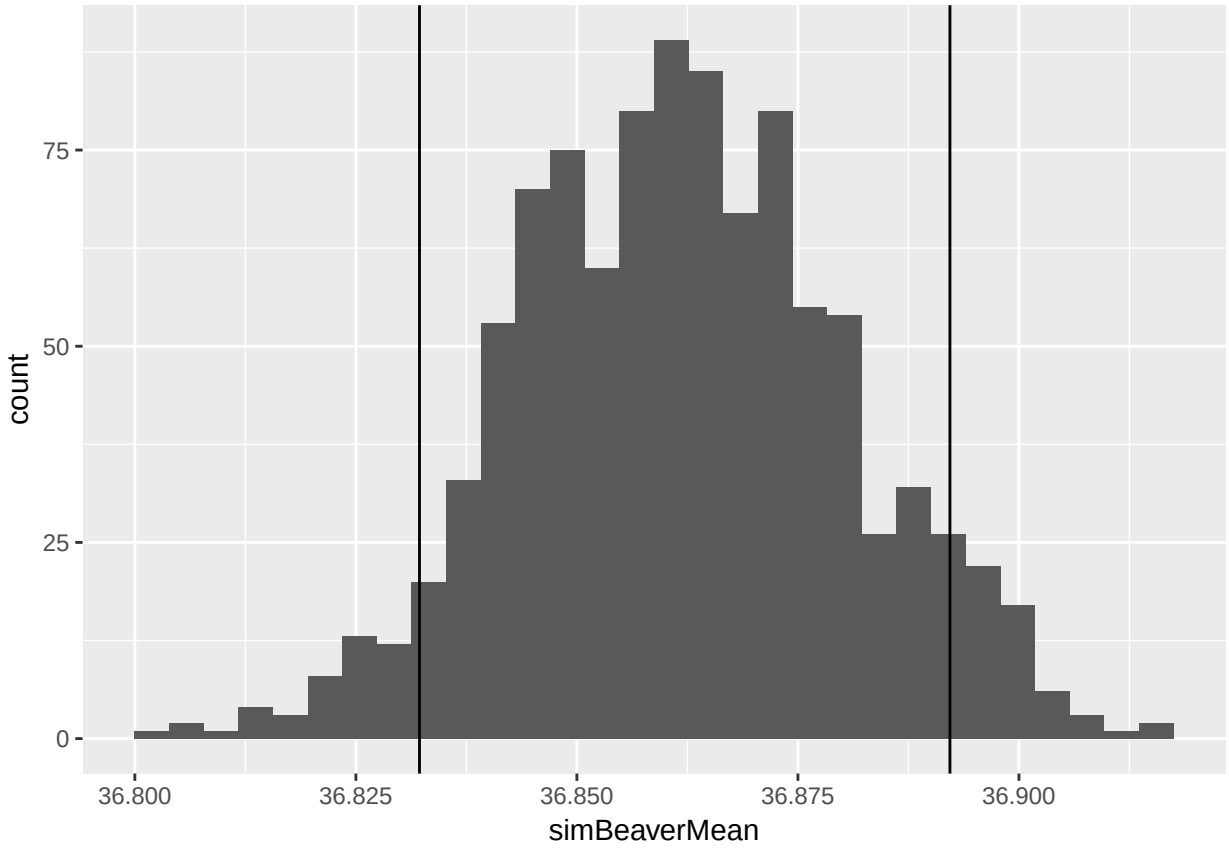
```
bmean = mean(beaver1$temp)
bsd = sd(beaver1$temp)
nits = 1000
simBeaverMean = matrix(NA,nits,1)
for(i in 1:nits){#for every iteration
  simBeaver = rnorm(n = length(beaver1$temp),mean = bmean,sd=bsd)
  simBeaverMean[i] = mean(simBeaver)

  }
```

OK, now we've calculated 1000 random means, let's see what that distribution looks like:

```
hiTemp = mean(beaver1$temp)+0.03
loTemp = mean(beaver1$temp)-0.03
ggplot()+geom_histogram(aes(x=simBeaverMean))+geom_vline(aes(xintercept=loTemp))+geom_vline(aes(xinterc
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



And now let's figure what where are 95% confidence interval is...

```
quantile(simBeaverMean,c(0.025,0.975))
```

```
##     2.5%    97.5%
## 36.82583 36.89860
```

Or, what probabilities our hi or low temperature ranges correspond to

```
beaverCDF = ecdf(simBeaverMean)
ourProbs = beaverCDF(c(loTemp,hiTemp))
totalExtremeProb = ourProbs[1]+(1-ourProbs[2])
```

Bootstrapping

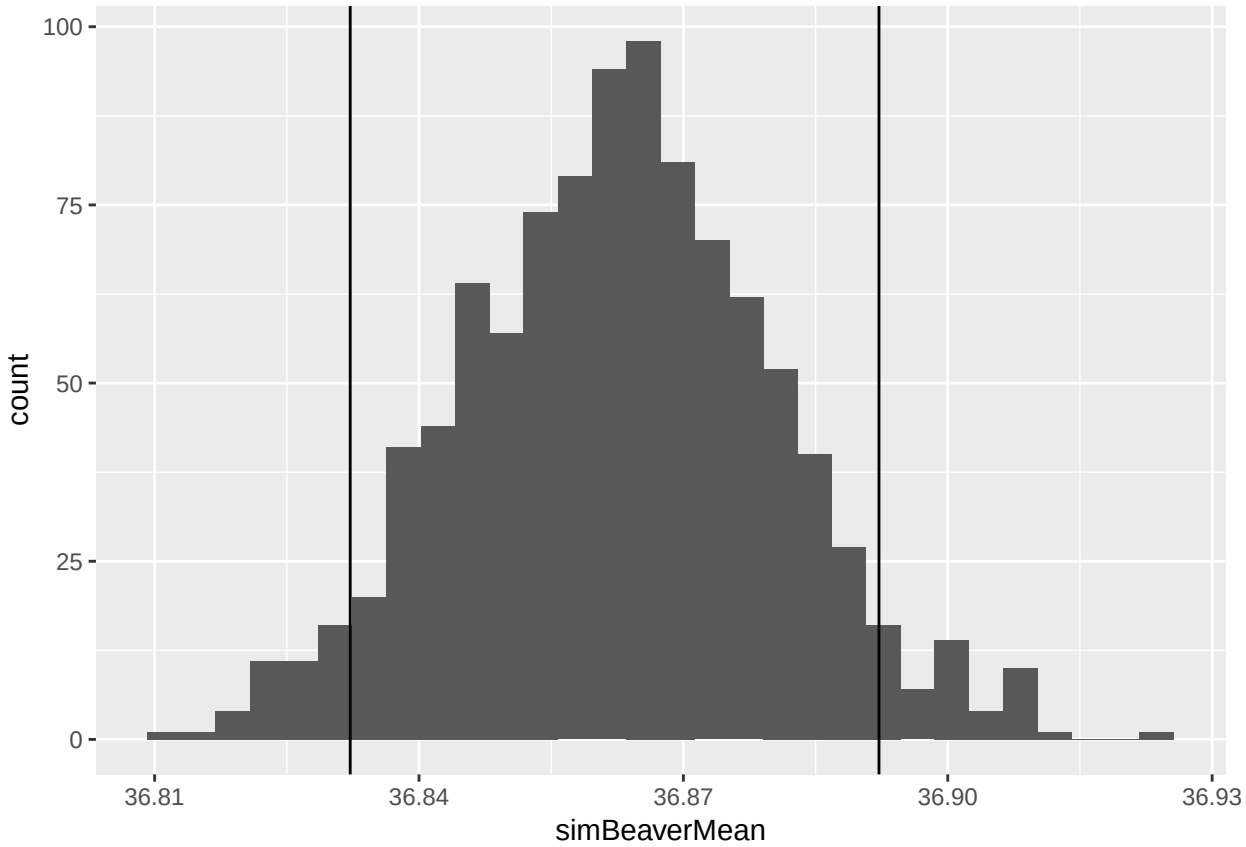
In bootstrapping, rather than specifying a distribution and its parameters from which to simulate the data, we resample our observations, with replacement, to get at our distribution:

```
nits = 1000
simBeaverMean = matrix(NA,nits,1)
for(i in 1:nits){
  simBeaver = sample(beaver1$temp,size = length(beaver1$temp),replace = TRUE)
  simBeaverMean[i] = mean(simBeaver)
}
```

Once we have that, we analyze that and plot as we did before:

```
#So now we have 1000 possible means from our dataset
#let's take a look
hiTemp = bmean+0.03
loTemp = bmean-0.03
ggplot()+geom_histogram(aes(x=simBeaverMean))+geom_vline(aes(xintercept=loTemp))+
  geom_vline(aes(xintercept = hiTemp))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
#lets use our simulated results to calculate a 95% confidence interval on our mean
quantile(simBeaverMean,c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 36.82754 36.89921
```

```r
#what about figuring out percentiles, given a certain critical value?
beaverCDF = ecdf(simBeaverMean)
ourProbs = beaverCDF(c(loTemp,hiTemp))
print(ourProbs)
```

```
## [1] 0.041 0.954
```

```r
totalExtremeProb = ourProbs[1]+(1-ourProbs[2])
print(totalExtremeProb)
```

```
## [1] 0.087
```

Jackknifing

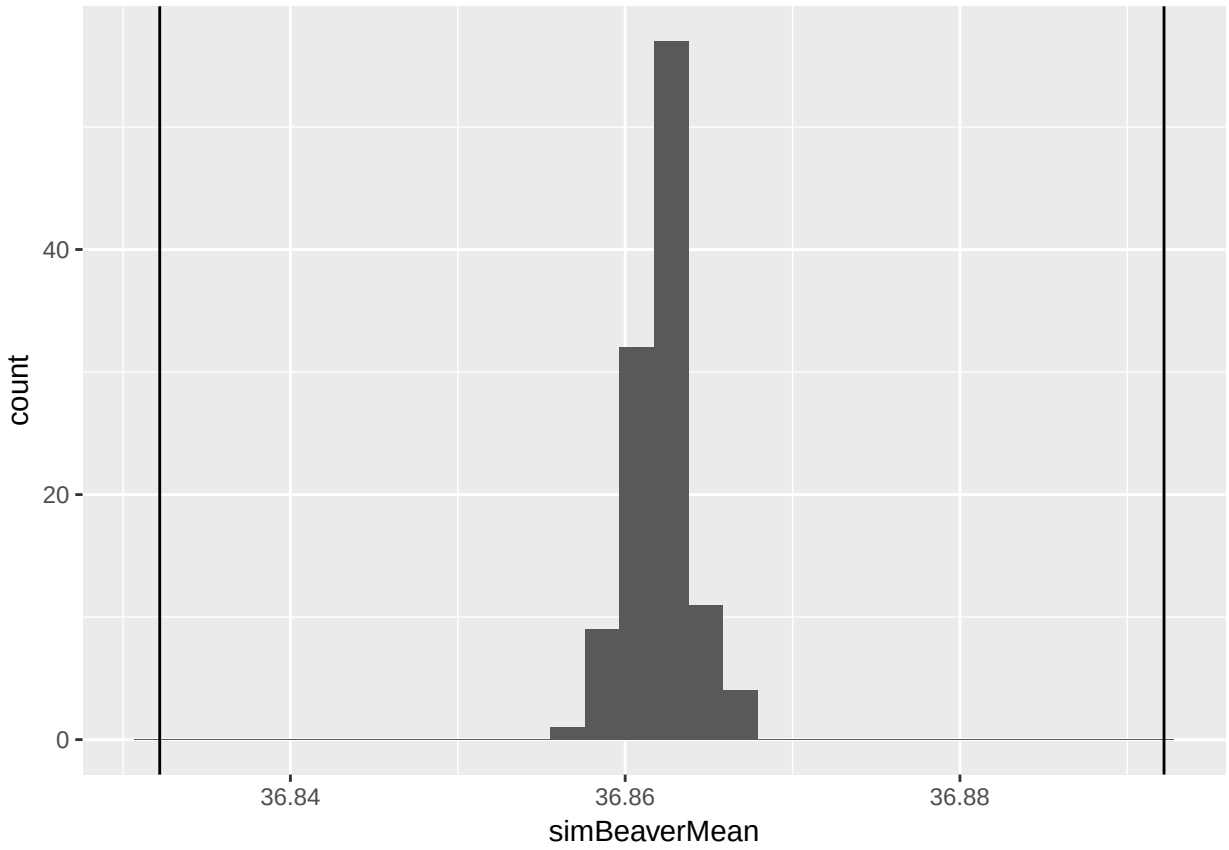
Something similar to bootstrapping, but not random and more explicit, is called jackknifing, where we repeat our analysis n times, removing one observation from our analysis each time through:

```r
#One last example: Jackknifing
simBeaverMean = matrix(NA,length(beaver1$temp),1)
for(i in 1:length(beaver1$temp)){
  simBeaver = beaver1$temp[-i]
  simBeaverMean[i] = mean(simBeaver)
}
```

Again, once we have our simulated values, we can analyze it the same way:

```r
#So now we have 1000 possible means from our dataset
#let's take a look
hiTemp = bmean+0.03
loTemp = bmean-0.03
ggplot()+geom_histogram(aes(x=simBeaverMean))+geom_vline(aes(xintercept=loTemp))+
  geom_vline(aes(xintercept = hiTemp))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
#lets use our simulated results to calculate a 95% confidence interval on our mean
quantile(simBeaverMean,c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 36.85892 36.86621
```

```
#what about figuring out percentiles, given a certain critical value?
beaverCDF = ecdf(simBeaverMean)
ourProbs = beaverCDF(c(loTemp,hiTemp))
print(ourProbs)
```

```
## [1] 0 1
```

```
totalExtremeProb = ourProbs[1]+(1-ourProbs[2])
print(totalExtremeProb)
```

```
## [1] 0
```

Side note - functions with multiple outputs.

One last thing. It's often useful to calculate values with multiple, and sometimes different kinds of outputs. But R only lets you export one thing from a function, and then it ends. Do this, you should take advantage of R's most general data type, the *list*. Think of a list as a bucket of any other kind of data with a name. Here's an example:

Write a function that you input a vector, and it returns both the mean, standard deviation and the input vector.

```r
awesome = function(vec){
  out = list()#initialize my list
  out$mean = mean(vec)#calculate and store the mean
  out$sd = sd(vec)#again for sd
  out$input = vec#and store the vector.

  return(out)
}


my.out = awesome(rnorm(100)) #let's try with 100 random datapoints
print(my.out$mean)
```

```
## [1] -0.04687396
```

```r
print(my.out$input)
```

```
##   [1] -2.133130183 -1.082234372 -0.569635905 -0.398712937 -0.312243409
##   [6]  0.335801864  0.820672549  0.085134168 -1.196670456 -1.106081375
##  [11] -1.343600232 -1.713713311  0.055124400 -0.311107874  0.962392496
##  [16]  0.397477634 -0.216879448 -0.684417636  0.722022085 -1.351432250
##  [21]  0.930396833  0.541666812  1.325408056 -0.415740695 -0.309942906
##  [26] -2.126578850 -0.343735482  0.447615073 -0.826214294  0.564219972
##  [31]  0.059887018  0.893194141 -0.205288581  1.815980061  0.122111876
##  [36] -0.913085006 -1.504263370  0.103707627  0.083611372  1.535183773
##  [41]  0.605272358 -0.485766737  0.621757557  0.483707378  0.016760940
##  [46]  1.406155411 -0.465937187 -1.293487107 -0.453916267 -0.962441499
##  [51]  0.208164413  0.421694739  0.606456743 -1.213176450  0.897020286
##  [56]  0.299057623 -1.349534462  0.733900553 -2.396534729  0.103809902
##  [61]  0.140221557 -0.977957204  1.105801307  0.038898781 -0.583102750
##  [66] -0.006794458 -0.282765086 -1.511296101  0.165256893 -1.827475579
##  [71]  0.100698678  1.407072458  1.246992095  1.873905842 -1.571088667
##  [76] -2.007192332  2.059132862  0.014770290 -0.999900504  2.706988063
##  [81]  0.223142325  0.561310285  0.601573921 -0.418303737  0.936429417
##  [86] -0.521805703 -0.538801875 -0.857413267  1.017003882  0.817099569
##  [91] -0.196653380 -1.629735391  0.619698777  0.810246574  1.439747735
##  [96]  0.908251869  1.048199959 -0.348793536  0.180068694 -0.950690980
```