# Rlecturenotes2022 11 08

Adam Tallman

2022-11-08

# Packages that need to be loaded

```
library(rlist)
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
## ✓ ggplot2 3.3.6      ✓ purrr   0.3.4
## ✓ tibble  3.1.8      ✓ dplyr   1.0.10
## ✓ tidyr   1.2.0      ✓ stringr 1.4.1
## ✓ readr   2.1.3      ✓ forcats 0.5.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✕ dplyr::filter() masks stats::filter()
## ✕ dplyr::lag()    masks stats::lag()
```

```
library(reshape)
```

```
## Warning: package 'reshape' was built under R version 4.2.2
```

```
##
## Attaching package: 'reshape'
##
## The following object is masked from 'package:dplyr':
##
##     rename
##
## The following objects are masked from 'package:tidyr':
##
##     expand, smiths
```

```
library(nhstplot)
```

```
## Warning: package 'nhstplot' was built under R version 4.2.2
```

# Sample variance

The sample variance is just the standard error squared. We can also represent it with V.

$$s^2 = \frac{\sum(X - \overline{x})^2}{n - 1}$$

The variance is a measurement of how far apart the numbers are in a dataset in relation to the mean.

$$V = \frac{\sum(X - \overline{x})^2}{n - 1}$$

The standard deviation is in the units of measurement of the values. But the variance is larger - that value squared.

# Exercise (Variance)

Calculate the standard deviation and variance for the reaction time of the forced task decision task chacobo data. Make sure to read in the data set that we made last class, which I have uploaded on moodle.

```
df <- read.csv("/Users/Adam/Desktop/experiment/chacobo.mcf.df.csv")
```

We can calculate the sample standard deviation “by hand”

```
rt <- df$reactionTime
rt.sd <- sqrt( ( sum((rt - mean(rt))^2 ) / ( length(rt) -1 ) ))
rt.sd
```

```
## [1] 2.986827
```

```
sd(rt)
```

```
## [1] 2.986827
```

The variance is just the standard deviation squared.

```
rt.sd^2
```

```
## [1] 8.921136
```

Or we can calculate it with the function var().

```
var(rt)
```

```
## [1] 8.921136
```

# T statistic

The p-value comes from the t statistic

$$t = \frac{x_1 - x_2}{\sqrt{\frac{V_1}{N_1} + \frac{V_2}{N_2}}}$$

The t statistic means roughly: the mean of group one (x1) minus the mean of group two (x2) divided by the square root of the variance of group one (V1) divided by its sample (N1) added to the variance of group two (V2) divided by group two's sample (N2).

It can also come from a z statistic. Z is the mean of the group minus the mean of the sample divided by the standard deviation.

$$Z = \frac{(x_1 - x)}{sd}$$

Once you have understood the t statistic you can understand the p value: its calculated mechanically from a "cumulative density function": its fairly easy to understand intuitively & geometrically, but calculating it by hand requires some calculus, which we won't cover in this course. Once you understand what a probability distribution is, and you understand the logic of the t statistic, the notion of a p value follows straightforwardly.

Once you understand these two ideas, hopefully you'll get some intuition about the underlying philosophy behind "classical statistics" "Frequentism": the idea behind frequentism is that the probability distribution represents some hypothetical state of affairs where the experiment were to be conducted on forever, but where the hypothesis you're testing is false: a very strange idea, and, I think, hard to understand without understanding what it was reacting against (the purported subjectvism of Bayesianism). You ask for the probability that the data you have were to occur if the imaginary wrong hypothesis were true. If this is making your head spin - don't worry, you are totally not alone. If you misinterpret p-values as some sort of inverse probability of your hypothesis being confirmed (lower p-value the better my hypothesis is), then you are interpreting them (wrongly) the way most scientists seem to.

If you think this sounds like a nonsensical way of thinking about probability (the probability of my data given an imaginary null hypothesis),and you are thinking there *must* be a better way of thinking of this, then you are actually on to something.

# What are probability density distributions?

A probability density distribution is a shape whose area is equal to 1. You can associate particular areas in a probability distribution with the probability of particular events occurring

Our frequency distributions can always be translated into probability distributions. The *difference* is that a probability distribution can be created by mathematical formula - they are thus more abstract than our frequency distribution. Note that when I was talking about "imaginary situations based on hypothetical probability distributions", I was talking about probability distributions.

The best way of understanding what a probability distribution is to simulate a frequency distribution and then build a probability distribution from that distribution.

# The Uniform distribution

The function runif() produces a bunch of random numbers following uniform distrbution. What's a uniform distribution? A distribution where everything is the same (usually or practically) bounded by a minimal and a maximal value.

runif() can actually give you a random number between two values. The syntax is like runif(number of values, minimum value, maximal value). Oh and unless you put in set.seed() at a specific value it will be a different number everytime!

```
set.seed(123)
runif(1, min = 0, max = 1)
```

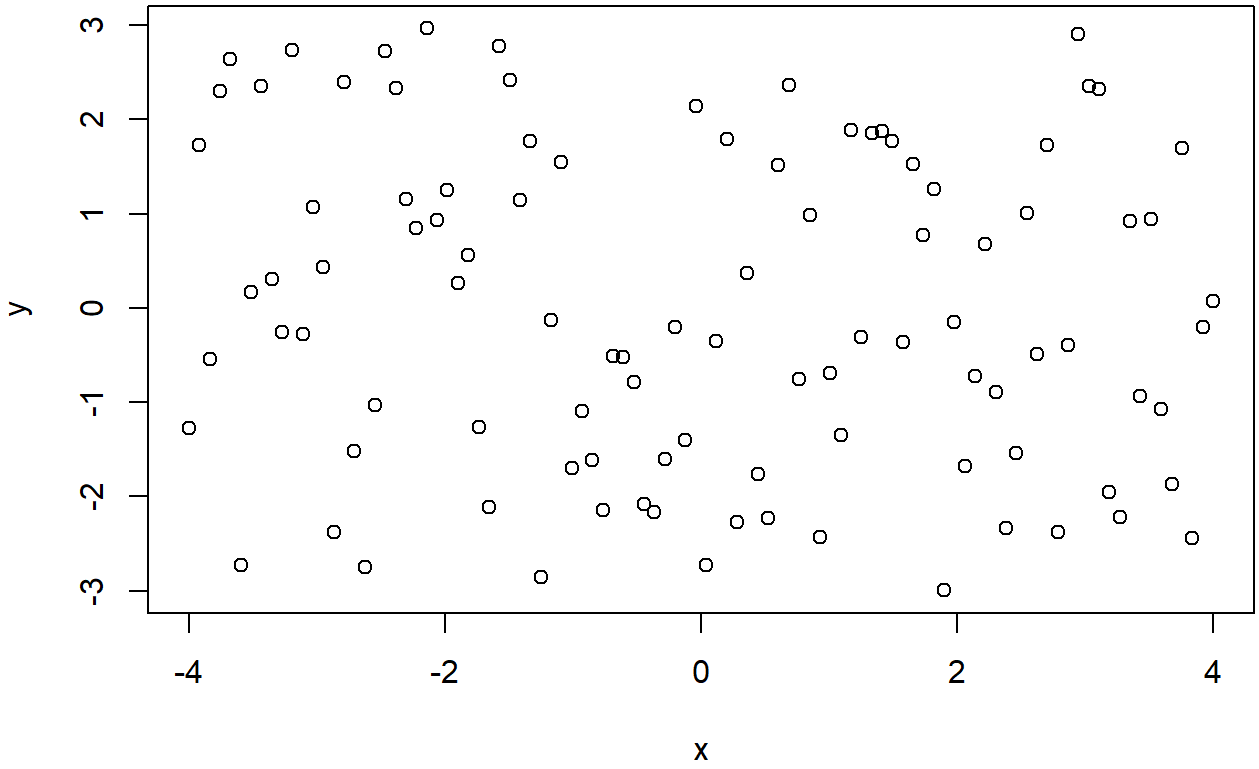
```
## [1] 0.2875775
```

Let's make some values according to a uniform distribution. Another function is to make non-random numbers in a sequence. We can use the function seq() for this.

```
x <- seq(-4, 4, length=100)
```

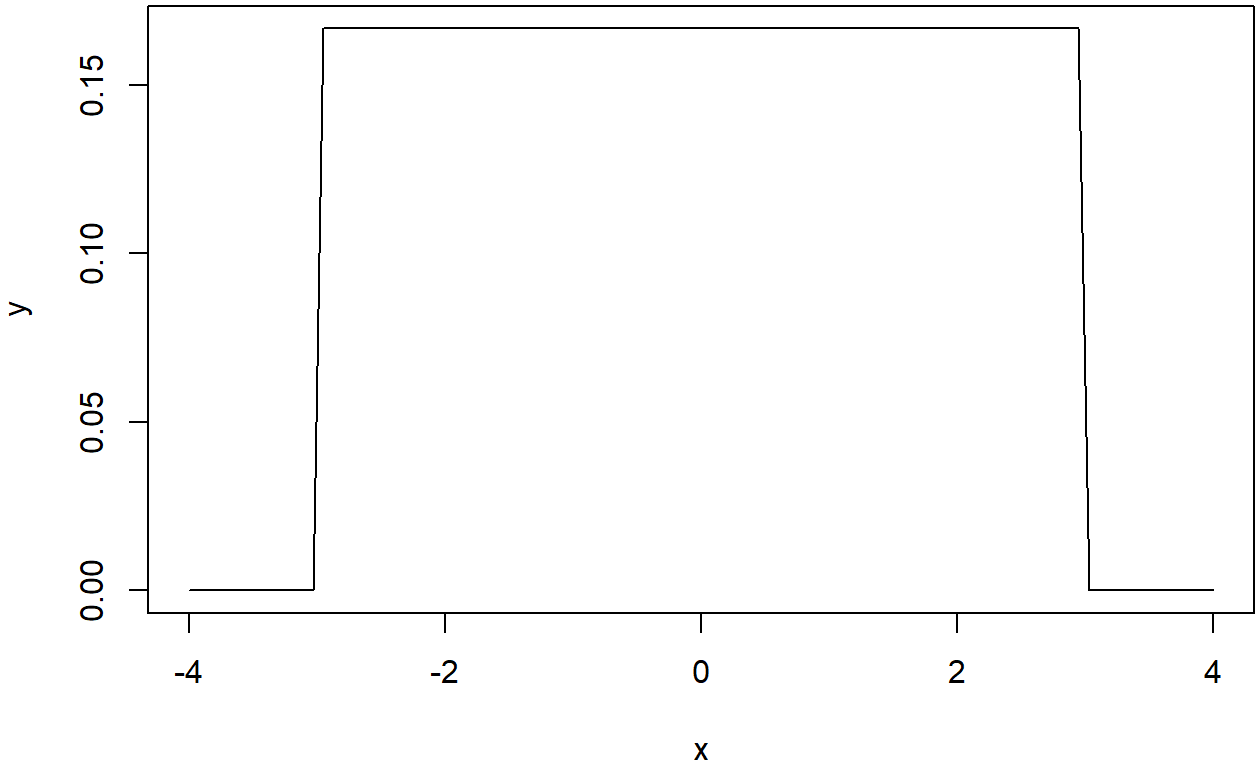
Imagine we want y values that are random. Let's get 100 of them so we can plot them in relation to our x value.

```
set.seed(123)
y <- runif(100, min = -3, max = 3)
plot(x,y)
```




There's an equivalent to the runif() function for probability density functions. It is dunif(). Its basically the same thing, *but* it makes a density function.

```
y <- dunif(x, min = -3, max = 3)
plot(x, y, type = 'l')
```




This is a probability distribution! How do we know - we know because the surface area equals 1…

```
a <- max(y)
b <- 3 - (-3)
a*b
```

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

Why is this a useful concept? We can calculate the probability of an event occurring on a specific are of the distribution. For example, if we want to know what the probability of x being between -3 and -2, its fairly easy.

```
a<-max(y)
b <- 1
a*b
```

```
## [1] 0.1666667
```

# Normal distribution

We introduced the normal/Gaussian distribution before. In R you can randomly generate a number from a normal distribution as well with rnorm().

```
set.seed(123)
rnorm(2)
```

```
## [1] -0.5604756 -0.2301775
```

R assumes the mean and standard deviations are 0 and 1 respectively unless you specify.

```
set.seed(123)
rnorm(2, mean=0, sd=1)
```

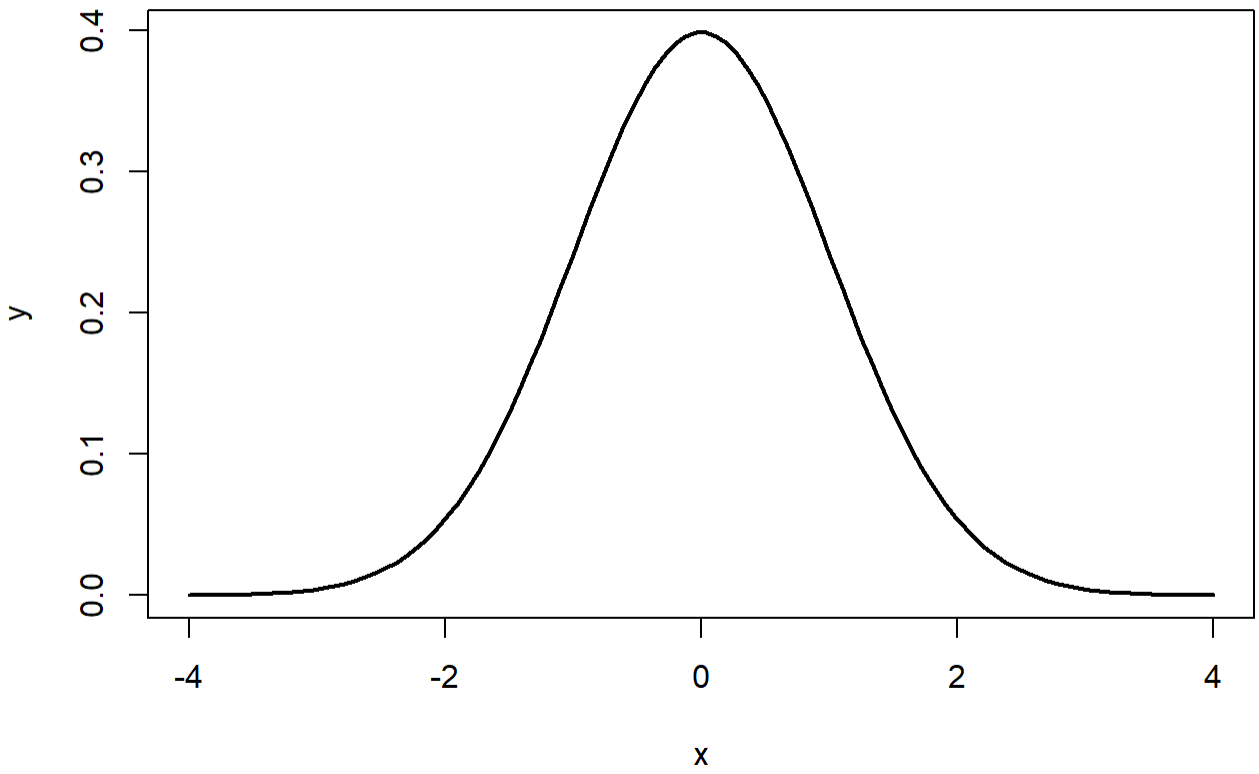
```
## [1] -0.5604756 -0.2301775
```

Let's make a bunch of y values from -4 to 4. dnorm() gives you bunch of values according to a probability distribution.

```
x <- seq(-4, 4, length=100)
y <- dnorm(x)
```

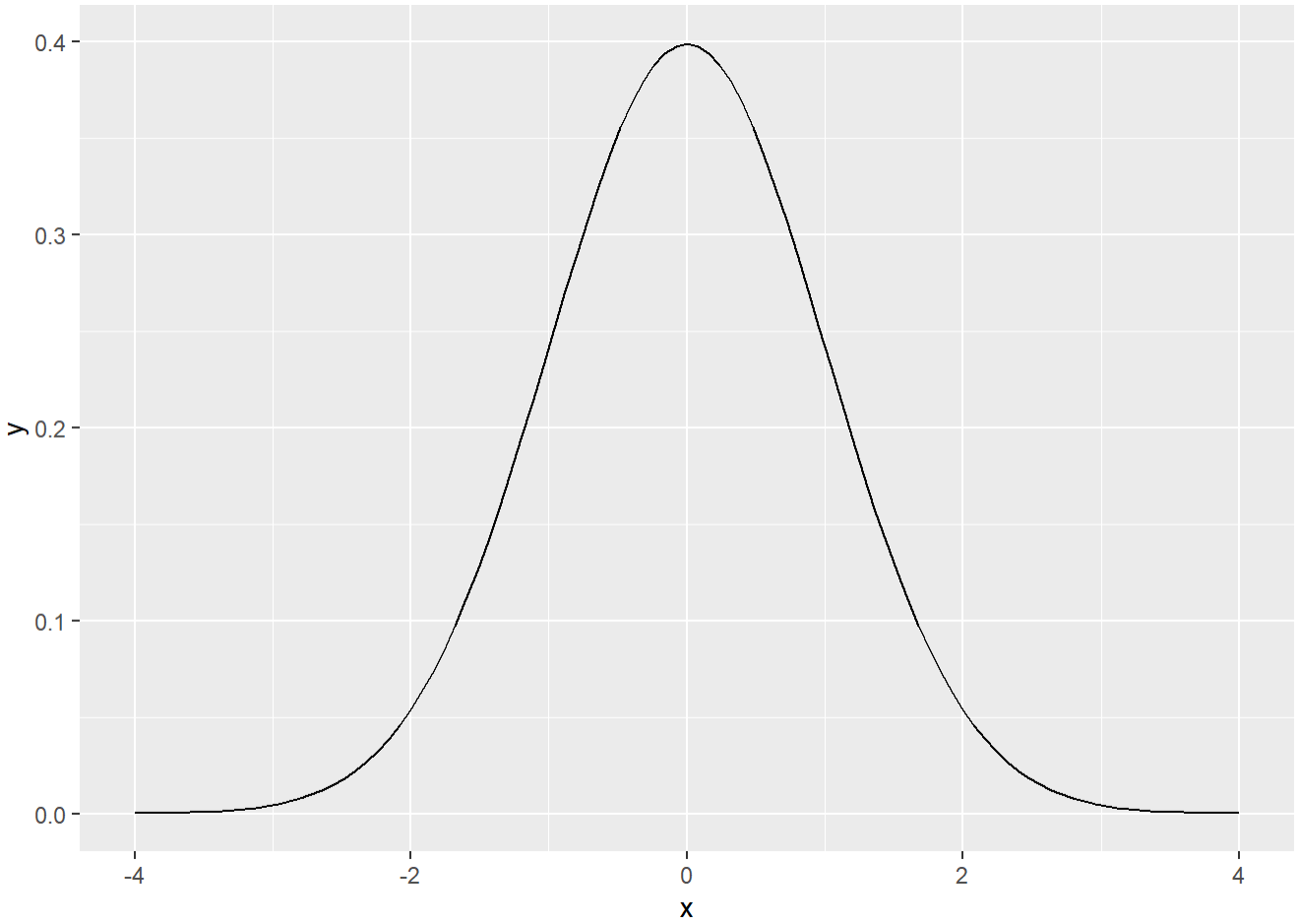
Here's the code for the plot. This lwd thing just makes the line a little thicker.

```
plot(x,y, type = "l", lwd = 2)
```



You can make this using ggplot with the following code as well.

```
ggplot(data.frame(x = c(-4, 4)), aes(x = x)) +
stat_function(fun = dnorm)
```




The area of this curve is 1. Calculating the area of specific areas using integral calculus, so we won't do that without using R functions.

# The binomial distrbution

Sometimes your values are just 0 or 1 (true or false). The binomial distribution is a distribution you get when you just have two values.

I guess its easiest to understand in reference to flipping a coin - imagine you flip a fair coin (p=0.5), 10 times. You flip it ten times once.

```
set.seed(123)
rbinom(1, 10, 0.5)
```

```
## [1] 4
```

Imagine a flip a coin 10 times, 10 times (or ten trials).

```
set.seed(123)
rbinom(10, 10, 0.5)
```

```
##  [1] 4 6 5 7 7 2 5 7 5 5
```

Imagine I flip a coin 10 times, with 10 trials. But the coin is biased. You only get heads 20% of the time.

```
set.seed(123)
rbinom(10, 10, 0.2)
```

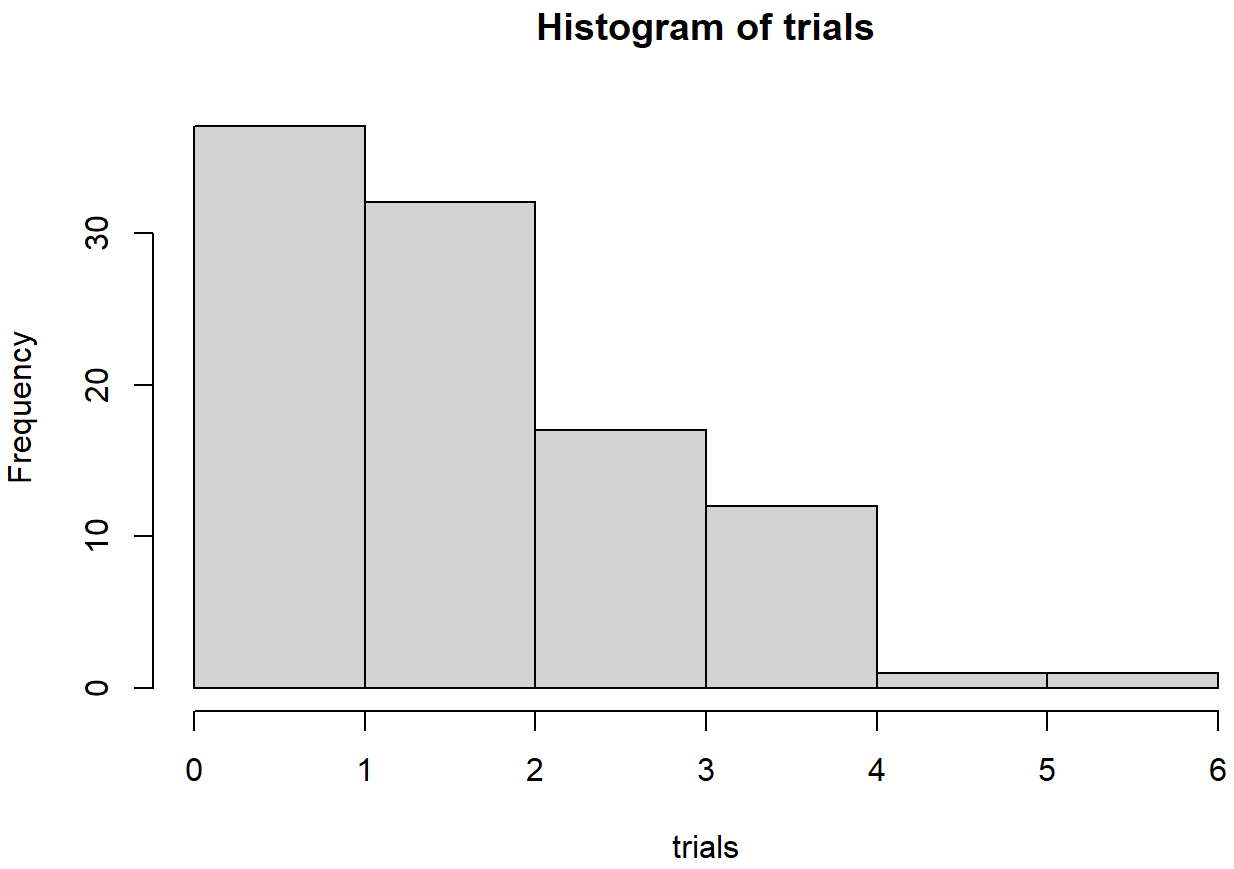
```
##  [1] 1 3 2 4 4 0 2 4 2 2
```

```
set.seed(123)
trials <- rbinom(100, 10, 0.2)
trials
```

```
##   [1] 1 3 2 4 4 0 2 4 2 2 4 2 2 2 0 4 1 0 1 4 4 3 2 6 2 3 2 2 1 1 4 4 3 3 0 2 3
##  [38] 1 1 1 1 2 2 1 1 1 1 2 1 3 0 2 3 1 2 1 1 3 4 1 2 0 2 1 3 2 3 3 3 2 3 2 3 0
##  [75] 2 1 2 2 1 1 1 2 2 3 0 2 5 4 4 1 1 2 1 2 1 1 3 0 2 2
```

Look this can make a histogram.

```
hist(trials,  breaks =6)
```



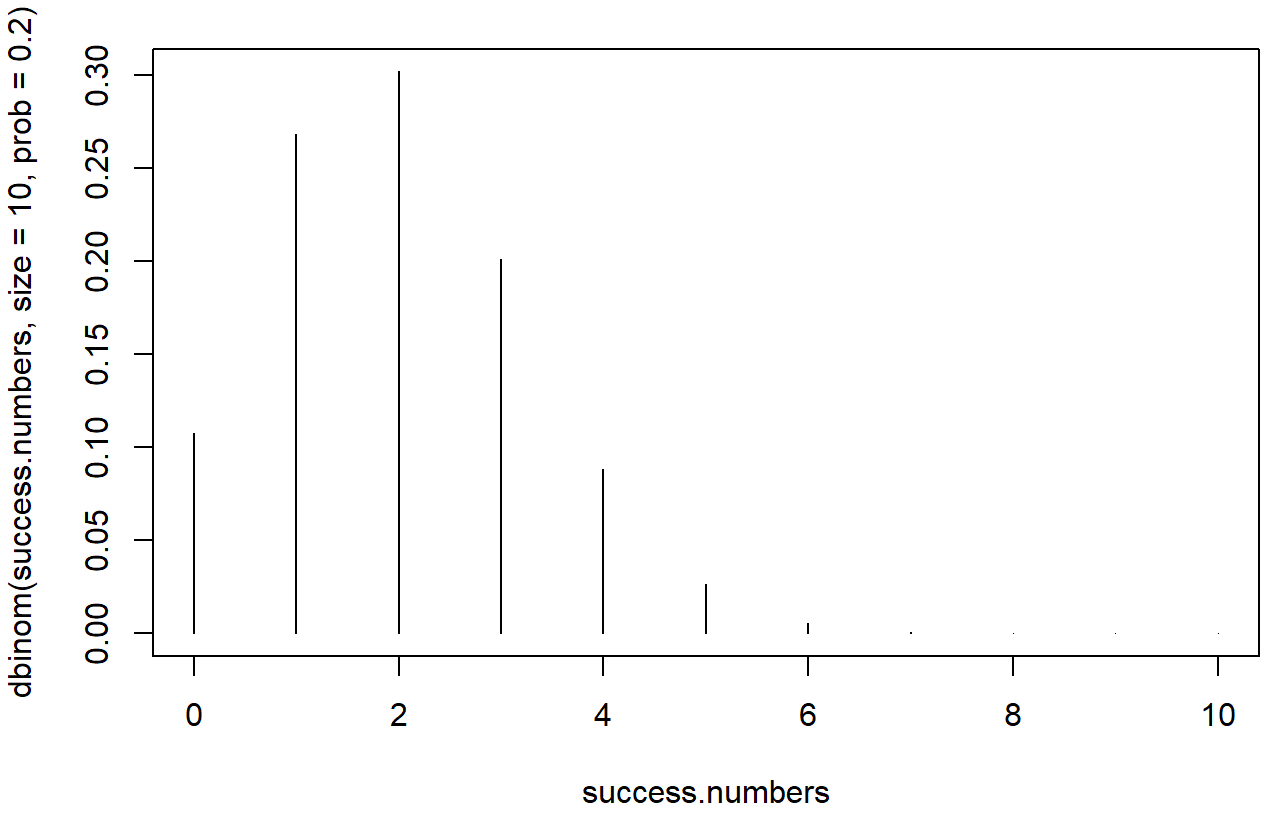

The distribution that relate binomial values according in relation to trials are binomial. You can use dbinom() to get you values in a probability distribution.

This asks something different: what is the probability that you get heads 100% of the time if you flip a coin 10 times in 10 trials.

```
dbinom(100, 10, 0.2)
```

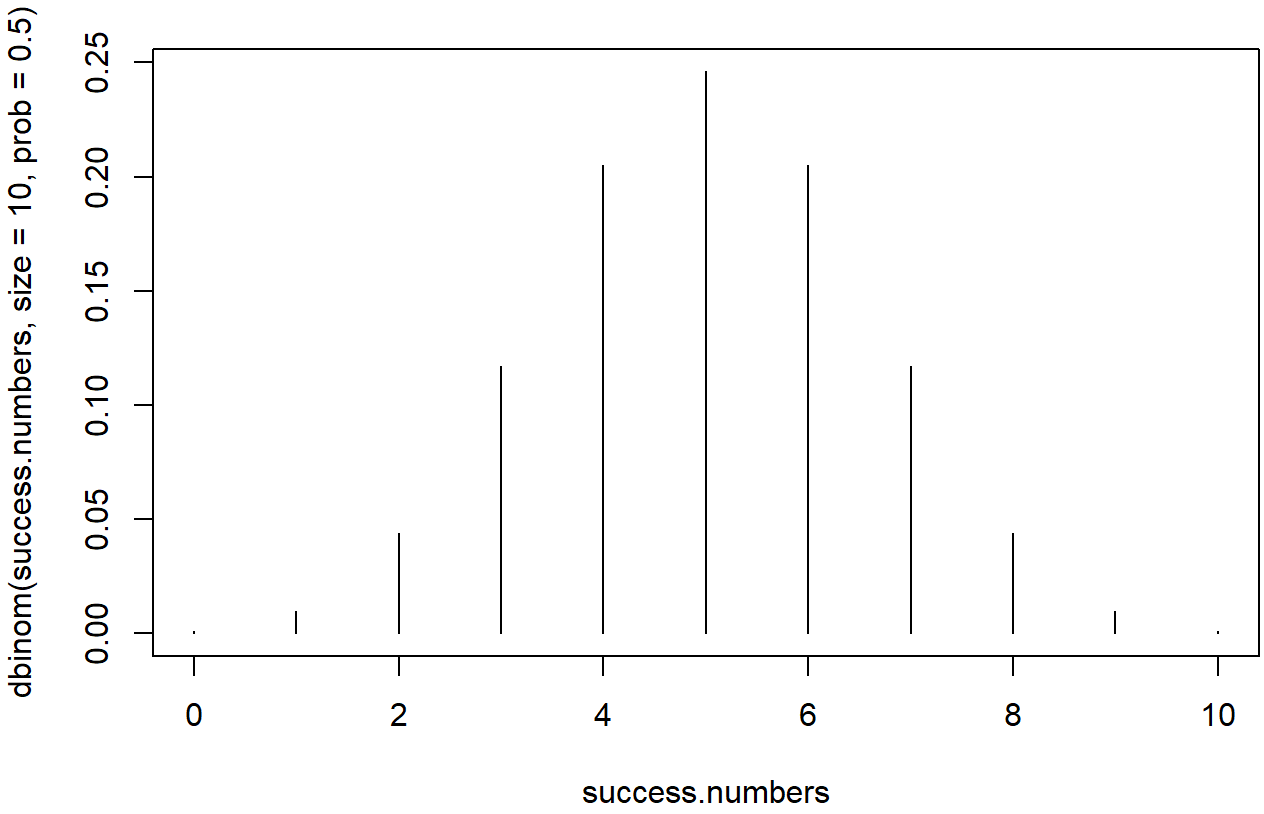
```
## [1] 0
```

```
set.seed(123)
success.numbers <- 0:100
plot(success.numbers, dbinom(success.numbers, size=10, prob=.2),type='h', xlim = c(0, 10))
```



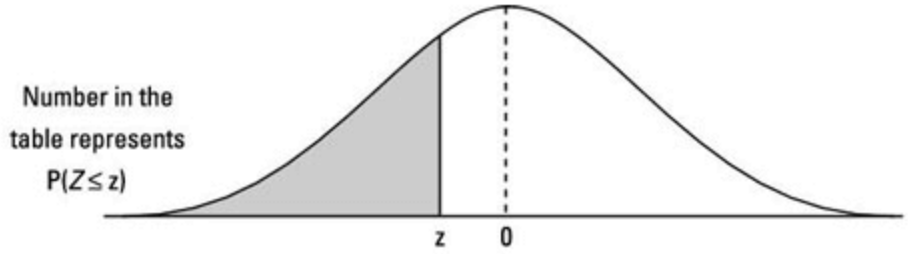

The distribution looks like this for a fair coin.

```
success.numbers <- 0:100
plot(success.numbers, dbinom(success.numbers, size=10, prob=.5),type='h', xlim = c(0, 10))
```




# T statistics to p values

You basically know the p value from the t statistic. So before computers, people had to rely on tables like this.




| z | 0.00 | 0.01 | 0.02 | 0.03 | 0.04 | 0.05 | 0.06 | 0.07 | 0.08 | 0.09 |
|---|---|---|---|---|---|---|---|---|---|---|
| –3.4 | .0003 | .0003 | .0003 | .0003 | .0003 | .0003 | .0003 | .0003 | .0003 | .0002 |
| –3.3 | .0005 | .0005 | .0005 | .0004 | .0004 | .0004 | .0004 | .0004 | .0004 | .0003 |
| –3.2 | .0007 | .0007 | .0006 | .0006 | .0006 | .0006 | .0006 | .0005 | .0005 | .0005 |
| –3.1 | .0010 | .0009 | .0009 | .0009 | .0008 | .0008 | .0008 | .0008 | .0007 | .0007 |
| –3.0 | .0013 | .0013 | .0013 | .0012 | .0012 | .0011 | .0011 | .0011 | .0010 | .0010 |
| –2.9 | .0019 | .0018 | .0018 | .0017 | .0016 | .0016 | .0015 | .0015 | .0014 | .0014 |
| –2.8 | .0026 | .0025 | .0024 | .0023 | .0023 | .0022 | .0021 | .0021 | .0020 | .0019 |
| –2.7 | .0035 | .0034 | .0033 | .0032 | .0031 | .0030 | .0029 | .0028 | .0027 | .0026 |
| –2.6 | .0047 | .0045 | .0044 | .0043 | .0041 | .0040 | .0039 | .0038 | .0037 | .0036 |
| –2.5 | .0062 | .0060 | .0059 | .0057 | .0055 | .0054 | .0052 | .0051 | .0049 | .0048 |
| –2.4 | .0082 | .0080 | .0078 | .0075 | .0073 | .0071 | .0069 | .0068 | .0066 | .0064 |
| –2.3 | .0107 | .0104 | .0102 | .0099 | .0096 | .0094 | .0091 | .0089 | .0087 | .0084 |
| –2.2 | .0139 | .0136 | .0132 | .0129 | .0125 | .0122 | .0119 | .0116 | .0113 | .0110 |
| –2.1 | .0179 | .0174 | .0170 | .0166 | .0162 | .0158 | .0154 | .0150 | .0146 | .0143 |
| –2.0 | .0228 | .0222 | .0217 | .0212 | .0207 | .0202 | .0197 | .0192 | .0188 | .0183 |
| –1.9 | .0287 | .0281 | .0274 | .0268 | .0262 | .0256 | .0250 | .0244 | .0239 | .0233 |
| –1.8 | .0359 | .0351 | .0344 | .0336 | .0329 | .0322 | .0314 | .0307 | .0301 | .0294 |
| –1.7 | .0446 | .0436 | .0427 | .0418 | .0409 | .0401 | .0392 | .0384 | .0375 | .0367 |
| –1.6 | .0548 | .0537 | .0526 | .0516 | .0505 | .0495 | .0485 | .0475 | .0465 | .0455 |
| –1.5 | .0668 | .0655 | .0643 | .0630 | .0618 | .0606 | .0594 | .0582 | .0571 | .0559 |
| –1.4 | .0808 | .0793 | .0778 | .0764 | .0749 | .0735 | .0721 | .0708 | .0694 | .0681 |
| –1.3 | .0968 | .0951 | .0934 | .0918 | .0901 | .0885 | .0869 | .0853 | .0838 | .0823 |
| –1.2 | .1151 | .1131 | .1112 | .1093 | .1075 | .1056 | .1038 | .1020 | .1003 | .0985 |
| –1.1 | .1357 | .1335 | .1314 | .1292 | .1271 | .1251 | .1230 | .1210 | .1190 | .1170 |
| –1.0 | .1587 | .1562 | .1539 | .1515 | .1492 | .1469 | .1446 | .1423 | .1401 | .1379 |
| –0.9 | .1841 | .1814 | .1788 | .1762 | .1736 | .1711 | .1685 | .1660 | .1635 | .1611 |
| –0.8 | .2119 | .2090 | .2061 | .2033 | .2005 | .1977 | .1949 | .1922 | .1894 | .1867 |
| –0.7 | .2420 | .2389 | .2358 | .2327 | .2296 | .2266 | .2236 | .2206 | .2177 | .2148 |
| –0.6 | .2743 | .2709 | .2676 | .2643 | .2611 | .2578 | .2546 | .2514 | .2483 | .2451 |
| –0.5 | .3085 | .3050 | .3015 | .2981 | .2946 | .2912 | .2877 | .2843 | .2810 | .2776 |
| –0.4 | .3446 | .3409 | .3372 | .3336 | .3300 | .3264 | .3228 | .3192 | .3156 | .3121 |
| –0.3 | .3821 | .3783 | .3745 | .3707 | .3669 | .3632 | .3594 | .3557 | .3520 | .3483 |
| –0.2 | .4207 | .4168 | .4129 | .4090 | .4052 | .4013 | .3974 | .3936 | .3897 | .3859 |
| –0.1 | .4602 | .4562 | .4522 | .4483 | .4443 | .4404 | .4364 | .4325 | .4286 | .4247 |
| –0.0 | .5000 | .4960 | .4920 | .4880 | .4840 | .4801 | .4761 | .4721 | .4681 | .4641 |

P values from the t statistic

Basically, the t statistic is some point on the x axis of a probability distribution. Then we ask what the probability is of events/numbers in before (if t is negative) or after (if t is positive) the t statistic….

It's easiest if we can simulate some values and then look at how the p values and t statistics change with respect to each other. Let's choose an imaginary linguistic example. For instance, I measure the duration of strong (stressed) vs weak (unstressed) vowels.

```
set.seed(123)
strong.vowels <- rnorm(50, mean = 90, sd = 3)
weak.vowels <- rnorm(50, mean = 70, sd = 3)
strong <- rep("yes", times =50, length.out=50)
weak <- rep("no", times =50, length.out=50)
vowel.durations <- list.append(strong.vowels, weak.vowels)
vowel.durations
```

```
##   [1] 88.31857 89.30947 94.67612 90.21153 90.38786 95.14519 91.38275 86.20482
##   [9] 87.93944 88.66301 93.67225 91.07944 91.20231 90.33205 88.33248 95.36074
##  [17] 91.49355 84.10015 92.10407 88.58163 86.79653 89.34608 86.92199 87.81333
##  [25] 88.12488 84.93992 92.51336 90.46012 86.58559 93.76144 91.27939 89.11479
##  [33] 92.68538 92.63440 92.46474 92.06592 91.66175 89.81426 89.08211 88.85859
##  [41] 87.91588 89.37625 86.20381 96.50687 93.62389 86.63067 88.79135 88.60003
##  [49] 92.33990 89.74989 70.75996 69.91436 69.87139 74.10581 69.32269 74.54941
##  [57] 65.35374 71.75384 70.37156 70.64782 71.13892 68.49303 69.00038 66.94427
##  [65] 66.78463 70.91059 71.34463 70.15901 72.76680 76.15025 68.52691 63.07249
##  [73] 73.01722 67.87240 67.93597 73.07671 69.14568 66.33785 70.54391 69.58333
##  [81] 70.01729 71.15584 68.88802 71.93313 69.33854 70.99535 73.29052 71.30554
##  [89] 69.02221 73.44642 72.98051 71.64519 70.71620 68.11628 74.08196 68.19922
##  [97] 76.56200 74.59783 69.29290 66.92074
```

```
prominence <- list.append(strong, weak)
prominence
```

```
##   [1] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [13] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [25] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [37] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [49] "yes" "yes" "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
##  [61] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
##  [73] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
##  [85] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
##  [97] "no"  "no"  "no"  "no"
```

We make a dataframe by putting these two things together.

```
set.seed(123)
simulated.vowel.data <- data.frame(vowel.durations, prominence)
summary(simulated.vowel.data)
```

```
##  vowel.durations  prominence
##  Min.   :63.07   Length:100
##  1st Qu.:70.50   Class :character
##  Median :80.33   Mode  :character
##  Mean   :80.27
##  3rd Qu.:89.77
##  Max.   :96.51
```

```
simulated.vowel.data[1:5,]
```

```
##   vowel.durations prominence
## 1        88.31857        yes
## 2        89.30947        yes
## 3        94.67612        yes
## 4        90.21153        yes
## 5        90.38786        yes
```

```
simulated.vowel.data[51:56,]
```

```
##    vowel.durations prominence
## 51        70.75996         no
## 52        69.91436         no
## 53        69.87139         no
## 54        74.10581         no
## 55        69.32269         no
## 56        74.54941         no
```

Looks like it worked. A t statistic is just calculated based on the following formula:

$$t = \frac{x_1 - x_2}{\sqrt{\frac{V_1}{N_1} + \frac{V_2}{N_2}}}$$

The formula should be understood as a formalization of the following intuitions: (i) t will be larger if the groups are more different with respect to their means; (i) t will be smaller if the variance of either of the groups is larger; (iii) t will be larger if the sample population is larger. So larger t's are more convincing evidence that there is a distinction.

```
t <- ( mean(strong.vowels) - mean(weak.vowels) ) / (sqrt( ( var(strong.vowels)/50 ) + ( var(w
eak.vowels)/50 ) ) )
t
```

```
## [1] 35.78984
```

Okay: now let's do all of this with smaller differences between the groups: what happens to the t statistic.

```
set.seed(123)
n <- 50
strong.vowels <- rnorm(n, mean = 75, sd = 3)
weak.vowels <- rnorm(n, mean = 70, sd = 3)
strong <- rep("yes", times =n, length.out=n)
weak <- rep("no", times =n, length.out=n)
vowel.durations <- list.append(strong.vowels, weak.vowels)
prominence <- list.append(strong, weak)
t <- (mean(strong.vowels) - mean(weak.vowels)) / (sqrt((var(strong.vowels)/n) + (var(weak.vow
els)/n)))
t
```

```
## [1] 8.488782
```

Let's say we increase the variance: what should we change in the simulated values and what will happen with the t statistic? We increase the variance by increasing the standard deviation (they are related numbers).

```
set.seed(123)
n <- 50
strong.vowels <- rnorm(n, mean = 75, sd = 10)
weak.vowels <- rnorm(n, mean = 70, sd = 10)
strong <- rep("yes", times =n, length.out=n)
weak <- rep("no", times =n, length.out=n)
vowel.durations <- list.append(strong.vowels, weak.vowels)
prominence <- list.append(strong, weak)
t <- (mean(strong.vowels) - mean(weak.vowels)) / (sqrt((var(strong.vowels)/n) + (var(weak.vow
els)/n)))
t
```

```
## [1] 2.118536
```

Let's see what happens to the t statistic when we increase the number of observations - the N.

```
set.seed(123)
n <- 100
strong.vowels <- rnorm(n, mean = 75, sd = 10)
weak.vowels <- rnorm(n, mean = 70, sd = 10)
strong <- rep("yes", times =n, length.out=n)
weak <- rep("no", times =n, length.out=n)
vowel.durations <- list.append(strong.vowels, weak.vowels)
prominence <- list.append(strong, weak)
t <- (mean(strong.vowels) - mean(weak.vowels)) / (sqrt((var(strong.vowels)/n) + (var(weak.vow
els)/n)))
t
```

```
## [1] 5.248661
```

The p value is calculated from the t statistic in relation to a probability density distribution.

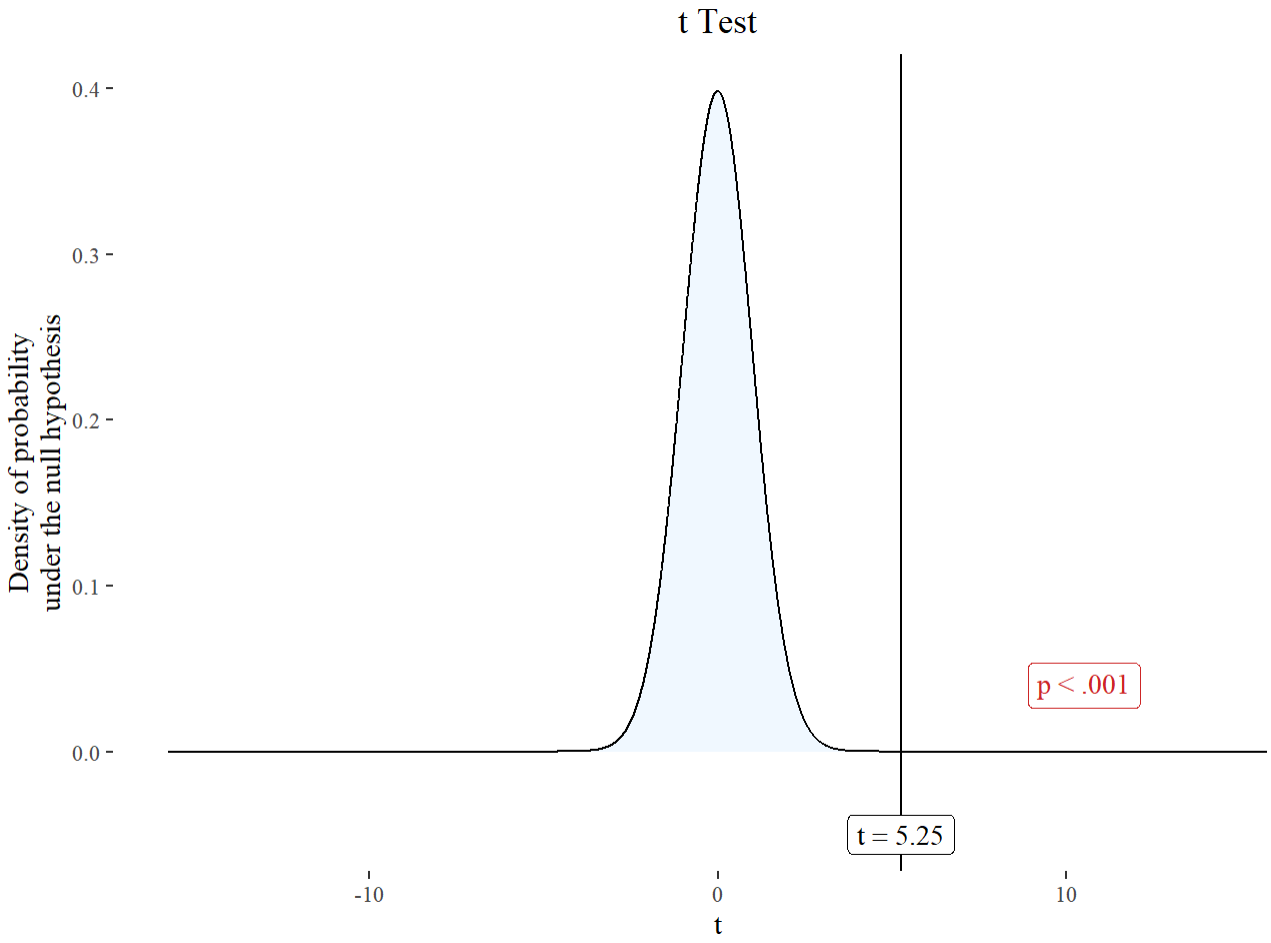
There's actually two types of p values - one tailed vs two tailed. One tailed is where you are reporting the probability of getting one side of the distribution, two tailed is when you consider both sides. I'll explain the one sided p value first. Here's the formula (df means degrees of freedom, which should be N - 1, since we now have two 100 samples, we get 198).

```
pvalue.one.sided = pt(-abs(t), df=198)
pvalue.one.sided
```

```
## [1] 1.964694e-07
```

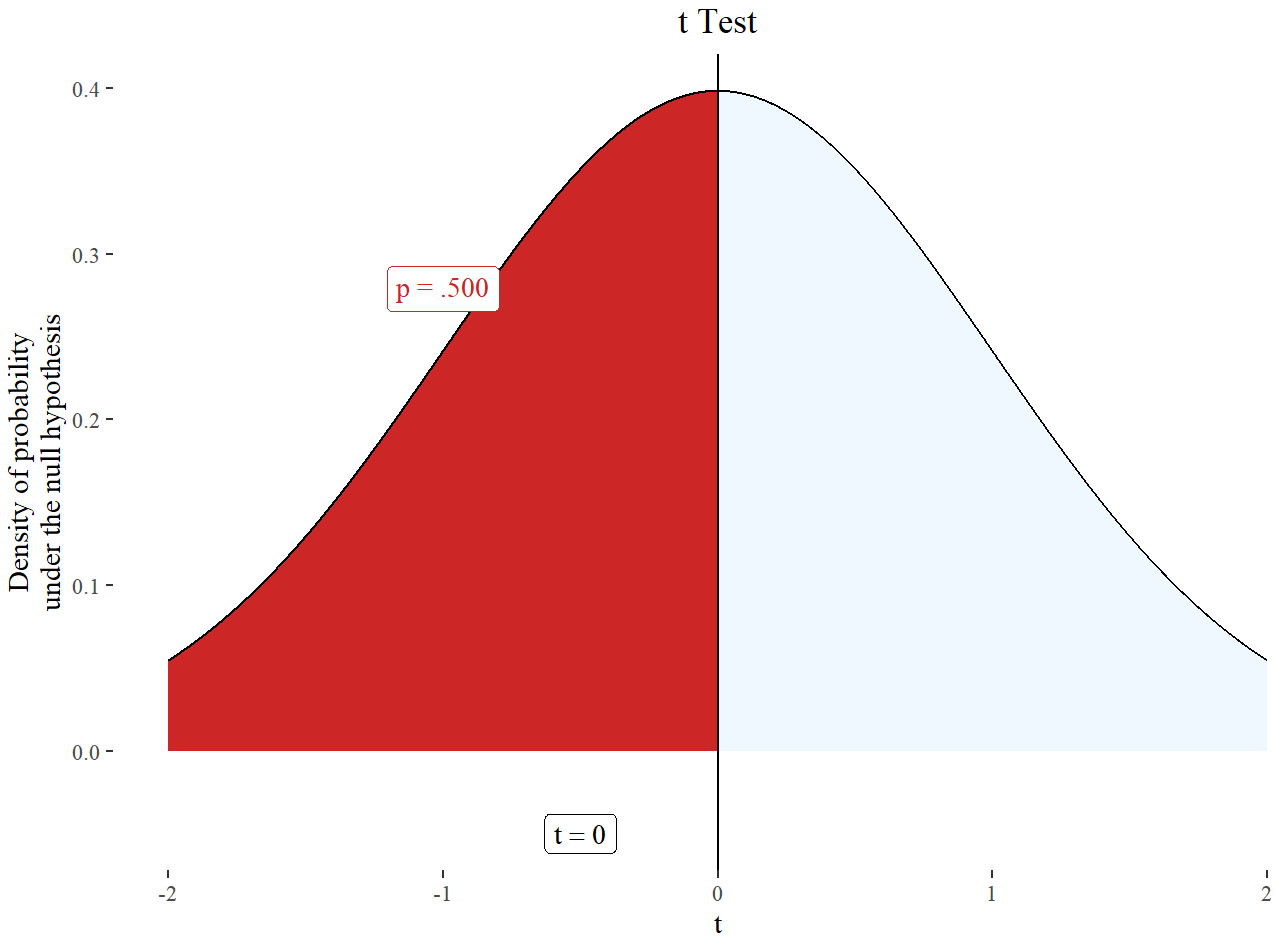
There's a function for plotting p-values (which you will never use in research, it is useful for teaching and conceptualizing what p values are).

```
plotttest(t=t, df=198, tails = "one")
```




The p value is the area in red. As t approaches 0 the p-value will get higher.

```
plotttest(t=0, df=198, tails = "one")
```

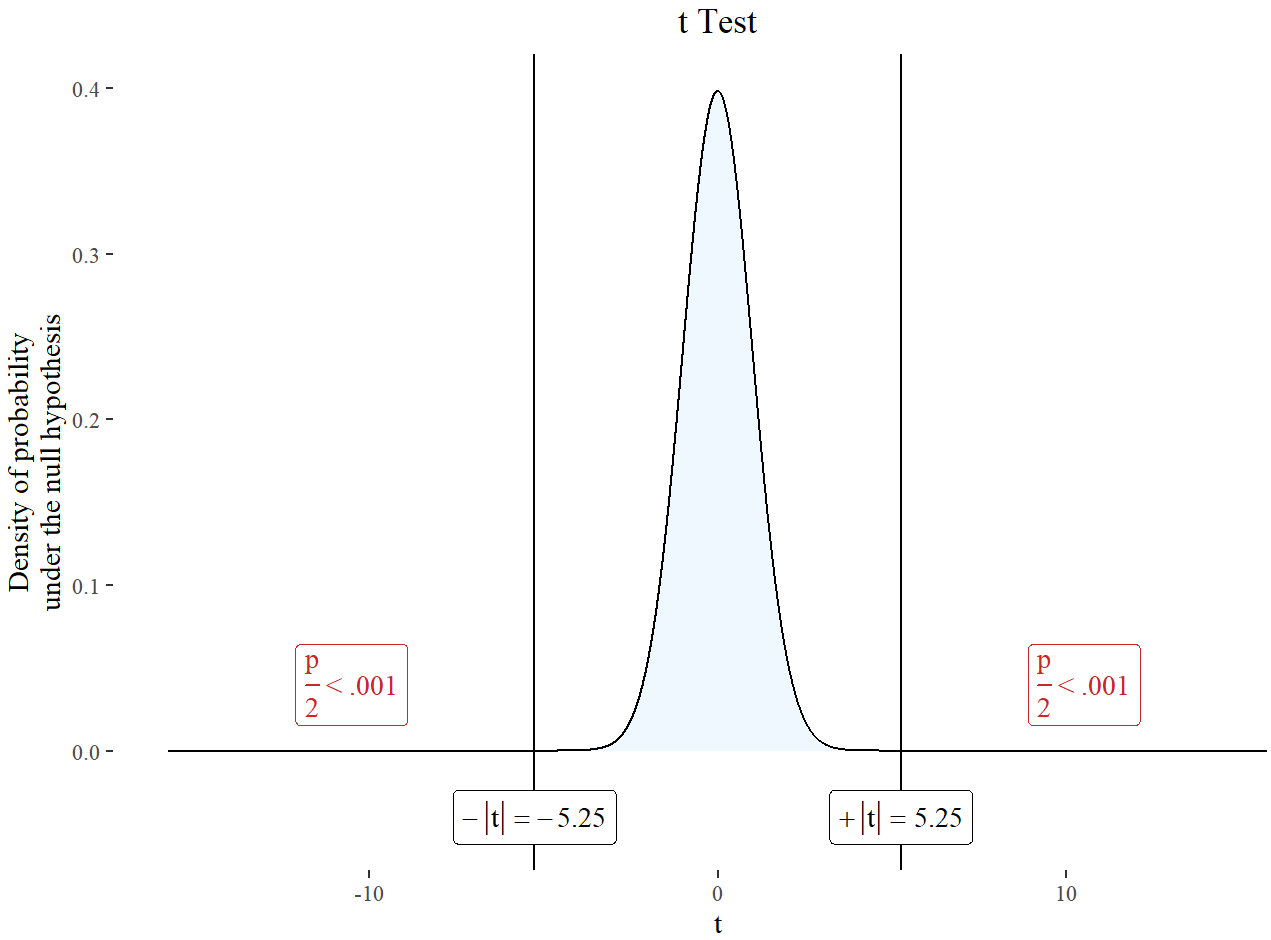



A two sided p value is where we consider both sides of the distribution. Basically it just calculates t and its negation together. To get the two sided p value you have to calculate the same value times two

```
pvalue.two.sided = 2*pt(-abs(t), df=198)
pvalue.two.sided
```

```
## [1] 3.929389e-07
```

```
plotttest(t=t, df=198, tails = "two")
```




What does this mean in terms of statistical inference? A two sided test would be used when you know the groups are different but you don't know in what direction. Say you know the vowels are different, but you don't know if strong vowels are supposed to longer or shorter than weak vowels (so really we should only be using one sided t tests).

# A worked example (araona vowels)

Lets use some real data.

```
araona.vowels <- read.csv("/Users/Adam/Desktop/Introduction to stats and R/Stats/araonavowel
s.csv")
```

```
head(araona.vowels)
```

```
##   Speaker Stress duration.ms intensity.db
## 1      MM      n       0.177           75
## 2      MM      y       0.151           76
## 3      MM      n       0.067           69
## 4      MM      n       0.220           72
## 5      MM      y       0.127           74
## 6      MM      n       0.070           70
```

```
summary(araona.vowels)
```

```
##    Speaker             Stress           duration.ms     intensity.db
##  Length:1965        Length:1965        Min.   :0.004   Min.   :30.00
##  Class :character   Class :character   1st Qu.:0.077   1st Qu.:55.00
##  Mode  :character   Mode  :character   Median :0.107   Median :71.00
##                                        Mean   :0.130   Mean   :66.44
##                                        3rd Qu.:0.149   3rd Qu.:77.00
##                                        Max.   :3.483   Max.   :89.00
```

These are just duration and intensity measurements for stress in Araona, which predictably occurs on the second syllable.

Let's separate the vowels into whether they are stressed or not.

```
araona.vowels.unstressed <- araona.vowels[araona.vowels$Stress =="n",]
araona.vowels.stressed <- araona.vowels[araona.vowels$Stress =="y",]
unstressed.durations.ms <- araona.vowels.unstressed$duration.ms
stressed.durations.ms <- araona.vowels.stressed$duration.ms
summary(unstressed.durations.ms)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0040  0.0700  0.0980  0.1238  0.1370  3.4830
```

$$t = \frac{x_1 - x_2}{\sqrt{\frac{V_1}{N_1} + \frac{V_2}{N_2}}}$$

```
t.stat.numerator <- mean(stressed.durations.ms) - mean(unstressed.durations.ms)
t.stat.numerator
```

```
## [1] 0.01193075
```

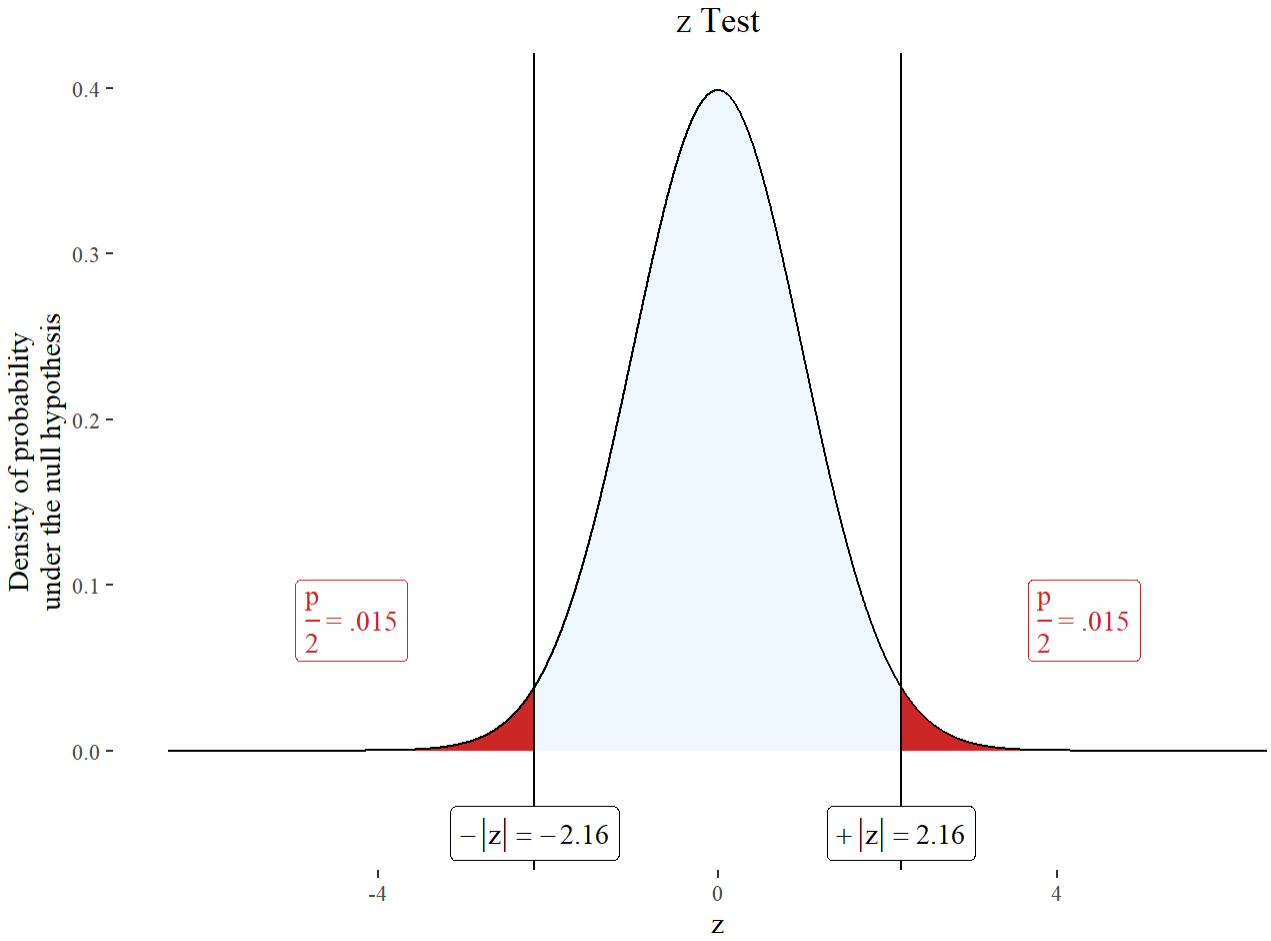
```
t.stat.denominator <- sqrt( ( var(stressed.durations.ms) / length(stressed.durations.ms) ) +
( var(unstressed.durations.ms) / length(unstressed.durations.ms) ) )
t.stat.denominator
```

```
## [1] 0.005522873
```

```
t.stat <- t.stat.numerator / t.stat.denominator
t.stat
```

```
## [1] 2.160243
```

```
plotztest(t.stat, tails = "two")
```




We can also do this test using t.test() function as in the following:

```
t.test(stressed.durations.ms,unstressed.durations.ms, var.equal = FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  stressed.durations.ms and unstressed.durations.ms
## t = 2.1602, df = 1793.8, p-value = 0.03089
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.001098805 0.022762687
## sample estimates:
## mean of x mean of y
## 0.1357015 0.1237708
```

# Null hypothesis testing and p-hacking

The terminology here is as follows: \

*null hypothesis*: there is no difference between your groups. *alternative to the null hypothesis*: the hypothesis you are interested in testing. *alpha*: the value your p value has to be lower than for you to reject the null hypothesis.

Alpha is typically set at 0.05 - which is arbitrary. It was invented by R.A. Fischer. When the p value is below your alpha you say the results are “statistically significant” … which is a really misleading way of saying “of low probability in relation to an arbitrary selected sampling statistic (imaginary probability distribution)”. \

There’s some more terminology used in “frequentist” statistics (from Pearson & Neyman rather than Ronald Fischer): \

*Type 1 error*: You reject the null, but your hypothesis is wrong (false positive) *Type 2 error*: Your don’t reject the null, but your hypothesis is true (true negative)

When you set the alpha level at 0.05, you are saying that you will commit a Type 1 error 5% of the time. Every time you do a test the probability that you will commit a Type 1 error increases.

The classical problem with statistical testing is that it does not easily control for the possibility of multiple testing. *P-hacking* refers to cases where researchers cheat (probably inadvertently) in order to find a statistically significant result.

*P-hacking* isn’t just a product of the logic of statistical testing - its also a problem with the way publications and career advancement work in many fields. “Null results” - that is cases where no effect was found - are often not considered to be worthy of publication. Since scientists need publications to further their careers, they are encouraged to figure out ways to make their results significant without following the logic of frequentist statistical testing (Stuart Ritchie (2020) “Science Fictions: How, Fraud, Bias, Negligence, and Hype Undermine the Search for Truth”).

Another problem is that scientists are biased: if they don’t get a result they like they assume the test was done wrong and try again and again to get the result they are looking for. Often scientific hypotheses are vague such that they can be compatible (or seem to be compatible) with multiple different statistical hypotheses. If you don’t get the right result, just try a different version of the same hypothesis. Since increases the probability of finding a result by accident. Remember based on what the p-value is, we should expect to find an accidental statistically significant value 5% of the time.

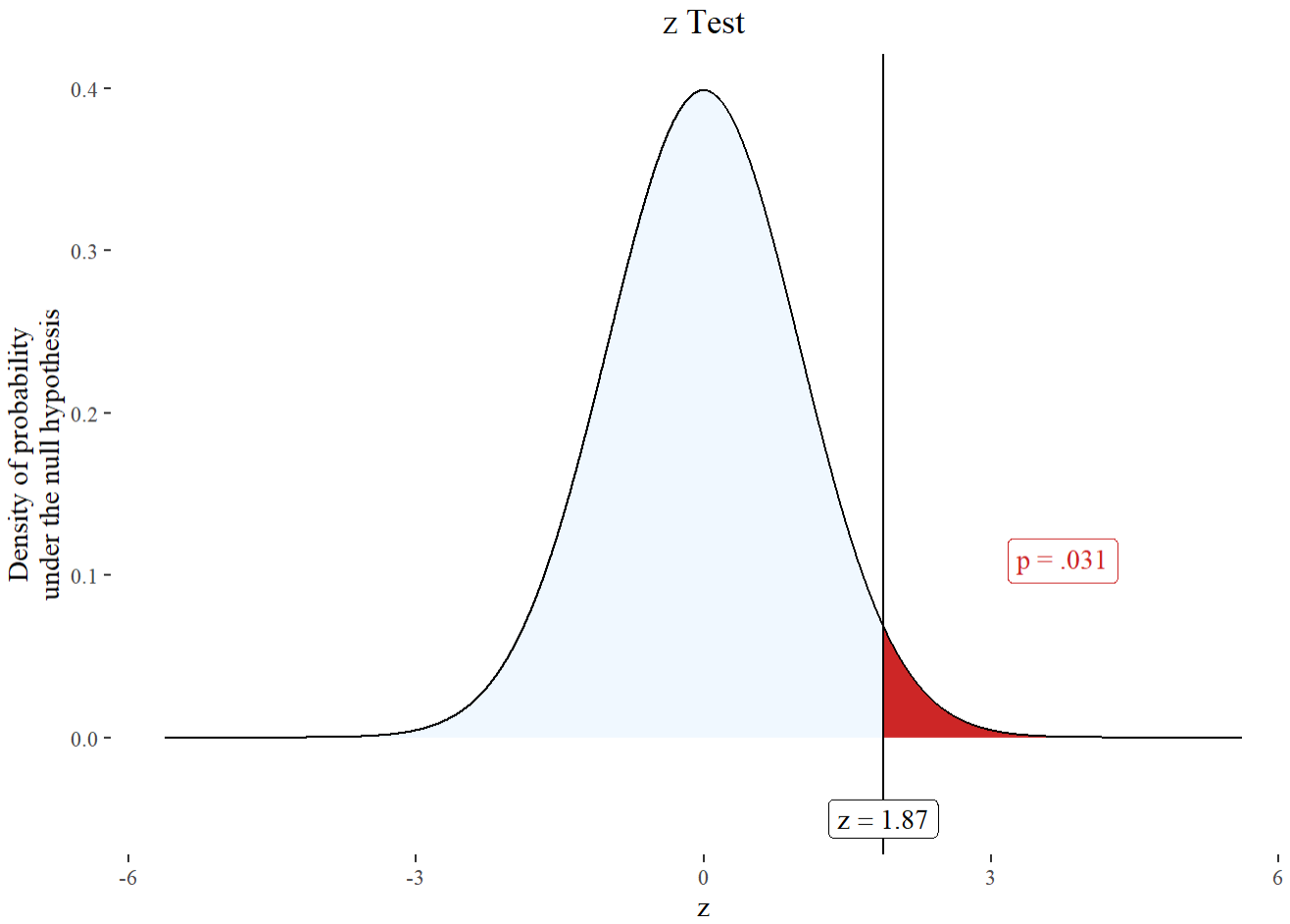
Another reason p-hacking results is because the p-values don’t make much sense philosophically. Why are we trying to ask questions about the probability of our data instead of the probability of our hypothesis? There’s a lot of discussion about misinterpreting statistical significance, but there is also reason to believe that p-values were designed to be misinterpreted (Aubrey Clayton (2021) “Bernouilli’s Fallacy”).

It’s actually fairly straight forward to simulate how p-hacking actually works. First, you should understand that p-values can be high just because you don’t have enough data.

```
set.seed(123)
n <- 6
strong.vowels <- rnorm(n, mean = 85, sd = 10)
weak.vowels <- rnorm(n, mean = 80, sd = 10)
strong <- rep("yes", times =n, length.out=n)
weak <- rep("no", times =n, length.out=n)
vowel.durations <- list.append(strong.vowels, weak.vowels)
prominence <- list.append(strong, weak)
t <- (mean(strong.vowels) - mean(weak.vowels)) / (sqrt((var(strong.vowels)/n) + (var(weak.vow
els)/n)))
t
```

```
## [1] 1.873338
```

```
plotztest(t, tails = "one")
```



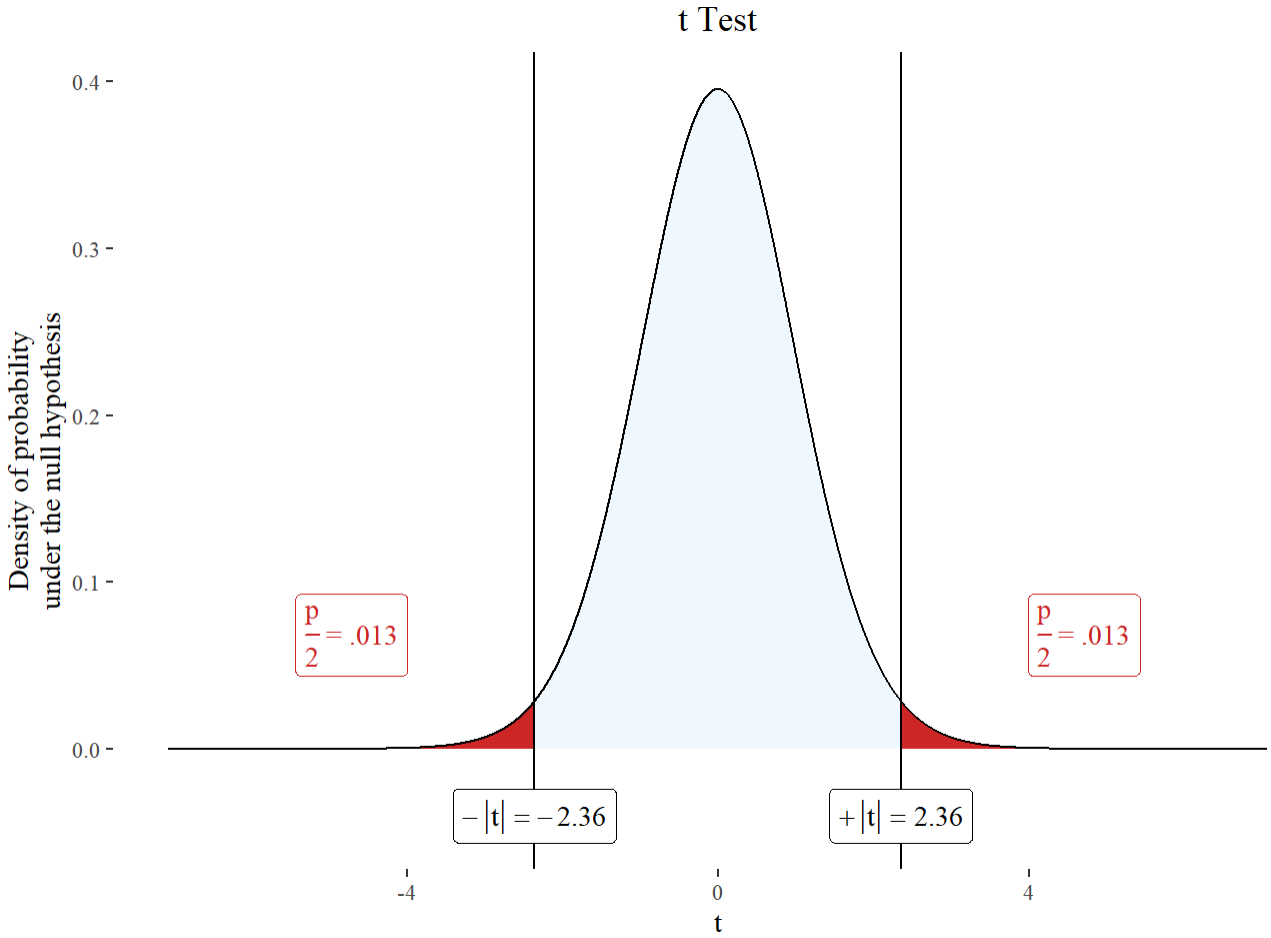

Not significant! And we know there should be a difference because we simulated the data to be different. So here we have created a *Type 2 error*, just by having low N.

We can simulate the opposite situation.

So, let's imagine we have *a lot* of possible hypotheses to test.

```
set.seed(1234)
n <- 30
vowel.durations <- rnorm(n, mean = 80, sd=10)
h1 <- c("strong","weak")
h1 <- sample(h1, n, replace=TRUE, prob=c(0.5, 0.5))
h2 <- c("strong","weak")
h2 <- sample(h2, n, replace=TRUE, prob=c(0.5, 0.5))
h3 <- c("strong","weak")
h3 <- sample(h3, n, replace=TRUE, prob=c(0.5, 0.5))
h4 <- c("strong","weak")
h4 <- sample(h4, n, replace=TRUE, prob=c(0.5, 0.5))
h5 <- c("strong","weak")
h5 <- sample(h5, n, replace=TRUE, prob=c(0.5, 0.5))
h6 <- c("strong","weak")
h6 <- sample(h6, n, replace=TRUE, prob=c(0.5, 0.5))
df <- data.frame(vowel.durations, h1,h2,h3,h4,h5,h6)
```

```
plotttest(t.test(vowel.durations~h4, data=df))
```




If you followed my formula here exactly, this one should have came out significant. But this is actually because I tried multiple different random groupings until I got one that is significant.

# (95%) confidence intervals

``A confidence interval shows the likely range in which the mean would fall if the sampling exercise were to be repeated" (Crawley 2015:61)

The interval gets wider as unreliability goes up.

If I say I think that between 35 - 40% of people will vote for Angela Merkel's party in a 95% confidence interval it means: Under repeated sampling in identical conditions, the true value would be between 35 - 40%, 95% of the time.

The formula for the confidence interval for a mean value is calculated as follows.

$$CI = \overline{x} \pm t_{\frac{1-p}{2},df} \times SE$$

If we are comparing means then its

$$CI = \overline{x}_1 - \overline{x}_2 \pm t * \sqrt{\frac{s_p}{n_1} + \frac{s_p}{n_2}}$$

$$SE = \frac{V}{N} = \frac{SD^2}{N}$$

What would it mean for a confidence interval to straddle zero? \

It would mean that there is basically no difference between the groups, or that the difference / relationship is just as likely to be positive as negative. \

What does it mean if the interval is relatively large? \

The larger the interval the less certain you are.

```
set.seed(123)
n <- 20
strong.vowels <- rnorm(n, mean = 90, sd = 5)
weak.vowels <- rnorm(n, mean = 80, sd = 5)
strong <- rep("yes", times =n, length.out=n)
weak <- rep("no", times =n, length.out=n)
vowel.durations <- list.append(strong.vowels, weak.vowels)
prominence <- list.append(strong, weak)
df <- data.frame(vowel.durations, prominence)
t <- (mean(strong.vowels) - mean(weak.vowels)) / (sqrt((var(strong.vowels)/n) + (var(weak.vow
els)/n)))
t
```

```
## [1] 7.669864
```

First we calculate the pooled variance, that's what sp stands for the in formula above.

```
pooled.variance = ((n-1)*var(strong.vowels)+(n-1)*var(weak.vowels))/(n+n-2)
```

Then we calculate the margin of error (qt() spits out the inverse probability cumulative density of the Student t-distribuion)

```
margin <- qt(0.975,df=n+n-1)*sqrt(pooled.variance/n + pooled.variance/n)
```

From the margin we can calculate the lower and the higher interval.

```
lowerinterval <- (mean(strong.vowels) - mean(weak.vowels)) - margin
upperinterval <- (mean(strong.vowels) - mean(weak.vowels)) + margin
```

```
lowerinterval
```

```
## [1] 8.07288
```

```
upperinterval
```

```
## [1] 13.85593
```

```
t.test(vowel.durations~prominence, data=df)
```

```
##
##  Welch Two Sample t-test
##
## data:  vowel.durations by prominence
## t = -7.6699, df = 37.082, p-value = 3.648e-09
## alternative hypothesis: true difference in means between group no and group yes is not equ
al to 0
## 95 percent confidence interval:
##  -13.86072  -8.06809
## sample estimates:
##  mean in group no mean in group yes
##          79.74371          90.70812
```

The t.test function flipped the signs. But that's okay. Its really the same result. Note that some researchers consider the confidence intervals to be more informative than the p value. I tend to agree.

``On all counts, the confidence intervals seems clearly superior to the traditional significance testing approach. Significance tests focus on just one null hypothesis value, whereas confidence intervals display the entire range of hypothetical values of a parameter that cannot be rejected. Although the confidence interval can be used as a significance test, there is no need to it to be used that way, and in fact such an interpretation willfully ignores most of the information being provided by a confidence interval… Confidence intervals enhance comparisons between research replications and enable researchers to move toward cumulative knowledge based on replications. Significance tests are not readily able to do this and, as has been argued by various authors (…), significance tests may actually blind researchers to cumulative evidence." (Smithson 2003: 12)

``…significance tests fail to contribute to cumulative knowledge, whereas confidence intervals pave the way. There are several reasons for this, but the main reason is that merely reporting a p value (or worse still, whether or not a result is significant at the 0.05 level) obscures all the factors that might determine significance,such as sample size and effect size. The confidence interval alerts the research to both of these,

as well as enabling realistic comparisons between studies” (Smithson 2003: 13-14)