

# Table of contents

1. Introduction: You are already an experimentalist
2. Conditions
3. Items
4. Ordering items for presentation
5. Judgment Tasks
6. Recruiting participants
7. Pre-processing data (if necessary)
8. Plotting
9. Building linear mixed effects models
10. Evaluating linear mixed effects models using Fisher
11. Neyman-Pearson and controlling error rates
12. Bayesian statistics and Bayes Factors
13. Validity and replicability of judgments
14. The source of judgment effects
15. Gradience in judgments

Section 1:  
Design

Section 2:  
Analysis

Section 3:  
Application

# The most important lesson in stats: Statistics is a field of study, not a tool

Statistics is its own field. There is a ton to learn, and more is being discovered every day. Statisticians have different philosophies, theories, tastes, etc. They can't tell you the "correct" theory any more than we can tell them the "correct" theory of linguistics.

What we want to do is take this large and vibrant field, and convert it into a tool for us to use when we need it. This is a category mismatch.



Imagine if somebody tried to do that with linguistics. We would shake our heads and walk away...

But statistics is in a weird position, because other sciences do need the tools that they develop to get work done. And statistics wants to solve those problems for science. So we have to try to convert the field into a set of tools.

# What you will run for (most) papers

Obviously, I am not qualified to teach you the actual field of statistics. And there is no way to give you a complete understanding of the “tool version” of statistics that we use in experimental syntax in the time we have here.

So here is my idea. I am going to start by showing you the R commands that you are going to run for (most) of your experimental syntax papers. Then we will work backwards to figure out exactly what information these commands are giving you.

## 1. Load the lmerTest package

```
library(lmerTest)
```

## 2. Create a linear mixed effects model with your fixed factors (e.g., factor1 and factor2) and random factors for subjects and items.

```
model.lmer=lmer(responseVariable~factor1*factor2 + (1+factor1*factor2|  
subject) + (1|item), data=yourDataset)
```

## 3. Run the anova() function to derive F statistics and p-values using the Satterthwaite approximation for degrees of freedom.

```
anova(model.lmer)
```

# The results for our data

If we run the following code in the script called `linear.mixed.effects.models.r`:

```
wh.lmer = lmer(zscores~embeddedStructure*dependencyLength + (1|subject)
+ (1|item), data=wh)
```

And then use the `summary()` and `anova()` functions, we get the following results:

`summary(wh.lmer)`

```
Fixed effects:
              Estimate Std. Error      df t value Pr(>|t|)
(Intercept)    0.81129    0.06415 171.34000  12.647 < 2e-16 ***
embeddedStructure2 -0.25244    0.08789 192.15000  -2.872 0.004531 **
dependencyLength2 -0.34913    0.08789 192.15000  -3.973 0.000101 ***
embeddedStructure2:dependencyLength2 -1.00138    0.12458 192.33000  -8.038 9.06e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

`anova(wh.lmer)`

```
Analysis of Variance Table of type III with Satterthwaite
approximation for degrees of freedom
              Sum Sq Mean Sq NumDF  DenDF  F.value    Pr(>F)
embeddedStructure    31.616   31.616     1 192.32  146.189 < 2.2e-16 ***
dependencyLength     40.255   40.255     1 192.32  186.133 < 2.2e-16 ***
embeddedStructure:dependencyLength 13.973   13.973     1 192.32   64.611 9.048e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this section we want to try to understand what the model above is modeling, and what the information in the summaries is telling us.

# Theories, models, and hypothesis tests

## **Substantive Theories**

As scientists, theories are what we really care about. Substantive theories are written in the units of that science; e.g., syntactic theories are written in terms of features, operations, tree-structures, etc.

## **Mathematical Models**

We want to find evidence for our theories. But what counts as evidence? One possible answer (among many) is: (i) a successful theory will predict observable data, therefore (ii) we can use a measure of how well a theory predicts the data as evidence for/against a theory. If we adopt this view, we need to link our theories to observable data in a way that lets us quantify that relationship. In short, we need a mathematical model that relates our theory to the data. This opens up lots of doors for us. We can create metrics to evaluate how good a model is, and compare models for goodness. And we can use probability theory to answer questions like “how likely is this data given this theory?”, “how likely is this theory given this data?”.

## **Hypothesis Tests**

Once we have models, and metrics for comparing them, we may want to formalize a criterion for choosing one model/theory over another. In other words, a test.

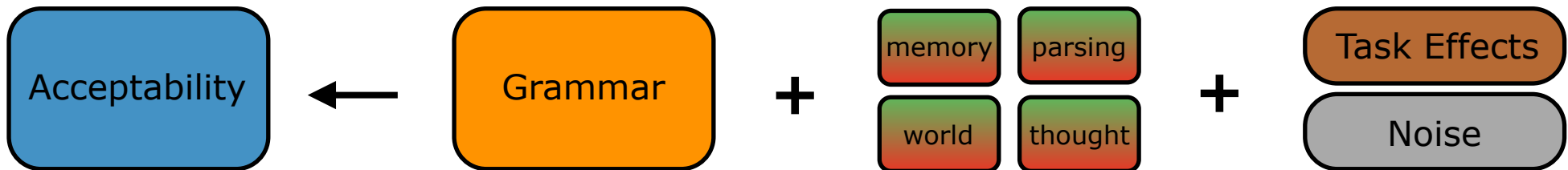
# Constructing a model for our theory

## The theory of wh-islands:

Our theory is that there is constraint on the extraction of wh-words out of embedded questions.

## Our model:

We already have a model in mind for our theory. We think that this constraint will affect acceptability. So we need a model of acceptability that has a spot for this constraint.



So all we need to do is translate this model of acceptability into a specific equation for our experiment. Here is what it is going to look like:

$$\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(0,1)} + \beta_2 \text{dependency}_{(0,1)} + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_i$$

Now let's spend the next several slides building this equation so you can see where it came from.

# This is a model to predict every judgment

We have 224 judgments in our dataset. We want a model that can explain every one of them. We capture this with the  $i$  subscript:

$$\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(0,1)} + \beta_2 \text{dependency}_{(0,1)} + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_i$$



This is shorthand for 1 to 224

$$\text{acceptability}_1 = \beta_0 + \beta_1 \text{structure}_0 + \beta_2 \text{dependency}_0 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_1$$

$$\text{acceptability}_2 = \beta_0 + \beta_1 \text{structure}_0 + \beta_2 \text{dependency}_1 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_2$$

$$\text{acceptability}_3 = \beta_0 + \beta_1 \text{structure}_1 + \beta_2 \text{dependency}_0 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_3$$

$$\text{acceptability}_4 = \beta_0 + \beta_1 \text{structure}_1 + \beta_2 \text{dependency}_1 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_4$$

...

$$\text{acceptability}_{224} = \beta_0 + \beta_1 \text{structure}_1 + \beta_2 \text{dependency}_1 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_{224}$$

Also notice that when we write out the individual equations for each judgment in our dataset, certain other numbers become concrete. The subscript on the structure and dependencies factors becomes a specific number (0 or 1), and the  $i$  subscript on the  $\varepsilon$  term takes the same value as the judgment.

# Coding the variables

The factors in our experiment are **categorical** (non-island/island, short/long).

Categorical variables can either be turned into 0 and 1 (treatment coding), or into -1 and 1 (effects coding). There is a difference between them that we will talk about in a few minutes. But for now, let's choose 0 and 1, like so:

## **structure**

non-island = 0

island = 1

## **dependency**

short = 0

long = 1

Now look at the first four equations below. Can you see which condition each one represents?

$$\text{acceptability}_1 = \beta_0 + \beta_1 \text{structure}_0 + \beta_2 \text{dependency}_0 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_1$$

$$\text{acceptability}_2 = \beta_0 + \beta_1 \text{structure}_0 + \beta_2 \text{dependency}_1 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_2$$

$$\text{acceptability}_3 = \beta_0 + \beta_1 \text{structure}_1 + \beta_2 \text{dependency}_0 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_3$$

$$\text{acceptability}_4 = \beta_0 + \beta_1 \text{structure}_1 + \beta_2 \text{dependency}_1 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_4$$

...

The first is **non-island** because its structure is 0, and it is **short** because its dependency is also 0. The fourth is **island** because its structure is 1, and it is **long** because its dependency is 1.



# What are the Betas?

The betas in this equation are coefficients. They are the numbers that turn the 0s and 1s into an actual effect on acceptability.

The idea is that you multiply the beta by the 0 or 1 in the factor to get an effect. So when the factor is 0, there is no effect. And when the factor is 1, you get an effect that is the same size as the beta.

$$\text{acceptability}_1 = \beta_0 + \beta_1 \text{structure}_0 + \beta_2 \text{dependency}_0 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_1$$

$$\text{acceptability}_2 = \beta_0 + \beta_1 \text{structure}_0 + \beta_2 \text{dependency}_1 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_2$$

$$\text{acceptability}_3 = \beta_0 + \beta_1 \text{structure}_1 + \beta_2 \text{dependency}_0 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_3$$

$$\text{acceptability}_4 = \beta_0 + \beta_1 \text{structure}_1 + \beta_2 \text{dependency}_1 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_4$$

...

It is important to note that each beta is constant.  $\beta_1$  is always  $\beta_1$ . It doesn't have another subscript that varies for each judgment (unlike the  $\varepsilon$  term). This is why each beta can be seen as an effect.

$\beta_1$  is the effect of having **island** structure.

$\beta_2$  is the effect of having a **long** dependency.

# structure<sub>1</sub>:dependency<sub>1</sub> is the violation

The structure<sub>1</sub>:dependency<sub>1</sub> term looks strange because it is the **interaction term** (the colon is a way of notating this). It is the special extra effect that occurs when the levels of the two factors are both 1. Basically, you multiply the two numbers together (0\*0, 0\*1, 1\*0, or 1\*1), and then multiply the result by  $\beta_3$ .

In our **substantive theory**, this mathematical term captures the effect of a **violation**. The **island/long** condition (1,1) is the only condition that meets the structural description of the island constraint.

$$\text{acceptability}_1 = \beta_0 + \beta_1\text{structure}_0 + \beta_2\text{dependency}_0 + \beta_3\text{structure}_0:\text{dependency}_0 + \varepsilon_1$$

$$\text{acceptability}_2 = \beta_0 + \beta_1\text{structure}_0 + \beta_2\text{dependency}_1 + \beta_3\text{structure}_0:\text{dependency}_1 + \varepsilon_2$$

$$\text{acceptability}_3 = \beta_0 + \beta_1\text{structure}_1 + \beta_2\text{dependency}_0 + \beta_3\text{structure}_1:\text{dependency}_0 + \varepsilon_3$$

$$\text{acceptability}_4 = \beta_0 + \beta_1\text{structure}_1 + \beta_2\text{dependency}_1 + \beta_3\text{structure}_1:\text{dependency}_1 + \varepsilon_4$$

...

The interaction term does nothing for the first three conditions, because it is equivalent to a 0 then. In the fourth condition (1,1) it is a 1. In this condition, that 1 is multiplied by  $\beta_3$  to add to the effect. This means that  **$\beta_3$  is the size of the violation effect** (it is the **DD score** from earlier!). Note that this is only true with treatment (0,1) coding. The coefficients have different interpretations with different codings.

# $\varepsilon$ is the error term

If you just look at the betas and factors, you will quickly see that we can only generate 4 acceptability judgments: one for each condition in our experiment (00, 01, 10, 11). But we have 224 values that we need to model. And that is where the  $\varepsilon$  term comes in.

The  $\varepsilon$  term is an error term. It is the difference between the value that the model predicts and the actual value of the judgment. This is why it varies in its subscript: we need a different  $\varepsilon$  term for each judgment.

$$\text{acceptability}_1 = \beta_0 + \beta_1 \text{structure}_0 + \beta_2 \text{dependency}_0 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_1$$

$$\text{acceptability}_2 = \beta_0 + \beta_1 \text{structure}_0 + \beta_2 \text{dependency}_1 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_2$$

$$\text{acceptability}_3 = \beta_0 + \beta_1 \text{structure}_1 + \beta_2 \text{dependency}_0 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_3$$

$$\text{acceptability}_4 = \beta_0 + \beta_1 \text{structure}_1 + \beta_2 \text{dependency}_1 + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_4$$

...

This may seem like a hack, but it is principled. The other parts of our model capture the things that we manipulated in our experiment. The error term captures all of the things that we couldn't control: individual differences in the participants, differences in the items, effects of the task, etc. (And we will see later that we can model some of these things, at least a little bit).

# We minimize the $\varepsilon$ 's to estimate $\beta$ 's

Once you've specified your model (as we have here), the next step is to find the coefficients that make for a good model.

One way to define "good" is to say that a good model will minimize the amount of stuff that is unexplained. Well, all of our unexplained stuff is captured by the  $\varepsilon$  terms, so this means that we want to minimize  $\varepsilon$ .

Here is a toy example with 3 values and a simple model with only one beta:

Let's imagine we have three judgments to model (2,3,4). If we choose the value 4 for the coefficient of  $\beta_0$ , we get  $\varepsilon$  terms (-2, -1, 0), which we can square and sum to derive a sum of squares.

$acc_i = \beta_0$	+	$\varepsilon_i$	SS=5
2 = 4	+	-2	
3 = 4	+	-1	
4 = 4	+	0	

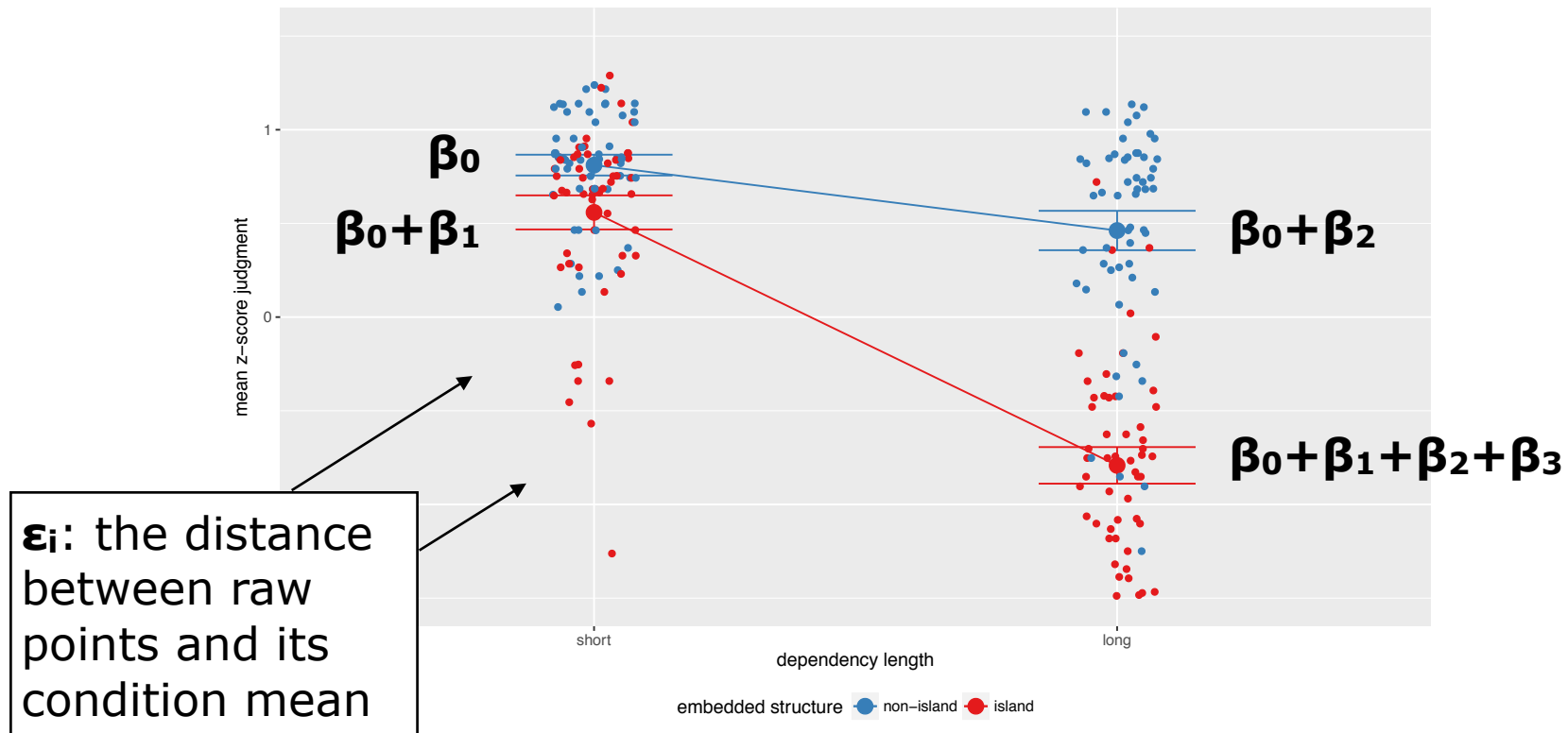
Now, let's imagine we have the same data, but we choose 3 for the coefficient of  $\beta_0$ . Now we get smaller error terms, and consequently a smaller SS. This is a better model, because less is unexplained.

$acc_i = \beta_0$	+	$\varepsilon_i$	SS=2
2 = 3	+	-1	
3 = 3	+	0	
4 = 3	+	1	

# Putting it all together

You specify the model for R. That was the command we entered into the console. R will then find the best value of the coefficients for the data that you gave it.

$$\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(0,1)} + \beta_2 \text{dependency}_{(0,1)} + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_i$$



And you might recall that this is exactly the 2x2 logic that we discussed earlier.

# The R command

Now that we understand our linear model, we can compare it to the R command that we ran at the beginning of this section. I will color parts so that you can see the correspondence:

$$\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(0,1)} + \beta_2 \text{dependency}_{(0,1)} + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_i$$



```
lmer(zscores ~ embeddedStructure + dependencyLength + embeddedStructure:dependencyLength +  
(1|subject) + (1|item), data=wh)
```

You don't need to specify the intercept ( $\beta_0$ ) in the command. R includes one by default (you can, however, tell it not to estimate an intercept if you want).

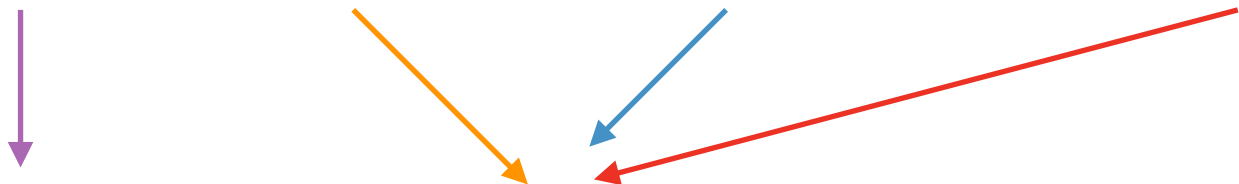
You don't need to specify the error term ( $\varepsilon_i$ ) in the command. Again, R includes one by default.

You will also notice that lmer() formula contains extra bits: (1|subject) and (1|item). That is because the top model only has **fixed effects**. The (1|subject) and (1|item) terms are **random effects**. We will turn to those next.

# The R command - a shortcut

You may have noticed that the command I just showed you is not exactly the command in the script (or on the slide at the beginning of this section). That is because there is a shortcut in R for specifying two factors and an interaction:

$$\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(0,1)} + \beta_2 \text{dependency}_{(0,1)} + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_i$$



```
lmer(zscores ~ embeddedStructure * dependencyLength + (1|subject) + (1|item), data=wh)
```

When you want all three effects, you can use the `*` operator instead of a `+`. R will automatically expand this to all three components:

embeddedStructure  
dependencyLength  
embeddedStructure:dependencyLength

It is a nice shortcut that really saves you time if you have more than two factors, because they grow in squares (remember, a 2x2x2 will have 8 components, and 2x2x2x2 will have 16).

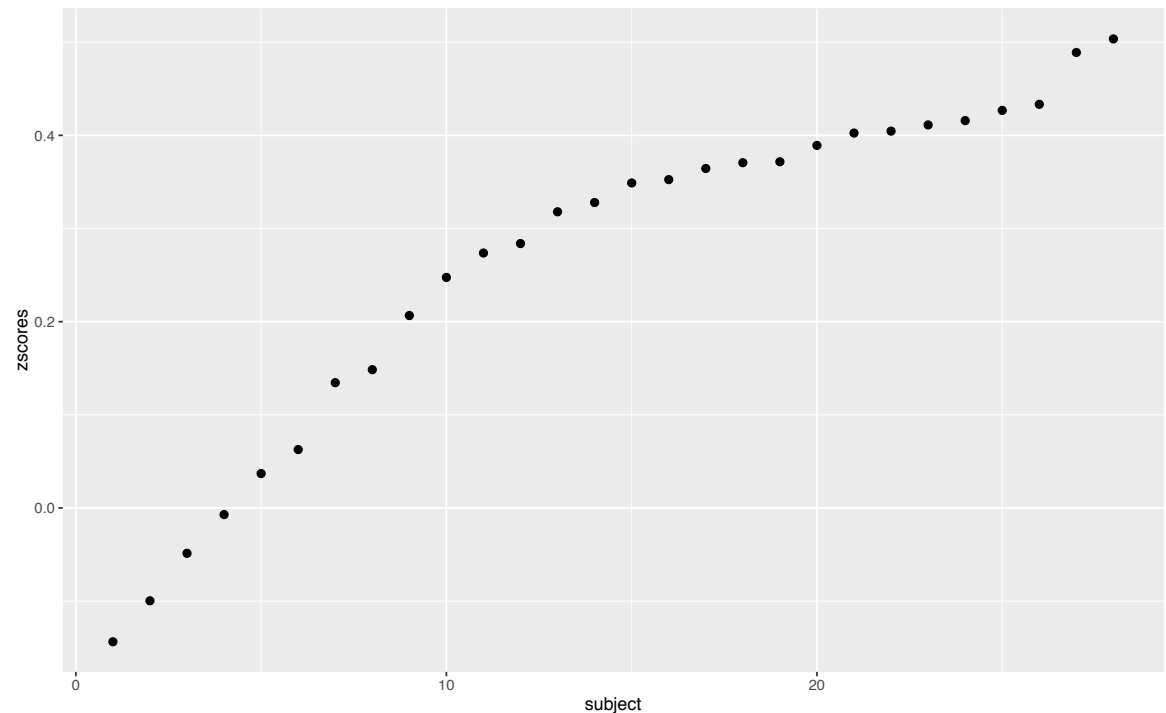
# Subject differences

Let's talk about the first term (1|subject). As the name suggests, this term captures differences between the subjects in our dataset.

```
lmer(zscores ~ embeddedStructure * dependencyLength + (1|subject) + (1|item), data=wh)
```

The plot at the right shows the mean rating of the 4 experimental conditions for each subject. As you can see, there is quite a bit of variability.

The (1|subject) term in the model tells R to estimate an intercept for each subject. This intercept is added to each subjects judgments to try to account for these differences.



Basically, instead of having these subject differences contaminate the effects of interest, or having these differences sit in an error term, this asks the model to estimate them. The code for this plot is in [subject.item.differences.r](#).

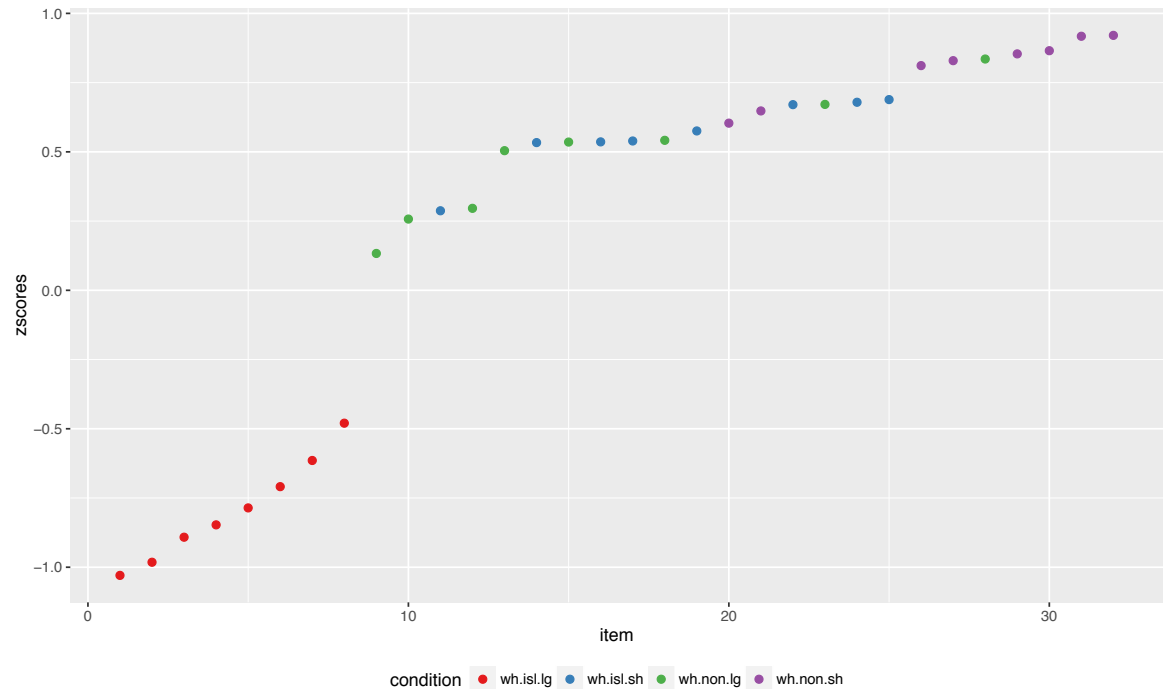


# Item differences

The second term, (1|item), is similar. As the name suggests, this term captures differences between the items in our dataset.

```
lmer(zscores ~ embeddedStructure * dependencyLength + (1|subject) + (1|item), data=wh)
```

Once again, we can plot the means of each item to see their differences. Now, we expect differences between items based on their condition. But as you can see by the colors (colors = condition), there are differences between items within a single condition.



This code asks R to estimate an intercept for each item, and add it whenever that item is being modeled. This makes sure that it isn't contributing to the other (important) effects, or to the error term. The code for this plot is in [subject.item.differences.r](#).

# Fixed factors vs Random factors

Now, you may have noticed that our experimental factors look different from these subject and item factors in the R command. This is because the former are **fixed factors** and the latter are **random factors**.

```
lmer(zscores ~ embeddedStructure * dependencyLength + (1|subject) + (1|item), data=wh)
```



There are two common ways to define the difference between fixed and random factors. The first is operational, the second is mathematical:

1. **Fixed factors** are factors whose **levels must be replicated exactly** in order for a replication to count as a replication.

**Random factors** are factors whose **levels will most likely not be replicated exactly** in a replication of the experiment.

2. **Fixed factors** are factors whose **levels exhaust the full range of possible level values** (as they are defined in the experiment).

**Random factors** are factors whose **levels do not exhaust the full range of possible level values**.

# Random intercepts and slopes

One last note about random factors. So far, we've only specified random intercepts — one value for each subject and one value for each item. But we can also specify **random slopes**. A random slope specifies a different value based on the values of the fixed factors (remember in our linear model, it is the fixed factors that specify the slopes of the lines).

The code for this looks complicated at first glance, but it isn't. We simply copy the fixed factor structure into the random subject term:

```
lmer(zscores ~ embeddedStructure * dependencyLength +  
(1+embeddedStructure*dependencyLength|subject) + (1|item), data=wh)
```

The 1 in the code tells R to estimate an intercept for each subject. The next bit tells R to estimate three more random coefficients per subject: one for embeddedStructure, one for dependencyLength, and one for the interaction embeddedStructure:dependencyLength.

There is a “best practices” claim in the field (Barr et al. 2013) that you should specify the “maximal” random effects structure licensed by your design. These means specifying random slopes if your design allows it.

The problem is that maximal random effects structure sometimes don't converge (R can't find a solution). In that case, you need to use a simpler model like an intercepts-only model.

# This is a linear mixed effects model

A model that only has fixed effects is usually just called a linear model, though it is perhaps more correctly a linear fixed effects model.

A model that has both fixed factors and random factors is called a mixed model, so if it is linear, it is a linear mixed effects model.

```
lmer(zscores ~ embeddedStructure * dependencyLength + (1|subject) + (1|item), data=wh)
```



In R, there is a package called lme4 that exists to model linear mixed effects models. You could load lme4 directly, and create the linear mixed effects model above. The function lmer() is a function from lme4.

We are using the package lmerTest to run our models. The lmerTest package calls lme4 directly (when you installed it, it also installed lme4). The reason we are using lmerTest is that lmerTest also includes some functions that let us calculate inferential statistics, like the F-statistic, and p-values. The lme4 package doesn't do that by itself.

# The Random slopes model in our script

Our script [linear.mixed.effects.models.r](#) contains the code for both an intercept-only model and a random slopes model. You should try running them.

What you will find is that the intercept-only model runs fine, but the slopes model fails to converge. Like I said, this happens with random slopes models.

It turns out that the problem with the model is our coding of the factors. We used [treatment coding](#), but for some reason (that I don't understand), the coding is causing a problem for the model.

The model will converge with a different coding scheme called [effect coding](#). This appears to be a pattern: many random slopes models will fail to converge with treatment coding, but succeed with effect coding.

So what should we do? Well, the coding doesn't affect things like F-statistics, t-statistics, and p-values. Those will be the same regardless of the coding scheme. So if that is all you care about, go ahead and change the coding.

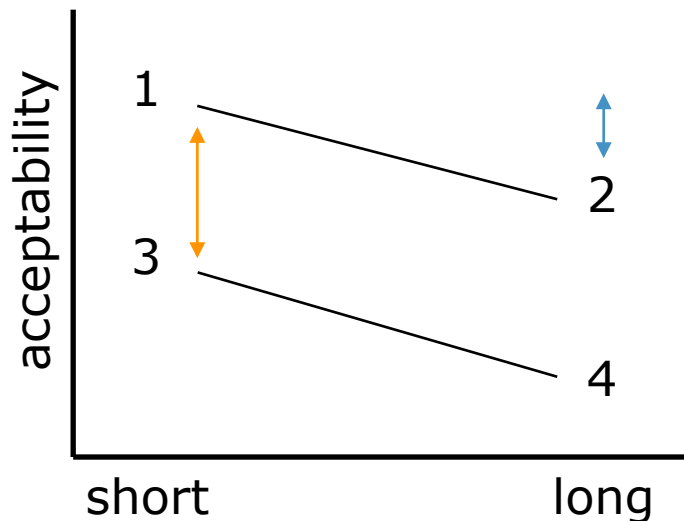
What does change is the interpretation of the coefficients in the model. In the next few slides, I will show you this change in interpretation. But the bottom line is that if the interpretation is important to you, you either need to drop the random slopes, or translate the effect coding estimates into treatment coding estimates by hand.

# Simple effects vs Main effects

The first step to understanding the difference between treatment coding and effect coding is to understand the difference between simple effects and main effects:

**Simple effects** are a difference between two conditions.

Typically, a simple effect is defined relative to one condition, the baseline condition. So if condition 1 were the baseline condition, we could define two simple effects:



The effect of 1 vs 2.

The effect of 1 vs 3.

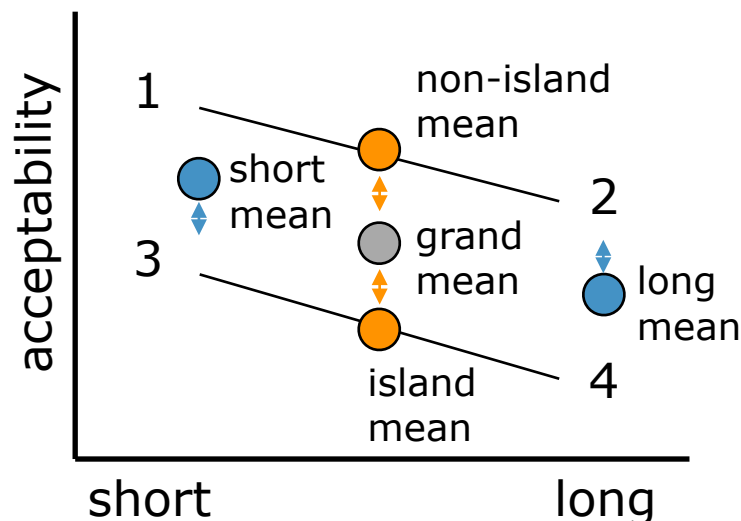
The effect of 1 vs 4 is the sum of these two (in this example).

# Simple effects vs Main effects

The first step to understanding the difference between treatment coding and effect coding is to understand the difference between simple effects and main effects:

**Main effects** are the difference between the grand mean of all conditions and the average of one level across both levels of the other factor.

Again, in a 2x2 design we can define two main effects: `embeddedStructure` and `dependencyLength`. Each one goes in two directions (one positive, one negative)



The blue arrows are the main effect of `dependencyLength` (positive and negative change from the grand mean)

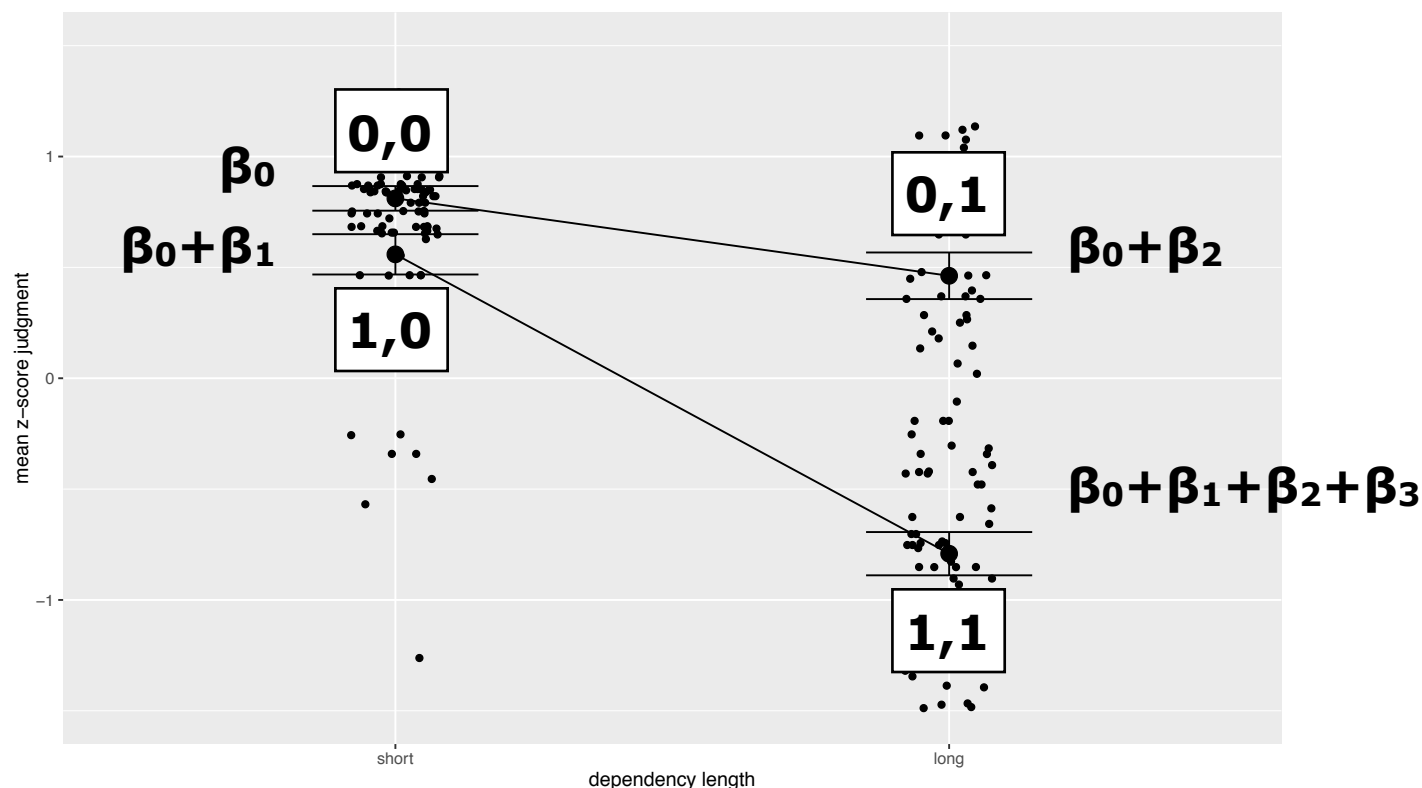
The orange arrows are the main effect of `embeddedStructure` (positive and negative change from the grand mean)

Each condition is a combination of the two main effects (in this example).

# Treatment coding reveals simple effects

In [treatment coding](#), each level is either 0 or 1. This is what we've been using so far. Treatment coding is great when one of your conditions can be considered a baseline in your theory.

$$\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(0,1)} + \beta_2 \text{dependency}_{(0,1)} + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_i$$



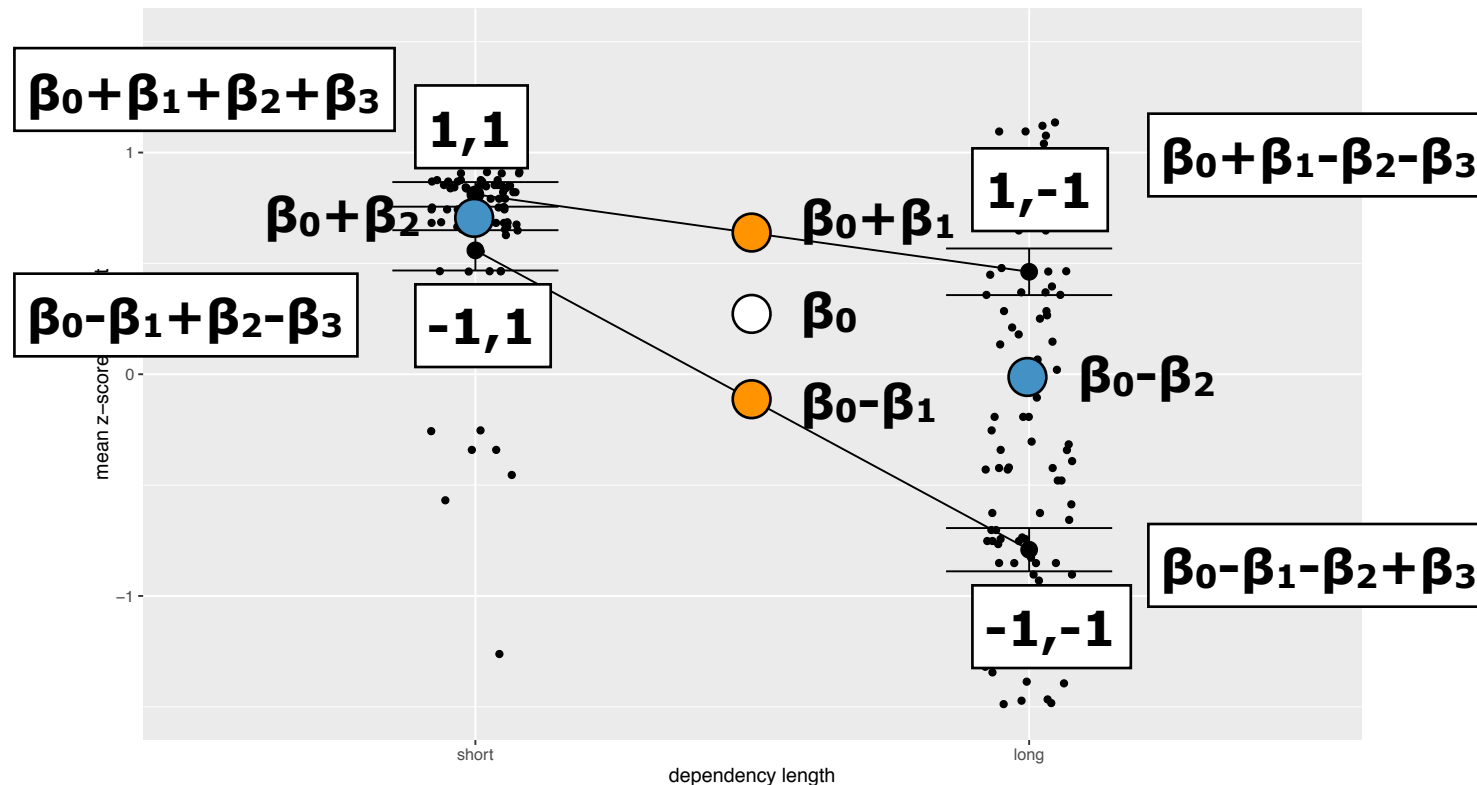
Treatment coding coefficients show you **simple effects**: the difference between the baseline condition and another condition. It works well for some designs, and less so for others (e.g., when you have no clear baseline).



# Effect coding reveals main effects

In **effect coding**, the factors are given the values **1 or -1**. This doesn't change the model that we specify, but it changes the interpretation of the coefficients. Effects coding is helpful when there is no clear "baseline" condition.

$$\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(1,-1)} + \beta_2 \text{dependency}_{(1,-1)} + \beta_3 \text{structure}_1 : \text{dependency}_1 + \varepsilon_i$$



Effect coding coefficients show you main effects. But be careful. Main effects are not straightforward to interpret when there is an interaction (because the interaction contaminates them).

# Choosing a contrast coding

Contrast coding is primarily about interpreting the coefficients in your model. If you don't care about trying to interpret those, then the contrast coding scheme will rarely matter. Contrast coding has **no** effect on statistics like F and t, and will not impact the p-values that F-tests and t-tests give you.

If you care about interpreting the coefficients, then you have to use your scientific knowledge to figure out which one is best for you.

**Treatment coding** is best if you have a clear baseline condition, and care about simple effects (differences from the baseline).

**Effect coding** is best when you don't have a clear baseline, or when you care about main effects (average effects of a factor). If you do care about main effects, remember that the presence of an interaction makes it impossible to interpret main effects (because the interaction contaminates them).

Finally, there are two times where it is better, mathematically, to use **effect coding**:

1. Some random slopes models won't converge with **treatment coding**, but will converge with **effect coding** (like our random slope model).
2. If you are mixing categorical and continuous factors, **treatment coding** can introduce heteroscedasticity (variable variance). **Effect coding** does not.

# Table of contents

1. Introduction: You are already an experimentalist
2. Conditions
3. Items
4. Ordering items for presentation
5. Judgment Tasks
6. Recruiting participants
7. Pre-processing data (if necessary)
8. Plotting
9. Building linear mixed effects models
10. Evaluating linear mixed effects models using Fisher
11. Neyman-Pearson and controlling error rates
12. Bayesian statistics and Bayes Factors
13. Validity and replicability of judgments
14. The source of judgment effects
15. Gradience in judgments

Section 1:  
Design

Section 2:  
Analysis

Section 3:  
Application

# Let's look at the coefficients of the intercept model (wh.lmer)

Here is the output of `summary(wh.lmer)` for treatment coding:

```
Fixed effects:
              Estimate Std. Error    df t value Pr(>|t|)
(Intercept)    0.81129    0.06415 171.34000  12.647  < 2e-16 ***
embeddedStructure2 -0.25244    0.08789 192.15000  -2.872  0.004531 **
dependencyLength2 -0.34913    0.08789 192.15000  -3.973  0.000101 ***
embeddedStructure2:dependencyLength2 -1.00138    0.12458 192.33000  -8.038  9.06e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The  $\beta$ 's for the model are listed under Estimate. Go ahead and check these numbers against the graph of our condition means.

Here is the output of `summary(wh.lmer)` for effect coding:

```
Fixed effects:
              Estimate Std. Error    df t value Pr(>|t|)
(Intercept)    0.26016    0.03497  27.13000   7.439 5.14e-08 ***
embeddedStructure1  0.37657    0.03114 192.33000  12.091  < 2e-16 ***
dependencyLength1  0.42491    0.03114 192.33000  13.643  < 2e-16 ***
embeddedStructure1:dependencyLength1 -0.25034    0.03114 192.33000  -8.038  9.06e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Just for fun, we can also look at the  $\beta$ 's from effect coding. As you can see, they are very different. You can check them against the  $\beta$ 's for treatment coding (you can translate between the two using the formulae in the previous slides, though it takes some effort).

Also notice there are some statistical things to the right in these readouts, such as t values and p-values... and notice that **they don't change based on coding!**

# Anova(wh.lmer) yields F-statistics and p-values

Here is the output of `anova(wh.lmer)` for both coding types:

```
Analysis of Variance Table of type III with Satterthwaite
approximation for degrees of freedom

              Sum Sq Mean Sq NumDF  DenDF F.value    Pr(>F)
embeddedStructure      31.616   31.616     1 192.32 146.189 < 2.2e-16 ***
dependencyLength      40.255   40.255     1 192.32 186.133 < 2.2e-16 ***
embeddedStructure:dependencyLength 13.973   13.973     1 192.32  64.611 9.048e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Although the `summary()` function had statistics in it (t statistics and p-values), I want to focus on the `anova()` function. This is the same information that you would get from a fixed effects ANOVA, which I think is useful for relating mixed effects models to standard linear models.

There are two pieces of information here that I want to explain in more detail: the **F statistic** and the **p-value**. These are the two pieces of information that `anova()` adds to our interpretation. With that, we will have (i) the graphs, (ii) the model and its estimates, (iii) the F statistic, and (iv) the p-value. Together, those 4 pieces of information provide a relatively comprehensive picture of our results.

Someday, it will be worth it for you to explore the Sum of Squares and df values, but for now, we can set them aside as simply part of the calculation of F's and p's respectively.

# The F statistic is about evaluating models

There are two common dimensions along which models are evaluated: their adequacy and their simplicity.

## 1. **Adequacy:** We want a model that minimizes error

We've already encountered this. We used **sum of squares** to evaluate the amount of error in a model. We chose the coefficients (the model) that minimized this error.

## 2. **Simplicity:** We want a model that estimates the fewest parameters

We can measure simplicity with the **number of parameters that are estimated from the model**. A model that estimates more parameters is more complicated, and one that estimates fewer parameters is simpler.

The intuition behind this is that **models are supposed to teach us something**. The more the model uses the data, the less the model itself is contributing.

The models we've been constructing are estimating 4 parameters from the data:  $\beta_0$ ,  $\beta_1$ ,  $\beta_2$ , and  $\beta_3$

# Degrees of Freedom as a measure of simplicity

We can use **degrees of freedom** as a measure of **simplicity**.

$df = \text{number of data points} - \text{number of parameters estimated}$

$$df = n - k$$

Notice that df makes a natural metric for simplicity for three reasons:

1. It is based on the number of parameters estimated, which is our metric.
2. It captures the idea that a model that estimates 1 parameter to explain 100 data points ( $df=99$ ) is better than a model that estimates 1 parameter to explain 10 ( $df=9$ ).
3. The values of df work in an intuitive direction: higher df is better (simpler) and lower df is worse.

# In practice, there is a tension between adequacy and simplicity

**Adequacy** seeks to minimize error. **Simplicity** seeks to minimize the number of parameters that are estimated from the data.

Imagine that you have 224 data points, just like our data set. A model with 224 parameters would predict the data with no error whatsoever because each parameter would simply be one of the data points. (This the old saying “the best model of the data is the data.”). This would have perfect adequacy.

But this model would also be the most complicated model that one can have for 224 data points. It would teach us nothing about the data.

This tension is not a necessary truth. There could be a perfect model that predicts all of the data without having to look at the data first. But in practice, there is a **tension between adequacy and simplicity**.

To put this in terms of our metrics, this means there will be a tension between **sum of squares** and **degrees of freedom**.

So what we want is a way to **balance this tension**. We want a way to know if the df we are giving up for lower error is a good choice or not.



# A transactional metaphor

One way to think about this is with a metaphor. As a modeler, you want to eliminate error. You can do this by spending df. If you spend all of your df, you would have zero error. But you'd also have no df left. We have to assume that df is inherently valuable (you lose out on learning something) since you can spend it for stuff (lower error). So you only want to spend your df when it is a good value to do so.

Thinking about it this way, the question when comparing models is **whether you should spend a df to decrease your error**. The simple model keeps more df. The complex model spends it. The simple model has more error. The complex model has less error because it spent some df. Which one should you use?

**Simple:** spends no df

$Y_i = \beta_0$	+	$\epsilon_i$
2 = 4	+	-2
3 = 4	+	-1
4 = 4	+	0
df=3		SS=5

**Complex:** spent a df

$Y_i = \beta_0$	+	$\epsilon_i$
2 = 3	+	-1
3 = 3	+	0
4 = 3	+	1
df=2		SS=2

# A transactional metaphor

When you are faced with the prospect of spending df, there are two questions you ask yourself:

1. How much (lower) error can I buy with my df?
2. How much error does df typically buy me?

In other words, you want to compare the value of your df (in this particular instance), with the value of your df in general. If the value here is more than the value in general, you should spend it. If it is less, you probably shouldn't spend it, because that isn't a good deal.

We can capture this with a ratio:

How much error can I buy with my df?

---

How much error does df typically buy me?

If the ratio is high, it is a good deal, so you spend your df. If the ratio is low, it is a bad deal, so you don't spend your df.

# The F ratio

To cash out this intuition, all we need to do is calculate how much you can buy with your df, and then calculate the value you can expect for a df, and see if you are getting a good deal by spending the df.

the amount of error you can buy with a df =  $(SS_{\text{simple}} - SS_{\text{complex}}) / (df_{\text{simple}} - df_{\text{complex}})$

the amount of error df typically buys =  $SS_{\text{complex}} / df_{\text{complex}}$

Let's take a moment to really look at these equations.

The first takes the difference in error between the models and divides it by the difference in df. So that is telling you how much error you can eliminate with the df that you spent moving from one model to the next. Ideally, you would only be moving by 1 df to keep things simple.

The second equation takes the error of the complex model and divides it by the number of df in that model, giving you the value in error-elimination for each df. The complex model has the lowest error of the two models, so it is a good reference point for the average amount of error-elimination per df.

# The F ratio

So now what we can do is take these two numbers, and create a ratio:

$$F = \frac{(SS_{\text{simple}} - SS_{\text{complex}})/(df_{\text{simple}} - df_{\text{complex}})}{SS_{\text{complex}}/df_{\text{complex}}}$$

If F stands for the ratio between the amount of error we can buy for a df and a typical value for a df, then we can interpret it as follows:

If F equals 1 or less, then we aren't getting a good deal for our df. We are buying relatively little error by spending it. So we shouldn't spend it. We should use the simpler model, which doesn't spend the df.

If F equals more than 1, we are getting a good deal for our df. We are buying relatively large amounts of error-reduction by spending it. So we should spend it. We should use the more complex model (which spends the df) in order to eliminate the error (at a good value).

The F ratio is named after Ronald Fisher (1890-1962), who developed it, along with a lot of methods in 20th century inferential statistics.

# Our toy example

Here are our two models:

simple			complex		
$Y_i = \beta_0$	+	$\varepsilon_i$	$Y_i = \beta_0$	+	$\varepsilon_i$
2 = 4	+	-2	2 = 3	+	-1
3 = 4	+	-1	3 = 3	+	0
4 = 4	+	0	4 = 3	+	1
df=3		SS=5	df=2		SS=2

$$F = \frac{(SS_{\text{simple}} - SS_{\text{complex}})/(df_{\text{simple}} - df_{\text{complex}})}{SS_{\text{complex}}/df_{\text{complex}}} = \frac{(5-2)/(3-2)}{2/2} = 3$$

So in this case the F ratio is 3, which says that we can buy three times more error-elimination for this df than we would typically expect to get. So that is a good deal, and we should use that df. So the complex model is better by this metric (the F ratio).

# Our real example

Here is the output of `anova(wh.lmer)` for both coding types:

```
Analysis of Variance Table of type III with Satterthwaite
approximation for degrees of freedom

              Sum Sq Mean Sq NumDF DenDF F.value    Pr(>F)
embeddedStructure      31.616   31.616     1 192.32 146.189 < 2.2e-16 ***
dependencyLength      40.255   40.255     1 192.32 186.133 < 2.2e-16 ***
embeddedStructure:dependencyLength 13.973   13.973     1 192.32  64.611 9.048e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now let's look again at the output of the `anova()` function (which calculates F's) for our example data.

The first F in the list is for the factor `embeddedStructure`. This F is comparing two models:

simple:       $\text{acceptability}_i = \beta_0$

complex:     $\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(0,1)}$

The resulting F ratio is 146:1, so yes, the structure factor is pretty good value for the df spent.

# Our real example

Here is the output of `anova(wh.lmer)` for both coding types:

```
Analysis of Variance Table of type III with Satterthwaite
approximation for degrees of freedom

              Sum Sq Mean Sq NumDF  DenDF  F.value    Pr(>F)
embeddedStructure      31.616   31.616     1 192.32 146.189 < 2.2e-16 ***
dependencyLength       40.255   40.255     1 192.32 186.133 < 2.2e-16 ***
embeddedStructure:dependencyLength 13.973   13.973     1 192.32  64.611 9.048e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The second F in the results is for `dependencyLength`. Again, this is comparing two models:

simple:       $\text{acceptability}_i = \beta_0$

complex:     $\text{acceptability}_i = \beta_0 + \beta_2 \text{dependency}_{(0,1)}$

The resulting F ratio is 186:1, so yes, the dependency factor is pretty good value for the df spent.

# Our real example

Here is the output of `anova(wh.lmer)` for both coding types:

```
Analysis of Variance Table of type III with Satterthwaite
approximation for degrees of freedom

              Sum Sq Mean Sq NumDF  DenDF  F.value    Pr(>F)
embeddedStructure      31.616   31.616     1 192.32  146.189 < 2.2e-16 ***
dependencyLength      40.255   40.255     1 192.32  186.133 < 2.2e-16 ***
embeddedStructure:dependencyLength 13.973   13.973     1 192.32   64.611 9.048e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The final F is for the interaction of the two factors. This is still comparing two models, but in this case, the simpler model is the model with the two main effects present with no interaction (+), and the complex model adds the interaction (\*):

simple:  $\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(0,1)} + \beta_2 \text{dependency}_{(0,1)}$

complex:  $\text{acceptability}_i = \beta_0 + \beta_1 \text{structure}_{(0,1)} * \beta_2 \text{dependency}_{(0,1)}$

The resulting F ratio is 64:1, which again, is a good value, and suggests that it was a good idea to add the interaction term.



# Model comparison is not hypothesis testing

Let's be clear: model construction and comparison is its own exercise. Nothing we have done so far has been a formalization of a hypothesis test. We've just been talking about how to construct models, and how to compare two models that we've constructed using information that seems useful.

Also, there are other metrics for model evaluation and comparison that you should explore: adjusted  $R^2$ , BIC, AIC, etc.

I want to stress the fact that you can be interested in model construction and model comparison for its own purposes. Models are a tool that allows you to better understand your research question. You can see exactly how different factors contribute to the dependent variable.

This distinction between model construction/comparison and hypothesis testing is why lme4 doesn't come with p-values. It is a tool for model construction and comparison, while p-values are a tool for hypothesis testing.

That being said, I wouldn't make you learn about F ratios if they couldn't be used for hypothesis testing. And lmerTest, which as the name suggests is designed to turn linear mixed effects models into hypothesis tests, wouldn't give you the F's if they weren't useful for tests. So let's do that now.

# Null Hypothesis Significance Testing

When people think of hypothesis testing, the first approach that comes to mind is [Null Hypothesis Significance Testing](#).

NHST was not the first approach to statistics that was developed ([Bayes Theorem](#) is from 1763, Karl Pearson developed many components of statistics in the 1890s and 1900s, Gosset developed the t-test in 1908). NHST is also not the currently ascendant approach (Bayesian statistics are ascending).

But NHST dominated 20th century statistics (both in theory and practice) so it is still a standard approach in experimental psychology, and it is very much necessary for reading papers published in the last 75 years.

Pedagogically speaking, I am not sure if it is better to begin with NHST, and then move to Bayes, or better to start with Bayes, and then move to NHST. For now, I think it is safer to start with NHST, and move to Bayes if you are interested.

That way, even if you don't have the time to look into Bayes in detail, you still have the NHST tools necessary to (i) publish papers, and (ii) read existing papers. You can cross the Bayes bridge if the field ever comes to it.

# Two approaches to NHST

It turns out that there are two major approaches to NHST. They are very similar in mathematical appearance, so it is easy to think that they are identical. But they differ philosophically (and in some details), so it is important to keep them separated.

**Ronald Fisher** was the first person to try to wrangle the growing field of statistics into a unified approach to hypothesis testing. His NHST was the first attempt, and may still be the closest to the way scientists think about NHST. We'll start with the **Fisher approach**.

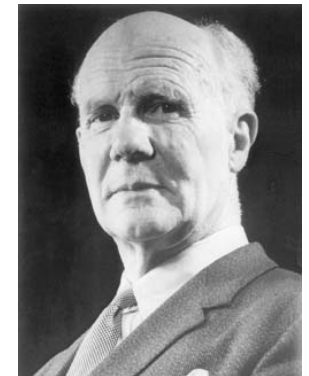


Ronald A. Fisher  
(1890-1962)

**Neyman** and **Pearson** were fans of Fisher's work, but thought there were some deficiencies in his approach. So they tried to rectify that. It turns out that they simply had a different conception of probability and hypothesis testing. We'll talk about the Neyman-Pearson approach second.



Jerzy Neyman  
(1894-1981)



Egon Pearson  
(1895-1980)

# Fisher's NHST

Under Fisher's NHST, there is only one hypothesis under consideration. Perhaps ironically, it is the most uninteresting hypothesis you could consider. It is called the **null hypothesis**, or  $H_0$ .

**H<sub>0</sub>:** The **null hypothesis**. This states that there is no effect in your data (e.g., no difference between conditions, no interaction term, etc).

For Fisher's NHST, the goal of an experiment is to **disprove the null hypothesis**.



"Every experiment may be said to exist only in order to give the facts a chance of disproving the null hypothesis." - Fisher (1966)

To do this, Fisher's NHST calculates the probability of **the observed data** under **the assumption that the null hypothesis is true**, or  $p(\text{data} | \text{null hypothesis})$ .

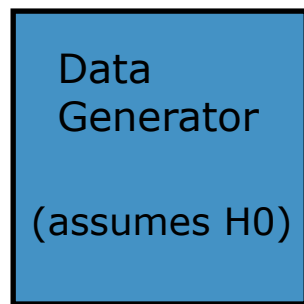
This leads to **Fisher's disjunction**:

If  $p(\text{data} | \text{null hypothesis})$ , called the p-value, is low, then you can conclude either: (i) the null hypothesis is incorrect, or (ii) a rare event occurred.

# Fisher's NHST logic, stated a different way

There are two steps to a statistical test under Fisher's NHST approach:

**Step 1:** Calculate  $P(\text{data} \mid \text{null hypothesis})$



data1  
data2  
data3  
...

One way to think about this is that you are creating a data generating device that assumes the null hypothesis, and generates **all possible data sets**.

Then you use the distribution of generated data to calculate the probability of the observed data

$$P(\text{data} \mid H_0) = \frac{\text{observed data}}{\text{generated data}}$$

**Step 2:** Make an inference about the null hypothesis

For Fisher,  $p(\text{data} \mid H_0)$  is a measure of the strength of evidence against the null hypothesis. If it is low, that is either strong evidence against the null hypothesis, or evidence that something really rare occurred.

# This logic is enough to interpret our results

Once we have the logic of NHST, we can go back to the results that R and lmerTest gave us, and interpret those results. (Sure, it would be nice to be able to calculate the results for ourselves, but R does this for us.)

Here is the output of `anova(wh.lmer)` for both coding types:

```
Analysis of Variance Table of type III with Satterthwaite
approximation for degrees of freedom

              Sum Sq Mean Sq NumDF  DenDF  F.value    Pr(>F)    ***
embeddedStructure      31.616   31.616     1 192.32 146.189 < 2.2e-16 ***
dependencyLength       40.255   40.255     1 192.32 186.133 < 2.2e-16 ***
embeddedStructure:dependencyLength 13.973   13.973     1 192.32  64.611 9.048e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The first thing to note is that in this case, our p-values are in scientific notation. This is because they are really small:

structure	p = .0000000000000000022
length	p = .0000000000000000022
interaction	p = .00000000000000009048

These are incredibly small, so under Fisher's logic, we say that there is either very strong evidence that the null hypothesis is false, or something very rare occurred (i.e., the null hypothesis is true, but we got a result at the very end of the distribution of possible null hypothesis results).

# This logic is enough to interpret our results

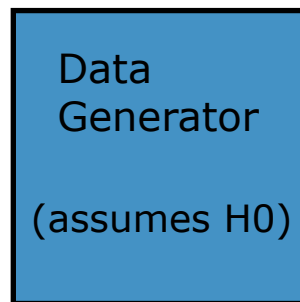
Once we have the logic of NHST, we can go back to the results that R and lmerTest gave us, and interpret those results. (Sure, it would be nice to be able to calculate the results for ourselves, but R does this for us.)

Here is the output of `anova(wh.lmer)` for both coding types:

```
Analysis of Variance Table of type III with Satterthwaite
approximation for degrees of freedom

              Sum Sq Mean Sq NumDF DenDF F.value    Pr(>F)    ***
embeddedStructure      31.616   31.616     1 192.32 146.189 < 2.2e-16 ***
dependencyLength       40.255   40.255     1 192.32 186.133 < 2.2e-16 ***
embeddedStructure:dependencyLength 13.973   13.973     1 192.32  64.611 9.048e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The second thing to note is that these p-values are based on the F statistics that lmerTest calculated for each effect.



F1  
F2  
F3  
...

In principle, you can use any summary statistic you want (and you may know that there are many summary statistics in the literature). You could even use the sample mean.

The F is a nice statistic to use because it gives us even more information than just a p-value — remember, it tells us how much value we got for that df.

# This logic is enough to interpret our results

Once we have the logic of NHST, we can go back to the results that R and lmerTest gave us, and interpret those results. (Sure, it would be nice to be able to calculate the results for ourselves, but R does this for us.)

Here is the output of `anova(wh.lmer)` for both coding types:

```
Analysis of Variance Table of type III with Satterthwaite
approximation for degrees of freedom

              Sum Sq Mean Sq NumDF DenDF F.value    Pr(>F)
embeddedStructure      31.616   31.616     1 192.32 146.189 < 2.2e-16 ***
dependencyLength       40.255   40.255     1 192.32 186.133 < 2.2e-16 ***
embeddedStructure:dependencyLength 13.973   13.973     1 192.32  64.611 9.048e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Finally, note that the readout puts asterisks next to the p-values to tell you if they are below .05, .01, etc.

It is tempting to think of this as just a nice way to quickly visualize the results, but there is something much deeper going on here. The **precise p-value is necessary for the Fisher approach to NHST**, the **asterisks are there for the Neyman-Pearson approach**.

We will talk about this more later, but in a nutshell, the Neyman-Pearson approach asks whether the p-value is below a pre-specified threshold. The exact number doesn't matter, it is just whether it is below the threshold. These asterisks implement several common thresholds.



# The logic of Fisher p-values

First and foremost, p-values are only one small piece of information. You also have your graphs, the model coefficients, and evaluation statistics like F.

But if you are going to use p-values, you need to be clear about what the p-value is telling you. It is the probability of obtaining the observed results, or results more extreme, under the data generation model of the null hypothesis).

Here are some other bits of information you may want to know. Unfortunately, p-values are not these other things:

1. The probability of the null hypothesis being true:  $p(H_0 \mid \text{data})$
2. The probability of your hypothesis of interest being true:  $p(H_1 \mid \text{data})$
3. The probability of incorrectly rejecting the null hypothesis (a false rejection).
4. The probability that you can replicate your results with another experiment.

The problem is that plenty of people think that p-values give these bits of information. That is false. There are literally dozens of papers out there trying to correct these misconceptions.

# The math underlying NHST

Though the logic is enough to interpret the results that R and lmerTest give us, you may want to study the math that NHST approaches use to generate the reference distribution for the null hypothesis. It will give you the flexibility to run (and even create) your own analyses, and it will help you understand the hypothesis tests at a deeper level.

There are basically three approaches to generating the null reference distribution in NHST. I will review each briefly in the next few slides:

## **1. Randomization methods.**

The basic idea is to take your observed data points, and randomize the condition labels that you attach to them.

## **2. Bootstrap methods.**

The basic idea is to use your sample as a population, and sample from it to generate a (population-based) reference distribution.

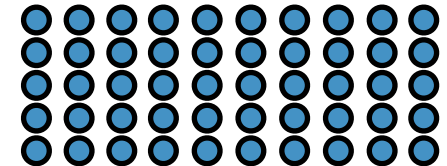
## **3. Analytic methods.**

Most people imagine analytic methods when they think of stats. The idea here is that there are test statistics whose distribution is invariant under certain assumptions. We can use these known distributions to calculate p-values analytically (with an equation).

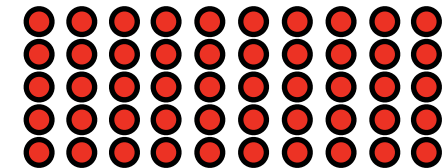
# Randomization Methods

Let's use an example to demonstrate how to generate a reference distribution for the null hypothesis using randomization. Let's focus on two conditions for simplicity:

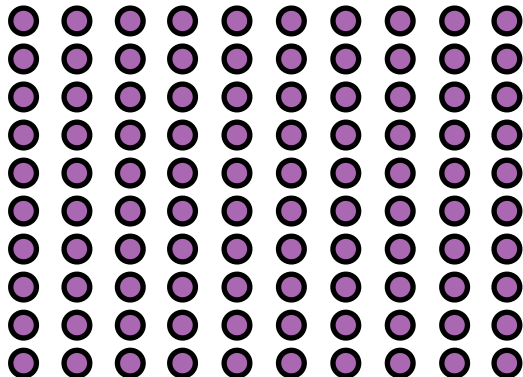
**control:** What do you think **that Jack stole** \_\_\_?



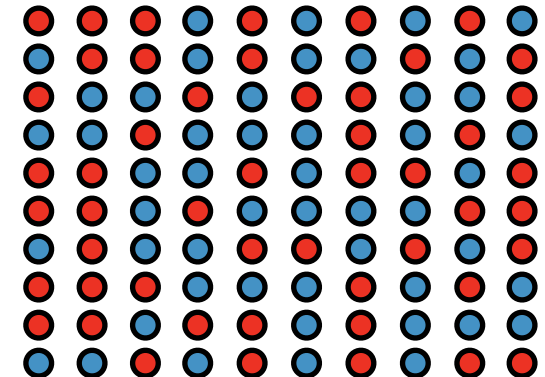
**target:** \* What do you wonder **whether Jack stole** \_\_\_?



Here is the critical insight of randomization tests: Even though I have **labeled** these observations **control** and **target**, under the null hypothesis they are all just from the same label, **null**. So, this assignment of labels is arbitrary under the null hypothesis. And if the assignment is arbitrary, then I should be able to randomly re-arrange the labels.

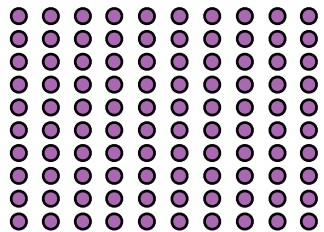


Randomly assign labels to these points because these labels are arbitrary under the null hypothesis.

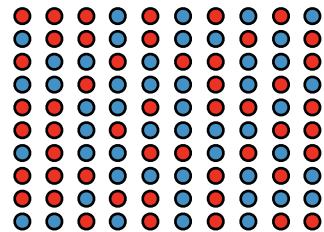


# Randomization: generating the distribution

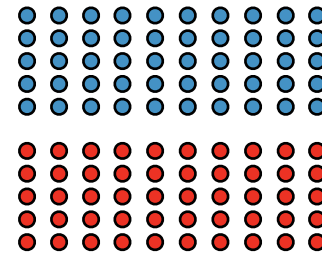
Start with the full data set



Randomly assign labels



Calculate the test statistic

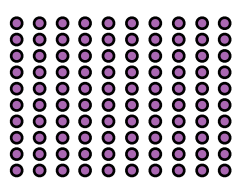


$$\rightarrow \bar{x}_c = 1$$

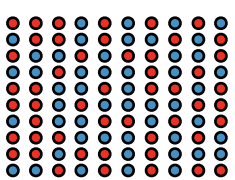
$$\rightarrow \bar{x}_t = .3$$

$$\rightarrow \bar{x}_c - \bar{x}_t = .7$$

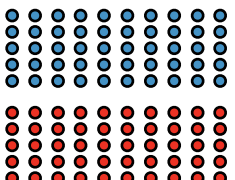
Then we repeat the process. With small samples, we can create every possible combination of labels, and have a complete distribution of possible test statistics. With large samples, this isn't possible, so we collect a large number of randomizations, like 10,000, and approximate the distribution.



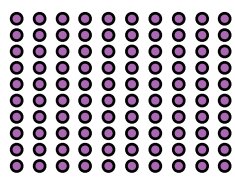
→



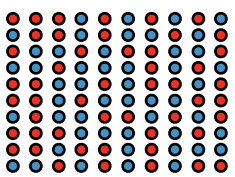
→



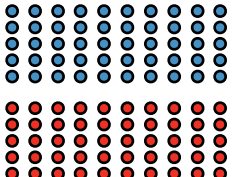
$$\bar{x}_c - \bar{x}_t = -.5$$



→



→



$$\bar{x}_c - \bar{x}_t = .3$$

We then collect all of the test statistics together to form a reference distribution under the null hypothesis.

See [randomization.r](#) for code to do this!

... to completion or 10,000

# Randomization: calculating a p-value

Now that we have a reference distribution, we ask the following question: **What is the probability of obtaining the observed result, or one more extreme, given this reference distribution?**

We say “or one more extreme” for two reasons. First, we can’t just ask about one value because our response scale is continuous (most likely, the probability of one value is 1/the number of values in our distribution). Second, if we have to define a bin, “more extreme” results make sense, because those are also results that would be less likely under the null hypothesis.

If you calculated all possible randomizations, then you can use this formula for p-values:

$$p = \frac{\text{observations equal, or more extreme}}{\text{number of randomizations}}$$

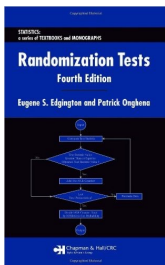
If you randomly sampled the randomizations, then the above will underestimate the true p-value (because your sampled distribution is missing some extreme values). You can correct for this by adding 1 to the numerator and denominator:

$$p = \frac{\text{observations equal, or more extreme} + 1}{\text{randomly sampled randomizations} + 1}$$

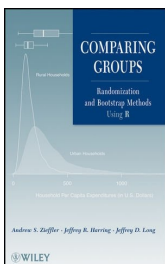
# Randomization: More info

Randomization tests are incredibly powerful and incredibly flexible. I would say that if you want to do pure NHST, without mixed effects, then randomization tests should be your first choice.

Even Fisher admitted that randomization tests should be the gold standard for NHST. But in the 1930s, computers weren't accessible enough to make randomization tests feasible for anything but very small experiments. So he developed analytic methods for larger experiments. But he said that the analytic methods are only valid insofar as they give approximately the same result as randomization methods.



The best reference for randomization tests is Edgington and Onghena (now 2007), **Randomization Tests**. Be warned that it is written like a reference, and not like a textbook. But if you need to know something about randomization tests, it is fantastic.



For a textbook experience, I like Zieffler, Harring, and Long's (2011) **Comparing Groups: Randomization and Bootstrap Methods using R**. It is an introduction to NHST using Randomization and Bootstrap methods, which is a nice idea in the computer age.

# Inferences: samples vs populations

If you want to make **causal inferences** about whether your treatment had an effect in your **sample**, you have to **randomly assign** units to treatments.

If you can't randomly assign your units to treatments, you can't be sure that your treatment is causing the effect. The effect could be caused by properties of the two groups.

As the name implies, randomization tests assume that you randomly assigned units to treatments. And because of this assumption, randomization tests allow you to make **causal inferences about the effect of your treatment in the sample**.

If you want to make **inferences about populations**, you have to **randomly sample the units from the population**.

If you can't randomly sample your units, you can't be sure that your results hold for the entire population. You can still make inferences about **your sample**, which is generally all you want to do anyway, but you can't claim that your treatment will have an effect in a population.

If you want to make **claims about a population**, then you can't use randomization tests. You have to add the idea of sampling from a population to the test. What you want are **bootstrap methods**.

# Bootstrap methods

If you are running a bootstrap, I assume you are interested in making inferences about populations. Here are the first three steps:

- Step 1:** Randomly sample your participants (or other experimental units) when running your experiment. This is necessary if you want to make inferences about population parameters from the samples.
- Step 2:** Choose a test statistic. Usually this is the mean, but it could be one of the other possible statistics.
- Step 3:** Define the null hypothesis as no difference between population parameters, e.g.,  $\mu_A - \mu_B = 0$

Let's assume that we did randomly sample (not true, but let's assume it for demonstration purposes, and let's use means/mean differences for our test statistic.



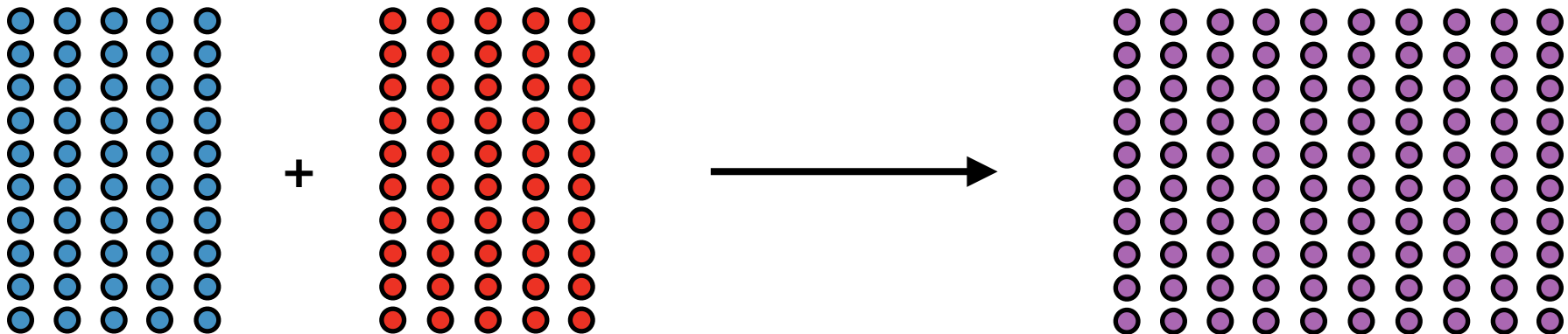
# Bootstrap methods

**Step 4:** Define the single population under the null hypothesis.

This is our first tricky step. We don't have an empirical measurement of our population. If we did, we wouldn't need to sample from it! So what do we do?

Well, we can use our experimental sample as an approximation because it was randomly sampled.

Under the null hypothesis, we only have one population, so that means we can combine all of the values from both conditions together into one group:



# Bootstrap methods

**Step 5:** Calculate the reference distribution for your test statistic under the null hypothesis (one population).

This is our next tricky step. We want to use our sample as an approximation of our distribution. This means randomly sampling from our sample in order to derive a reference distribution.

In this case we want to **randomly sample with replacement**, which means that after each participant is selected, we replace it so that it could be **selected again in the very same sample!** Up until now, we have been sampling **without replacement**, which means that each participant could only be **selected once per sample**.

<b>with replacement</b>		$\{1,2\}$	$\{1,1\}$		<b>without replacement</b>		$\{1,2\}$
$\{1,2,3\}$ , choose 2	$\rightarrow$	$\{1,3\}$	$\{2,2\}$		$\{1,2,3\}$ , choose 2	$\rightarrow$	$\{1,3\}$
		$\{2,3\}$	$\{3,3\}$				$\{2,3\}$

We do this to approximate a population that is much larger than our sample (possibly infinitely large). Values will still be chosen according to their probability, but they won't artificially disappear because our sample size is small.

# Bootstrap methods

**Step 5:** Calculate the reference distribution for your test statistic under the null hypothesis (one population).

So here is what we do:

First, we randomly sample with replacement two samples from our observed sample. We call these **bootstrap replicates**. They are replicates because they are other possible samples that we could have obtained in our experiment. They are bootstrap replicates because this procedure is called the bootstrap method.

Second, we calculate the mean for each bootstrap replicate, and then calculate the mean difference.

Third, we save this mean difference (as the first value in our reference distribution).

Then we repeat this process a large number of times (e.g., 10,000) to derive a reference distribution called the **bootstrap distribution**.

# Bootstrap methods

**Step 6:** Calculate the probability of your observed statistic (e.g., difference between means in your experiment), or one more extreme, from your reference distribution.

Now that we have a reference distribution that approximates the exact distribution pretty well, all we need to do is use our old formula (plus correction) to calculate the p-value:

$$p = \frac{\text{outcomes equal, or more extreme} + 1}{\text{randomly sampled outcomes} + 1}$$

What we've just done is called a **non-parametric bootstrap**, because we didn't make any assumptions about the parameters of the population. Instead, we used our (combined) sample as a proxy for the population.

**Parametric:** A parametric test is one in which the parameters of the population are known (or assumed)

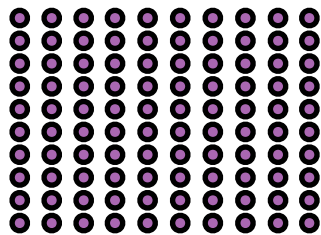
**Non-parametric:** A non-parametric test is one in which the parameters of the population are unknown (or not assumed)

# A parametric bootstrap

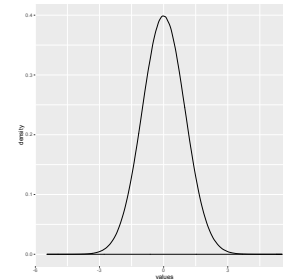
The parametric bootstrap has the same steps as the non-parametric bootstrap. The only difference is in the population that the replicates are drawn from!

1. Define a population to draw the replicates from:

**non-parametric:** the sample is used as a proxy



**parametric:** a probability model for the population with certain parameters



2. Sample with replacement from the population to derive a reference distribution.
3. Calculate the probability of the data given the reference set.

# A parametric bootstrap

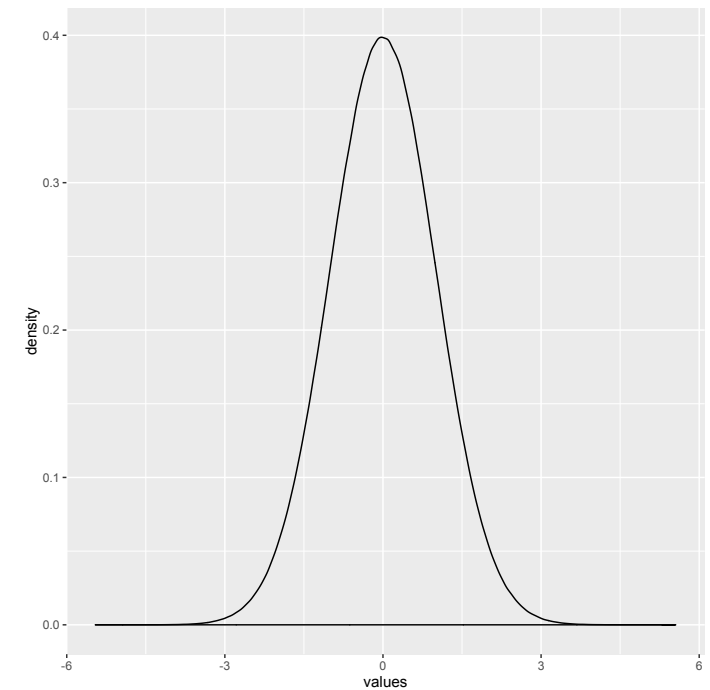
The only challenge in the parametric bootstrap is picking the correct probability model for your population. How do you know what parameters to pick?

In principle, you could pick any probability model that you think underlies the generation of your data. In practice, if you are ever doing one of these analyses, you will probably choose a **normal distribution**.

The normal distribution is the “bell curve”. It has some useful properties that make it a good choice for many applications:

1. It is the probability model underlying a large number of phenomena.
2. It can be completely parameterized with 2 parameters: the mean and the standard deviation using the following equation:

$$y = \frac{1}{\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma}$$

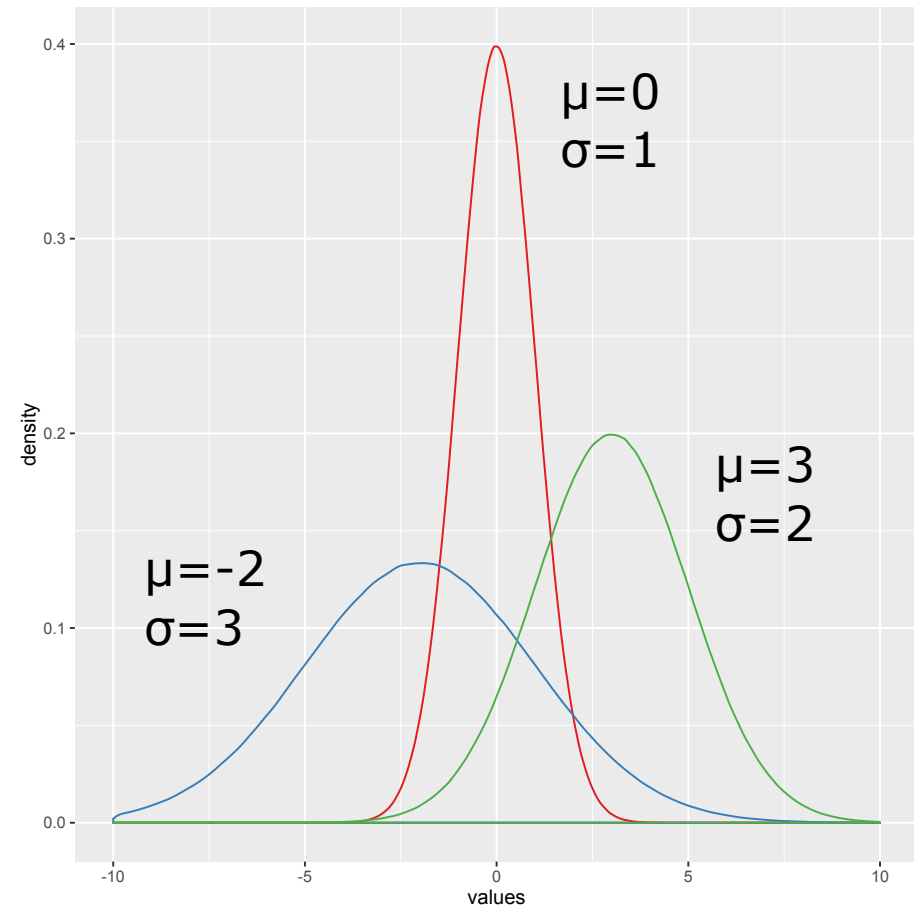


# A parametric bootstrap

The only problem with the normal distribution is that it is a **family of distributions**. Every member of the family follows the equation, but they each use a different value for the mean and standard deviation:

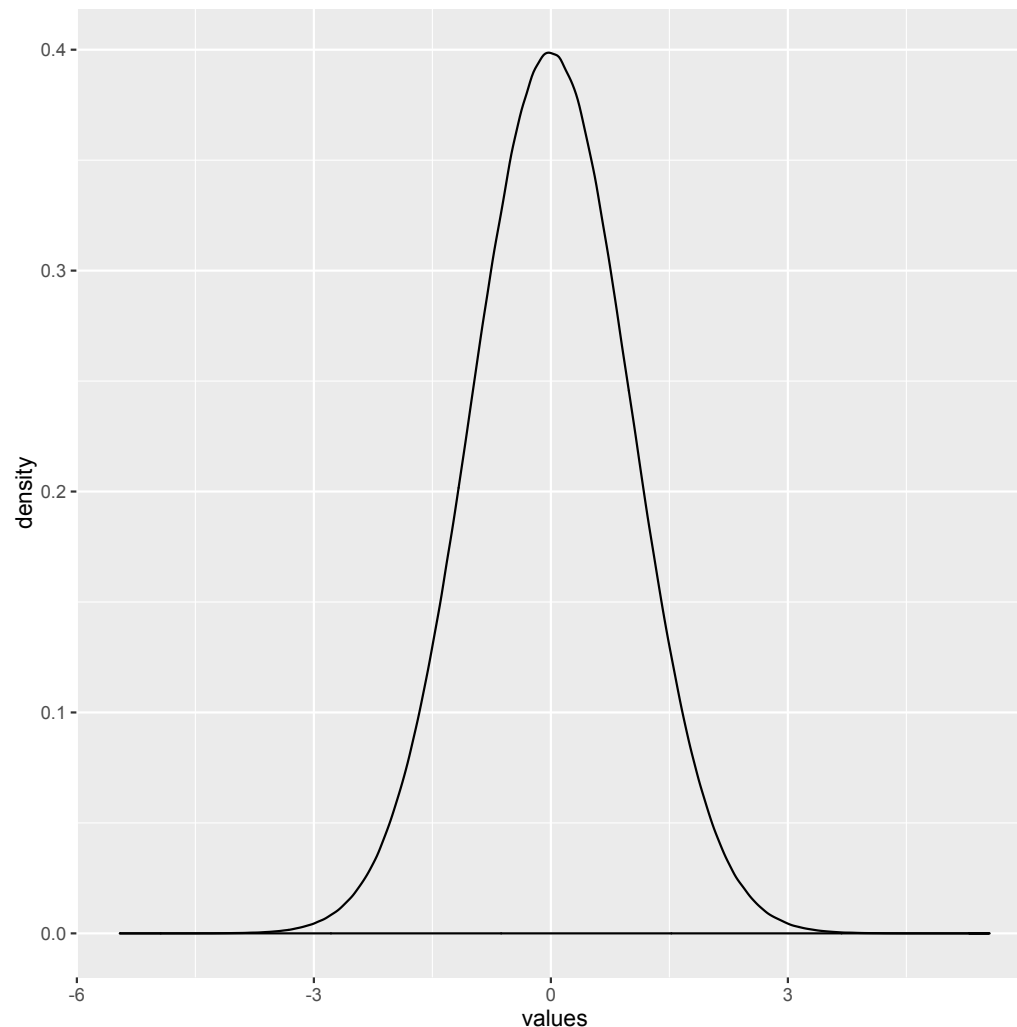
Believe it or not, these are all normal distributions.

The difference is that each one has a different mean (so a different location on the x-axis), and a different standard deviation (a different width on the x-axis, which also means a different height).



# A parametric bootstrap

Instead of trying to guess the mean and standard deviation, you can use the **standard normal distribution**, which is just a normal distribution with a mean of 0, and a standard deviation of 1. It is easy to work with.





# Converting our data to the standard normal distribution: z-scores appear again!

It is easy enough to use the standard normal distribution as our probability model for the population. R even gives us the built-in function `rnorm()`, which randomly samples from the standard normal distribution by default.

The problem is that our observed values are not on the same scale (the mean of the combined group of both of our condition is not 0). So we won't be able to compare our observed values to the reference distribution.

This is actually easy to fix. We can simply convert the values in our combined group into the standard normal distribution scale using our old friend the **z-score transformation**.

In this case, we are applying the z-score transformation to our combined data set (the thing that represents the full population), not each participant. That's the only difference. It is the same equation:

$$Z = \frac{X - \text{mean}}{\text{standard deviation}}$$

The result is that each value will be equal to its distance from the mean (as if the mean were 0), and that distance will be measured in units equal to the standard deviation. So our observed values will be on the same scale as our reference distribution!

# A parametric bootstrap

Now that we've decided on a probability function, and re-scaled our observed data, we simply carry out the bootstrap procedure like before:

First, we randomly sample with replacement two samples from our probability model. We call these **bootstrap replicates**. They are replicates because they are other possible samples that we could have obtained in our experiment. They are bootstrap replicates because this procedure is called the bootstrap method.

Second, we calculate the mean for each bootstrap replicate, and then calculate the mean difference.

Third, we save this mean difference (as the first value in our reference distribution).

Then we repeat this process a large number of times (e.g., 10,000) to derive a reference distribution called the **bootstrap distribution**.

Finally we calculate a p-value using the standard formula (and correction).

The script **bootstrap.r** contains code to run both a non-parametric and parametric bootstrap.

# Analytic methods

Because randomization and bootstrap methods are so computationally intensive, early 20th century statisticians could not use them. These people were smart. They developed analytic methods that give approximately the same result as randomization and bootstrap methods. And then shared them with the world.

The basic idea of analytic methods is that we need test statistics that have known, or easily calculable, reference distributions. We can't use the mean, because the distribution of the mean will vary based on the experiment (the data type, the design, etc). We need statistics that are relatively invariant, so that we can calculate the distribution once, and use it for every experiment in all of the different areas of science.

There are both **parametric** and **non-parametric** analytic methods, just like there are both parametric and non-parametric bootstrap methods. And there are a ton of different test statistics with different properties that are suited for different experimental situations.

For pedagogical reasons, I am going to focus on the **F statistic**.

# The F statistic is parametric

When people talk about parametric statistics, there is a typical cluster of three assumptions that they usually have in mind. The F statistic is parametric in this way - its distribution is predictable only if these assumptions are met:

**Normally distributed errors:**

The error terms in the linear model are normally distributed (which will be true if the population(s) of participants are normally distributed)

**Independence:**

The observed responses are independent (in repeated-measures designs this means the pairs of responses are independent)

**Homogeneity of variance:**

The variances of the samples are equal (homogeneous). This is always true when the null hypothesis is true, but also must be true when the null hypothesis is false.

There is a fourth assumption that typically accompanies these four under the rubric “parametric”, but it is not about the distribution of the statistic. It is about the inferences that can be drawn from it.

**Random Sampling:** Participants are randomly sampled from a population

# The F-distribution

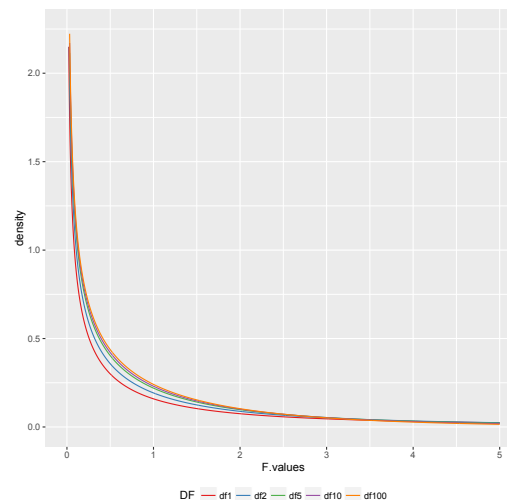
The distribution of the F statistic (called the Fisher-Snedecor distribution), is useful for analytic methods because it does not vary based on things like the mean or scale of the data. Instead, it is completely determined by two numbers, typically called  $df_1$  and  $df_2$ , or  $df_{num}$  and  $df_{den}$ , because of their relationships to the degrees of freedom in our calculation of F.

$$F = \frac{(SS_{simple} - SS_{complex}) / (df_{simple} - df_{complex})}{SS_{complex} / df_{complex}}$$

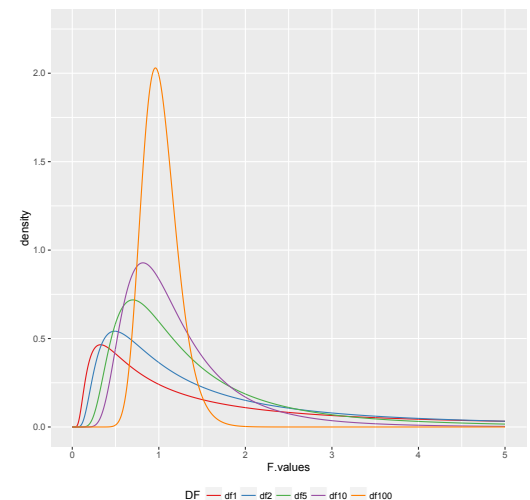
$$df_{num} = df_{simple} - df_{complex}$$

$$df_{den} = df_{complex}$$

If you want the equation for the probability density function, you can see it on the wikipedia page for the F distribution: <https://en.wikipedia.org/wiki/F-distribution>. It is fairly complicated, so I won't reproduce it here. But I will show you how the distribution varies with different dfs. In a 2x2, our df will be 1, as in the left figure. I include the right just to show you the full range of the F distribution. These plots are in [f.distribution.r](https://f.distribution.r).



$$df_{num} = 1$$



$$df_{num} = 100$$

$$df_{den} = 1, 2, 10, 100$$

# The ANOVA approach to F

The term ANOVA is just another way of saying F-test. It is actually the primary way, because most people think about tests, not about the statistics that they are using in that test.

ANOVA stands for **AN**alysis Of **V**ariance. What you should be thinking at this point is that we have never once discussed analyzing variance, so how is it that the F-tests that we have been discussing are analyses of variance?

Well, it turns out that there is a completely different, but equally valid, way of thinking about the F-ratio. Instead of a measure of error minimization per degrees of freedom, you can think of it as a ratio between two estimates of the population variance: the numerator is an estimate based on the sample means, and the denominator is an estimate based on the sample variance. (Don't worry, this will make more sense soon!)

$$F = \frac{\text{estimated } \sigma^2, \text{ based on sample means}}{\text{estimated } \sigma^2, \text{ based on the two sample variances}}$$

This is mathematically equivalent to the model comparison approach that I taught you, but conceptually different. I prefer model comparison; but most stats courses prefer the analysis of variance method. So now I will connect them for you!

# Analysis of Variance

The first thing to realize about what we've been doing so far is that we've seen two ways to use samples to estimate the variance of a population.

**Option 1:** Use the variance of the sample as an estimate

Recall from our first lecture that the variance of a sample ( $s^2$ ) can be used as an unbiased estimate of the population variance ( $\sigma^2$ ) if we use  $(n-1)$  in the calculation:

$$s^2 = \frac{\sum(Y_i - \bar{Y})^2}{(n-1)} = \text{estimate of } \sigma^2$$

In the case of an independent measures ANOVA, you actually have **two samples**! So you can come up with an even better estimate of  $\sigma^2$  by **averaging** the two estimates! (If one estimate is good, the average of two estimates will be better!) Here is a formula to let you do that for two samples:

$$\text{mean } s^2 = \frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{(n_1-1) + (n_2-1)}$$

# Analysis of Variance

The first thing to realize about what we've been doing so far is that we've seen two ways to use samples to estimate the variance of a population.

**Option 2:** Use the variance of two (or more) means

Now, this estimate you probably didn't even notice. The basic idea has two steps.

First, the variance of two (or more) means provides an estimate of the variance of the sampling distribution of means (the variability in all of the means that you could get if you repeatedly sampled from a population:  $\sigma_{\bar{Y}}^2$ ).

$$\text{estimate of } \sigma_{\bar{Y}}^2 = \frac{\sum(\bar{Y}_j - \bar{\bar{Y}})^2}{(j-1)}$$

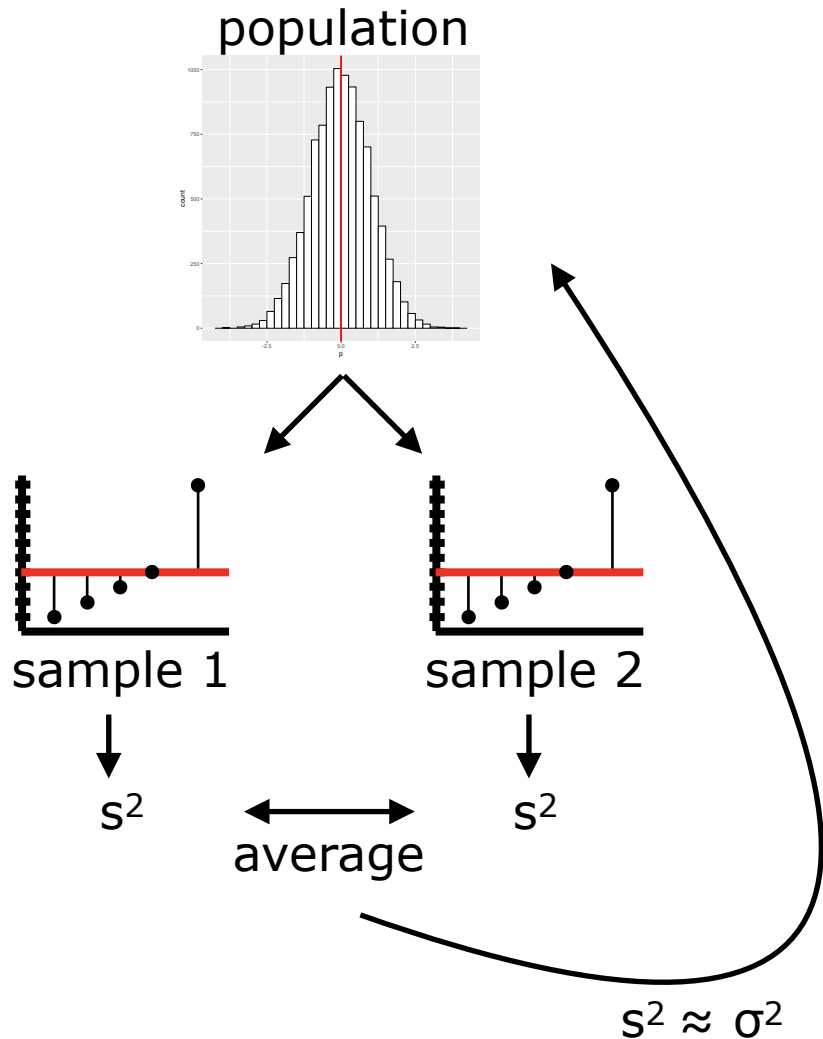
Second, the variance of sampling means ( $\sigma_{\bar{Y}}^2$ ) can be used to calculate the population mean:

$$\sigma_{\bar{Y}}^2 = \frac{\sigma^2}{n} \quad \text{therefore,} \quad \sigma^2 = n(\sigma_{\bar{Y}}^2)$$

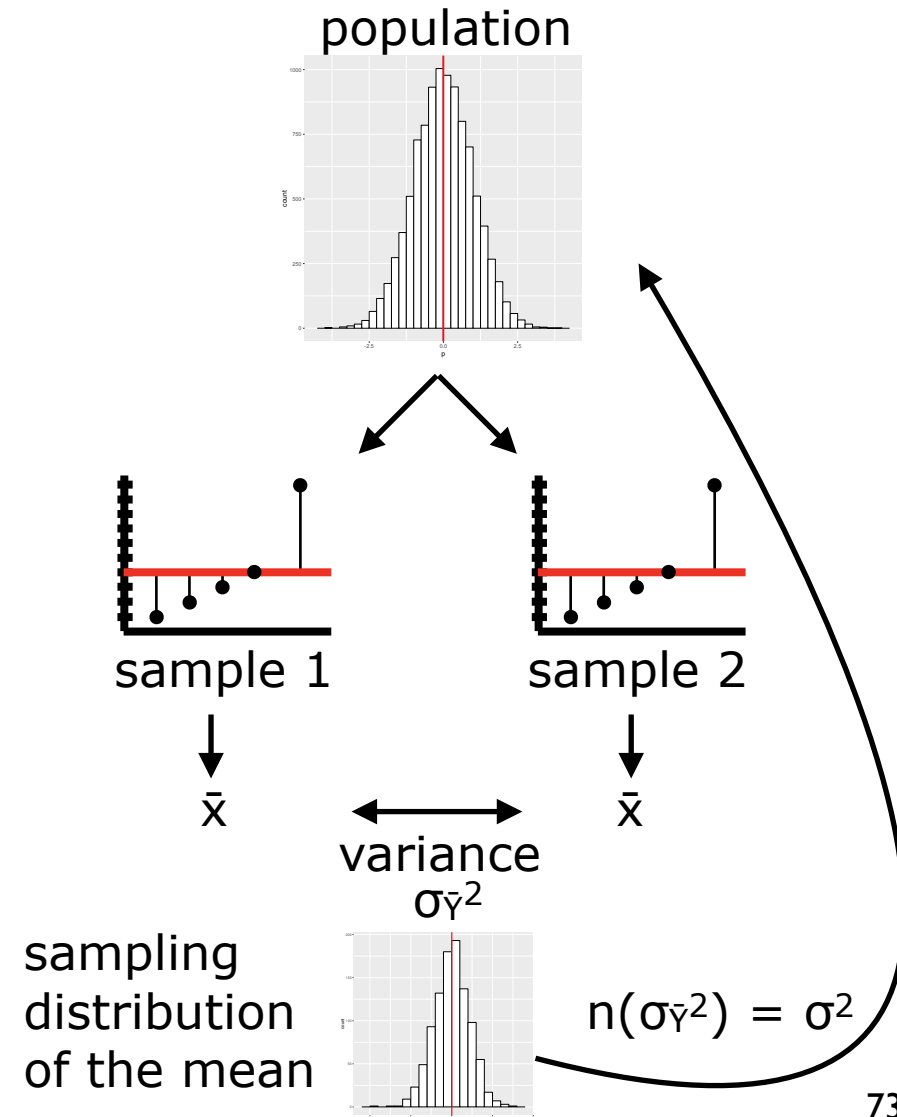


# Two estimates of population variance ( $\sigma^2$ )

Based on sample variance  
aka denominator  
aka within groups



Based on sample means  
aka numerator  
aka between groups



# Comparing the two estimates

We call the estimated variance based on the sample variances (Option 1) the **Within Groups Mean Squared Error, or  $MS_W$** .

The reason we call it this is because “mean squared error” is just another way to say variance; and it was an estimate that was calculated by averaging the variance of the two groups (within the groups).

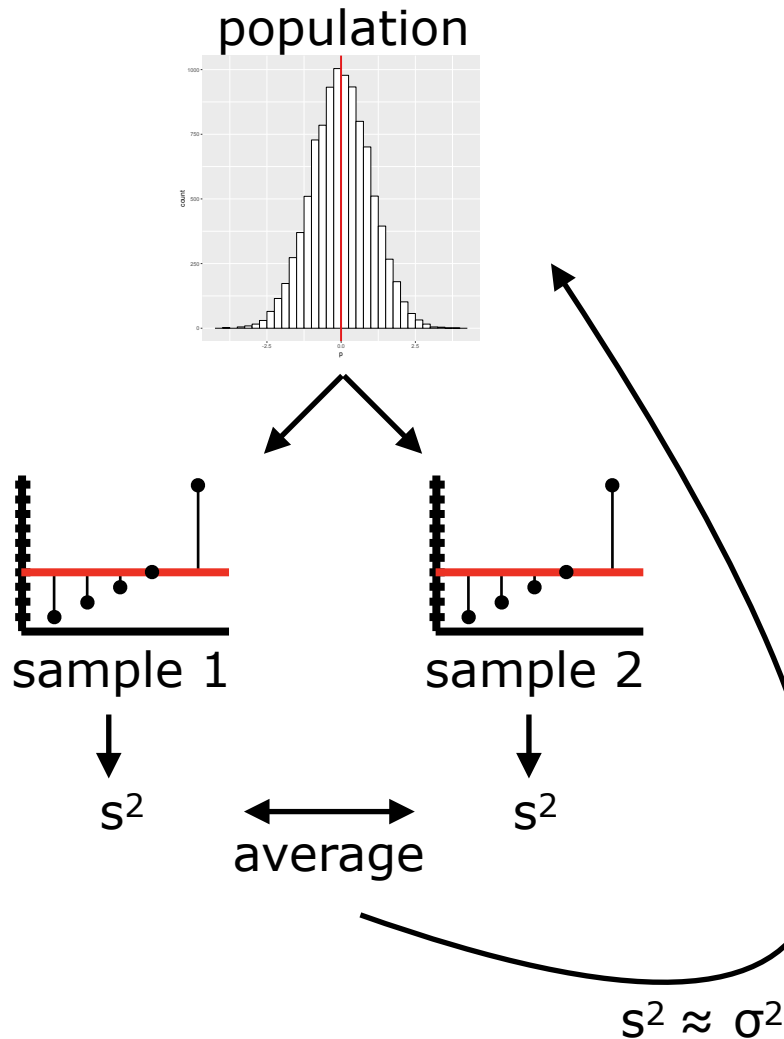
Assuming that variances are equal in both groups regardless of the hypothesis (null or alternative), which is an important assumption of ANOVAs, the  $MS_W$  **will not change based on whether the null hypothesis is true or false!**

We call the estimated variance based on the sample means (Option 2) the **Between Groups Mean Squared Error, or  $MS_B$** . This is because it used the variance between the means of the two groups to estimate the variance (mean squared error) of the population.

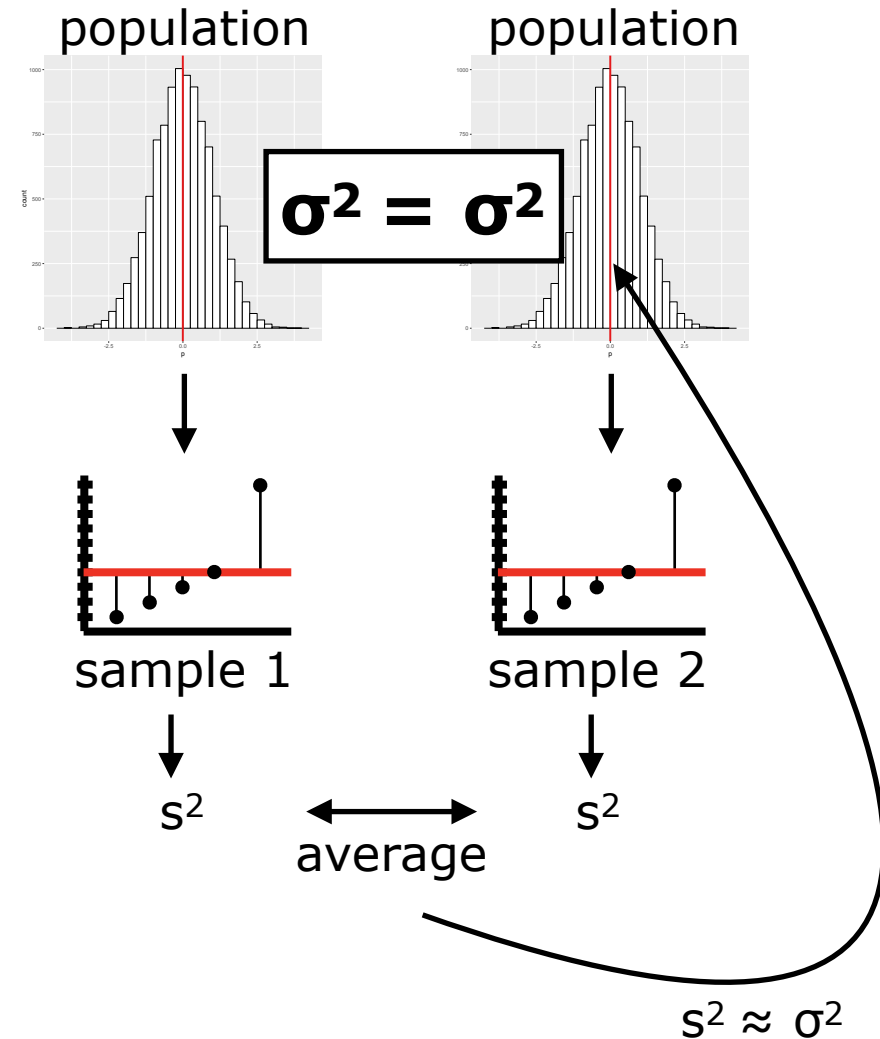
Now here is the neat thing. The  $MS_B$  will absolutely change depending on whether the null hypothesis is true or false. If the **null hypothesis is true**, then this estimate will be approximately the same as  $MSE_{WG}$ . But if the **null hypothesis is false**, this estimate will be **larger**. This is because the two means don't come from the same population, so they will likely be more different than two means that come from the same population.

# Within groups variance does not change based on the hypothesis

Within Groups  
Null Hypothesis is True

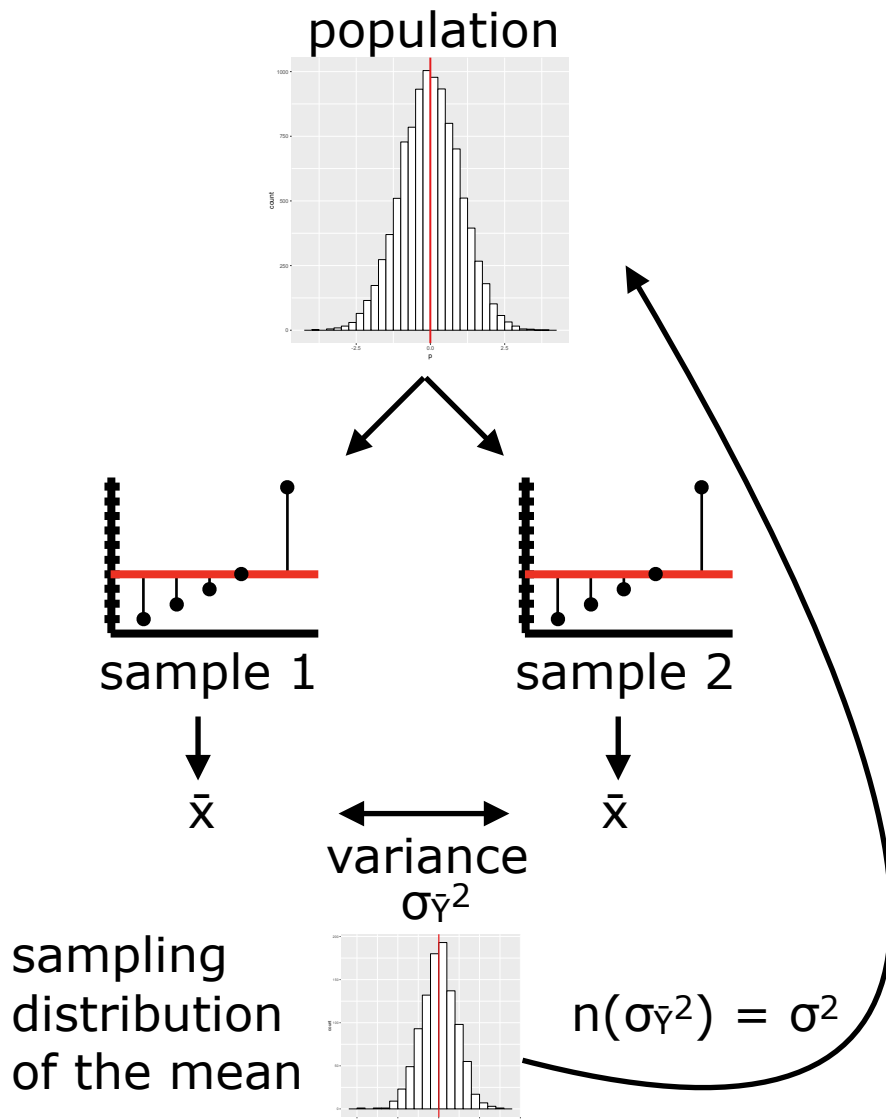


Within Groups  
Null Hypothesis is False

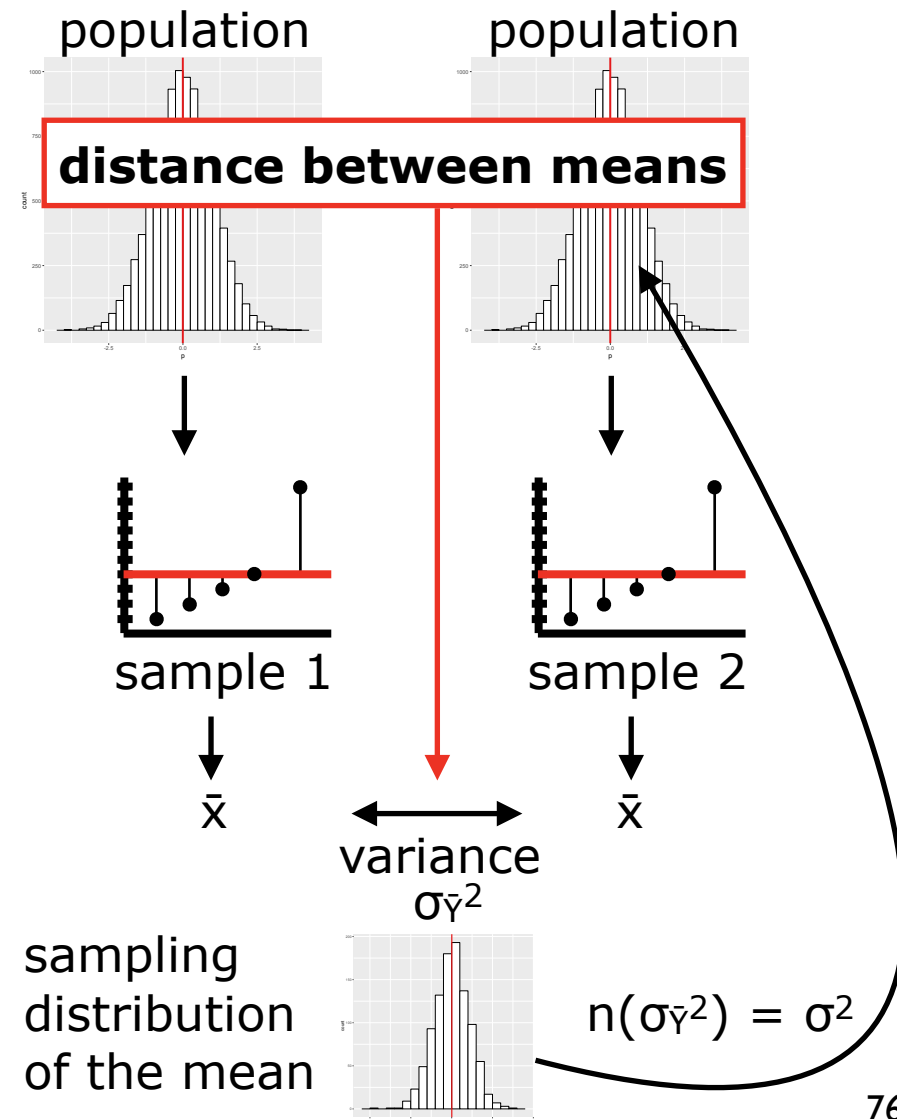


# Between groups variance does change based on the hypothesis

Between Groups  
Null Hypothesis is True



Between Groups  
Null Hypothesis is False



# This is also the F-ratio!

And now, yet another mind blowing moment:

$$F = \frac{MS_B}{MS_W}$$

Since  $MS_B$  gets larger when the null hypothesis is false,  $F$  will be larger (and will be close to 1 when the null is true).

Yup, the ratio between the estimate of the population variance based on mean variation and the estimate of the population variance based on sample variances is identical to the F-ratio that we've been talking about!

$$F = \frac{(SS_{\text{simple}} - SS_{\text{complex}})/(df_{\text{simple}} - df_{\text{complex}})}{SS_{\text{complex}}/df_{\text{complex}}}$$

We call the way we've been talking about the F-ratio the **model comparison approach**, because it emphasizes the comparison of two models. We call the new approach the **analysis of variance approach**, hence ANOVA. They are mathematically equivalent (I will leave it to you to work out the math), and they are equally valid for defining the F statistic for a test. Although I prefer using the model comparison approach, both are equally valid ways of thinking about F-tests.

# And just FYI, $F = t^2$

We haven't looked at t-tests at all in this class, but some of you may have heard of them. A t-test is a way of comparing one mean to 0, or two means to each other, using the t-statistic. What you may find interesting is that F and t are related. F is  $t^2$ .

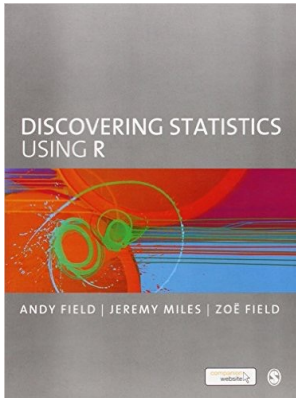
We can see this easily with our toy example from earlier. Let's calculate both an F for these two models, and a t for the complex model versus the constant in the simple model.

simple			complex		
$Y_i$	$= \beta_0 X_{0-i}$	$+ \epsilon_i$	$Y_i$	$= \beta_0 X_{0-i}$	$+ \epsilon_i$
2	= 4	+ -2	2	= 3	+ -1
3	= 4	+ -1	3	= 3	+ 0
4	= 4	+ 0	4	= 3	+ 1
df=3		SS=5	df=2		SS=2

$$F = \frac{(SS_{\text{simple}} - SS_{\text{complex}})/(df_{\text{simple}} - df_{\text{complex}})}{SS_{\text{complex}}/df_{\text{complex}}} = \frac{(5-2)/(3-2)}{2/2} = 3$$

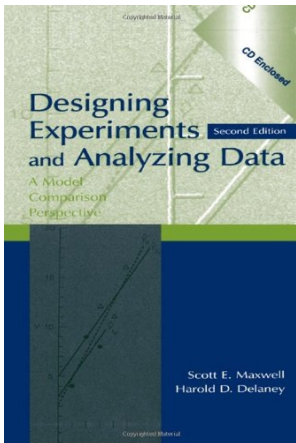
$$t = \frac{\bar{Y} - \mu}{\sqrt{\left(\frac{s^2}{n}\right)}} = \frac{3 - 4}{\sqrt{\left(\frac{1}{3}\right)}} = -1.732051$$

# Analytic methods: more information



## **Discovering Statistics Using R Field, Miles, and Field**

This book is a comprehensive introduction to (analytic) statistics, and it is a great introduction to R (and plotting with R). It is very readable (and at times, amusing), and covers all of the things that are covered in fundamental statistics courses.



## **Designing Experiments and Analyzing Data Maxwell and Delaney**

This is probably the best book on the model comparison approach to F-tests there is. It is also a beast of a book. But well worth it if you really want to understand F-tests. There is no R here. This is math.