Introduction to R through Mediation and Moderation

Amanda Montoya

May 17, 2017

Morning Session 9:00am - 11:30am

Introductions

Instructor

Course

- Download files from github.com/akmontoya/UW2017
- Play around! Try things out! Figure out what works and what doesn't. Ask why...

What is R?

- Open source software environment
  - Freely available, users can see how functions are written
- Language is functional (vs. imerpative or object oriented)
  - I have heard people describe R as object oriented before
- Language is interpretted (conversation between human readable code and computer binary)
- Uses a command line interface
- Runs in batch mode (give list of commands, execute all commands, get results)

Try opening R and taking a look at it

What is Rstudio?

- A much nicer way to use R on a regular basis
- User interface which includes the console from R and many other things
- Makes saving script and running code easy

Try opening Rstudio and taking a look

RStudio Continued...

User Interface

- Console
- Source
- Environment/History
- Files / Plots / Packages / Help / Viewer

You can reorganize the windows to look however you like

Try opening a new R script (File –> New File –> Rscript)

R Basics

- Calculator
- Objects and Variables
- Extracting elements of objects
- Boolean Logic
- Descriptive Statistics
- Using Dataframes
- Bascis of functions
- Histograms and Plots

R is a Calculator

Start by typing in the console in R. Type command then press enter.

Basic operators in R:

- + Addition

```
2 + 4
```

```
## [1] 6
```

- - Subtraction

```
24.5 - 6.7243
```

```
## [1] 17.7757
```

- * Multiplication

```
16 * 9.5
```

```
## [1] 152
```

- ^ Exponentiation

```
10 ^ 8
```

```
## [1] 1e+08
```

- / Division

```
7 / 3.5
```

```
## [1] 2
```

- %/% Modular Division

```
9%/%5
```

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

- %% Remainders

```
  9%%5
```

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

- Combinations of these

```
(1924 + 13) / 24
```

```
## [1] 80.70833
```

Does anyone remember PEMDAS?

Now try in the Source Window

- Type command, then press "Run" or Ctrl+R
- Source can be used to build a file of your commands
- Use # at the start of a line to create a comment
- Use spaces around operators to made code easier to read

```r
#I'm going to add the numbers 2 and 4
2 + 4
```

```
## [1] 6
```

```r
#Now I'll take 6 and divide by 2
6 / 2
```

```
## [1] 3
```

```r
#Parentheses go before exponentiation
(6 + 4) ^ 7 / 12
```

```
## [1] 833333.3
```

Objects and Variables in R

Objects

- Objects arethings we can manipulate in R.
- Data and functions can be objects in R.

Variables

- Storage location and associated name containing an object
- Everything gets saved into your environment

Basic idea: Save an object as a variable

```
six <- 4 + 2

six
```

```
## [1] 6
```

Data Types

(Important for being able to read documentation and error messages)

- Double: Numeric not whole number (e.g. 5.67, 1/2)
- Integer: Numeric whole number (e.g. 3, -2)
- Logical: TRUE and FALSE
- Character: aka "String" (e.g. Bob)

Object Types

- Atomic vector / scalar: Single object
- Vector: row or column of objects of the same type
- Matrix: a 2 dimensional collection of objects of the same type
- Array: An any dimensional collection of objects of the same type
- List: a row/column of objects of different types
- Dataframe: a 2 dimensional collection of objects with all the same type in a given column
- Many, many, more

Functions for understanding objects

```
years <- seq(1861,2017)
length(years)
```

```
## [1] 157
```

```
typeof(years)
```

```
## [1] "integer"
```

```
is.vector(years)
```

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

```r
years2 <- c(years, "UWRules")
length(years2)

typeof(years2)

years2

is.vector(years2)
```

```r
years3 <- list(years, "UWRules")
length(years3)

typeof(years3)

years3

is.vector(years3)
```

Variable names

- R is case sensitive
- Step 1: Make it interpretable to you
- Step 2: Make it interpretable to others
- Avoid single letter names
- Names CANNOT start with numbers
- Popular naming conventions for multiword names
  - camelCase: easy to use, can be unintuitive with statistics conventions (e.g. t-test, F-test)
  - Use a delimiter (e.g. _ or .): slower to type, easy to read, makes names longer

The ever present battle between = and <-.

Creating different types

```
c(1, 2, 3)

matrix(data = 1:100, nrow = 5, ncol = 10,
       byrow = FALSE #default)

matrix(data = 1:100, nrow = 5, ncol = 10,
       byrow = TRUE)

list(c(1, 2, 3), "Bob", years)
```

Practice

1. Create a script file, include your name, the date, and "Practice 1" in the comments
2. Create V1, V2, and V3 (on next slide) as separate vectors. What data type is each vector?
3. Can you make V1, V2, and V3 into a matrix where each variable is a column? each variable is a row? Can you figure out how to make this into a dataframe?
4. Create a list with your first name, last name, last 4 digits of your phone number, and your favorite number.
5. Use the `seq` function to create a vector of the years from your birth to 2015 (if your birthday has not happened yet) and 2016 (if your birthday has already occured in 2017). What is the length of this vector? It should be your age.

V1	V2	V3
19	4.0	“A”
20	3.6	“A-”
18	4.0	“A”
23	2.2	“C”
22	3.0	“B”
19	3.4	“B+”

Extracting elements of objects

For objects which have many elements you may want to select one of the them.

Use brackets, then numbers to indicate the element. Vectors require one number. Matrices require a row then column, separated by a comma

```
shortvec <- seq(1:10)
shortvec[6]
```

```
## [1] 6
```

```
egmat <- matrix(1:100, ncol = 5)
egmat[7,3]
```

```
## [1] 47
```

Extracting multiple elements of an object

You can use the `:` to denote "through" where `a:b` will denote the ath through the bth element.

```
shortvec[6:9]
```

```
## [1] 6 7 8 9
```

```
egmat[7:10,3:4]
```

```
##       [,1] [,2]
## [1,]   47   67
## [2,]   48   68
## [3,]   49   69
## [4,]   50   70
```

To select a whole row or whole column leave the space blank

```
shortvec[]
```

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

```
egmat[,3]
```

```
##  [1] 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
```

```
egmat[7,]
```

```
## [1]  7 27 47 67 87
```

To select everything but a certain index (or indexes) use a negative operator

```
shortvec[-6]

egmat[7,-3]

egmat[,-3]
```

Logical Operators

You can make logical statements in R which can be used creatively in subsetting data, learning about the data, and many other uses.

Logic statements are used to create logical objects (e.g. TRUE or FALSE).

Mathematical logical statements:

- Equal / Not Equal: == / !=
- Greater than / greater than or equal to: > / >=
- Less than / less than or equal to: < / <=

For coding purposes, I like to include logicals in parentheses.

```
(3 == 4)
```

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

```
(3 != 4)
```

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

```
(3 < 4)
```

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

```
(3 >= 4)
```

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

Applying logic to elements

You can use logcal statements to evaluate all the elements of a vector/matrix/etc.

```
(shortvec <= 3)
```

```
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FAI
```

```
shortvec[shortvec <= 3] #ponder this one
```

```
## [1] 1 2 3
```

```
(egmat > 24)
```

```
##       [,1]  [,2] [,3] [,4] [,5]
##  [1,] FALSE FALSE TRUE TRUE TRUE
##  [2,] FALSE FALSE TRUE TRUE TRUE
```

Boolean Logic

Boolean logic can be used to combine statements and negate them.

Consider a set of integers from 0 - 100

AND: *both* statements must be true for the statement to be true. E.g. X is greater than 4 AND less than 6 (this will be true for the number 5 only)

OR: *either* statement or *both* can be true for statement to be true. E.g. X is greater than 6 OR less than 4 (this is true for all number except 5)

NOT: The statement must be false for the negation to be true. E.g. X is NOT greater than 6 (this will be true for numbers 0 - 5)

Boolean Logic in R

```
#AND is represented with an ampersand &
shortvec
```

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

```
(shortvec <= 3) & (shortvec > 1)
```

```
##  [1] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FAI
```

```
#OR is represented with a vertical bar |
(shortvec <= 3) | (shortvec > 6)
```

```
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TR
```

```
#Note that OR is an INCLUSIVE OR
(shortvec <= 3) | (shortvec > 2)
```

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

Boolean Logic in R

```
#NOT is represented with an exclamation point !
!(shortvec  <= 3)
```

```
##  [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TR
```

```
#Combining these can get complex
(!(shortvec <= 3) | (shortvec  == 2))
```

```
##  [1] FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TR
```

Practice

All questions use egmat

1. Create a logical matrix from egmat which indicates which elements are greater than 12 AND less than or equal to 8.
2. Select the elements from egmat which are NOT greater than 54 OR greater than 98.
3. Try out this code, what is it doing? What would happen if we reordered shortvec? Try it out.

```
egmat[egmat != shortvec]
```

```
##  [1]  11  12  13  14  15  16  17  18  19  20  21  22  23
## [18]  28  29  30  31  32  33  34  35  36  37  38  39  40
## [35]  45  46  47  48  49  50  51  52  53  54  55  56  57
## [52]  62  63  64  65  66  67  68  69  70  71  72  73  74
## [69]  79  80  81  82  83  84  85  86  87  88  89  90  91
## [86]  96  97  98  99 100
```

Practice

```
# 1. egmat: elements > 8 AND <= 12
(egmat > 8) & (egmat <= 12)

#2. egmat:  elements are NOT > 54 OR > 98.
egmat[!(egmat > 54) | (egmat > 98)]

#3. What would happen if we reordered shortvec? Try it out
egmat[egmat != shortvec]
shortvecrev <- order(shortvec, decreasing = TRUE)
shortvecrev
egmat != shortvecrev

#elementwise comparison, cycling
shortvecplus <- c(shortvec, seq(91, 100))
egmat != shortvecplus

#4. Hint: Consider using modular division and/or
# remainders %/%, %%
```

Vaughn, 2017 Study 2

Using Dataframes

The first thing you'll need to do is read in a dataframe from some outside source. You can do your data maintenance in SPSS or Excel then same the dataset as txt, csv, etc.

```
#Locate file on your computer
vaughndatabig <- read.csv(file = "C:\\Users\\Akmontoya\\Des
head(vaughndatabig)
summary(vaughndatabig)

vaughndata <- read.csv(file = "C:\\Users\\Akmontoya\\Deskto
head(vaughndata)
foot(vaughndata)
names(vaughndata)
```

Head and foot give the first and last 6 rows of a dataset. Good to check that things read in reasonably.

Selecting parts of a dataframe

Just like a matrix you can use indices to select part of the matrix.

```
vaughndata[12:14,1:5]
```

```
##    study nscond as1 cs1 rs1
## 12     5      1   6   6   4
## 13     5      1   5   5   5
## 14     5      1   4   5   3
```

```
head(vaughndata[,2:3])
```

```
##   nscond as1
## 1      1   2
## 2      1   1
## 3      1   6
## 4      1   2
## 5      1   4
```

Summarizing a dataframe

The summary and structure functions are good for understanding the dataset as a whole.

```
str(vaughndata)

summary(vaughndata)
```

Attaching a dataset

When you are working with a dataset a lot you might consider attaching it. So you don't have to type it's name over and over. Beware this can be dangerous.

```
vaughndata$as1[1:6]
```

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

```
attach(vaughndata)
as1[1:6]
```

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

```
detach()
#as1[1:6]
```

A Cautionary Tale

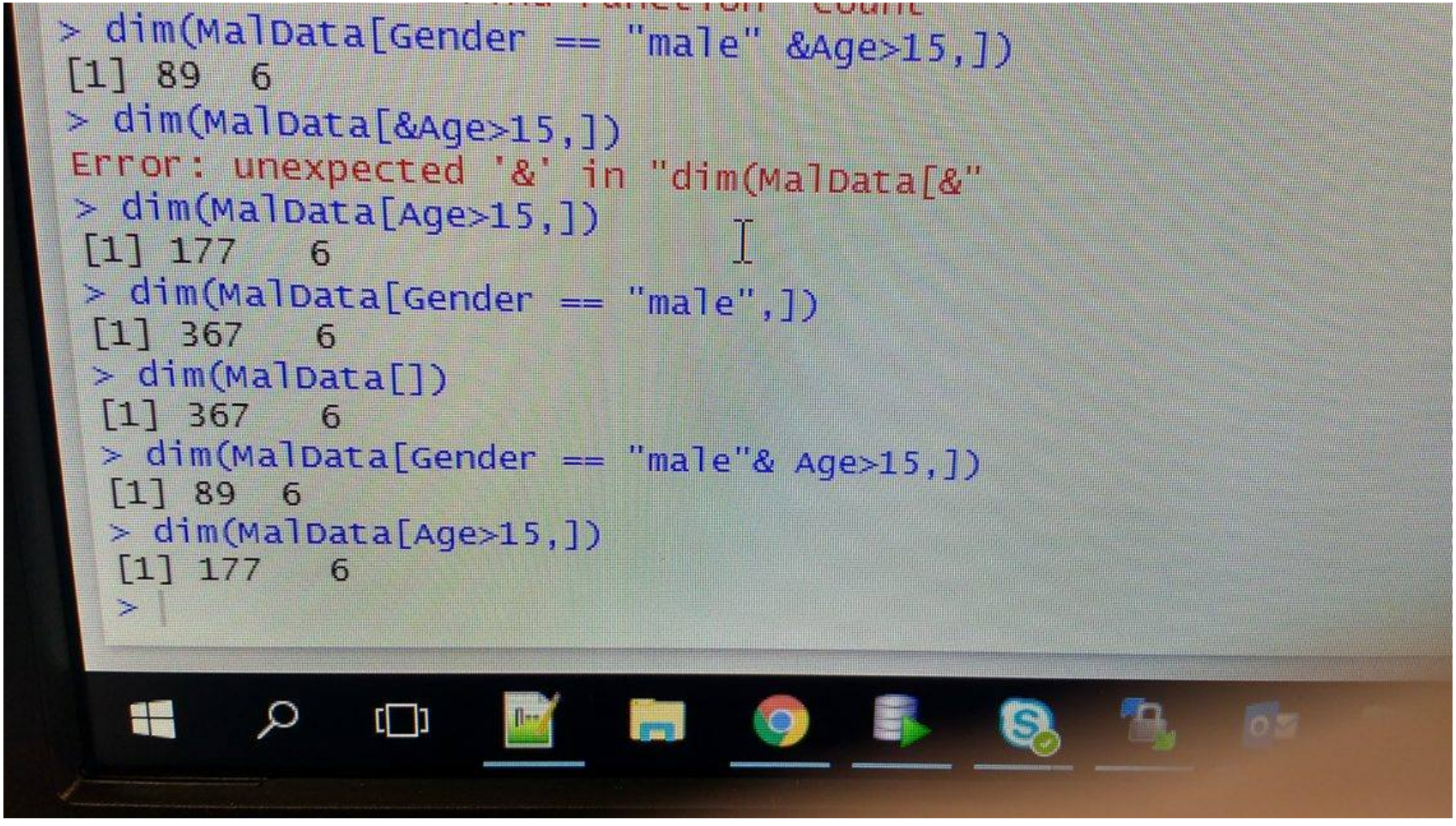


Figure 1: What's going on?

Missing Values

Missing values are represented by `NA` in R.

```
head(is.na(vaughndata$howdist))
```

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

```
sum(is.na(vaughndata$howdist))
```

```
## [1] 486
```

Sorting / Ordering

Be very careful about sorting/ordering functions in R. Many do not maintain the structure of a dataset. They'll just reorder a column, and leave all other columns as they were.

```
#order gives rank of each element
order(c(5,1,3,2))
```

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

```
vaughndata <- vaughndata[order(vaughndata$age),]
```

Subsetting

The subset function allows you to create new dataframe which are a subset of an existing dataframe.

```
lownsdata <- subset(vaughndata, nscond == 1)
highnsdata <- subset(vaughndata, nscond == 2)

lownsfemale <- subset(vaughndata,
                      (nscond == 1)&(gender == 2))
```

Practice! Dataset exploration!

1. Create two new datasets. One for males (gender $= 1$) and one for females (gender $= 2$). Are there more missing values in the lownsdata or the highnsdata on the howdist variable?
2. Look at the variables that start with "ethnic". Why do you think there are so many missing values? Is this a problem?
3. Challenge: Can you find a way to first sort by condition (`nscond`) and then age?

Descriptive Statistics

There are a variety of summary functions which can be useful for calculating summary statistics.

```
mean(vaughndata$relativepromotion)
```

```
## [1] -0.1292247
```

```
range(vaughndata$relativepromotion)
```

```
## [1] -6  6
```

```
sd(vaughndata$relativepromotion)
```

```
## [1] 1.705874
```

Descriptive Statistics

```
cor(vaughndata$autonomy, vaughndata$competence)
```

```
## [1] 0.7138203
```

```
cov(vaughndata$autonomy, vaughndata$competence)
```

```
## [1] 1.545736
```

```
summary(vaughndata$autonomy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.333   4.500   4.417   5.667   7.000
```

Functions and Missing Values

When there are `NA` values in the data, many functions will give `NA` as the result.

```
mean(vaughndata$howdist)
```

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

```
mean(vaughndata$howdist, na.rm = TRUE) #FALSE is default
```

```
## [1] 1.882353
```

Functions in R

Functions are at the heart of R. Understanding how they work is incredibly important!

Functions will all start with a name then parenthesis a comma separated list of arguments and then closed parethesis.

There is an order to the arguements and if you don't label them, R will assume they are in order.

```
?sd
sd(x = vaughndata$relativepromotion, na.rm = TRUE)
sd(vaughndata$relativepromotion, TRUE)
sd(na.rm = TRUE, x = vaughndata$relativepromotion)
```

Functions in R

If you don't know the name of the arguments try pressing tab.

The descriptions are often very helpful!

Use the help pages

```
?dnorm
help(dnorm)
??"normal distribution"
```

The help page will show you what the defaults are

```
sd(x, na.rm = FALSE)
```

Outputs of Functions

What comes out of a function is often called a “return value"

Many of the help sections will have a “Value” section. This describes the return value from the function. Especially complex functions.

Let’s look at the help page for the `lm` function which we will use shortly.

Side-effects of Functions

Side-effects are something that occurs because you ran a function, but is not necessarily saved as a value.

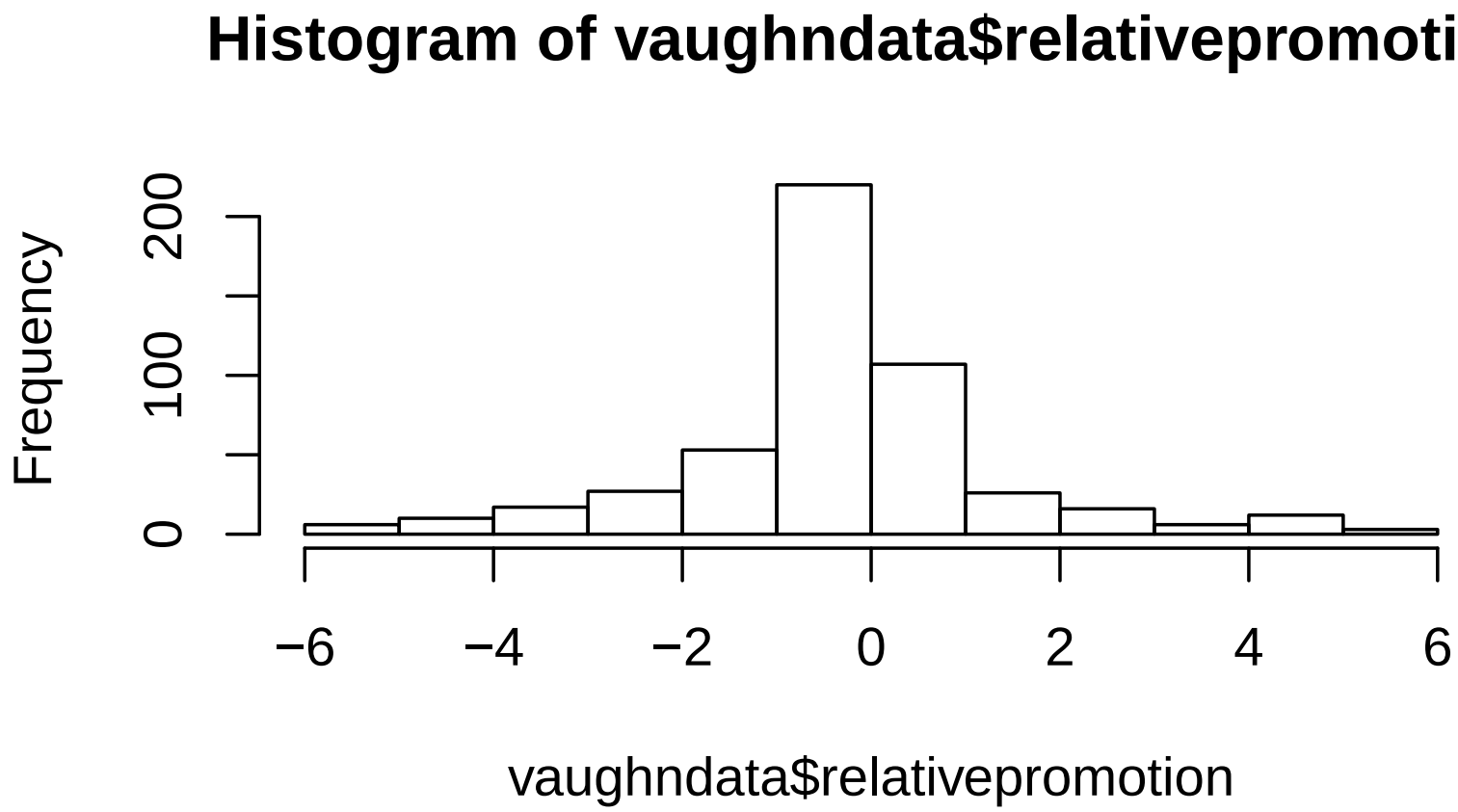
Some examples

- Plots
- Saving a file
- Posting to Twitter

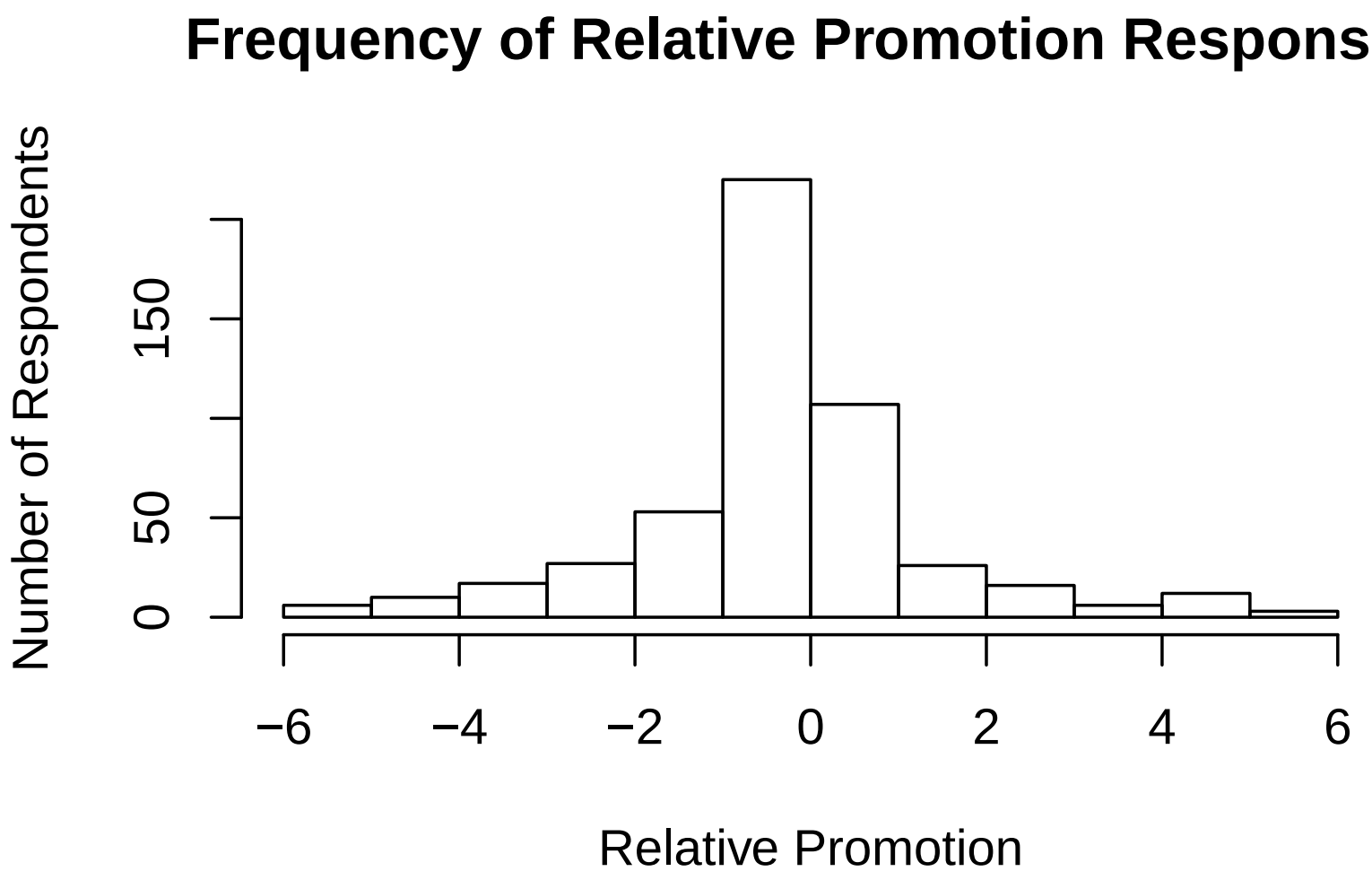
Basic Histograms

Histograms plot the frequency or density of certain responses

```
#?hist
hist(vaughndata$relativepromotion)
```



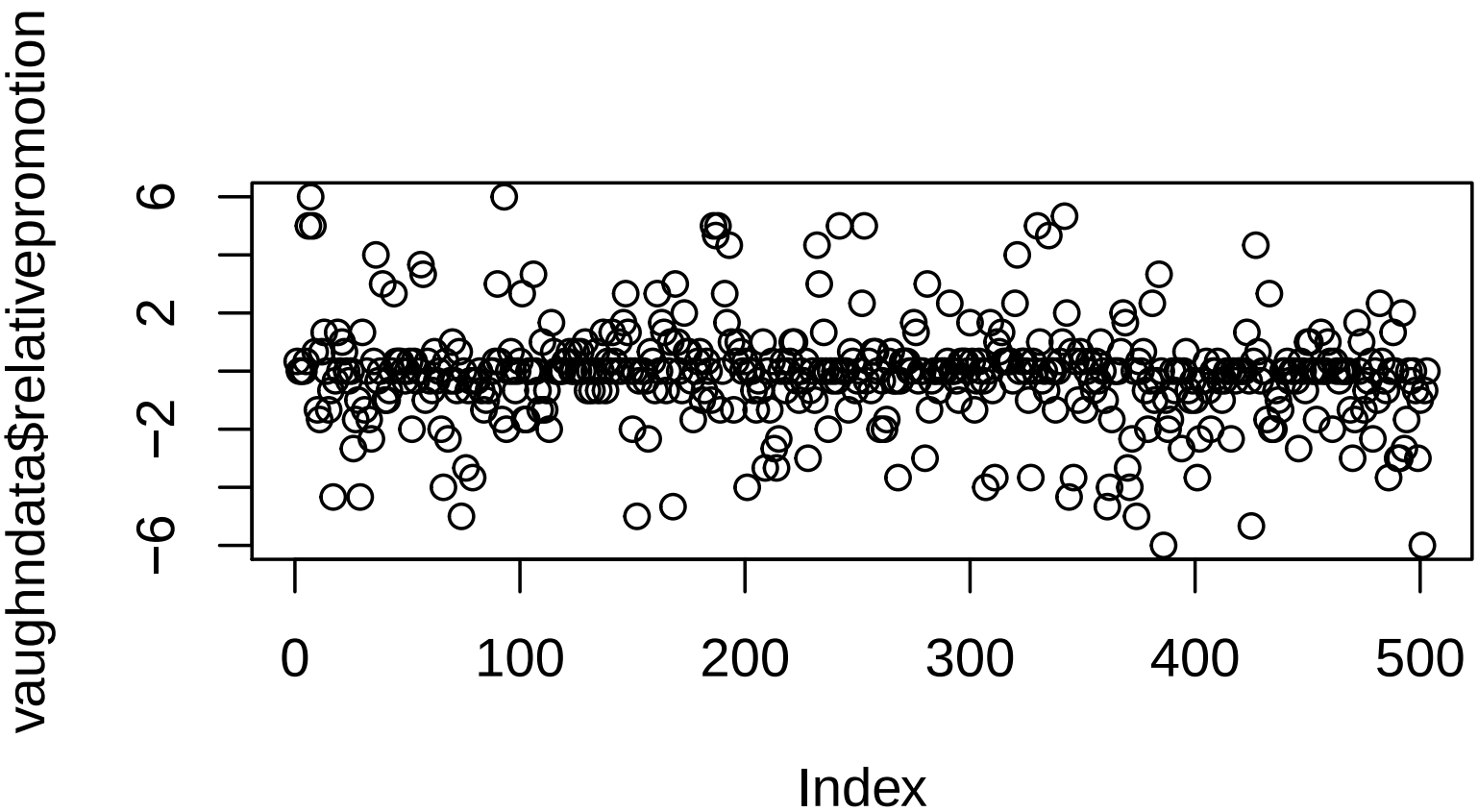
```
hist(x = vaughndata$relativepromotion, breaks = 16,
     xlab = "Relative Promotion",
     ylab = "Number of Respondents",
     main = "Frequency of Relative Promotion Responses")
```



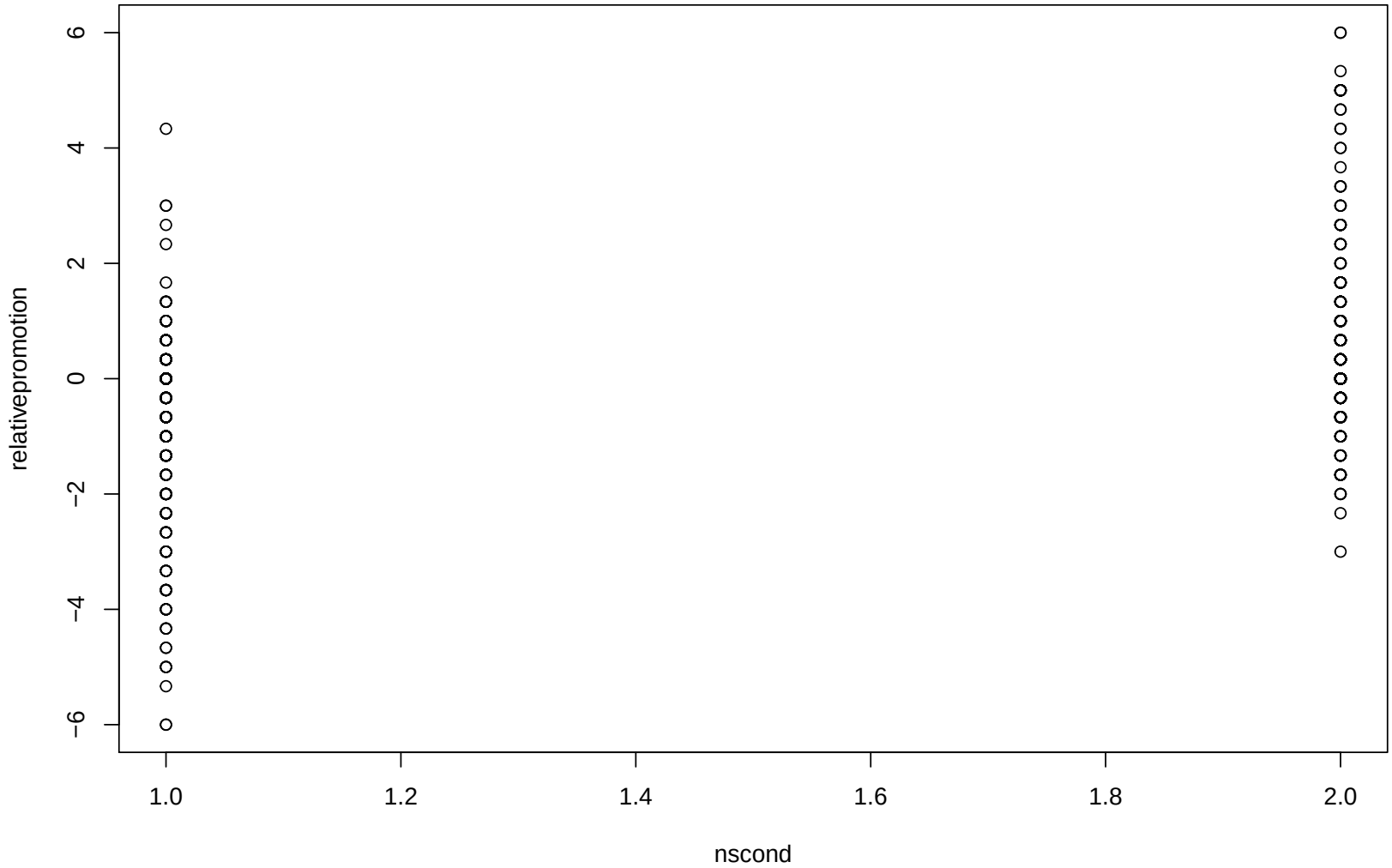
Basic plots

`plot()` function makes basic plots based on the "type"

```
#?hist
plot(vaughndata$relativepromotion)
```

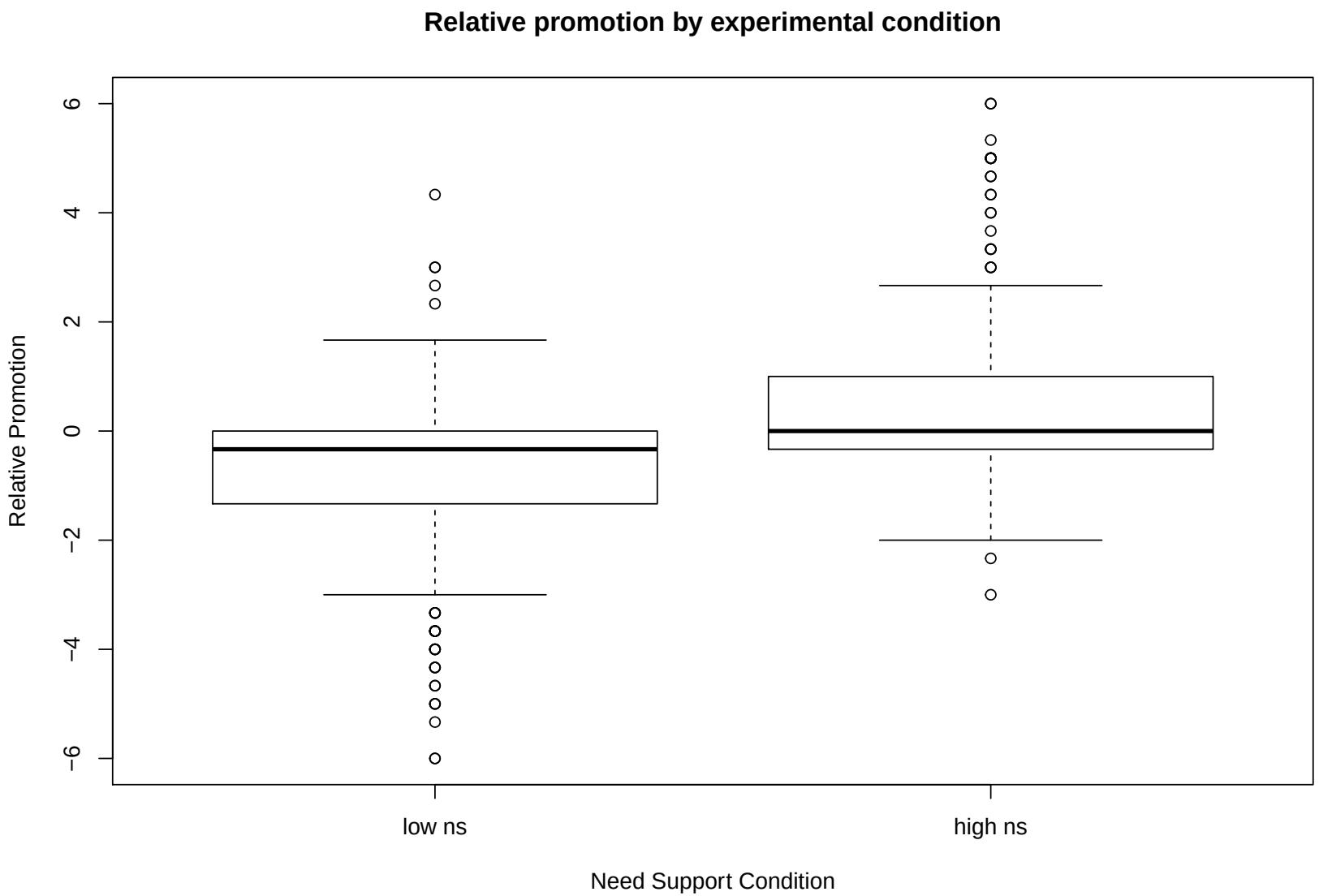


```r
with(vaughndata, plot(nscond, relativepromotion))
```




```r
#first value goes on X axis, second value on Y
```

```
vaughndata$nscondfact <- factor(vaughndata$nscond, labels =
with(vaughndata, plot(nscondfact, relativepromotion, ylab =
```



Linear Models

Regression and linear models are at the core of mediation and moderation analysis.

R makes doing regression easy and gives you a lot of the information you might want with fairly little effort.

lm formula

```
outcome ~ predictor1 #simple regression
outcome ~ predictor1 + predictor1 # multiple regression
outcome ~ predictor1 + predictor2 +
          predictor1*predictor2 #interaction
outcome ~ predictor1:predictor2 #shorter interactions
```

Using the Vaughn data

lm objects are pretty large and have a lot of information so save it as a variable, then look at different parts.

```
vaughnlm <- lm(relativepromotion ~ nscond,
                  data = vaughndata)
```

```r
summary(vaughnlm)

str(vaughnlm)

#See handout for output
```

Selecting elements of the lm object

You can save the coefficients and standard errors from these models

```
vaughnlm$coeff
```

```
## (Intercept)      nscond
##   -2.137922    1.334709
```

```
vcov(vaughnlm)
```

```
##             (Intercept)      nscond
## (Intercept)  0.04937748 -0.02954842
## nscond      -0.02954842  0.01963389
```

How might you get the variance for the coefficient for nscond only?

The fitted values and residuals

```
head(vaughnlm$fitted.values)
```

```
##         28         291          33          61         204
## -0.8032129   0.5314961 -0.8032129 -0.8032129   0.5314961
```

```
head(vaughnlm$residuals)
```

```
##         28         291          33          61         204
##  1.1365462 -0.5314961   0.8032129   0.8032129 -0.1981627
```

I like to save these as part of the dataset and plot them.

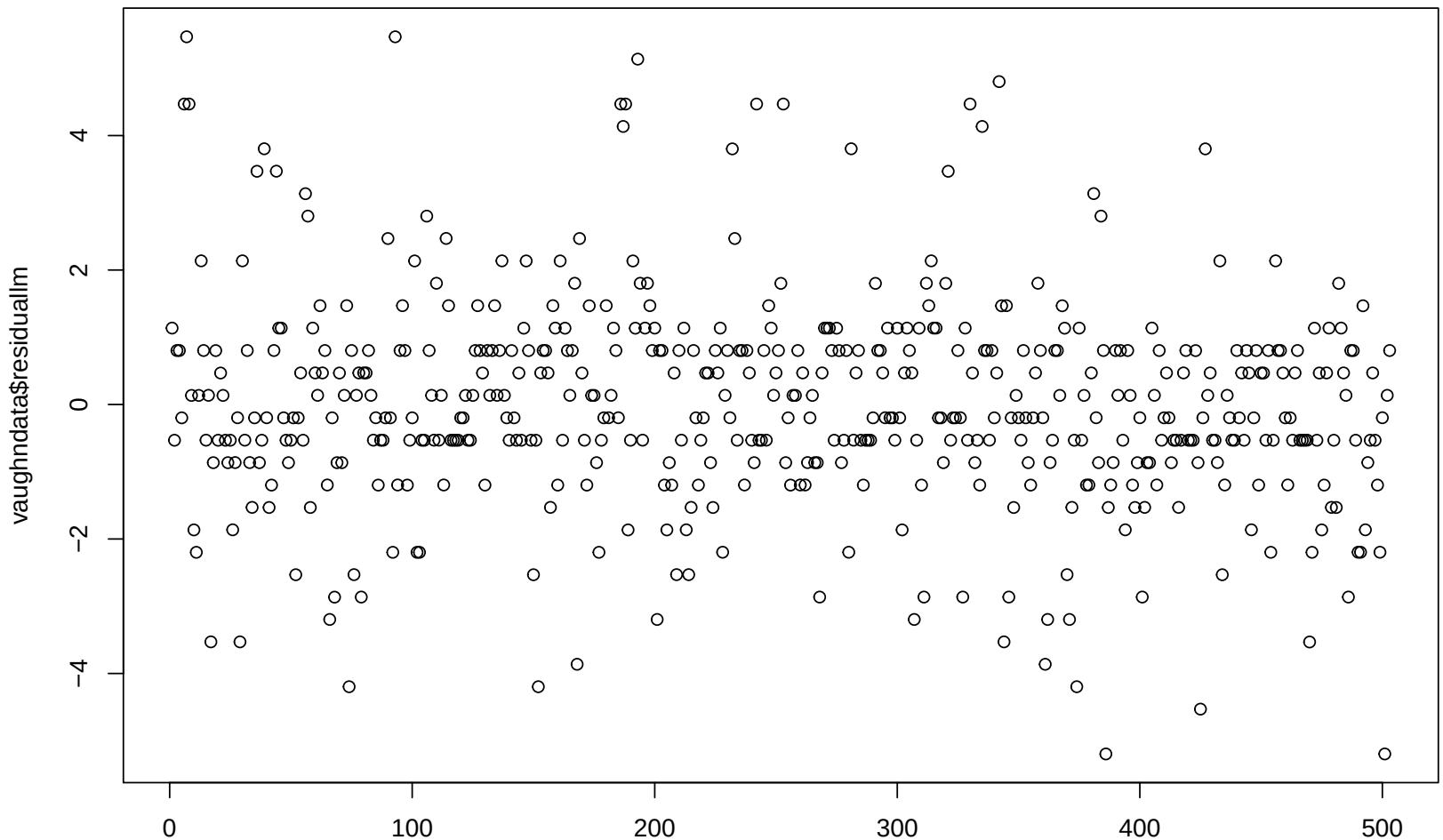
```
vaughndata$fittedlm <- vaughnlm$fitted.values
vaughndata$residuallm <- vaughnlm$residuals
```

Linear Models Assumptions

- Linear relationship between outcome and predictor(s)
- Normality of errors
- Homoskedasticity of errors
- Random sampling

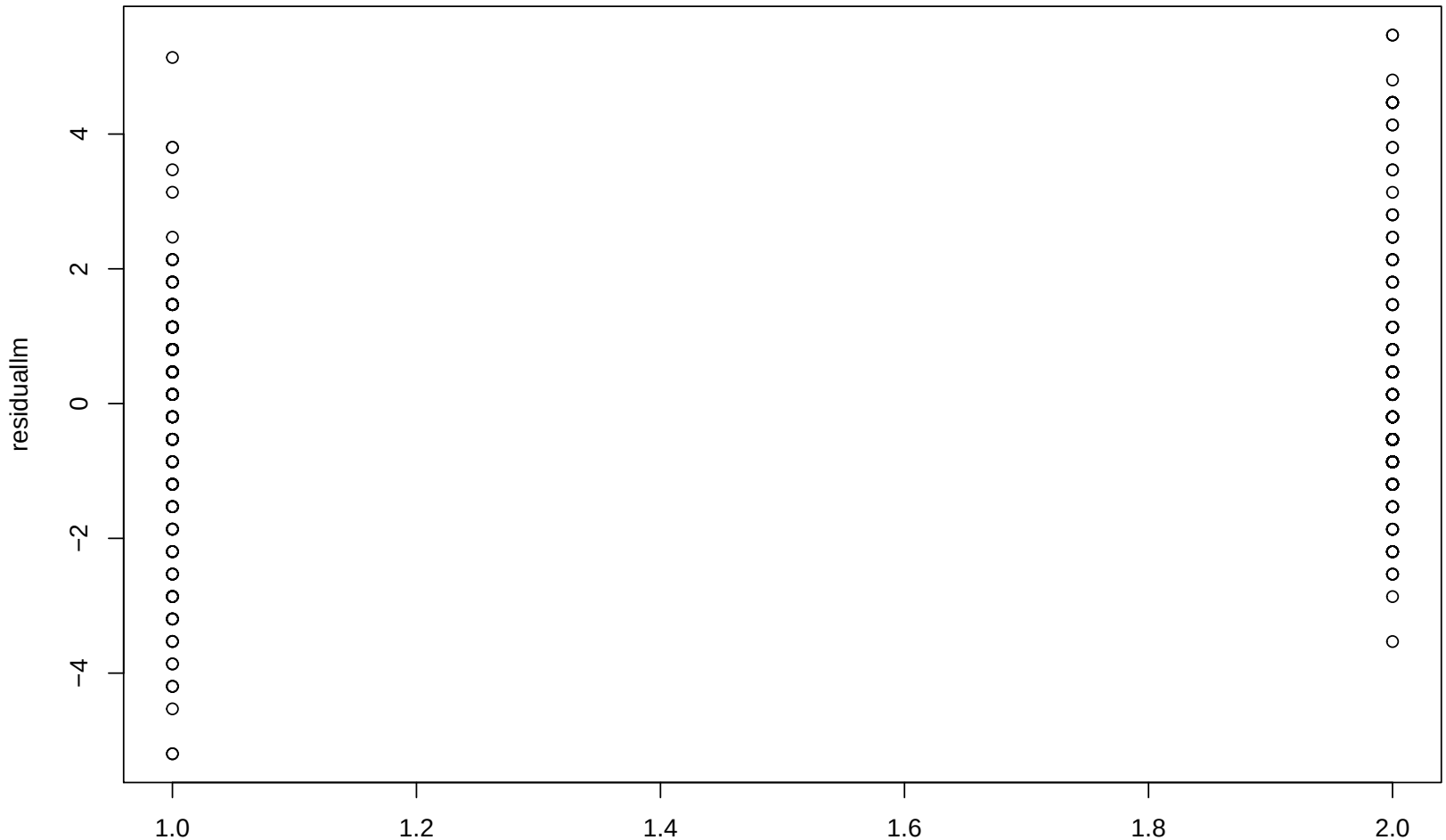
Residual plots

```
#homoskedasticity
plot(vaughndata$residuallm)
```



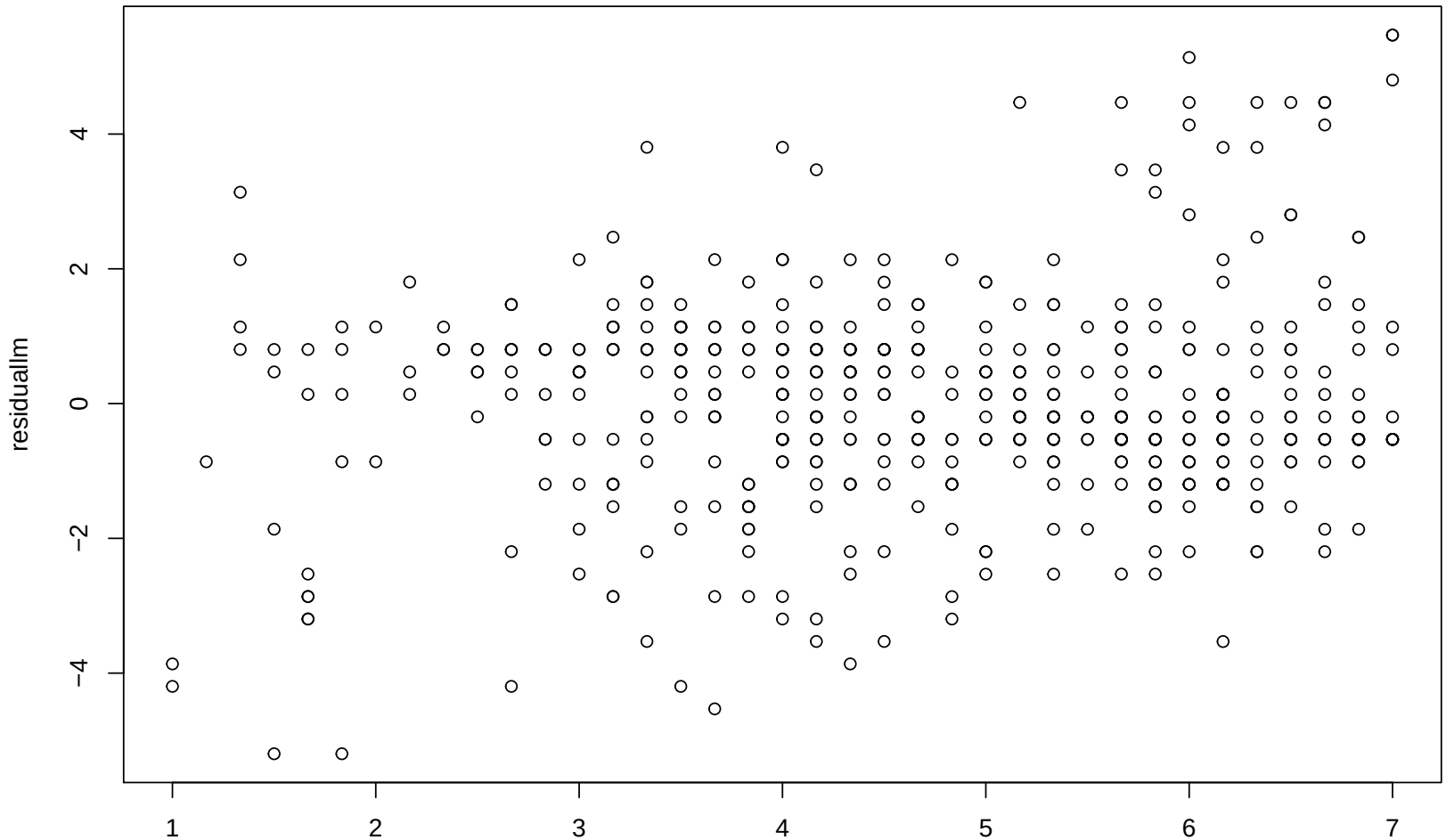
Residuals by condition

```
#homoskedasticity
with(vaughndata, plot(nscond, residuallm))
```



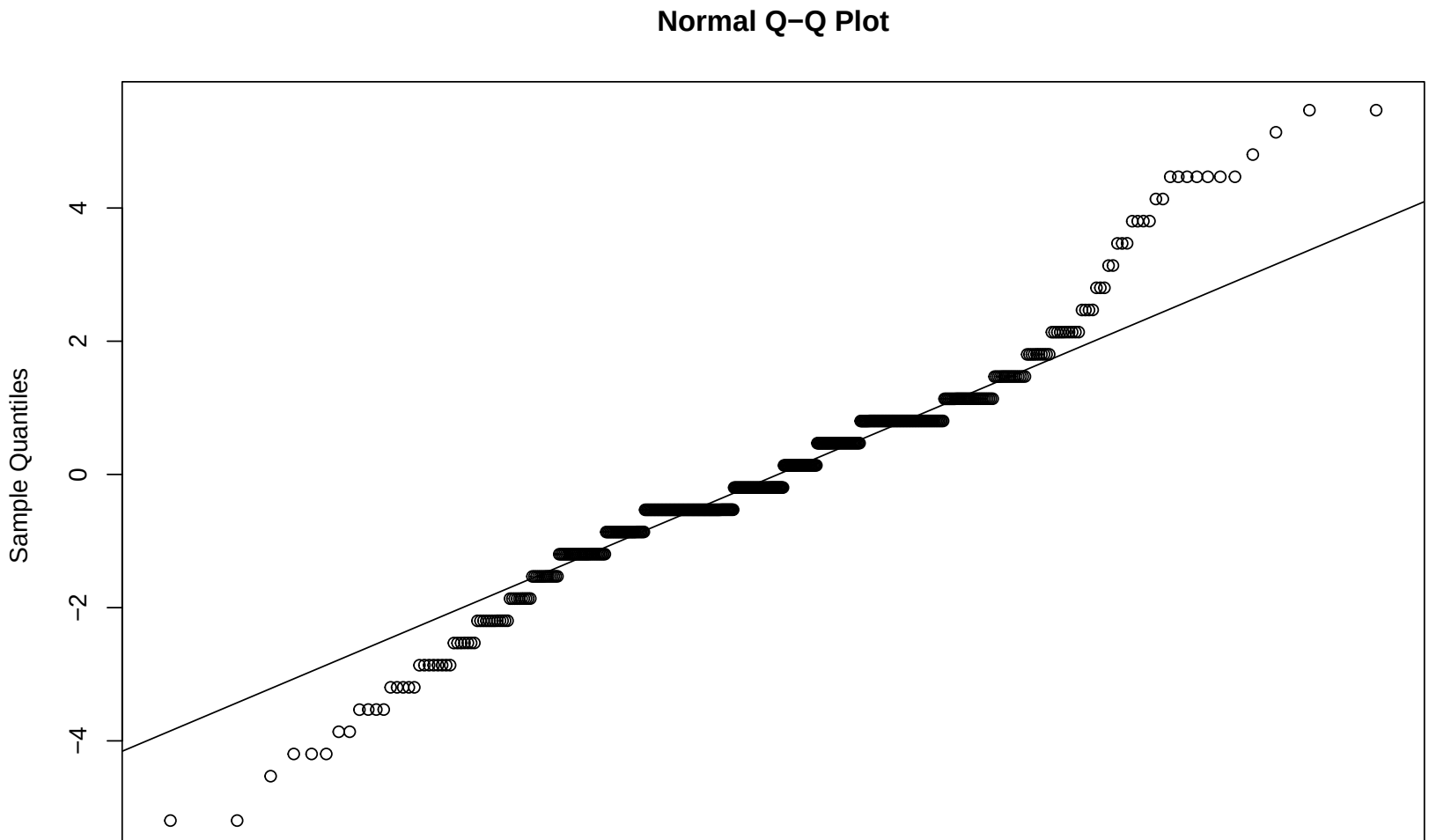
Residuals by competency

```
#confounders?
with(vaughndata, plot(competence, residuallm))
```



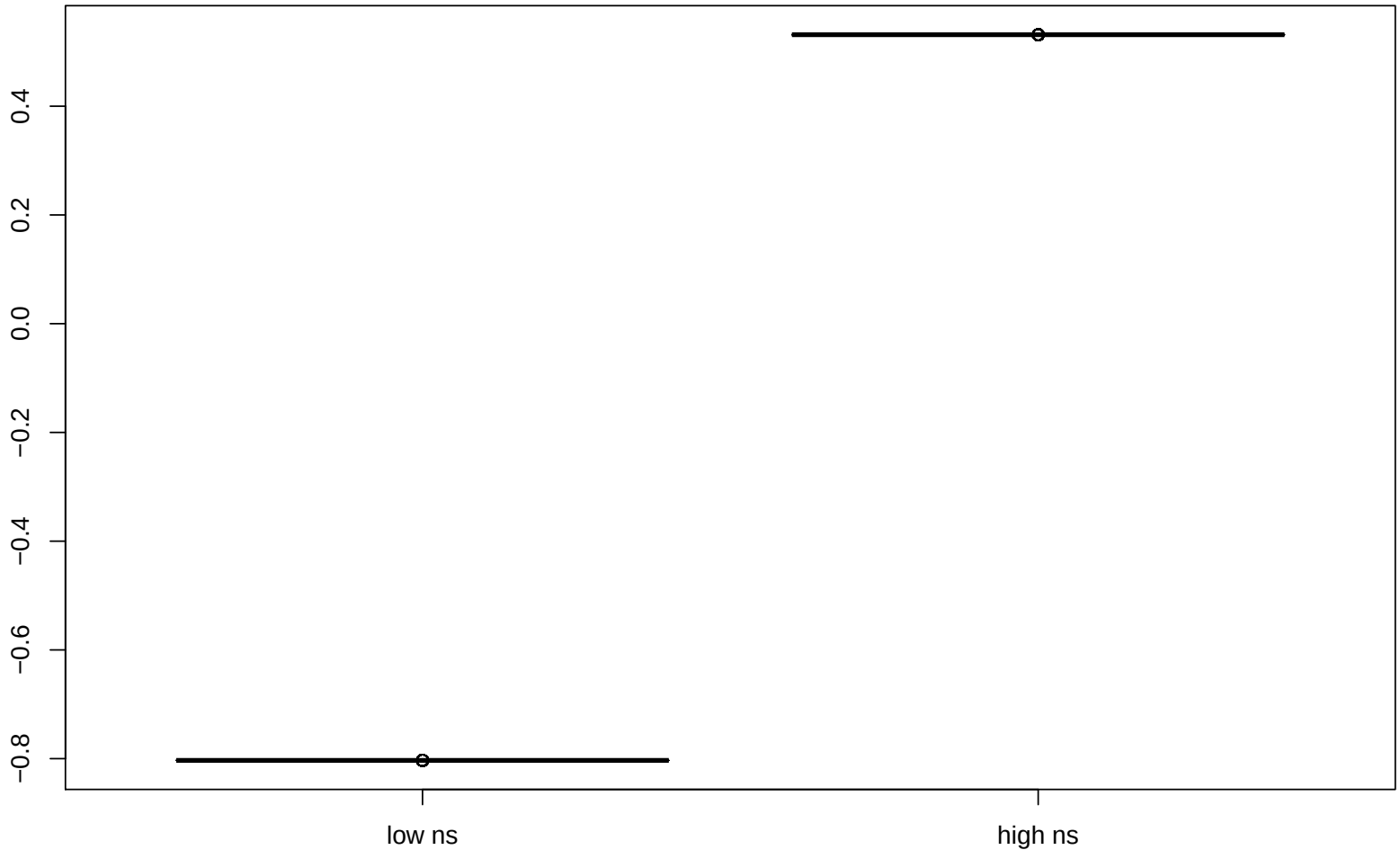
QQplots

```
#normality
qqnorm(vaughndata$residuallm)
qqline(vaughndata$residuallm)
```

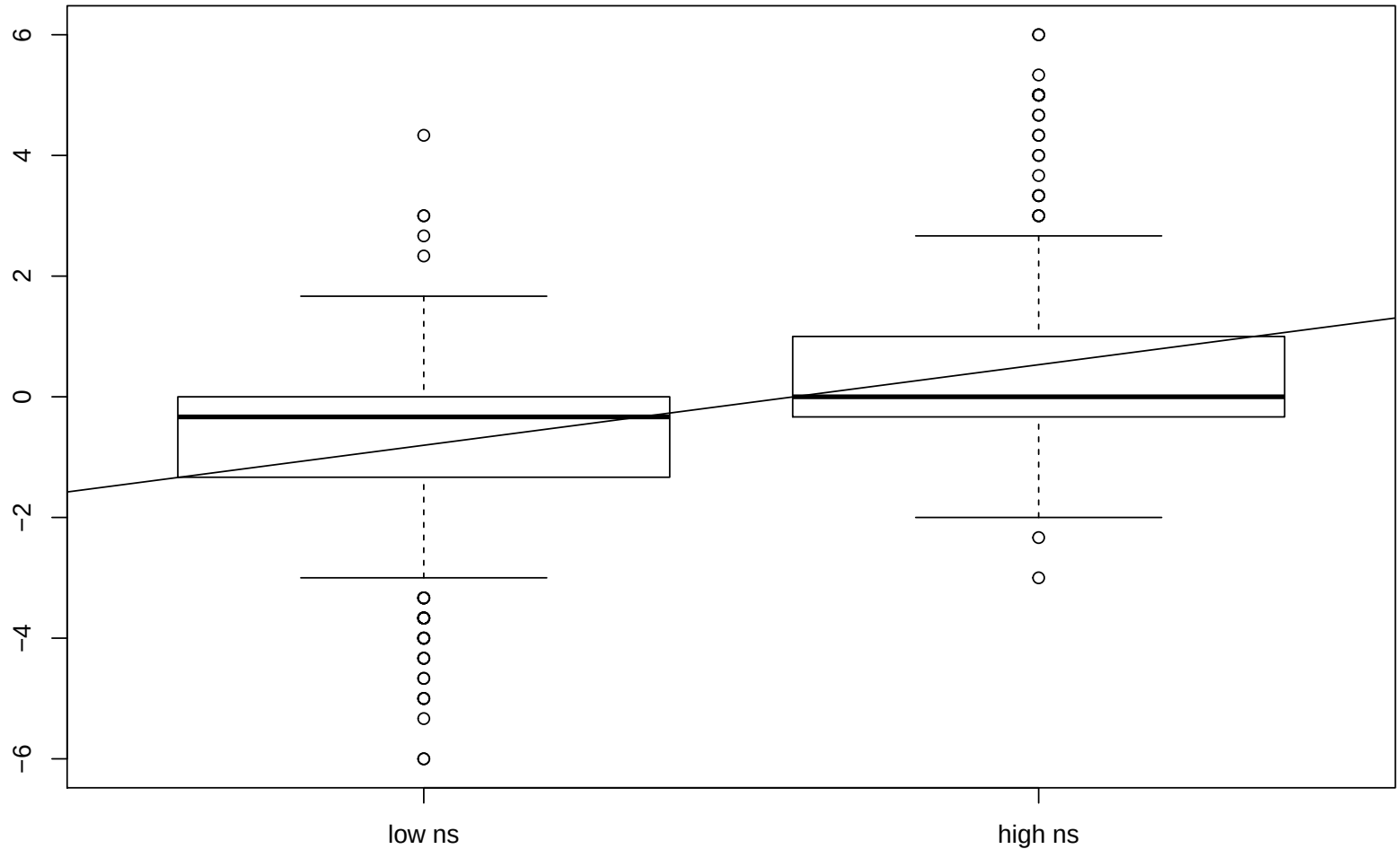


Fitted values

```
with(vaughndata, plot(nscondfact, fittedlm), type = "p")
```



```
with(vaughndata, plot(nscondfact, relativepromotion))
abline(vaughnlm)
```



Practice 3

1. Plot autonomy against relative promotion. Does it look like there might be some relationship between the two? Does this relationship look linear? Try using the `with()` function.
2. Make a new linear model predicting relative promotion from autonomy. Is there a significant relationship between autonomy and relative promotion? Write a sentence describing your results, like you might in a paper.
3. Save the fitted and residual values as part of the dataframe. Make sure to make new variable names, so you don't save over previous variables.
4. Plot the residuals in at least 3 different ways. Do you see any evidence of assumption violations?
5. Plot the original data for autonomy and relative promotion. Add the fitted line over this plot.
6. Challenge: Repeated this exercise but including nscond as a predictor. What additional plots might you make now that you have two predictors? Do both predictors significantly predict

Afternoon Session 1:00pm - 4:00pm