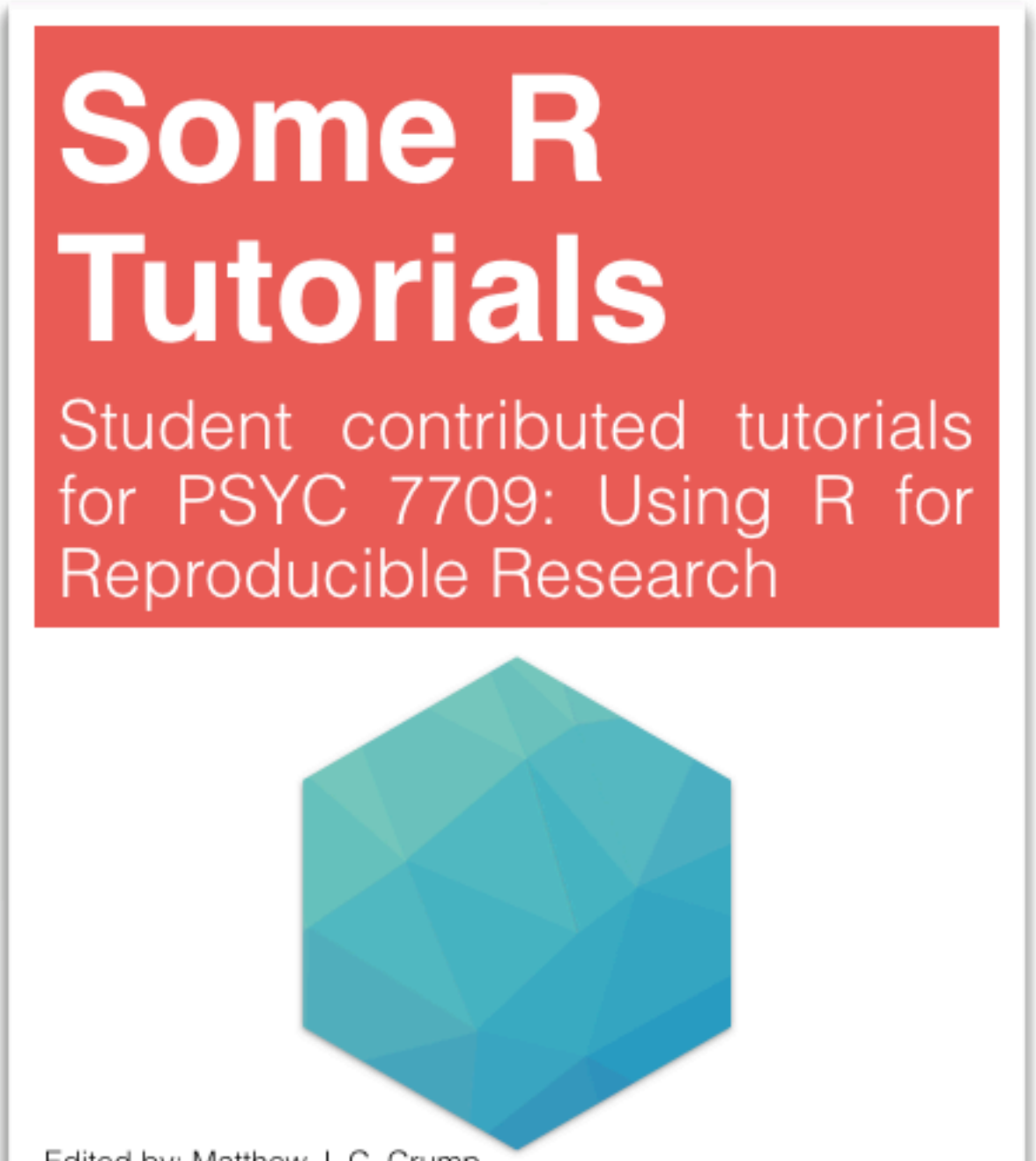
PSYC 7709: Using R for Reproducible Research

Matthew Crump

2018: Last compiled 2019-05-20

Contents

Preface


Some R Tutorials
Student contributed tutorials for PSYC 7709: Using R for Reproducible Research
Edited by: Matthew J. C. Crump
Authors: Kamal Abdelrahman, Aira Contreras, Zoren Degtyarev, Javante Foster, Ana-Louise Franz, Troy Funderburk, Melissa Horger, Jeffrey Kravitz, Ana Lakshin, Marsha Manigat, Richard

Crump, Matthew J. C. (2018). Open tools for writing open interactive textbooks (and more).

This is a tutorial and set of working examples for creating web-based textbooks using a collection of open-source tools.

Chapter 1

Overview

Something about the course and each of the chapters.

Chapter 2

Bootstrapped Mediation Tutorial

Author: Jeff Kravtiz

This tutorial will a) review the concepts of mediation, moderation, and bootstrapping; b) demonstrate how to complete bootsrapped mediation analyses in R; and c) generate hypothetical datasets to explore various possible outcomes.

2.1 Mediation, Moderation, and Bootstrapping

Basic statistical analyses like ANOVAs allow us to test the effect of one variable on another. However, we often want to go beyond testing these simple relationships to examine more complex relationships, allowing us to test theoretically rich ideas. Mediation and Moderation analyses are two commonly used statisitcal techniques that allow us to explore these complex relationships.

2.1.1 Mediation - Overview

Mediation analyses help us examine the **mechanism** of the relationship between two variables. This technique answers the question of *how* variables are related. A mediator variable **explains the relationship** between an independent variable and a dependent variable.

For example, consider a case in which we know that mindfulness meditation increases happiness in people. Although there is clear evidence that mindfulness increases happiness, research has not yet explored why this is the case. Suppose that we think that mindfulness decreases stress, which in turn increases happiness. In other words, we think that mindfulness only indirectly affects happiness *by means of* reducing stress.

2.1.2 Moderation - Overview

Moderation analyses help us examine the **conditions** in which two variables are related. This technique answers the question of *when* variables are related. A moderator variable **affects the strength and/or direction** of the relationship between an independent variable and a dependent variable.

Consider again the above case in which we know that mindfulness meditation increases happiness in people. Various studies examine this effect, and while most studies produce positive findings, some produce null findings. We suspect that there certain situations in which mindfulness does not increase happiness. Specifically, we hypothesize that mindfulness can only function properly when people have a sufficient attention span. In other words, we expect that the effect of mindfulness on happiness *depends on* an individual's attention span.

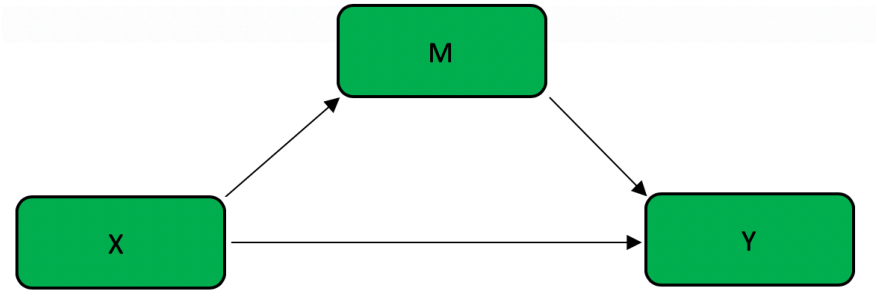
2.1.3 Bootstrapping

Bootstrapping is a statistical method that utilizes random resampling with replacement to estimate a population parameter. This technique samples from a given dataset to estimate a parameter when it would otherwise be impossible or impractical to do so. In this way, the dataset is treated as the population, and each random sample aims to replicate a potential score within the true population. The amount of samples varies, but usually falls between 1,000 and 10,000.

One advantage of the bootstrap method is that it produces confidence intervals for your statisitcal estimate. This gives us valuable information of the likely value of a parameter, whereas a p value simply gives us a single number that estimate the likelihood of our statistic assuming that the null is true.

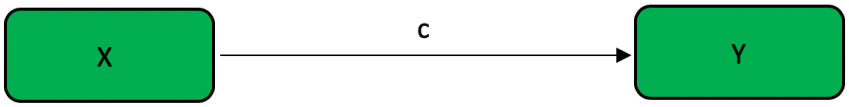
2.2 Mediation - Effects

The purpose of mediation analyses is to determine if the effect of an independent variable (X) on a dependent variable (Y) can be explained by a mediating variable (M). This can be visualized in the following figure:

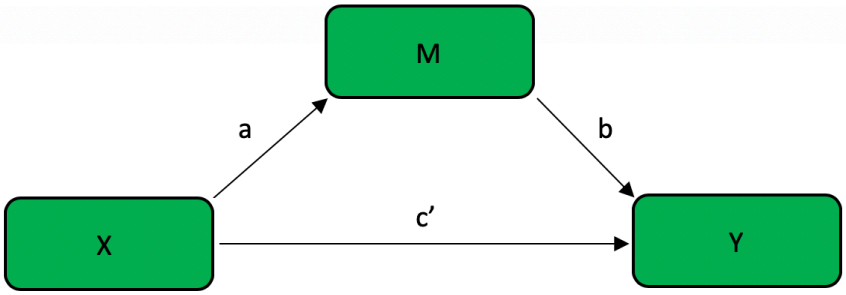


There are various different effects to consider when conducting a mediation analysis.

- The **total effect** is the total effect of X on Y, with no consideration of mediator variables. This is also known as the **c path**



- The **direct effect** is the effect of X on Y after removing the contribution of M. This is known as the **c' path**
- The effect of X on M is known as the **a path**
- The effect of M on Y is known as the **b path**
- The **indirect effect** is the effect of X on Y *through M*. This is equivalent to the product of the a path and the b path, sometimes known as the **ab path**



2.3 Bootstrapped Mediation in R

This tutorial demonstrates how to conduct bootstrapped mediation analyses using the `mediate` function in the `mediation` package. As such, the first step is to install and load the `mediation` package.

```
install.packages("mediation")
library(mediation)
```

2.3.1 Structure

In this section, I will first introduce and then breakdown the structure of mediation using the `mediate` function. The basic structure is as follows:

```
X <-
Y <-
M <-


model.m <- lm(M ~ X)
model.y <- lm(Y ~ X + M)

library(mediation)
mediation_results <- mediate(model.m = model.m,
                        model.y = model.y,
                        sims =
                        boot = TRUE,
                        mediator = "   ",
                        treat = "   ")
summary(mediation_results)
```

2.3.2 Defining Variables

The next step in mediation is defining your variables. As I mentioned earlier, X will denote the independent variable; Y will denote the dependent variable; and M will denote the mediator variable. We will assign X, Y, and M to be vectors of equal length representing participants' scores on each of these variables.

```
X <-
Y <-
M <-
```

In order to demonstrate a practical example, let us revisit the aforementioned example in which we want to investigate whether mindfulness increases happiness by decreasing stress. Suppose that we have one randomly-assigned, between-subjects independent variable of mindfulness training, with one experimental condition in which 50 participants practice mindfulness meditation for 20 minutes (dummy coded 1), and one control condition in which 50 participants close their eyes for 20 minutes (dummy coded 0). We measure both stress levels and happiness, each on a 1 - 7 likert scale, where 7 indicates minimum stress or maximum happiness.

```
X_mindfulness <- c(rep(0,50),rep(1,50))
Y_happiness <- runif(100, min = 1, max = 7)
M_stress  <- runif(100, min = 1, max = 7)
```

2.3.3 Defining Models

To use the `mediate` function, we must first define two linear models:

- `model.m`, predicting M from X
- `model.y`, predicting Y from X and M

```
model.m <- lm(M ~ X)
model.y <- lm(Y ~ X + M)
```

In our example, it would look like this:

```
model.m <- lm(M_stress ~ X_mindfulness)
model.y <- lm(Y_happiness ~ X_mindfulness + M_stress)
```

2.3.4 Mediate

Next, we call the `mediate` function with the following arguments:

- `model.m` = `model.m`
- `model.y` = `model.y`
- `sims` = *the number of resamples that you want to use*
- `boot` = `TRUE`
- `mediator` = *your mediator variable,* ***in quotes***
- `treat` = *your independent variable,* ***in quotes***

```
mediation_results <- mediate(model.m = model.m,
                        model.y = model.y,
                        sims =
                        boot = TRUE,
                        mediator = "  ",
                        treat = "  ")
summary(mediation_results)
```

In our example, it would look like this:

```
full_mediation <- mediate(model.m = model.m,
                        model.y = model.y,
                        sims = 500,
                        boot = TRUE,
                        mediator = "M_stress",
```

```
                       treat = "X_mindfulness")
summary(full_mediation)
```

2.4 Generating Datasets to Interpret Outcomes

Because there are various possible outcomes of a mediation analysis, it is useful to generate datasets to model and interpret each outcome. The possible results of a mediation analysis include a) full mediation, in which the effect of the independent variable on the dependent variable is completely explained by the mediator variable; b) partial mediation, in which in which the effect of the independent variable on the dependent variable is partially explained by the mediator variable, but the independent variable still has some direct effect on the dependent variable; and c) no mediation.

Let's first look at the results of our running example:

```
summary(mediation_results)
```

```
##
## Causal Mediation Analysis
##
## Nonparametric Bootstrap Confidence Intervals with the Percentile Method
##
##                Estimate 95% CI Lower 95% CI Upper p-value
## ACME             0.0214      -0.0833         0.12    0.87
## ADE              0.0391      -0.6306         0.69    0.98
## Total Effect     0.0605      -0.6295         0.69    0.94
## Prop. Mediated   0.3534      -2.3995         1.52    0.97
##
## Sample Size Used: 100
##
##
## Simulations: 500
```

- `ACME` is the Average Causal Mediation Effect, or the **indirect effect** (ab path)
- `ADE` is the Average Direct Effect, or the **direct effect** (c' path)
- `Total Effect` is the **total effect** (c path)
- Our output also gives us confidence intervals and p values for each of these effects

Because this data was generated randomly, it is likely that none of the effects are significant. If we impose a structure on our data when generating our example datasets, we can examine each of the possible outcomes of mediation.

2.4.1 Full Mediation

In the case of full mediation, the relationship between mindfulness (X) and happiness (Y) is completely explained by stress (M).

```
# Full Mediation

X_mindfulness <- c(rep(0,50),rep(1,50))
Y_happiness <- c(rep(3,25),rep(4,25),rep(5,25),rep(6,25))
M_stress  <- c(rep(3,25),rep(4,25),rep(5,25),rep(6,25))
```

```
model.m <- lm(M_stress ~ X_mindfulness)
model.y <- lm(Y_happiness ~ X_mindfulness + M_stress)

library(mediation)
full_mediation <- mediate(model.m = model.m,
                          model.y = model.y,
                          sims = 500,
                          boot = TRUE,
                          mediator = "M_stress",
                          treat = "X_mindfulness")
summary(full_mediation)
```

```
##
## Causal Mediation Analysis
##
## Nonparametric Bootstrap Confidence Intervals with the Percentile Method
##
##                 Estimate 95% CI Lower 95% CI Upper p-value
## ACME            2.00e+00     1.81e+00          2.2  <2e-16 ***
## ADE            -2.33e-15    -4.07e-15          0.0    0.92
## Total Effect    2.00e+00     1.81e+00          2.2  <2e-16 ***
## Prop. Mediated  1.00e+00     1.00e+00          1.0  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Sample Size Used: 100
##
##
## Simulations: 500
```

In this case, we know we have a full mediation because we have a significant total effect and indirect effect (ACME), but not a direct effect (ADE). The significant total effect tells us that X affects Y. The significant indirect effect tells us that there was a mediating effect of M. The non-significant direct effect tells us that all of the effect of X on Y was explained by M, showing a full mediation.

2.4.2 Partial Mediation

```
# Partial Mediation

X_mindfulness <- c(rep(0,50), rep(1,50))
Y_happiness <- c(rep(3,25), rep(4,25), rep(5,25), rep(6,25))
M_stress  <- c(rnorm(25,3,1), rnorm(25,4,1), rnorm(25,5,1), rnorm(25,6,1))


model.m <- lm(M_stress ~ X_mindfulness)
model.y <- lm(Y_happiness ~ X_mindfulness + M_stress)

library(mediation)
partial_mediation <- mediate(model.m = model.m,
                          model.y = model.y,
                          sims = 500,
                          boot = TRUE,
                          mediator = "M_stress",
```

```
                       treat = "X_mindfulness")
summary(partial_mediation)
```

```
##
## Causal Mediation Analysis
##
## Nonparametric Bootstrap Confidence Intervals with the Percentile Method
##
##                Estimate 95% CI Lower 95% CI Upper p-value
## ACME              0.400        0.267         0.57  <2e-16 ***
## ADE               1.600        1.396         1.83  <2e-16 ***
## Total Effect      2.000        1.819         2.21  <2e-16 ***
## Prop. Mediated    0.200        0.134         0.28  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Sample Size Used: 100
##
##
## Simulations: 500
```

Here we know we have a partial mediation because we have a significant total effect, indirect effect (ACME), and direct effect (ADE). The significant total effect tells us that X affects Y. The significant indirect effect tells us that there was a mediating effect of M. The significant direct effect tells us that although there was a mediating effect of M, there was still a direct relationship between X and Y, showing a partial mediation.

2.4.3 No Mediation

```
# No Mediation

X_mindfulness <- c(rep(0,50), rep(1,50))
Y_happiness <- c(rep(3,25), rep(4,25), rep(5,25), rep(6,25))
M_stress  <- runif(100, min = 1, max = 7)

model.m <- lm(M_stress ~ X_mindfulness)
model.y <- lm(Y_happiness ~ X_mindfulness + M_stress)

library(mediation)
no_mediation <- mediate(model.m = model.m,
                        model.y = model.y,
                        sims = 500,
                        boot = TRUE,
                        mediator = "M_stress",
                        treat = "X_mindfulness")
summary(no_mediation)
```

```
##
## Causal Mediation Analysis
##
## Nonparametric Bootstrap Confidence Intervals with the Percentile Method
##
##                Estimate 95% CI Lower 95% CI Upper p-value
## ACME            -0.0036      -0.0425         0.02    0.76
```

```
## ADE              2.0036       1.7942         2.22  <2e-16 ***
## Total Effect     2.0000       1.7952         2.21  <2e-16 ***
## Prop. Mediated  -0.0018      -0.0224         0.01    0.76
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Sample Size Used: 100
##
##
## Simulations: 500
```

Here we know we have no mediation because we have a non-significant indirect effect (ACME), which means that the relationship between X and Y was not at all explained by M.

Chapter 3

A tutorial for using the lme function from the nlme package.

Author: Melissa Horger

3.1 The nlme package

nlme is a package for fitting and comparing linear and nonlinear mixed effects models.

It let's you specify variance-covariance structures for the residuals and is well suited for repeated measure or longitudinal designs.

3.1.1 Similar packages

One similar package is lme4. It allows you to fit outcomes whose distribution is not Gaussian and crossed random effects. Some pros include that it stores data more effiently due to the use of sparse matrices and it works well with clustered data sets.

3.1.2 What's included in nlme?

nlme contains sample data, statistical functions, matrices, and a lattice framework.

Examples of sample data

1. MathAchSchool
    - A dataset (160x7) that contains school-level demographic data such as student enrollment, academic progress, and SES.

```
library(nlme)
print(MathAchSchool[1:5,])
```

```
##      School Size   Sector PRACAD DISCLIM HIMINTY MEANSES
## 1224   1224  842   Public   0.35   1.597       0  -0.428
## 1288   1288 1855   Public   0.27   0.174       0   0.128
## 1296   1296 1719   Public   0.32  -0.137       1  -0.420
## 1308   1308  716 Catholic   0.96  -0.622       0   0.534
## 1317   1317  455 Catholic   0.95  -1.694       1   0.351
```

2. BodyWeight
   - A data set (176x4) describing the weight of rats over time while consuming different diets.

```
print(BodyWeight[1:5,])
```

```
## Grouped Data: weight ~ Time | Rat
##   weight Time Rat Diet
## 1    240    1   1    1
## 2    250    8   1    1
## 3    255   15   1    1
## 4    260   22   1    1
## 5    262   29   1    1
```

3. Earthquake
   - A dataset (182x5) listing seismic measurements of 23 large earthquakes in western North America between 1940 and 1980

```
print(Earthquake[1:5,])
```

```
## Grouped Data: accel ~ distance | Quake
##     Quake Richter distance soil accel
## 132    20       5      7.5    1 0.264
## 133    20       5      8.8    1 0.263
## 134    20       5      8.9    1 0.230
## 135    20       5      9.4    1 0.147
## 136    20       5      9.7    1 0.286
```

Examples of functions 1. anova.lme - This compares the likelihoods of fitted models. It will produce an AIC and BIC and can be used to compare null and predictive models or models with different predictors and/or interactions.

```
anova.lme(object, ..., test=TRUE,type = "sequential/marginal", adjustSigma, Terms, L, verbose)
```

2. corMatrix - A function to generate the correlation matrix of an object/dataset.

```
corMatrix(object, ...)
```

3. gapply - Applies a function to a distinct set of rows in a data frame. - To use this, the rows must first be identified using the "groups" function

```
gapply(object, which, FUN, form, level, groups, .)
```

4. lme - We will learn about this function extensively in the following sections

3.1.3 Using the nlme package

3.1.3.1 Begin by installing the nlme package

Found on the CRAN repository

Website: https://svn.r-project.org/R-packages/trunk/nlme

```
install.packages("nlme")
```

3.1.3.2 Load the package (and other relevant packages)

```
library(ggplot2)
library(nlme)
library(dplyr)
library(knitr)
```

3.2 The lme function

This generic function fits a linear mixed-effects model in the formulation described in Laird and Ware (1982) but allowing for nested random effects. The within-group errors are allowed to be correlated and/or have unequal variances.

3.2.1 Some important considerations

1. Need repeated measures from a single subject The data may be longitudinal, but they also may not.
2. Can account for correlations within individuals within the random effects
3. Uses maximum likelihood estimates

3.2.2 The arguments for this function

lme(model, data, fixed, random, groups, start, correlation, weights, subset, method, na.action, naPattern, control, verbose)

3.3 An example: Does the number of daily naps impact infant performance on a thing?

```
#creating a data set

Subs <- rep(c(seq(1:10)), 4)

Month <- c(rep(c(1), 10), rep(c(5), 10), rep(c(10), 10), rep(c(15), 10))

Naps <- c(rep(c(3), 10), 2, 3, 2, 1, 2, 3, 2, 3, 2, 3, 2, 2, 2 ,2, 3, 3, 2, 2, 1, 2, 3, 1, 2, 2, 1, 1, 1

Napsfactor <- as.factor(Naps)

#Let's assume that infants' performance will get better with time. I altered the possible sampling dist

scores <- c(runif(10, 1, 7), runif(10, 8, 15), runif(10, 16, 22), runif(10, 23, 30))


dataset <- data.frame(Subs, Month, Naps, scores, Napsfactor)

#save(dataset,file="horger.RData")
```

```
load("data/horger.RData")
```

```
#Data should be set up in long format and look similar to this.
print(dataset)
```

```
##    Subs Month Naps    scores Napsfactor
## 1     1     1    3  1.204228          3
## 2     2     1    3  4.215435          3
## 3     3     1    3  6.530265          3
## 4     4     1    3  3.651179          3
## 5     5     1    3  2.085147          3
## 6     6     1    3  4.913243          3
## 7     7     1    3  2.683738          3
## 8     8     1    3  3.372563          3
## 9     9     1    3  4.621265          3
## 10   10     1    3  3.105155          3
## 11    1     5    2 13.572888          2
## 12    2     5    3 10.021787          3
## 13    3     5    2 14.754621          2
## 14    4     5    1 11.818922          1
## 15    5     5    2 13.203504          2
## 16    6     5    3 12.327272          3
## 17    7     5    2 13.829133          2
## 18    8     5    3  9.144146          3
## 19    9     5    2  9.209897          2
## 20   10     5    3  8.146771          3
## 21    1    10    2 20.521925          2
## 22    2    10    2 19.207385          2
## 23    3    10    2 19.908420          2
## 24    4    10    2 21.929604          2
## 25    5    10    3 16.036066          3
## 26    6    10    3 19.170270          3
## 27    7    10    2 16.097766          2
## 28    8    10    2 17.480285          2
## 29    9    10    1 20.111028          1
## 30   10    10    2 18.278805          2
## 31    1    15    3 23.089659          3
## 32    2    15    1 26.585482          1
## 33    3    15    2 24.839186          2
## 34    4    15    2 24.917255          2
## 35    5    15    1 23.046358          1
## 36    6    15    1 28.955977          1
## 37    7    15    2 26.056173          2
## 38    8    15    1 28.870685          1
## 39    9    15    2 26.720957          2
## 40   10    15    1 26.898722          1
```

3.3.1 The experimental design.

This is a 4x3 within subject design. Infants are assessed at 4 time points - 1 month, 5 months, 10 months, and 15 months. There are 3 levels of napping - 1, 2, or 3 naps per day.

3.3.2 Data analysis

We will run a conditional growth model because we are including predictors. Subsequent fixed and random effects are now "conditioned on" the predictors (age and number of naps).

```
#Conditional growth model
tutorial<-lme(scores ~ Month * Naps, random = ~ Month | Subs, data=dataset)


#Because we are using a random sample, may need to rerun the scores several times for this piece of cod
```

lme(model, random, data)

model - scores ~ Month * Naps

We expect scores will be influenced by how old infants are (Month) and the number of Naps they take per day. There may be an interaction between these predictors.

random - random = ~ Month | Subs

Random error comes from the fact that this is a within subjects design. The same infants are assessed at 1 month, 5 months, 10 months, and 15 months.

data - data=dataset

Using the data set we created previously.

3.3.3 Displaying our results

```
tutorial
```

```
## Linear mixed-effects model fit by REML
##   Data: dataset
##   Log-restricted-likelihood: -81.67538
##   Fixed: scores ~ Month * Naps
## (Intercept)       Month        Naps  Month:Naps
##  10.8206754   1.1464573  -2.8094967   0.1105195
##
## Random effects:
##  Formula: ~Month | Subs
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev       Corr
## (Intercept) 4.728576e-01 (Intr)
## Month       1.238514e-05 0
## Residual    1.732905e+00
##
## Number of Observations: 40
## Number of Groups: 10
```

We can move the results to a nicer table using the summary function

```
#summarize an lme object - our solution
tut <- summary(tutorial)
tabl = tut$tTable
tabl
```

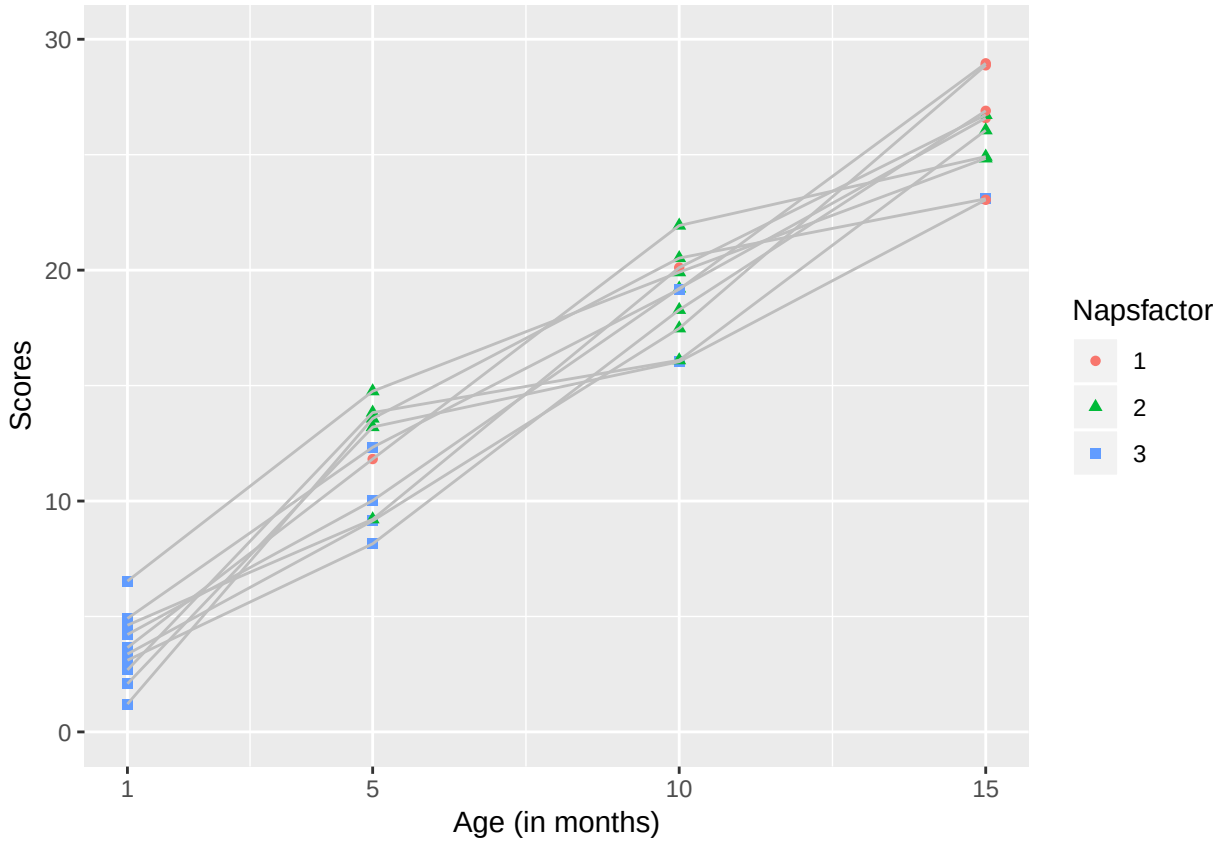
```
##                  Value  Std.Error DF   t-value      p-value
## (Intercept) 10.8206754 2.59927655 27  4.162957 2.872090e-04
```

```
## Month        1.1464573 0.21022323 27  5.453523 9.028044e-06
## Naps        -2.8094967 0.92823701 27 -3.026702 5.381092e-03
## Month:Naps   0.1105195 0.08248428 27  1.339886 1.914511e-01
```

From this analysis, we would conclude that there is a main effect of age, infants performance improved with age, but there is no effect of number of naps. It was trending in the correct direction as the only negative slope.

3.3.4 Graphing our results

```
library(ggplot2)
plot<- ggplot(dataset, aes(x=Month, y=scores,  color=Napsfactor, shape = Napsfactor, group = Subs), xli
  geom_point()+
  geom_line(color="grey")

plot + scale_x_continuous(name="Age (in months)", limits=c(1, 15), breaks =c(1,5,10,15)) +
  scale_y_continuous(name="Scores", limits=c(0, 30))
```



This kind of graph allows us to see the developmental trajectory of individual infants. It highlights the fact that there are 2 developmental trends occuring here- Infants' performance on the assessment is improving with time and the average number of naps they take is decreasing with time.

3.4 Continuing our example: Single main effect versus two main effects or an interaction

When making the original data set, I intentionally biased the data to show a developmental curve by increasing the sampling distribution for each age range. I could do something similar to bias our data to support the impact of taking fewer naps

```
#Create a new data set
Subs <- rep(c(seq(1:10)), 4)

Month <- c(rep(c(1), 10), rep(c(5), 10), rep(c(10), 10), rep(c(15), 10))

Naps <- c(rep(c(3), 10), 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 3, 3, 3 ,2,2,  2, 2,2,1, 1, 2, 2, 2, 2, 2, 1, 1,
Napsfactor <- as.factor(Naps)

secondscores <- c(runif(10, 1, 10), runif(5, 5, 10),runif(5, 9, 17), runif(3, 10, 15), runif(5, 14, 22)
```

```
seconddataset <- data.frame(Subs, Month, Naps, secondscores, Napsfactor)
#save(seconddataset, file="horger2.RData")
```

```
load("data/horger2.Rdata")
print(seconddataset)
```

```
##    Subs Month Naps secondscores Napsfactor
## 1     1     1    3     7.621794          3
## 2     2     1    3     7.794263          3
## 3     3     1    3     3.168637          3
## 4     4     1    3     3.415735          3
## 5     5     1    3     8.465790          3
## 6     6     1    3     7.971604          3
## 7     7     1    3     8.553510          3
## 8     8     1    3     7.115589          3
## 9     9     1    3     8.379775          3
## 10   10     1    3     8.840234          3
## 11    1     5    3     7.416176          3
## 12    2     5    3     7.583637          3
## 13    3     5    3     5.250097          3
## 14    4     5    3     6.014483          3
## 15    5     5    3     5.372417          3
## 16    6     5    2     9.927262          2
## 17    7     5    2    12.315161          2
## 18    8     5    2    13.054388          2
## 19    9     5    2    15.337895          2
## 20   10     5    2     9.226133          2
## 21    1    10    3    10.863713          3
## 22    2    10    3    12.495100          3
## 23    3    10    3    10.480098          3
## 24    4    10    2    20.347891          2
## 25    5    10    2    16.173209          2
## 26    6    10    2    20.144221          2
## 27    7    10    2    14.771556          2
## 28    8    10    2    20.676701          2
## 29    9    10    1    23.473947          1
## 30   10    10    1    21.244746          1
```

```
## 31    1    15    2    22.276560          2
## 32    2    15    2    21.167274          2
## 33    3    15    2    22.405964          2
## 34    4    15    2    19.556372          2
## 35    5    15    2    18.950262          2
## 36    6    15    1    25.115982          1
## 37    7    15    1    23.578970          1
## 38    8    15    1    23.747330          1
## 39    9    15    1    26.066104          1
## 40   10    15    1    28.513882          1
```

3.4.1 Did the manipulation work?

```
#Summary stats from our first dataset
demos <- dataset %>%
    group_by(Month, Naps) %>%
  summarise(mean_score = mean(scores, na.rm=TRUE))



#Summary stats from our second dataset
seconddemos <- seconddataset %>%
    group_by(Month, Naps) %>%
  summarise(mean_secondscore = mean(secondscores, na.rm=TRUE))


print(demos)
```

```
## # A tibble: 10 x 3
## # Groups:   Month [4]
##    Month  Naps mean_score
##    <dbl> <dbl>      <dbl>
##  1     1     3       3.64
##  2     5     1      11.8
##  3     5     2      12.9
##  4     5     3       9.91
##  5    10     1      20.1
##  6    10     2      19.1
##  7    10     3      17.6
##  8    15     1      26.9
##  9    15     2      25.6
## 10    15     3      23.1
```

```
print(seconddemos)
```

```
## # A tibble: 8 x 3
## # Groups:   Month [4]
##   Month  Naps mean_secondscore
##   <dbl> <dbl>            <dbl>
## 1     1     3             7.13
## 2     5     2            12.0
## 3     5     3             6.33
## 4    10     1            22.4
## 5    10     2            18.4
```

```
## 6    10     3            11.3
## 7    15     1            25.4
## 8    15     2            20.9
```

It may or may not because we're still drawing a random sample, but the trend should be clearer than during the first example.

###Now run the analysis again

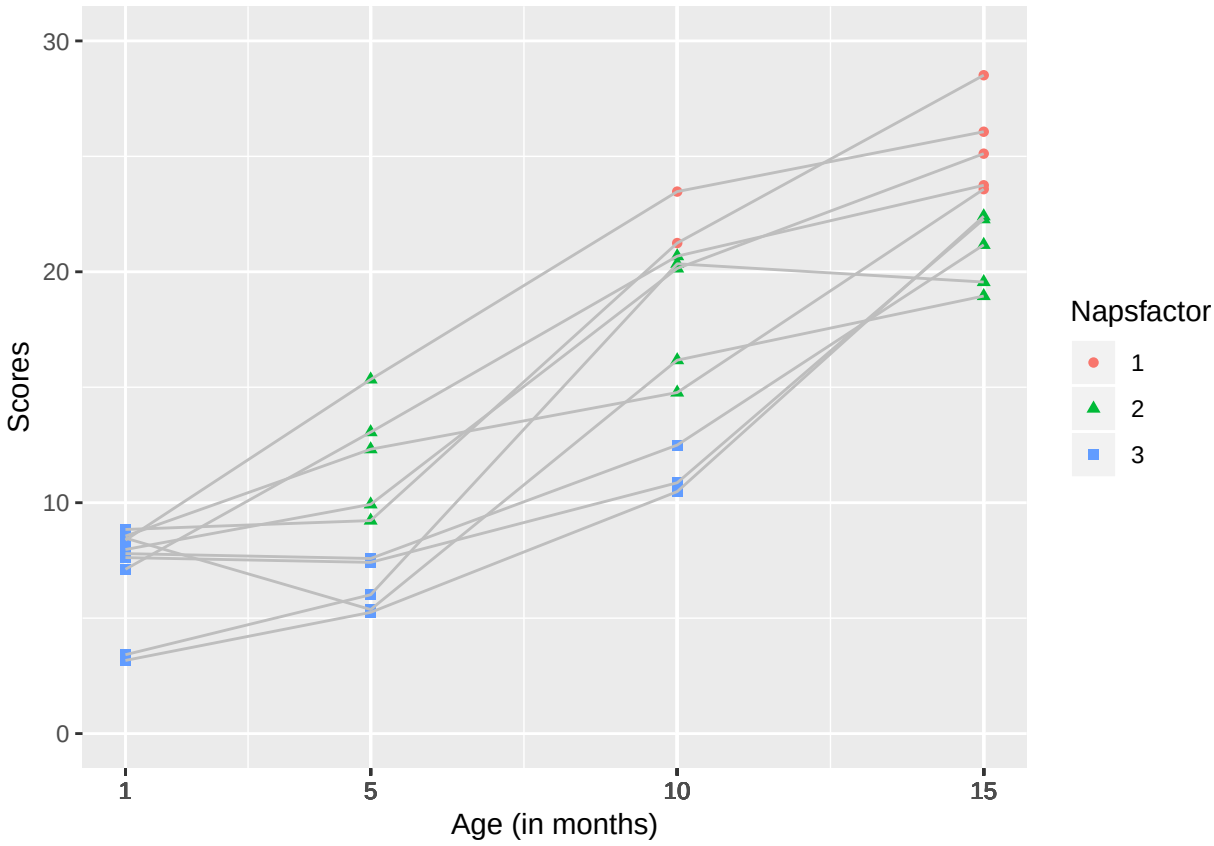
```
#Run the analysis again

secondtutorial <- lme(secondscores ~ Month * Naps, random = ~ Month | Subs, data=seconddataset)

secondtutorial
```

```
## Linear mixed-effects model fit by REML
##   Data: seconddataset
##   Log-restricted-likelihood: -90.36513
##   Fixed: secondscores ~ Month * Naps
## (Intercept)       Month        Naps  Month:Naps
##  16.8803891   1.0208719  -3.7260738  -0.1487096
##
## Random effects:
##  Formula: ~Month | Subs
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev       Corr
## (Intercept) 1.694427e-04 (Intr)
## Month       2.301061e-15 0
## Residual    2.291816e+00
##
## Number of Observations: 40
## Number of Groups: 10
```

```
#Create a table
secondtut<- summary(secondtutorial)
secondtabl = secondtut$tTable
secondtabl
```

```
##                  Value Std.Error DF   t-value      p-value
## (Intercept) 16.8803891 4.0453977 27  4.172739 0.0002798284
## Month        1.0208719 0.3172138 27  3.218246 0.0033430373
## Naps        -3.7260738 1.4256854 27 -2.613531 0.0144718579
## Month:Naps  -0.1487096 0.1227206 27 -1.211774 0.2360974216
```

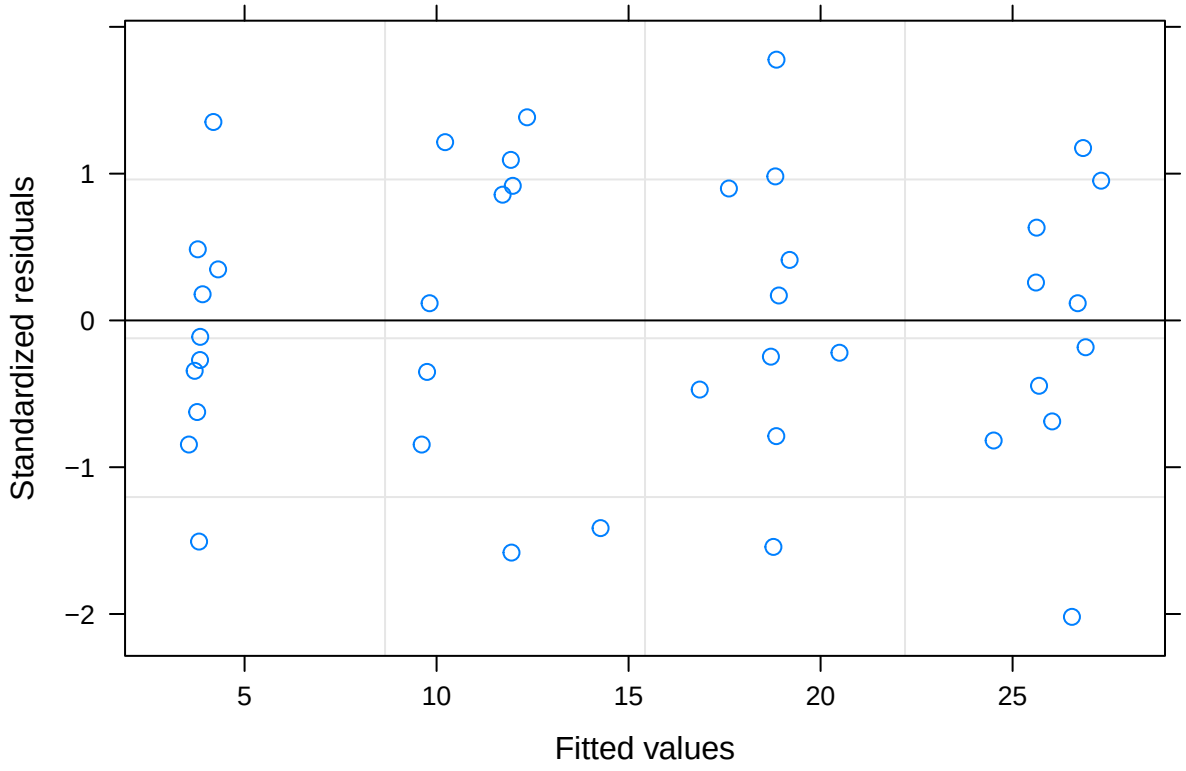
```
#Graph the results
secondplot<-
ggplot2::ggplot(seconddataset, aes(x=Month, y=secondscores,  color=Napsfactor, shape = Napsfactor, grou
  geom_point()+
  geom_line( color="grey")

secondplot + scale_x_continuous(name="Age (in months)", limits=c(1, 15), breaks = Month) +
  scale_y_continuous(name="Scores", limits=c(0, 30))
```



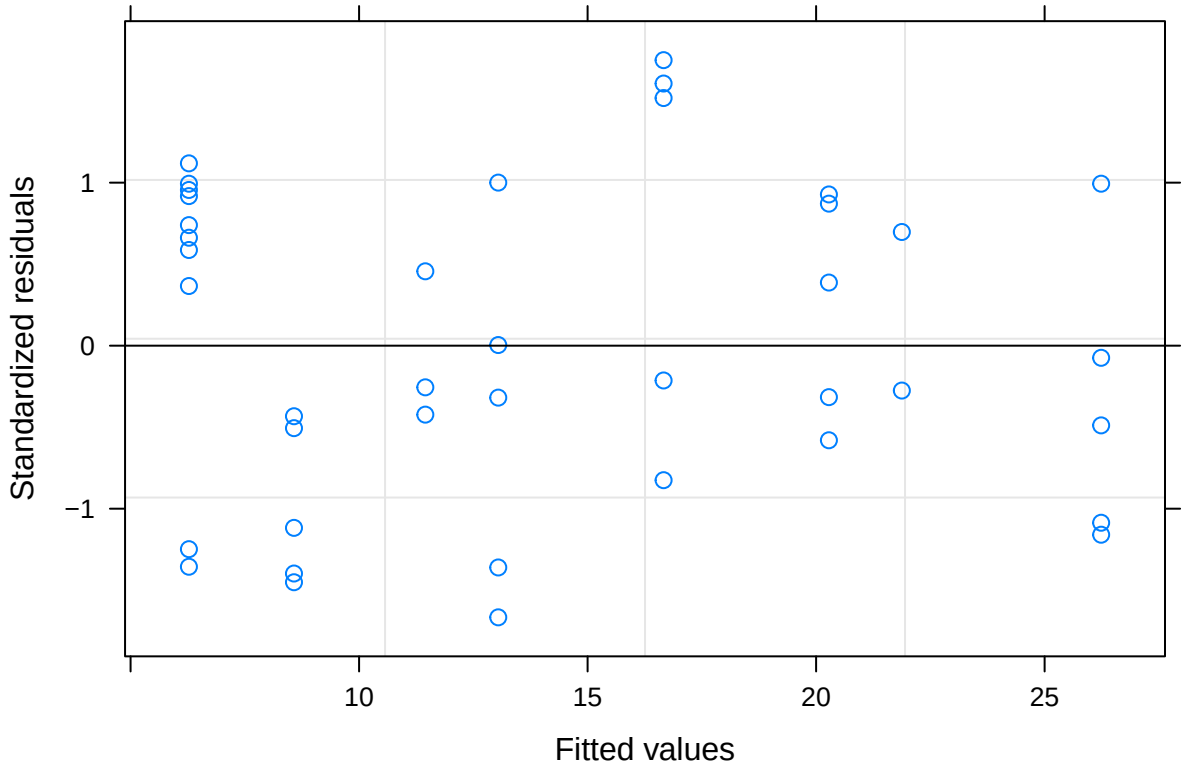
3.5 Plot the residuals

We can check the residuals to judge the fit of our models. The second tutorial should fit better because we set the data up that way.

```
plot(tutorial)
```




```
plot(secondtutorial)
```



Remember, for a well fitting regression, we want the plot of our residuals to meet the following criteria: (1) they're pretty symmetrically distributed (2) they're relatively small and (3) they don't follow a clear pattern

The second plot seems to meet these qualifications.

3.6 Writing up our results

```
summary(secondtutorial)
```

```
## Linear mixed-effects model fit by REML
##  Data: seconddataset
##        AIC      BIC    logLik
##   196.7303 209.3984 -90.36513
##
## Random effects:
##  Formula: ~Month | Subs
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev       Corr
## (Intercept) 1.694427e-04 (Intr)
## Month       2.301061e-15 0
## Residual    2.291816e+00
##
## Fixed effects: secondscores ~ Month * Naps
##                 Value Std.Error DF   t-value p-value
## (Intercept) 16.880389  4.045398 27  4.172739  0.0003
## Month        1.020872  0.317214 27  3.218246  0.0033
## Naps        -3.726074  1.425685 27 -2.613531  0.0145
## Month:Naps  -0.148710  0.122721 27 -1.211774  0.2361
##  Correlation:
##            (Intr) Month  Naps
## Month      -0.935
## Naps       -0.985  0.936
## Month:Naps  0.818 -0.947 -0.861
##
## Standardized Within-Group Residuals:
##        Min         Q1        Med         Q3        Max
## -1.6665261 -0.6414693 -0.1440354  0.8827551  1.7514202
##
## Number of Observations: 40
## Number of Groups: 10
```

A linear mixed effects model and conditional growth curve analysis was used to analyze infants' scores at 1, 5, 10, and 15 months old. Their scores were modeled with fixed effects of Month and Naps (1, 2, or 3) and random error to account for the within subjects design.There was a significant effect of Month- scores increased with age (Estimate= , SE= , p=).

```
#Estimate =
secondtabl[2,1]
```

```
## [1] 1.020872
```

```
#SE =
secondtabl[2,2]
```

```
## [1] 0.3172138
```

```
#p =
secondtabl[2,5]
```

```
## [1] 0.003343037
```

There was also a significant effect of Naps with fewer naps associated with better scores over time (Estimate= , SE= , p=).

```
#Estimate =
secondtabl[3,1]
```

```
## [1] -3.726074
```

```
#SE =
secondtabl[3,2]
```

```
## [1] 1.425685
```

```
#p =
secondtabl[3,5]
```

```
## [1] 0.01447186
```

The interaction was not significant.

Chapter 4

Data Shaping using Reshape(2) and Stats Packages

Author: Aira Contreras

4.1 Introduction

Research data sets can be overwrought with information that may not be of interest to the researcher; on one hand it could be that the researcher is using data sets from experiments or studies that were not conducted directly by them and therefore needs to carefully choose variables, or perhaps there were too many variables collected during the study and a particular analysis only requires a select few. It is also possible that the analysis software is only able to process the data if it is fed into the input in a certain manner. Whatever the limitations are, data shaping is a key step in data analysis and there are many tools available to assist with this task. R programmers have acknowledged the need for CRAN approved packages that aid with this endeavor and since 2003 have created packages such as Reshape to help others. This section will focus on the functionalities of the Reshape2 package distributed in 2012 by Hadley Wickham and the Stats 2001 by the R Core Team.

4.2 Different Data Inputs

Data input formats, for the purposes of this exercise, exist mainly in 2 forms: *wide format* or *long format*. It is important to understand what format the data is in prior to re-shaping or performing an analysis. This will allow you to correctly reshape the data prior to analysis or input into the software correctly. The next subsections describe the differences between the 2 formats and provide a visual example of the formats.

4.2.1 Long Format

Data is described to be in **long format** when each row represents 1 time point or variable. For example, if an experiment collected information on state population, state mortality, and state income then the output would be the state (e.g. California) and the variable (e.g. Income) in the first row, the state (e.g. California) and and the variable (e.g. Mortality) in the next row, and so forth until a full data output is created.

```
#install.packages("reshape2")
#install.packages("dplyr")
library(reshape2)
```

```
library(dplyr)

SubjectID<-1:50
df<- data.frame(SubjectID,state.x77)
example<-melt(df,id="SubjectID")
ex<-example%>%filter(SubjectID==1)
ex
```

```
##   SubjectID   variable    value
## 1         1 Population  3615.00
## 2         1     Income  3624.00
## 3         1 Illiteracy     2.10
## 4         1   Life.Exp    69.05
## 5         1     Murder    15.10
## 6         1    HS.Grad    41.30
## 7         1      Frost    20.00
## 8         1       Area 50708.00
```

This example demonstrates that all of the rows, which measure a distinct variable with its corresponding value, belong to 1 subject (i.e. 1). This format is useful for analysis of repeated measures experiments in which the same subject may be asked about different benchmarks during the course of the experiment.

4.2.2 Wide Format

Data is described to be in **wide format** when each row represents all of the individual variable responses for 1 subjected, separated by column. For example, if an experiment collected information on subjected responses at different intervals then the output would be the subject (e.g. 1) and the responses at each interval in separate columns (e.g. inter1, inter2, inter3, etc.) in the first row, the next subject (e.g. 2) and the responses at each interval in separate columns (e.g. inter1, inter2, etc.) in the next row and so fhorth until a full data output is created.

```
wide <- reshape(Indometh, v.names = "conc", idvar = "Subject",
                timevar = "time", direction = "wide")
wide[,1:8]
```

```
##    Subject conc.0.25 conc.0.5 conc.0.75 conc.1 conc.1.25 conc.2 conc.3
## 1        1      1.50     0.94      0.78   0.48      0.37   0.19   0.12
## 12       2      2.03     1.63      0.71   0.70      0.64   0.36   0.32
## 23       3      2.72     1.49      1.16   0.80      0.80   0.39   0.22
## 34       4      1.85     1.39      1.02   0.89      0.59   0.40   0.16
## 45       5      2.05     1.04      0.81   0.39      0.30   0.23   0.13
## 56       6      2.31     1.44      1.03   0.84      0.64   0.42   0.24
```

This example demonstrates that all of the rows show a distinct subject with the distinct variable measures in each individual column.

4.3 Reshape2 Package Functions

The Reshape2 package contains various functions that helps users place their data into either the *long format* or the *wide format* depending on what is needed by the software being used for analysis. Though there are many functions availalble and it is encouraged that all functions are reviewed (you can do so here: https://cran.r-project.org/web/packages/reshape2/reshape2.pdf), the primary focus of this section will be on the use of the **melt()** and **cast()** functions. For proper use and data reshaping, the functions are ideally

used in tandem, though as shown above, they can be used separately (i.e. only the melt function used for the separation of some data from others).

4.3.1 melt()

As it implies, the melt function strips away variables and/or structures from existing data sets and prepares the data for the cast function (or for analysis). Depending on the type of data that you may have (array, data frame, matrix), **melt()** should be able to accommodate the breakdown or reshaping of the data. A check for the use of the function called *melt_check()* is purported to exist, but the documentation on how to use this function is bare and I currently cannot make it run.

4.3.1.1 melt()/melt.default()/melt.data.frame()

The melt function requires the following inputs at minimum to work *melt(data,Subject Name Code Column, Variable Column Names in List Form)*. As is, the function does not remove any missing values (na.rm = FALSE), so if it is necessary to remove missing values from the data set, this should be done prior to the use of melt() or na.rm should be set as equal to **TRUE**.

```
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```
names(airquality) <- tolower(names(airquality))
aqm <- melt(airquality, id=c("month", "day"), na.rm=TRUE)

head(aqm)
```

```
##   month day variable value
## 1     5   1    ozone    41
## 2     5   2    ozone    36
## 3     5   3    ozone    12
## 4     5   4    ozone    18
## 6     5   6    ozone    28
## 7     5   7    ozone    23
```

The air quality data set contained 4 measured variables (Ozone, Solar.R, Wind, and Temp) over 2 time metrics (Month, Day). Using melt(), the data set was reshaped in aqm to show Month and Day as the independent variables, with the dependent variables Ozone, Solar R., Wind, and Temp shown as the dependent variables with their corresponding measurments. Note that because we did not specify the measured variable, all of the variables not designated as independent variables were designated as measured variables. In order to select specific measured/dependent variables, it is necessary to specify them as shown in the example below.

```
aqm2 <- melt(airquality, id=c("month", "day"),measure.vars = "ozone", na.rm=TRUE)

head(aqm2)
```

```
##   month day variable value
## 1     5   1    ozone    41
```

```
## 2     5   2    ozone    36
## 3     5   3    ozone    12
## 4     5   4    ozone    18
## 6     5   6    ozone    28
## 7     5   7    ozone    23
```

Here, again, the columns are the same as the aqm data set, however, because we specify *ozone* as the measured variable, only ozone will appear in the data set aqm2.

4.3.1.2 melt.array()

If the data you are working with is presented in an *array* as opposed to a data frame or matrix (meaning that there are a fixed number of values of a sinlge type) then you can use the melt.array() feature.

```
a <- array(c(1:23, NA), c(2,3,4))

a[,,1:2]
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

The above is an example of a random array that was created with 3 dimensions (row, column, depth).

```
melt(a)[1:10,]
```

```
##    Var1 Var2 Var3 value
## 1     1    1    1     1
## 2     2    1    1     2
## 3     1    2    1     3
## 4     2    2    1     4
## 5     1    3    1     5
## 6     2    3    1     6
## 7     1    1    2     7
## 8     2    1    2     8
## 9     1    2    2     9
## 10    2    2    2    10
```

Using the function melt, without any additional indicators will break the data down and out of the current structures that hold them. The first 3 columns represent the location of the data and the fourth column indicates the value in that given location. Based on the data output provided, it can be very easy to get lost or make mistakes. Additional data commands that name the columns may be helpful to keep track of the data as shown below.

```
melt(a, varnames = c("X","Y","Z"))[1:10,]
```

```
##    X Y Z value
## 1  1 1 1     1
## 2  2 1 1     2
```

```
## 3  1 2 1     3
## 4  2 2 1     4
## 5  1 3 1     5
## 6  2 3 1     6
## 7  1 1 2     7
## 8  2 1 2     8
## 9  1 2 2     9
## 10 2 2 2    10
```

4.3.2 cast()

The **cast()** function is what allows us to shape the data after we have melted it down into the ideal formats that are needed. The options exist to run *dcast()* or *acast()*. As you can imagine, the **dcast()** function best serves those that are intending to have a data frame or matrix output.

4.3.2.1 acast()

According to the README, the cast function provides outputs in the following format: x_var+x_2$_{y_var+y_2}$z_var. Though it may be an easy format to understand, it was not sufficient of an explanation for some. As such, please refer to the example below.

```
acast(aqm, day ~ month ~ variable)[1:10,,1:2]
```

```
## , , ozone
##
##     5  6   7   8  9
## 1  41 NA 135  39 96
## 2  36 NA  49   9 78
## 3  12 NA  32  16 73
## 4  18 NA  NA  78 91
## 5  NA NA  64  35 47
## 6  28 NA  40  66 32
## 7  23 29  77 122 20
## 8  19 NA  97  89 23
## 9   8 71  97 110 21
## 10 NA 39  85  NA 24
##
## , , solar.r
##
##      5   6   7   8   9
## 1  190 286 269  83 167
## 2  118 287 248  24 197
## 3  149 242 236  77 183
## 4  313 186 101  NA 189
## 5   NA 220 175  NA  95
## 6   NA 264 314  NA  92
## 7  299 127 276 255 252
## 8   99 273 267 229 220
## 9   19 291 272 207 230
## 10 194 323 175 222 259
```

As demonstrated from the data output above, the cast() function shaped the data such that the months are the column headers, divided across days in the rows, and then by variable as the overall "topic".

It is also possible to cast the data in such a way that the averages are obtained.

```
acast(aqm, month ~ variable, mean, margins = TRUE)
```

```
##          ozone  solar.r      wind     temp    (all)
## 5     23.61538 181.2963 11.622581 65.54839 68.70696
## 6     29.44444 190.1667 10.266667 79.10000 87.38384
## 7     59.11538 216.4839  8.941935 83.90323 93.49748
## 8     59.96154 171.8571  8.793548 83.96774 79.71207
## 9     31.44828 167.4333 10.180000 76.90000 71.82689
## (all) 42.12931 185.9315  9.957516 77.88235 80.05722
```

In the above example, the data once again separated by what was indicated (e.g. month and variable). We have added however, that the mean of these variables for each month be calculated and additionally, by adding the *margins = TRUE* command, we have asked for the means of each row to be included.

Overall, the takeaway for how to use cast should be as follows: acast(melted data set, rowvariablename$_{\text{columnvariablename}}$OPTIO

4.3.2.2 dcast()

Similar to *acast()*, **dcast()** reforms the data, however, it places it into a data fram or matrix format.

```
dcast(aqm, month ~ variable, mean, margins = c("month", "variable"))
```

```
##   month    ozone  solar.r      wind     temp    (all)
## 1     5 23.61538 181.2963 11.622581 65.54839 68.70696
## 2     6 29.44444 190.1667 10.266667 79.10000 87.38384
## 3     7 59.11538 216.4839  8.941935 83.90323 93.49748
## 4     8 59.96154 171.8571  8.793548 83.96774 79.71207
## 5     9 31.44828 167.4333 10.180000 76.90000 71.82689
## 6 (all) 42.12931 185.9315  9.957516 77.88235 80.05722
```

To use a different data set, the example below takes data from the ChickWeight set and melts it down and casts it as a data frame that displays only the time (row) and variable (column) and the corresponding means.

```
head(ChickWeight)
```

```
## Grouped Data: weight ~ Time | Chick
##   weight Time Chick Diet
## 1     42    0     1    1
## 2     51    2     1    1
## 3     59    4     1    1
## 4     64    6     1    1
## 5     76    8     1    1
## 6     93   10     1    1
```

```
names(ChickWeight) <- tolower(names(ChickWeight))
chick_m <- melt(ChickWeight, id=2:4, na.rm=TRUE)
dcast(chick_m, time ~ variable, mean)
```

```
##    time    weight
## 1     0  41.06000
## 2     2  49.22000
## 3     4  59.95918
## 4     6  74.30612
## 5     8  91.24490
## 6    10 107.83673
```

```
## 7    12 129.24490
## 8    14 143.81250
## 9    16 168.08511
## 10   18 190.19149
## 11   20 209.71739
## 12   21 218.68889
```

4.4 Stats Package Functions

The Stats package contains various functions that helps users place their data into either the *long format* or the *wide format* depending on what is needed by the software being used for analysis. Though there are many functions availalble and it is encouraged that all functions are reviewed (you can do so here: https://www.rdocumentation.org/packages/stats/versions/3.6.0), the primary focus of this section will be on the use of the **reshape()** function.

4.4.1 Reshape()

More readily than the melt() and cast() functions, reshape is able to guid the direction of the data as it reshapes and rebuilds the data set as indicated.

The reshape() functions require the basic inputs the melt() function did, with the added input of direction, which indicates whether the dataset should be a long format data set or a wide format dataset.

```
wide <- reshape(Indometh, v.names = "conc", idvar = "Subject",
                timevar = "time", direction = "wide")
wide[,1:5]
```

```
##    Subject conc.0.25 conc.0.5 conc.0.75 conc.1
## 1        1      1.50     0.94      0.78   0.48
## 12       2      2.03     1.63      0.71   0.70
## 23       3      2.72     1.49      1.16   0.80
## 34       4      1.85     1.39      1.02   0.89
## 45       5      2.05     1.04      0.81   0.39
## 56       6      2.31     1.44      1.03   0.84
```

```
state.x77 <- as.data.frame(state.x77)
long <- reshape(state.x77, idvar = "state", ids = row.names(state.x77),
                times = names(state.x77), timevar = "Characteristic",
                varying = list(names(state.x77)), direction = "long")
longa<-long%>%filter(state=="Alabama")

longa
```

```
##   Characteristic Population   state
## 1     Population    3615.00 Alabama
## 2         Income    3624.00 Alabama
## 3     Illiteracy       2.10 Alabama
## 4       Life Exp      69.05 Alabama
## 5         Murder      15.10 Alabama
## 6        HS Grad      41.30 Alabama
## 7          Frost      20.00 Alabama
## 8           Area   50708.00 Alabama
```

4.5 Summary

Data reshaping is important and there are many tools availalbe to assist users in re-formatting the data sets they have available. In addition to the packages discussed above, other packages have been developed by other R users to facilitate this process; the next generation of ReShape can be found in the tidyr package, and other packages like psych, include tools that help reformat data. Additionally with the use of additional packages like dplyr and forcats, additional power is added to the ability of the user to format and reformat as needed.

4.6 References

Kabacoff, R. I. (n.d.). Qhick-R: Reshape. Retrieved May 03, 2019, from https://www.statmethods.net/management/reshape.html

Wickham, H. (2012). reshape2: Flexibly reshape data: a reboot of the reshape package. R package version, 1(2).

Martin, K. G. (n.d.). The Wide and Long Data Format for Repeated Measures Data. Retrieved May 03, 2019, from https://www.theanalysisfactor.com/wide-and-long-data/

R Core Team (2012). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/

Chapter 5

How to Annotate a Graph Using GG Signif"

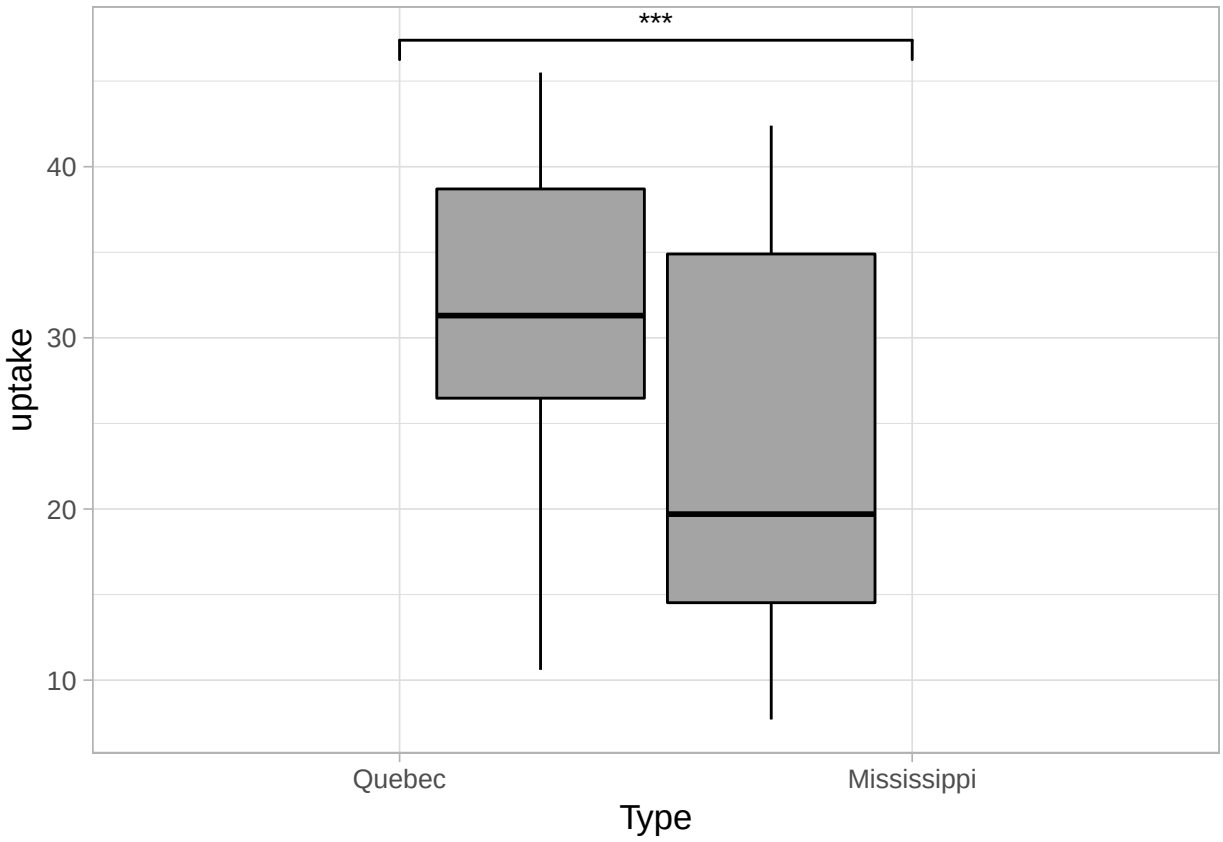
The `ggsignif` package is an extension of GGplot, the popular plotting package used for data visualization. GGplot is used by layering graphing, data and visualization components. GGsignif is used as an additional layer to the GGplot packaage that allows for calculation and annotation of statistical significance within graphs.

#How do you use GGsignif? 1st: load both packages

```
library(ggplot2)
library(ggsignif)
```

2nd: Plot your data using GGplot2 and add layer 'geom_signif'.

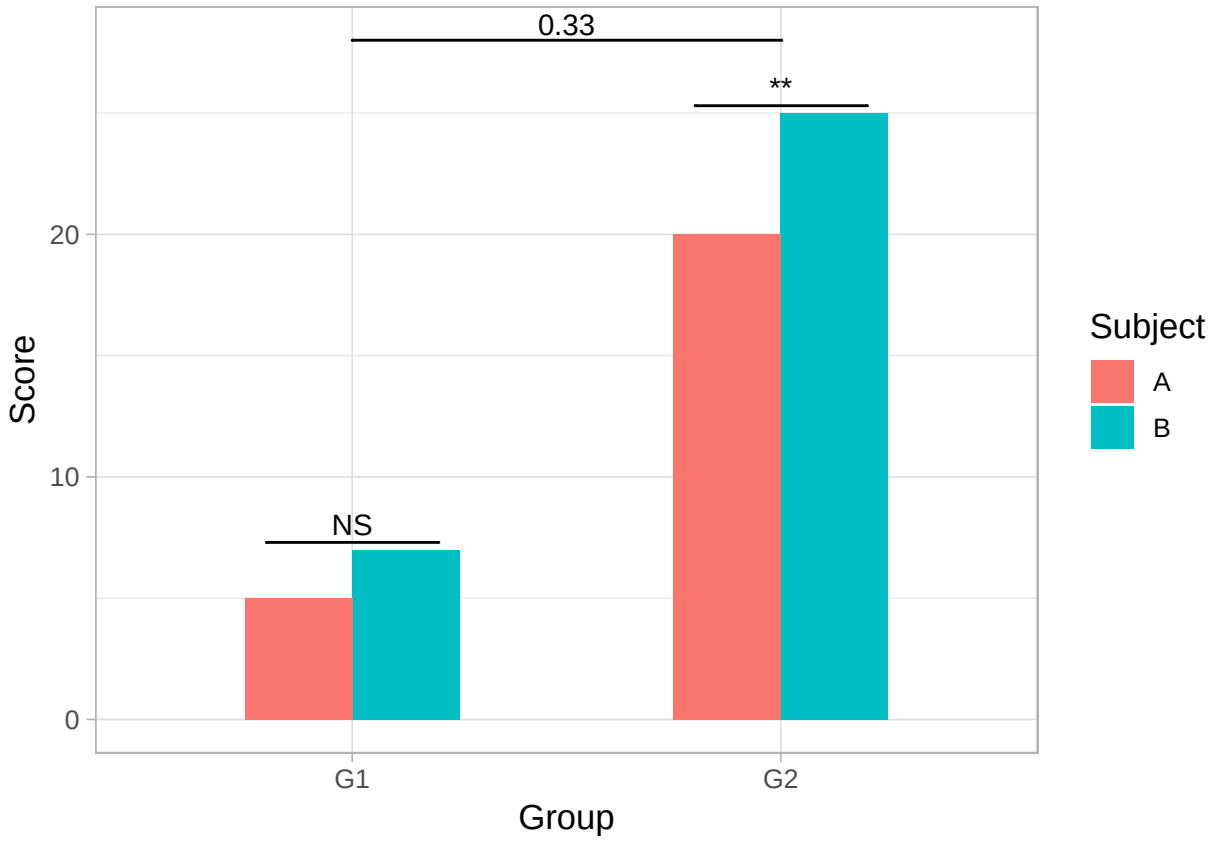
```
all.data<- CO2
library(dplyr)

All.data <- all.data %>%
  select(Type,Treatment,conc,uptake)

ggplot(All.data, aes(x=Type,y=uptake,group=Treatment))+
  geom_boxplot(fill='#A4A4A4', color = "black")+
  theme_light(base_size=13)+
  geom_signif(comparisons = list(c("Quebec","Mississippi")),map_signif_level = TRUE)
```



- The first input delineates which values need to be calculated for annotation i.e. comparison = list(c(here is where enter those values)). 'Map_signif_level' is used to control whether the output graph contains the actual signficance or stars (***).

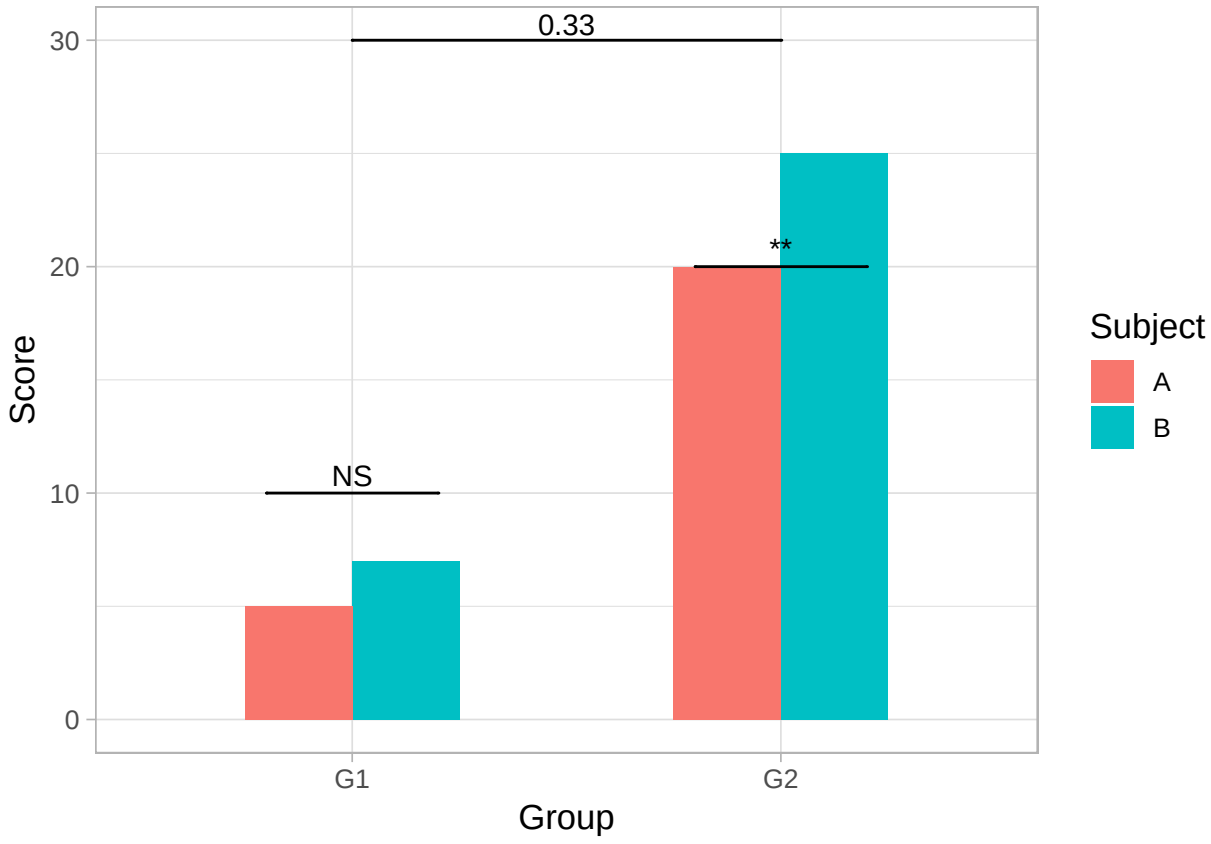
#Advanced Options You can also control exactly where the annotations will show up in the output graph.

```
dataf<- data.frame(Group<- c("G1","G1","G2","G2"),
                   Subject<- c("A","B","A","B"),
                   Score<- c(5,7,20,25))

ggplot(dataf, aes(Group,Score))+
  geom_bar(aes(fill = Subject), stat = "identity",
           position = "dodge", width = .5)+
  geom_signif(y_position = c(7.3,25.3), xmin = c(0.8,1.8),
              xmax = c(1.2,2.2), annotation = c("NS","**"),
              tip_length = 0)+
  geom_signif(comparisons = list(c("G1","G2")), y_position = 28,
              tip_length = 0, vjust = .1)+
  theme_light(base_size=13)
```



##'y_position' - Controls the placement of each individual groups' signficance annotation : y_position = c(height of first annotation, second annotation height on y-axis).

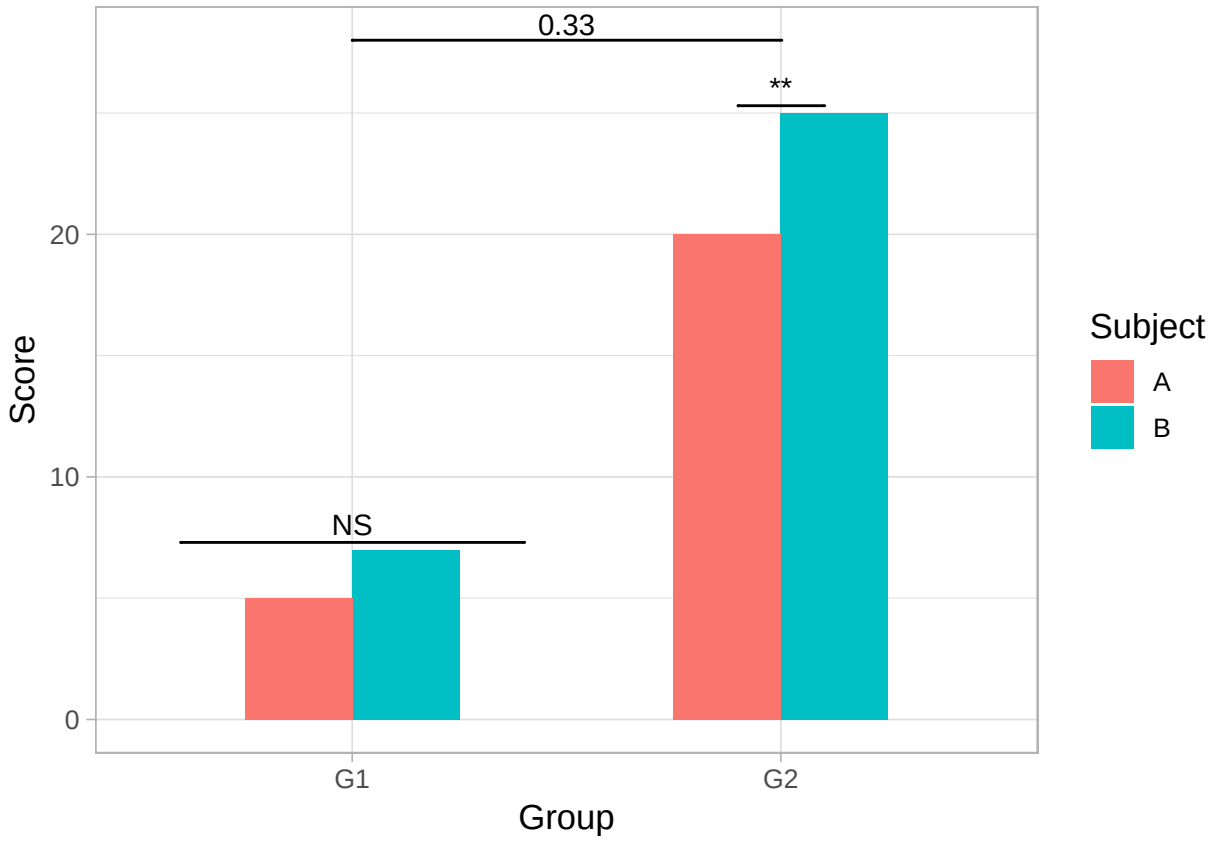
```
ggplot(dataf, aes(Group,Score))+
  geom_bar(aes(fill = Subject), stat = "identity",
           position = "dodge", width = .5)+
  geom_signif(y_position = c(10,20), xmin = c(0.8,1.8),
              xmax = c(1.2,2.2), annotation = c("NS","**"),
              tip_length = 0)+
  geom_signif(comparisons = list(c("G1","G2")), y_position = 30,
              tip_length = 0, vjust = .1)+
  theme_light(base_size=13)
```



5.1 'xmin' and 'xmax'

- The two work together to set the length of the annotation bar where the first number within xmin's paranthesis is the starting point and the first number in xmax is the end. In the first graph xmin = c(.8,1.8) and xmax = c(1.2,2.2), this means the first bar starts on the x-axis at .8 until 1.2 and the second bar's length is from 1.8 to 2.2.
- It can also be made shorter or longer:

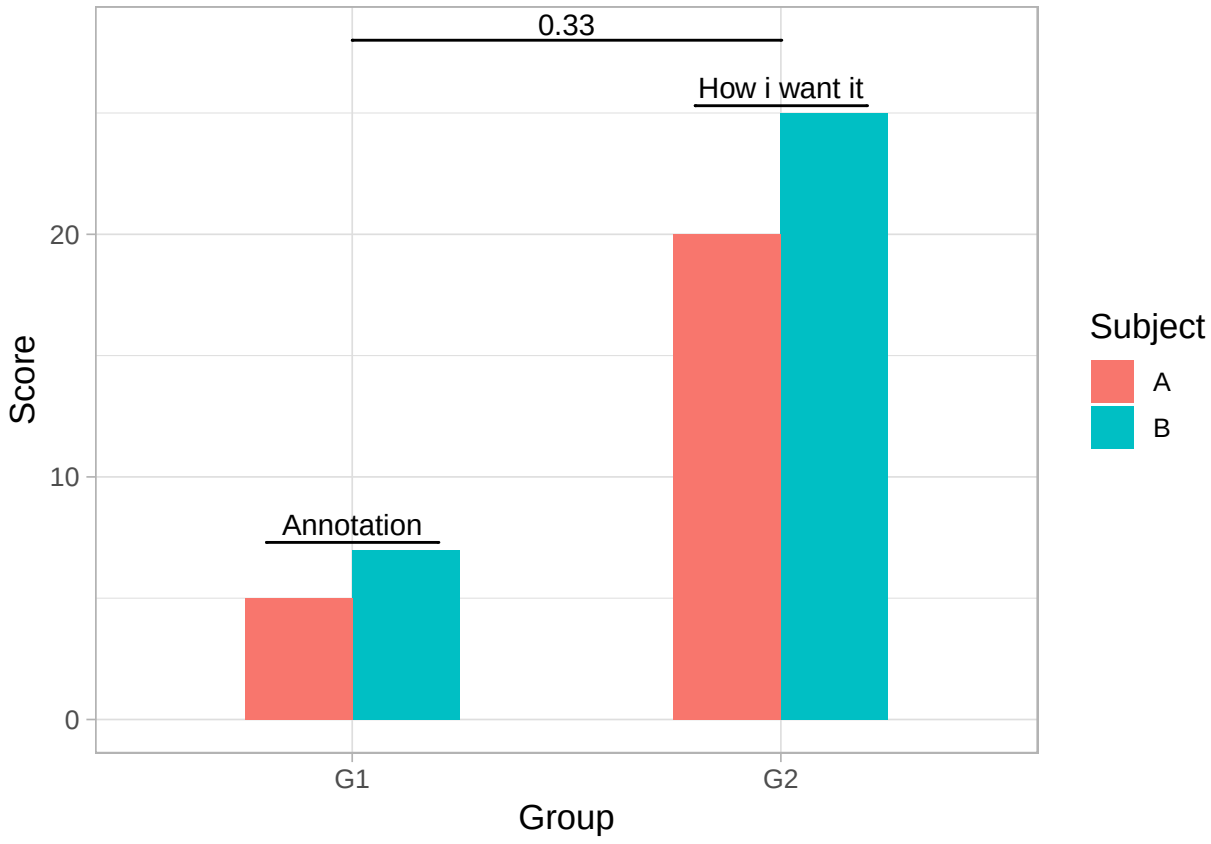
```
ggplot(dataf, aes(Group,Score))+
  geom_bar(aes(fill = Subject), stat = "identity",
           position = "dodge", width = .5)+
  geom_signif(y_position = c(7.3,25.3), xmin = c(0.6,1.9),
              xmax = c(1.4,2.1), annotation = c("NS","**"),
              tip_length = 0)+
  geom_signif(comparisons = list(c("G1","G2")), y_position = 28,
              tip_length = 0, vjust = .1)+
    theme_light(base_size=13)
```



5.2 'annotation'

- Here is where you type in the desired text or symbol for each bar.

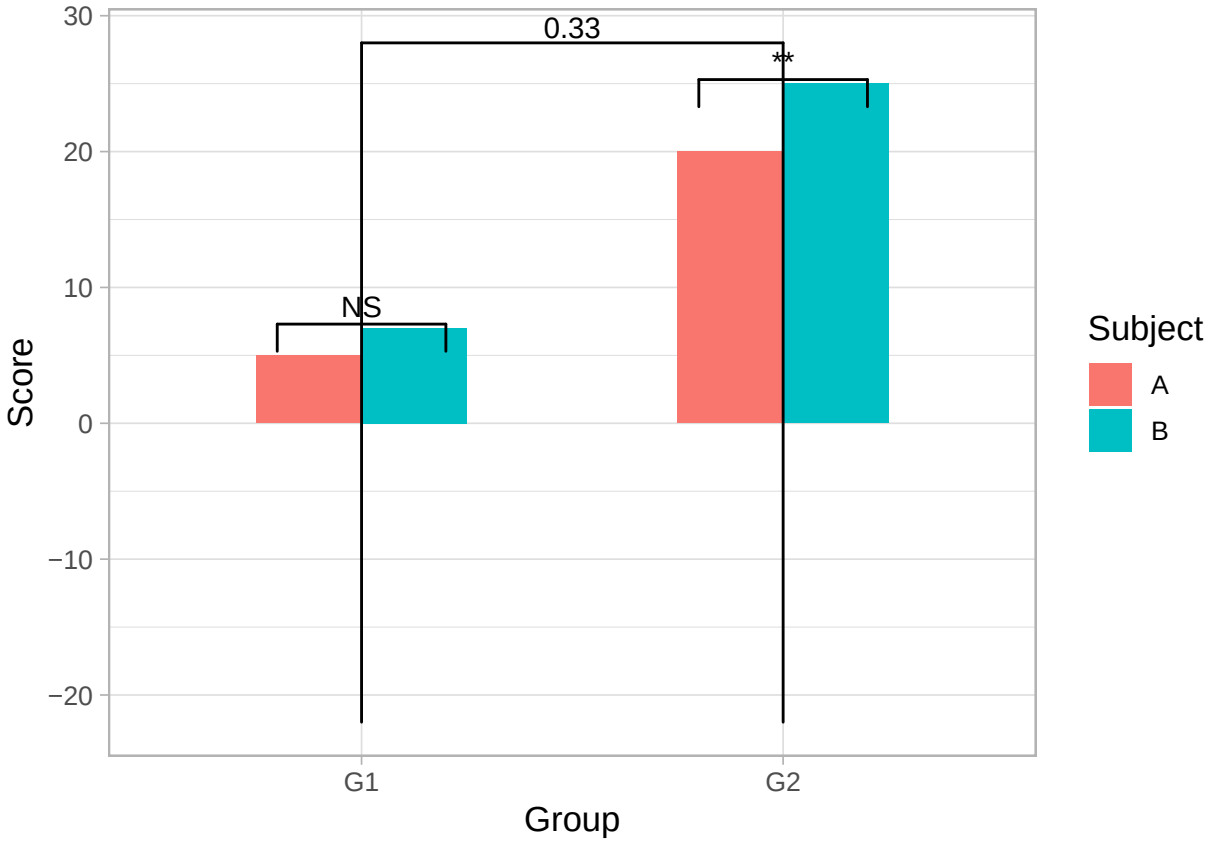
```
ggplot(dataf, aes(Group,Score))+
  geom_bar(aes(fill = Subject), stat = "identity",
           position = "dodge", width = .5)+
  geom_signif(y_position = c(7.3,25.3), xmin = c(0.8,1.8),
              xmax = c(1.2,2.2), annotation = c("Annotation","How i want it"),
              tip_length = 0)+
  geom_signif(comparisons = list(c("G1","G2")), y_position = 28,
              tip_length = 0, vjust = .1)+
  theme_light(base_size=13)
```



5.3 'tip_length'

- This gives the option to add vertical lines coming down on both ends of annotation bar of varying length.

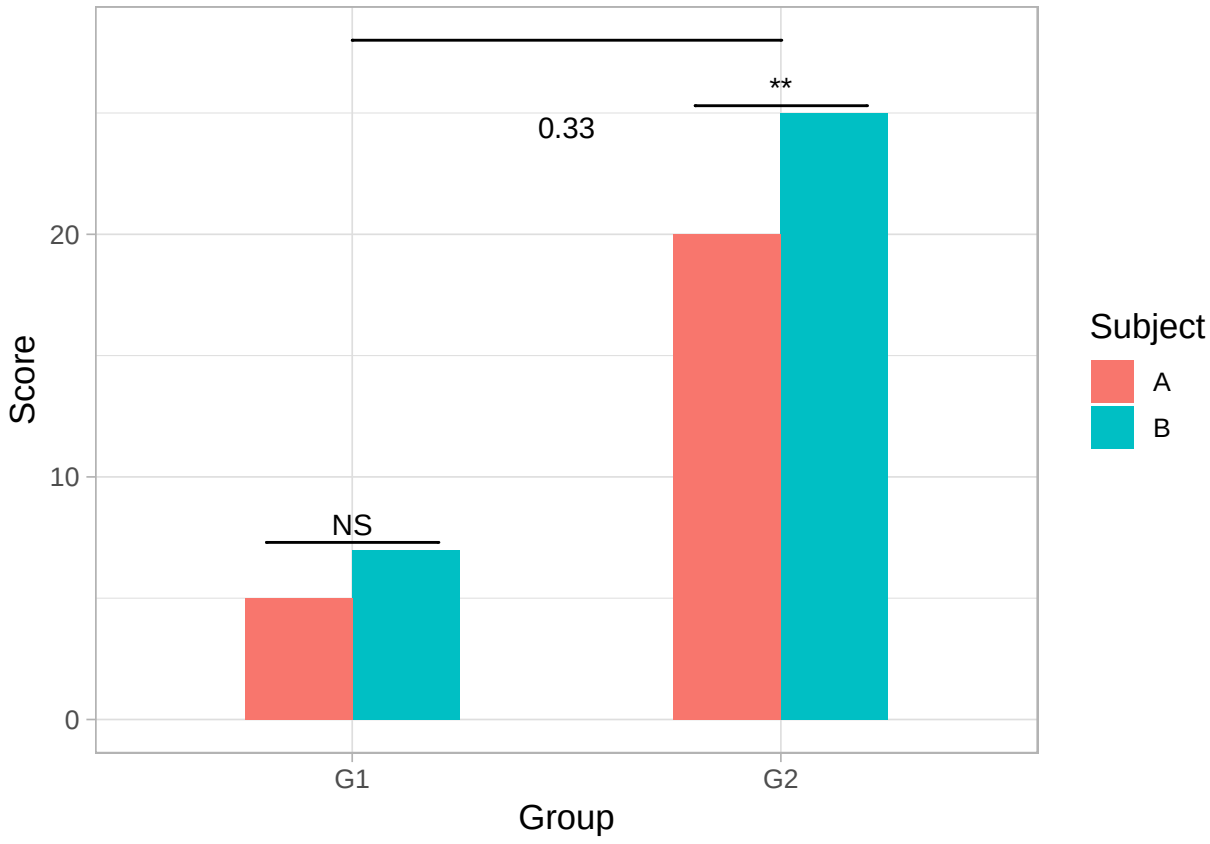
```
ggplot(dataf, aes(Group,Score))+
  geom_bar(aes(fill = Subject), stat = "identity",
           position = "dodge", width = .5)+
  geom_signif(y_position = c(7.3,25.3), xmin = c(0.8,1.8),
              xmax = c(1.2,2.2), annotation = c("NS","**"),
              tip_length = .1)+
  geom_signif(comparisons = list(c("G1","G2")), y_position = 28,
              tip_length = 2.5, vjust = .1)+
  theme_light(base_size=13)
```



5.4 'vjust'

- This option allows user to control placement of the "0.33" in the graph.

```
ggplot(dataf, aes(Group,Score))+
  geom_bar(aes(fill = Subject), stat = "identity",
           position = "dodge", width = .5)+
  geom_signif(y_position = c(7.3,25.3), xmin = c(0.8,1.8),
              xmax = c(1.2,2.2), annotation = c("NS","**"),
              tip_length = 0)+
  geom_signif(comparisons = list(c("G1","G2")), y_position = 28,
              tip_length = 0, vjust = 5)+
  theme_light(base_size=13)
```


0.33
**
NS
20
10
0
Score
G1
G2
Group
Subject
A
B