Intro to Analysis

Ben Chu

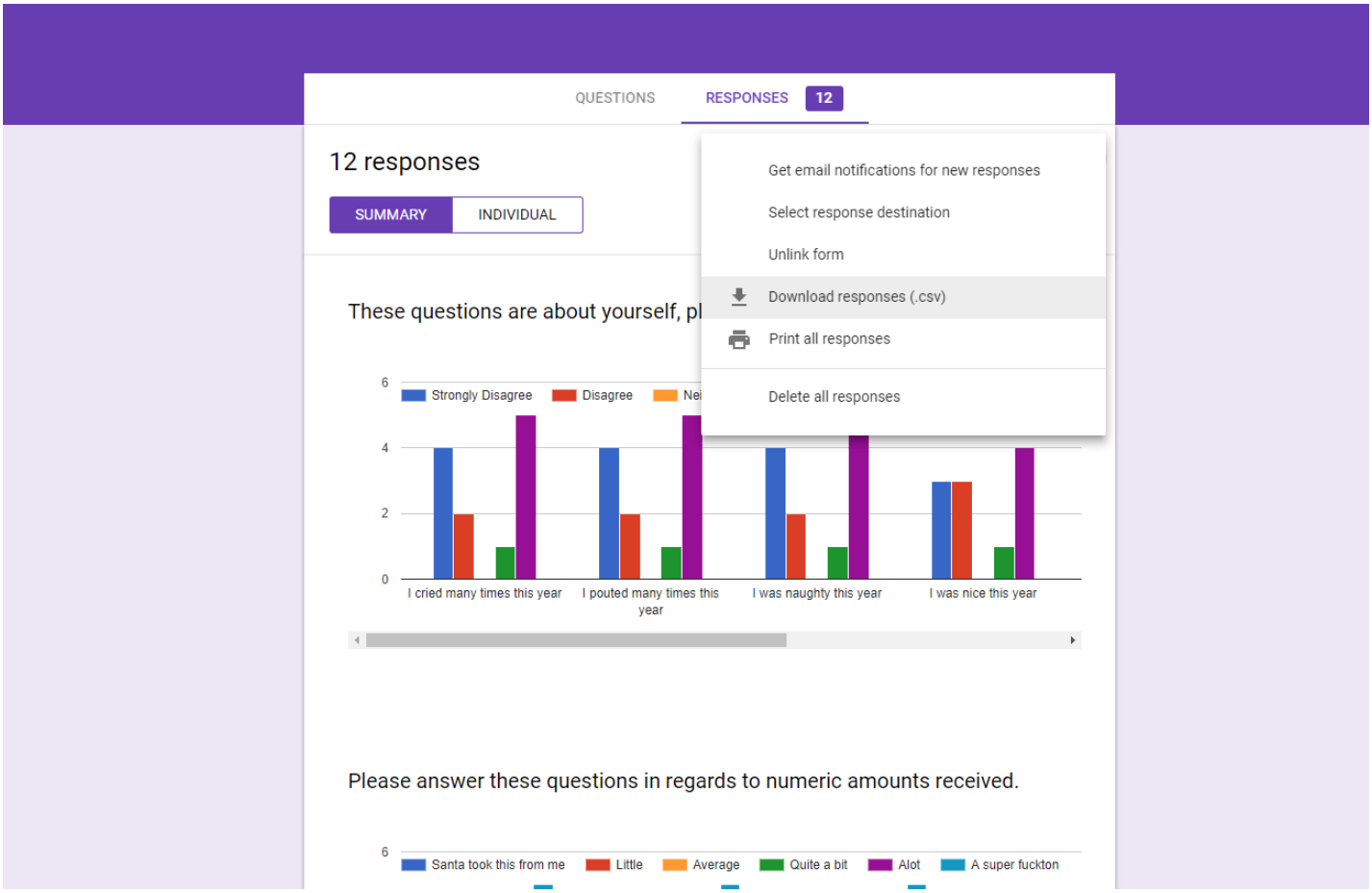
March 4, 2018

This is a brief introduction into data analysis.

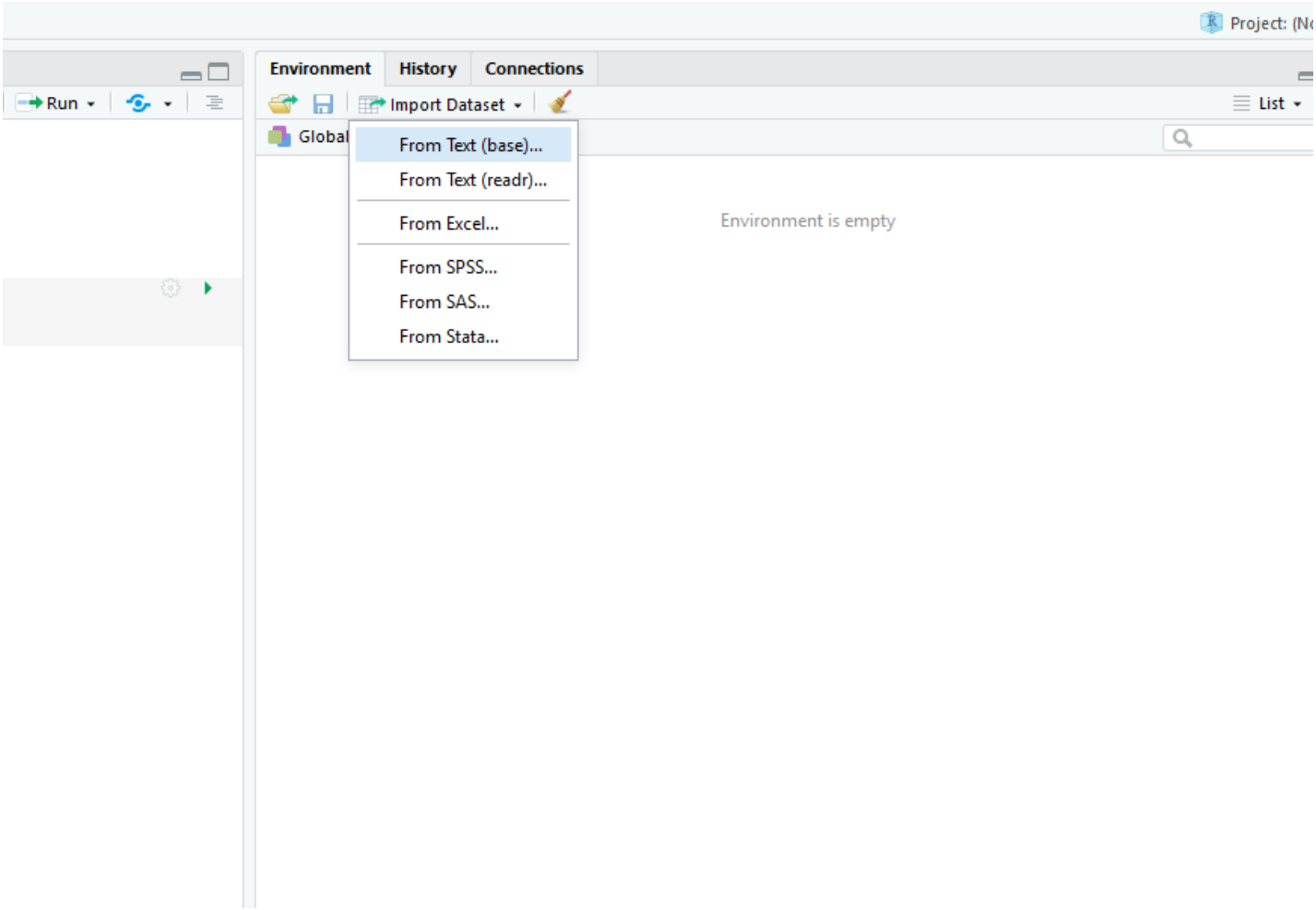
In this small handout, I will explain how to do basic data cleaning and analysis. This instruction is meant to show you how data scientists approach turning data into usable information. I want to further preface by saying there are many ways to clean data but this way is how I (Ben) usually approach cleaning data.

First things to do is to find the data itself.

Find the `.csv` file and export it. If you are using Google forms, please open your survey and switch over to the response section. Next to the selection, there is a dropdown that provides you with an option to download your responses as a `.csv` file. This file is a `comma seperated` which means it can be opened with very basic text files such as notepad or excel or word.



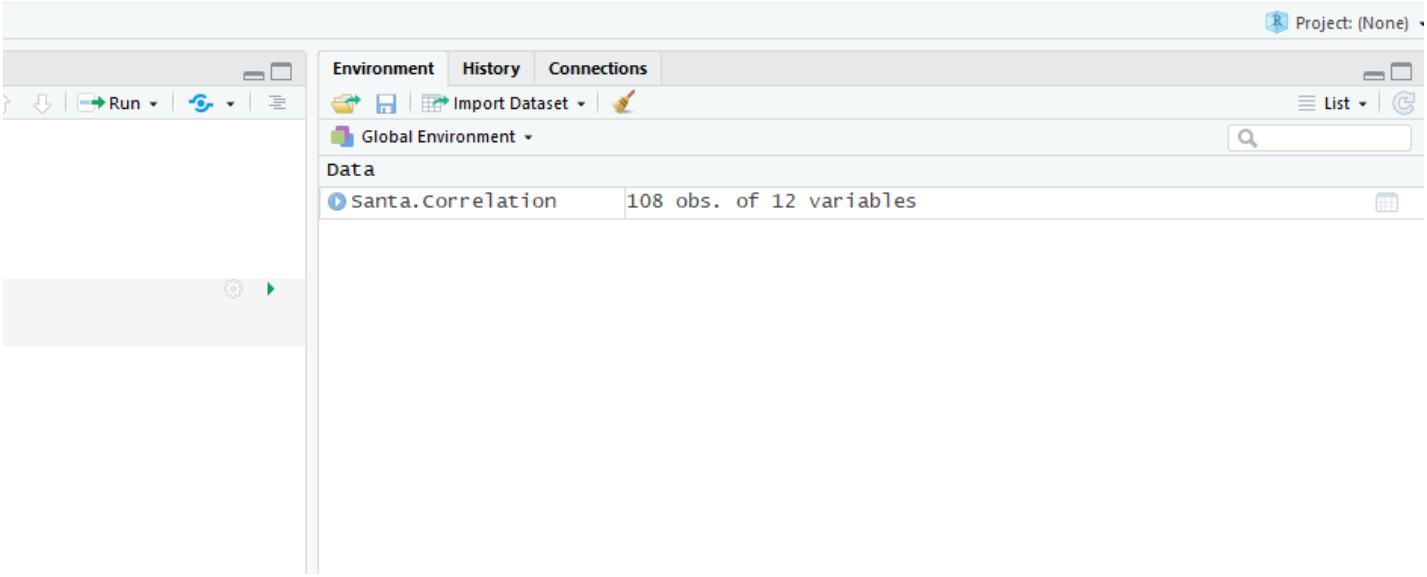
Next thing to do is to important it into your environment. My file ended up being saved as `Santa.Correlation.csv`



You can also do this via the command line.

```
Santa.Correlation <- read.csv("C:/Users/Branly Mclanbry/Desktop/Santa Correlation.csv")
```

Nice! it should appear in your global environment



The data itself is dirty so we have to take a few steps to clean it up first.

1. Install packages

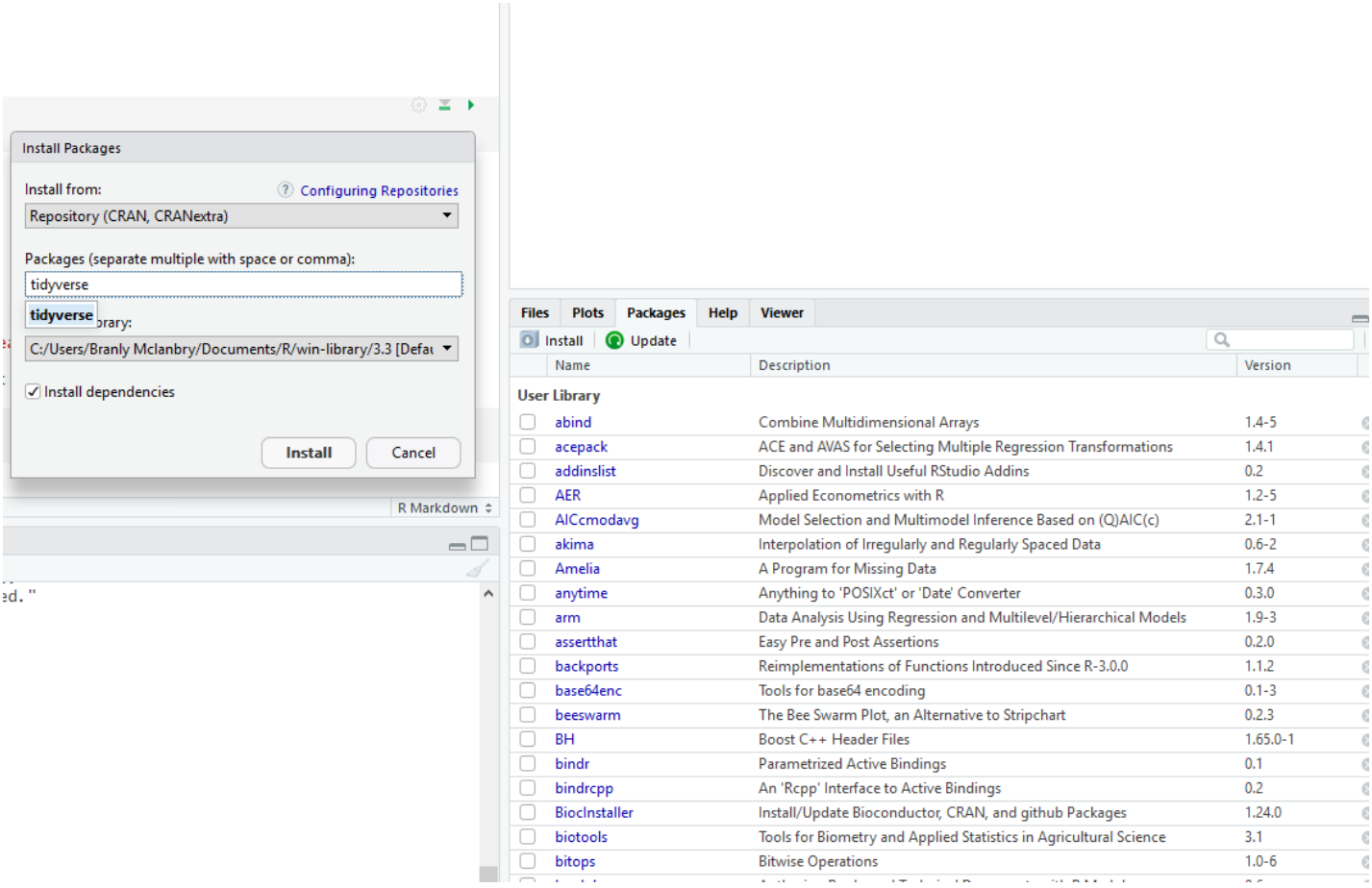
2. Make the item names less confusing
3. Turn items into numbers
4. Reverse Code
5. Composite items into constructs

The first thing we want to do is to install and activate some packages.

Packages are pre-built user submitted functions that makes life easier. An analogy would be like buying furniture from IKEA. They provide you with everything you need to do what you want, you simply need to follow instructions.

You can install it via the point and click install, but most people prefer to install it via the command line. The `tidyverse` package comes from a genius called `Hadley Wickham`. He's a r rockstar, a `r` ockstar because he essentially brainchilded code that makes it easier for people to understand and utilize. If I ever see him in public and I'm going to shake his hand and have him sign my forehead.

```
install.packages("tidyverse")
```



Nice!You can activate the package by using this function.

```
library(tidyverse)
```

Next thing we want to do is to make the variables we are using less complicated.

We want to turn items into less confusing names because `"These.questions.are.about.yourself..please.answer.in.full...I.cried.many.times.this.year."` is a really long unecessary item. THANKS GOOGLE

Let's shorten it a bit shall we?

From the `tidyverse` package, we want to use the `mutate` function to duplicate our variables with less complicated names. I'm going to create a new dataset arbitrarily called `data` and it will have the exact same information but with easier to understand names.

The `%>%` functions basically means "and then"".
The `mutate` function tells us to create a new variable based on previous inputs.
So the function below is telling us several things."Please create a new dataset called `data` using the information from `Santa.Correlation` and with that, duplicate the variable `These.questions.blabla` but call it `cried` " and so on.

```
data <- Santa.Correlation %>%
  mutate(cried = These.questions.are.about.yourself..please.answer.in.full...I.cried.many.times.this.year.,
         pouted = These.questions.are.about.yourself..please.answer.in.full...I.pouted.many.times.this.year.,
         naughty = These.questions.are.about.yourself..please.answer.in.full...I.was.naughty.this.year.,
         nice = These.questions.are.about.yourself..please.answer.in.full...I.was.nice.this.year.,
         bad = These.questions.are.about.yourself..please.answer.in.full...I.was.bad.this.year.,
         good = These.questions.are.about.yourself..please.answer.in.full...I.was.good.this.year.,
         good4 = These.questions.are.about.yourself..please.answer.in.full...I.was.good.for.goodness.sake. ,
         horns = Please.answer.these.questions.in.regards.to.numeric.amounts.received....Tiny.tin.horns. ,
         drums = Please.answer.these.questions.in.regards.to.numeric.amounts.received....Little.toy.drums. ,
         stockings = Please.answer.these.questions.in.regards.to.numeric.amounts.received....Stockings.stuffed.,
         gifts = Please.answer.these.questions.in.regards.to.numeric.amounts.received....Gifts.)
```

Note, I used the `names` function to get my variable names. All you have to do is call `names(Santa.Correlation)` to get all the variable names in your dataset. Which then allows you to copy and paste easily.

Great!

Let's turn some characters into numerics.

Now that we have understandable names, we need to create understandable and analyzable responses. This is because character codes can't be compared to each other.
Let's use the `head` function to look at the first 5 responses for `nice and the first 5 responses for` stockings`.

While they do make sense, they can't be statistically analyzed because statistics uses numbers and not character strings.

```
head(data$nice,5)
```

```
## [1] Strongly Agree                    Strongly Disagree Agree
## [5] Strongly Disagree
## 6 Levels:  Agree Disagree Neither Agree or Disagree ... Strongly Disagree
```

```
head(data$stockings,5)
```

```
## [1] A super fuckton         A super fuckton         Santa took this from me
## [4] A super fuckton         Little
## 6 Levels: A super fuckton Alot Average Little ... Santa took this from me
```

Yeesh, what a mess.
I want to say that this is one of the most thorough but complicated ways to recode variables. I am showing it to you this way because of how simple and straightforward it is. Albeit, burdensome.

Again, we will be using the `%>%` and `mutate` function. Now we will be throwing in a new function called `case_when`, which basically works in the following:
"when `nice` has a response which is `Strongly Disagree`, please relabel it as `1`." and so forth.

note I am creating a new variable called `nice.r` which is seperate from `nice` because it is recoded.

```
data <- data %>%
  mutate(nice.r = case_when(nice == "Strongly Disagree"         ~ 1,
                            nice == "Disagree"                  ~ 2,
                            nice == "Neither Agree or Disagree" ~ 3,
                            nice == "Agree"                     ~ 4,
                            nice == "Strongly Agree"            ~ 5))
```

I further want to talk about reverse coding. There are some questions that are worded in a negative manner. We need to have reversely coded items in order to have all the items go in the same direction.

For example, `cried` is reverse coded, but instead of having the scale direction of 1-5, we want 5-1.
Let's add on our previous code by having things be reverse coded. We want to create a new variable called `cried.r`.

```
data <- data %>%
  mutate(nice.r = case_when(nice == "Strongly Disagree"         ~ 1,
                            nice == "Disagree"                  ~ 2,
                            nice == "Neither Agree or Disagree" ~ 3,
                            nice == "Agree"                     ~ 4,
                            nice == "Strongly Agree"            ~ 5),
         cried.r = case_when(cried == "Strongly Disagree"          ~ 5,
                             cried == "Disagree"                   ~ 4,
                             cried == "Neither Agree or Disagree" ~ 3,
                             cried == "Agree"                      ~ 2,
                             cried == "Strongly Agree"              ~ 1))
```

There is a package called `car` which has a function `recode` which can make this VERY easy. But that's for you to find out

Now let's do it for all the variables.

```
data.numerics <- data %>%
  mutate(nice.r = case_when(nice == "Strongly Disagree"           ~ 1,
                            nice == "Disagree"                    ~ 2,
                            nice == "Neither Agree or Disagree"   ~ 3,
                            nice == "Agree"                       ~ 4,
                            nice == "Strongly Agree"              ~ 5),
         good.r = case_when(good == "Strongly Disagree"           ~ 1,
                            good == "Disagree"                    ~ 2,
                            good == "Neither Agree or Disagree"   ~ 3,
                            good == "Agree"                       ~ 4,
                            good == "Strongly Agree"              ~ 5),
         good4.r = case_when(good4 == "Strongly Disagree"         ~ 1,
                            good4 == "Disagree"                   ~ 2,
                            good4 == "Neither Agree or Disagree"  ~ 3,
                            good4 == "Agree"                      ~ 4,
                            good4 == "Strongly Agree"             ~ 5),
         cried.r = case_when(cried == "Strongly Disagree"         ~ 5,
                             cried == "Disagree"                  ~ 4,
                             cried == "Neither Agree or Disagree" ~ 3,
                             cried == "Agree"                     ~ 2,
                             cried == "Strongly Agree"            ~ 1),
         pouted.r = case_when(cried == "Strongly Disagree"        ~ 5,
                             cried == "Disagree"                  ~ 4,
                             cried == "Neither Agree or Disagree" ~ 3,
                             cried == "Agree"                     ~ 2,
                             cried == "Strongly Agree"            ~ 1),
         naughty.r = case_when(cried == "Strongly Disagree"       ~ 5,
                             cried == "Disagree"                  ~ 4,
                             cried == "Neither Agree or Disagree" ~ 3,
                             cried == "Agree"                     ~ 2,
                             cried == "Strongly Agree"            ~ 1),
         bad.r = case_when(cried == "Strongly Disagree"           ~ 5,
                             cried == "Disagree"                  ~ 4,
                             cried == "Neither Agree or Disagree" ~ 3,
                             cried == "Agree"                     ~ 2,
                             cried == "Strongly Agree"            ~ 1),
         horns.r = case_when(horns == "Santa took this from me " ~ 1,
                            horns == "Little"                     ~ 2,
                            horns == "Average"                    ~ 3,
                            horns == "Quite a bit"                ~ 4,
                            horns == "Alot"                       ~ 5,
                            horns == "A super fuckton"            ~ 6),
         drums.r = case_when(drums == "Santa took this from me " ~ 1,
                            drums == "Little"                     ~ 2,
                            drums == "Average"                    ~ 3,
                            drums == "Quite a bit"                ~ 4,
                            drums == "Alot"                       ~ 5,
                            drums == "A super fuckton"            ~ 6),
         stockings.r = case_when(stockings == "Santa took this from me "  ~ 1,
                            stockings == "Little"                      ~ 2,
                            stockings == "Average"                     ~ 3,
                            stockings == "Quite a bit"                 ~ 4,
                            stockings == "Alot"                        ~ 5,
```

```
                        stockings == "A super fuckton"          ~ 6),
        gifts.r = case_when(gifts == "Santa took this from me " ~ 1,
                        gifts == "Little"                       ~ 2,
                        gifts == "Average"                      ~ 3,
                        gifts == "Quite a bit"                  ~ 4,
                        gifts == "Alot"                         ~ 5,
                        gifts == "A super fuckton"              ~ 6))
```

Whew, that's alot of copy and pasting.

Composite scores.

Now that our items are all in the same direction, we can create a composite variable that is a combination of the items together. For example, we want to know the total niceness of a child on christmas night. It's the total of nice items divided by the number of nice items. Again, we will be using the `%>%` and `mutate` function.

```
data.total <- data.numerics %>%
  mutate(niceness = (nice.r + good.r + good4.r)/3)
```

Nice!
Now let's do it for our two scales.

```
data.total <- data.numerics %>%
  mutate(niceness = (nice.r + good.r + good4.r)/3,
         somanygifts = (horns.r + drums.r + stockings.r + gifts.r)/4)
```

Cool bean! Now we have composite scores.

Let's just take a look at our data in terms of means or ranges. Use the `describe` function from the `Hmisc` package and it'll provide a small summary of how our data looks. This function is important because it provides overall information during our write up. The function operates as `data$variable` . Essentially it is asking us to select our data, and from that data to choose a variable.
note describe is a super common function that is terribly titled. We need to specify which function from which package we are using. To do so, we just use the name of the package, and the function within the "::" tells use which function. `package::function` .

```
psych::describe(data.total$niceness)
```

```
##    vars  n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 20 2.77 1.56   2.33    2.71 1.98   1   5     4 0.28    -1.63 0.35
```

```
psych::describe(data.total$somanygifts)
```

```
##    vars  n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 15 4.17 1.64      4    4.19 2.97   2   6     4 -0.1    -1.77 0.42
```

Reliability

We want to know about the internal reliability of our items. This concept gets at the idea of “are each items measuring what they are supposed to be mesuring?” For example, If I wanted to ask questions of how nice a child was, I would want to ask questions like “how nice are you” or “have you been good?”. Asking a question like “What do you think about the color purple” might not be the best construct to ask for niceness.

We will be using the package `psych` and within that, the function of `alpha` which will provide us with Cronbach’s alpha.
We first need to be selecting the items we want to test using the `select` function.
Let’s call our overall scale `naughtynicelist` which is selected by nice items from our `data.total`

```
naughtynicelist <- data.total %>%
  select(nice.r,good.r,good4.r,cried.r,pouted.r,naughty.r,bad.r)
```

Super! How about another list for our overall gifts?

```
overallgifts <- data.total %>%
  select(horns.r, drums.r, stockings.r, gifts.r)
```

Now let’s look at the total Cronbach’s Alpha, but first we need to load the package `psych` .

```
library(psych)
alpha(naughtynicelist)
```

```
## Some items ( nice.r good.r good4.r ) were negatively correlated with the total scale and
## probably should be reversed.
## To do this, run the function again with the 'check.keys=TRUE' option
```

```
##
## Reliability analysis
## Call: alpha(x = naughtynicelist)
##
##   raw_alpha std.alpha G6(smc) average_r S/N  ase mean  sd
##       0.73      0.73    0.98      0.28 2.7 0.11  3.1 1.1
##
##  lower alpha upper     95% confidence boundaries
## 0.53 0.73 0.94
##
##  Reliability if an item is dropped:
##           raw_alpha std.alpha G6(smc) average_r S/N alpha se
## nice.r         0.78      0.78    0.99      0.38 3.6    0.087
## good.r         0.75      0.75    0.95      0.33 2.9    0.089
## good4.r        0.74      0.72    0.96      0.30 2.6    0.096
## cried.r        0.65      0.64    0.97      0.23 1.8    0.142
## pouted.r       0.65      0.64    0.97      0.23 1.8    0.142
## naughty.r      0.65      0.64    0.97      0.23 1.8    0.142
## bad.r          0.65      0.64    0.97      0.23 1.8    0.142
##
##  Item statistics
##            n raw.r std.r r.cor r.drop mean  sd
## nice.r    21  0.30  0.27  0.21  0.051  3.0 1.5
## good.r    22  0.46  0.44  0.43  0.215  2.9 1.7
## good4.r   21  0.55  0.53  0.51  0.309  2.8 1.8
## cried.r   21  0.79  0.77  0.78  0.653  3.2 1.7
## pouted.r  21  0.79  0.77  0.78  0.653  3.2 1.7
## naughty.r 21  0.79  0.77  0.78  0.653  3.2 1.7
## bad.r     21  0.79  0.77  0.78  0.653  3.2 1.7
##
## Non missing response frequency for each item
##              1    2    3    4    5 miss
## nice.r    0.19 0.24 0.19 0.10 0.29 0.05
## good.r    0.32 0.18 0.05 0.18 0.27 0.00
## good4.r   0.43 0.10 0.10 0.05 0.33 0.05
## cried.r   0.29 0.10 0.05 0.24 0.33 0.05
## pouted.r  0.29 0.10 0.05 0.24 0.33 0.05
## naughty.r 0.29 0.10 0.05 0.24 0.33 0.05
## bad.r     0.29 0.10 0.05 0.24 0.33 0.05
```

```
alpha(overallgifts)
```

```
##
## Reliability analysis
## Call: alpha(x = overallgifts)
##
##   raw_alpha std.alpha G6(smc) average_r S/N    ase mean  sd
##       0.97      0.97       1       0.9  37 0.0099  3.9 1.7
##
##  lower alpha upper     95% confidence boundaries
## 0.95 0.97 0.99
##
##  Reliability if an item is dropped:
##             raw_alpha std.alpha G6(smc) average_r S/N alpha se
## horns.r          0.96      0.96    0.95      0.88  23    0.016
## drums.r          0.97      0.97    0.96      0.91  31    0.012
## stockings.r      0.97      0.97    0.98      0.91  31    0.013
## gifts.r          0.97      0.97    0.96      0.90  28    0.014
##
##  Item statistics
##              n raw.r std.r r.cor r.drop mean  sd
## horns.r     18  0.98  0.98  0.98   0.96  3.8 1.7
## drums.r     17  0.95  0.95  0.95   0.92  3.9 1.7
## stockings.r 16  0.97  0.96  0.96   0.92  4.1 1.7
## gifts.r     16  0.96  0.96  0.96   0.93  4.1 1.8
##
## Non missing response frequency for each item
##                2    3    4    5    6 miss
## horns.r     0.33 0.17 0.11 0.11 0.28 0.18
## drums.r     0.29 0.24 0.06 0.12 0.29 0.23
## stockings.r 0.25 0.19 0.06 0.19 0.31 0.27
## gifts.r     0.31 0.12 0.06 0.12 0.38 0.27
```

We will be looking at the raw_alpha number. What this is telling us is how closely tied together our items are. I received a warning telling me that some items are negatively correlated. That might be the case that I forgot to reverse code them initially. However, in this case it might have been because I randomly selected answers.

Correlations

Let's take a look at the correlation between Santa's list and the amount of gifts received. I will be using the `cor.test` function with our two variables. The basic formula for finding correlations is the `data$variable` and the `$` requests the variable from the data.

```
cor.test(data.total$niceness,data.total$somanygifts)
```

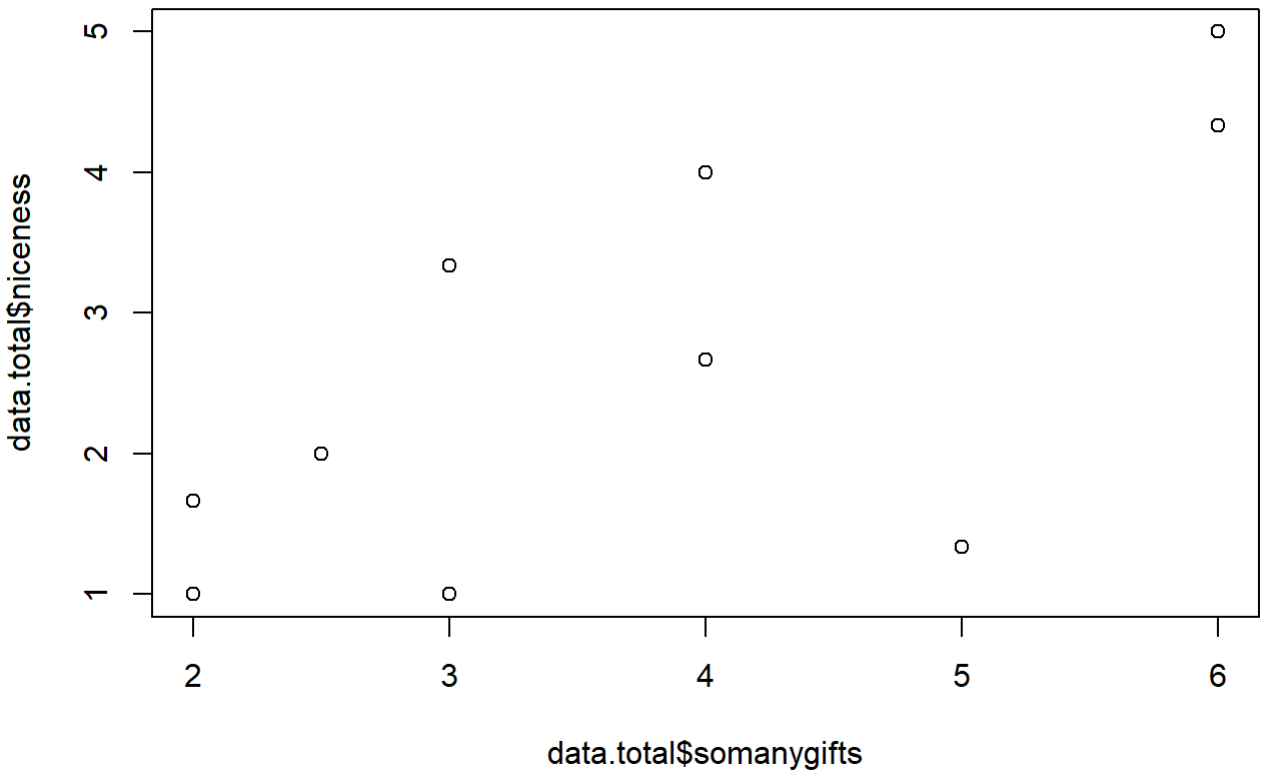
```
##
##  Pearson's product-moment correlation
##
## data:  data.total$niceness and data.total$somanygifts
## t = 4.4684, df = 11, p-value = 0.0009494
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.4519560 0.9386702
## sample estimates:
##       cor
## 0.8029819
```

Fantastic!
This output tells us several things!
The df = the number of participants.
P-value = the significance value 95% confidence interval = the range of possible correlations that exists
cor = the correlation coefficient.
In this case, we find a significant correlation between how nice you are and the amount of gifts received. #### APA write up. An APA write up might look like this. There is a positive correlation between an increased amount of perceived self-niceness and frequency of gifts received. $r = .80$, 95% CI [0.45,0.93], $p < .001$. As niceness increases reported amount of gifts also increase.

Plots

It might be hard to conceptualize the data, so instead of using our thinking brains, let's use our eyeballs. I am using the `plot` function which is a simple plot of data points.

```
plot(data.total$somanygifts, data.total$niceness)
```



##Regression So now that we find a correlation between the two. Let's ask the question: Does niceness predict overall gifts received?

We do so with the `lm` function which stands for linear model. The format is as follows. `independent variable ~ dependent variable, data` . Let's try it in action. In this particular function the `~` means "predicted by"

```
mymodel <- lm(somanygifts~niceness,data.total)
```

sweeeet let's find our more about our model by running a `summary` call on it.

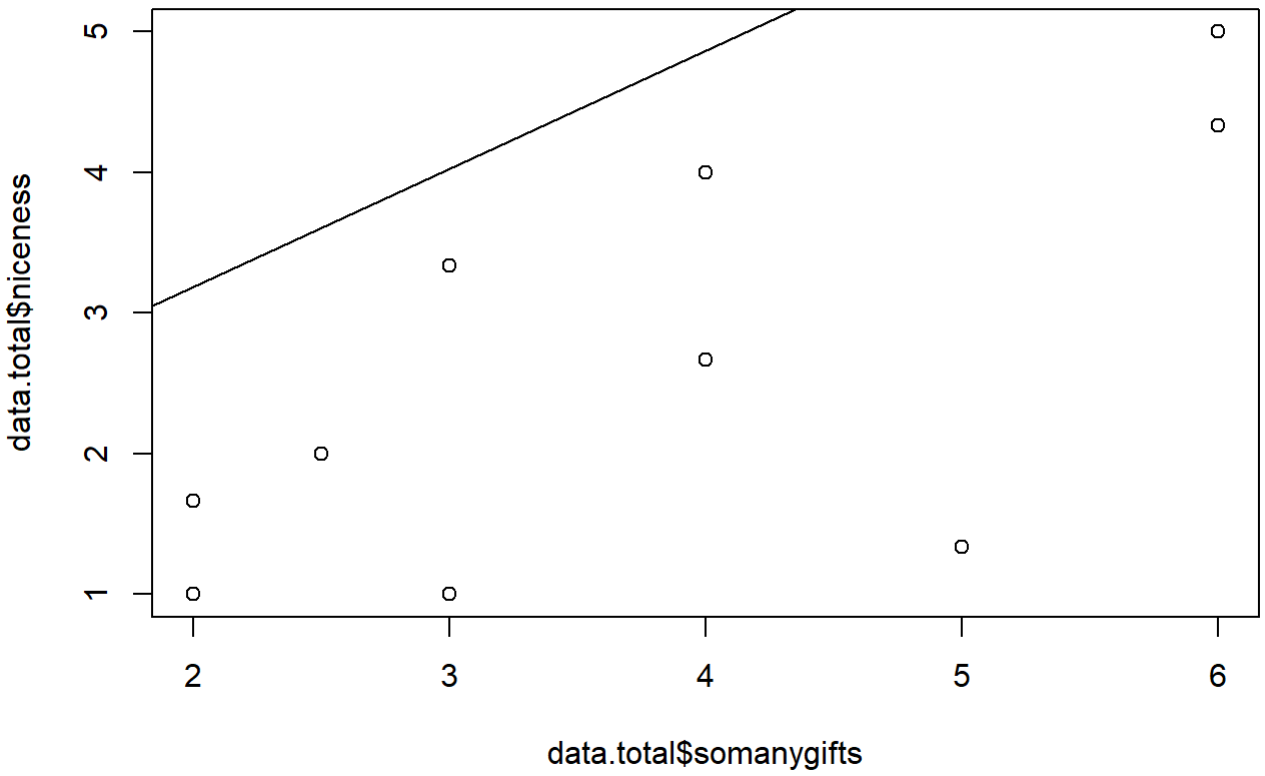
```
summary(mymodel)
```

```
##
## Call:
## lm(formula = somanygifts ~ niceness, data = data.total)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3068 -0.8678  0.2542  0.2907  2.3763
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.5017     0.6210   2.418 0.034107 *
## niceness      0.8415     0.1883   4.468 0.000949 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.036 on 11 degrees of freedom
##   (9 observations deleted due to missingness)
## Multiple R-squared:  0.6448, Adjusted R-squared:  0.6125
## F-statistic: 19.97 on 1 and 11 DF,  p-value: 0.0009494
```

note important things to notice are the estimate and significance value. Essentially (this is the APA write up), we find that niceness significantly predicts gifts. $r^2 = .64$, $F = 19.97$, $p < .001$, $b = .84$. As people are nicer, we expect them to receive more gifts.

Let's try that plot again, but this time add a `abline` on it. a abline is simply a straight line through the plot. "AB" stands for the intercept and the slope.

```
plot(data.total$somanygifts, data.total$niceness)
abline(lm(somanygifts~niceness,data.total))
```


data.total$niceness
1
2
3
4
5
2
3
4
5
6
data.total$somanygifts