# ENGE 5714 Course Notes 2021

Dr. Katz

2021-02-16

# Contents

# Preamble

This book will be a living document of notes for ENGE 5714 - Quantitative Research Methods in Engineering Education. I will try to keep it updated and post alerts about updates in our course Slack workspace.

# Chapter 1

# Week 1: Introductions

In week one we just reviewed some of the materials from the fall semester. By the end we discussed R and RStudio, but this first week was primarily about getting to know each other and the outline of the course.

# Chapter 2

# Week 2: Intro stats, Data & Distributions, Intro R & RStudio

This week, we discuss some very basic ideas related to statistics, data, and working in R.

## 2.1 First steps in R

We can create a new variable by assigning it a value with the `<-` operator. Let's create a vector of numbers 1 to 10 with the `seq()` function and then a separate vector that takes each of the x values, multiplies it by 2, and adds 3.

```
x <- seq(1:10)
y <- 2* x + 3
```

Just to make sure everything worked as expected, we can then just type `x` and `y` and R will print their values. We could also look in the "environment" window to see whether those variables (and their expected values) were actually created.

```
x
```

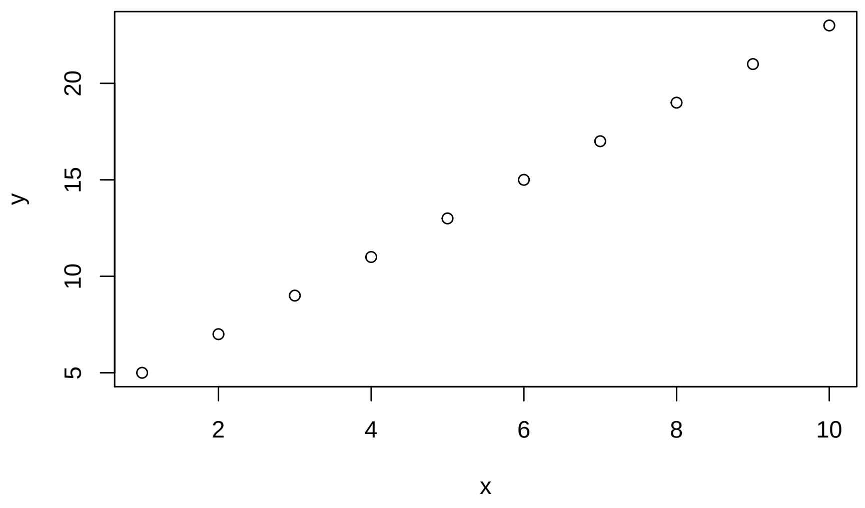
```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
y
```

```
##  [1]  5  7  9 11 13 15 17 19 21 23
```

So far, so good. If we want to quickly visualize this, we could create a simple scatter plot with the `plot()` command (note: we will come back to plotting data much more in week 3).

```
plot(x, y)
```



## 2.2 Getting your R environment set up

One of the first things you will have in any script or .rmd file is a section to load all the libraries that you use in that script.

You can install a library by using the install.packages() function, for example:

`install.packages("tidyverse")`, `install.packages("janitor")`, and `install.packages("psych")`

with this installed, you can then load the package using the library() function

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(janitor)
```

```
## Warning: package 'janitor' was built under R version 4.0.3
```

```
##
## Attaching package: 'janitor'
```

```
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library(psych)
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

## 2.3 Reading in data

A good first step when working in R is to check which directory you are working in with the `getwd()` function. You should get a directory in response.

```
getwd()
```

```
## [1] "C:/Users/akatz4/Desktop/test_course_note"
```

You can also check which files are in that directory with `list.files()`.

```
list.files()
```

```
##  [1] "_book"
##  [2] "_bookdown.yml"
##  [3] "_bookdown_files"
##  [4] "_output.yml"
##  [5] "01-Week_01.Rmd"
##  [6] "02-Week_02.Rmd"
##  [7] "03-Week_03.Rmd"
##  [8] "04-Week_04.Rmd"
##  [9] "05-Week_05.Rmd"
## [10] "06-Week_06.Rmd"
## [11] "book.bib"
## [12] "docs"
## [13] "ENGE_5714_2021_pre_survey.csv"
## [14] "Free Reduced Lunch by Schools and Grade Structures 2008-2017_final.csv"
## [15] "index.Rmd"
## [16] "packages.bib"
## [17] "preamble.tex"
## [18] "README.md"
## [19] "RExam.dat"
## [20] "seniorsurvey.csv"
## [21] "style.css"
## [22] "survey_student_info.csv"
## [23] "test_course_note.Rproj"
## [24] "test_course_notes.Rmd"
## [25] "test_course_notes_files"
```

If you notice that the file you are looking for is not there, then you can use setwd() to change your working directory

```
setwd("./Week 2/")
```

After that, make sure you have switched to the correct working directory `getwd()` and then `list.files()`.

Assuming you have directed yourself to the correct place, you can now read in the file(s) that you want to be working with. There are a *lot* of ways to do this. Since we will be spending a lot of time in class working with .csv files, we will focus on using the `read_csv()` function from the readr package (part of the tidyverse collection of packages). This function will read in the .csv file and store the data as a tibble (a tidyverse version of a data frame, which we can think of as a collection of observations stored in rows with values for variables for each observation stored in columns).

```
prior_survey <- read_csv("ENGE_5714_2021_pre_survey.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   student_id = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

## 2.4 Exploring the data

Now that we have loaded in the data, let's take a look at the csv. If we just run a line with the name of the tibble - i.e., `prior_survey` then we should receive a printout that shows the first several rows of that tibble and a listing of all the columns, along with the data types (i.e., double for numeric values, character for strings, etc) of each column.

```
prior_survey
```

```
## # A tibble: 24 x 49
##    student_id `I have taken a~ `I am intereste~ `I know what a ~
##         <dbl> <chr>            <chr>            <chr>
##  1          1 Somewhat disagr~ Somewhat agree   Strongly disagr~
##  2          2 Strongly disagr~ Neither agree n~ Somewhat agree
##  3          3 Strongly disagr~ Somewhat agree   Somewhat agree
##  4          4 Somewhat disagr~ Strongly agree   Strongly disagr~
##  5          5 Somewhat agree   Strongly agree   Somewhat agree
##  6          6 Somewhat disagr~ Somewhat agree   Somewhat disagr~
##  7          7 Strongly disagr~ Somewhat agree   Strongly disagr~
##  8          8 Somewhat agree   Somewhat agree   Somewhat agree
##  9          9 Strongly disagr~ Strongly agree   Somewhat agree
## 10         10 Neither agree n~ Strongly agree   Somewhat agree
## # ... with 14 more rows, and 45 more variables: `I know what a type II error
## #   is` <chr>, `I know what a (statistical) confidence level is` <chr>, `I know
## #   what a p value is` <chr>, `I know what p-hacking means` <chr>, `I know what
## #   statistical power means` <chr>, `I have heard of frequentist statistics
## #   before` <chr>, `I have heard of Bayesian statistics before` <chr>, `I have
## #   heard the term "parametric statistics" before` <chr>, `I have heard the
## #   term "non-parametric statistics" before` <chr>, `I know what a histogram
## #   is.` <chr>, `I know what a probability distribution is.` <chr>, `I know
## #   what a random variable is.` <chr>, `I know what a probability distribution
## #   function is.` <chr>, `I know what a cumulative distribution function
## #   is.` <chr>, `I know what the expectation of a random variable is.` <chr>,
## #   `I know how to calculate the variance of a random variable.` <chr>, `I know
## #   what a z score is.` <chr>, `I know how to calculate the correlation between
## #   two variables.` <chr>, `I know how to interpret the correlation coefficient
## #   between two variables` <chr>, `I have heard of linear regression` <chr>, `I
## #   know how to run a linear regression (in some software...or by hand, if I'm
## #   feeling wild).` <chr>, `I know how to interpret a linear
## #   regression.` <chr>, `I have heard of multiple regression` <chr>, `I know
## #   how to perform a multiple regression` <chr>, `I know how to interpret a
## #   multiple regression` <chr>, `I have heard of logistic regression.` <chr>,
## #   `I understand when to use a logistic regression.` <chr>, `I know how to
## #   interpret the results of a logistic regression` <chr>, `I have heard of
## #   t-tests` <chr>, `I have performed a t-test before` <chr>, `I know how to
## #   interpret the results of a t-test` <chr>, `I have heard of Analysis of
## #   Variance.` <chr>, `I understand when to run an Analysis of Variance
## #   (ANOVA)` <chr>, `I know how to interpret the results from an ANOVA` <chr>,
## #   `I have heard of a chi-square test` <chr>, `I have used a chi-square test
## #   before` <chr>, `I know how to interpret the results of a chi-square
## #   test` <chr>, `I have heard of cluster analysis before` <chr>, `I have used
## #   cluster analysis before` <chr>, `I know how to interpret the results of a
## #   cluster analysis` <chr>, `I have heard of factor analysis (either
## #   exploratory or confirmatory)` <chr>, `I have used factor analysis (either
## #   exploratory or confirmatory)` <chr>, `I know how to interpret the results
## #   of a factor analysis (either exploratory or confirmatory)` <chr>, `I
## #   already have R and Rstudio downloaded to my computer.` <chr>, `I have used
## #   R before` <chr>
```

When we do this, we see that there are a bunch of columns that have spaces in their names. This is okay (in the sense that R can handle this), but it can be a little frustrating to work with. Let's try cleaning the column names with `clean_names()` from the janitor package. This function will replace the spaces in the column names with underscores and make everything lower case. So, a column name like "I have take a statistics course before" will be changed to "i_have_taken_a_statistics_course_before".

```
prior_survey <- prior_survey %>% clean_names() # from janitor package
```

Look at the data in prior_survey again and see if anything looks different (hint: it should).

```
prior_survey
```

```
## # A tibble: 24 x 49
##    student_id i_have_taken_a_~ i_am_interested~ i_know_what_a_t~
##         <dbl> <chr>            <chr>            <chr>
##  1          1 Somewhat disagr~ Somewhat agree   Strongly disagr~
##  2          2 Strongly disagr~ Neither agree n~ Somewhat agree
##  3          3 Strongly disagr~ Somewhat agree   Somewhat agree
##  4          4 Somewhat disagr~ Strongly agree   Strongly disagr~
##  5          5 Somewhat agree   Strongly agree   Somewhat agree
##  6          6 Somewhat disagr~ Somewhat agree   Somewhat disagr~
##  7          7 Strongly disagr~ Somewhat agree   Strongly disagr~
##  8          8 Somewhat agree   Somewhat agree   Somewhat agree
##  9          9 Strongly disagr~ Strongly agree   Somewhat agree
## 10         10 Neither agree n~ Strongly agree   Somewhat agree
## # ... with 14 more rows, and 45 more variables:
## #   i_know_what_a_type_ii_error_is <chr>,
## #   i_know_what_a_statistical_confidence_level_is <chr>,
## #   i_know_what_a_p_value_is <chr>, i_know_what_p_hacking_means <chr>,
## #   i_know_what_statistical_power_means <chr>,
## #   i_have_heard_of_frequentist_statistics_before <chr>,
## #   i_have_heard_of_bayesian_statistics_before <chr>,
## #   i_have_heard_the_term_parametric_statistics_before <chr>,
## #   i_have_heard_the_term_non_parametric_statistics_before <chr>,
## #   i_know_what_a_histogram_is <chr>,
## #   i_know_what_a_probability_distribution_is <chr>,
## #   i_know_what_a_random_variable_is <chr>,
## #   i_know_what_a_probability_distribution_function_is <chr>,
## #   i_know_what_a_cumulative_distribution_function_is <chr>,
## #   i_know_what_the_expectation_of_a_random_variable_is <chr>,
## #   i_know_how_to_calculate_the_variance_of_a_random_variable <chr>,
## #   i_know_what_a_z_score_is <chr>,
## #   i_know_how_to_calculate_the_correlation_between_two_variables <chr>,
## #   i_know_how_to_interpret_the_correlation_coefficient_between_two_variables <chr>,
## #   i_have_heard_of_linear_regression <chr>,
## #   i_know_how_to_run_a_linear_regression_in_some_software_or_by_hand_if_im_feeling_wild <chr
## #   i_know_how_to_interpret_a_linear_regression <chr>,
## #   i_have_heard_of_multiple_regression <chr>,
## #   i_know_how_to_perform_a_multiple_regression <chr>,
## #   i_know_how_to_interpret_a_multiple_regression <chr>,
## #   i_have_heard_of_logistic_regression <chr>,
## #   i_understand_when_to_use_a_logistic_regression <chr>,
## #   i_know_how_to_interpret_the_results_of_a_logistic_regression <chr>,
## #   i_have_heard_of_t_tests <chr>, i_have_performed_a_t_test_before <chr>,
## #   i_know_how_to_interpret_the_results_of_a_t_test <chr>,
## #   i_have_heard_of_analysis_of_variance <chr>,
## #   i_understand_when_to_run_an_analysis_of_variance_anova <chr>,
## #   i_know_how_to_interpret_the_results_from_an_anova <chr>,
## #   i_have_heard_of_a_chi_square_test <chr>,
## #   i_have_used_a_chi_square_test_before <chr>,
## #   i_know_how_to_interpret_the_results_of_a_chi_square_test <chr>,
## #   i_have_heard_of_cluster_analysis_before <chr>,
## #   i_have_used_cluster_analysis_before <chr>,
## #   i_know_how_to_interpret_the_results_of_a_cluster_analysis <chr>,
## #   i_have_heard_of_factor_analysis_either_exploratory_or_confirmatory <chr>,
## #   i_have_used_factor_analysis_either_exploratory_or_confirmatory <chr>,
## #   i_know_how_to_interpret_the_results_of_a_factor_analysis_either_exploratory_or_
## #   i_already_have_r_and_rstudio_downloaded_to_my_computer <chr>,
## #   i_have_used_r_before <chr>
```

One other function that will we see more in the future is the `table()` function, which will create a table with the counts of the values for a variable. For example, if we wanted to quickly know how students answered the "I have taken a quantitative research methods course before" question, we can run the following:

```
table(prior_survey$i_have_taken_a_quantitative_research_methods_course_before)
```

```
##
## Neither agree nor disagree             Somewhat agree
##                          2                          5
##          Somewhat disagree             Strongly agree
##                          5                          2
##          Strongly disagree
##                         10
```

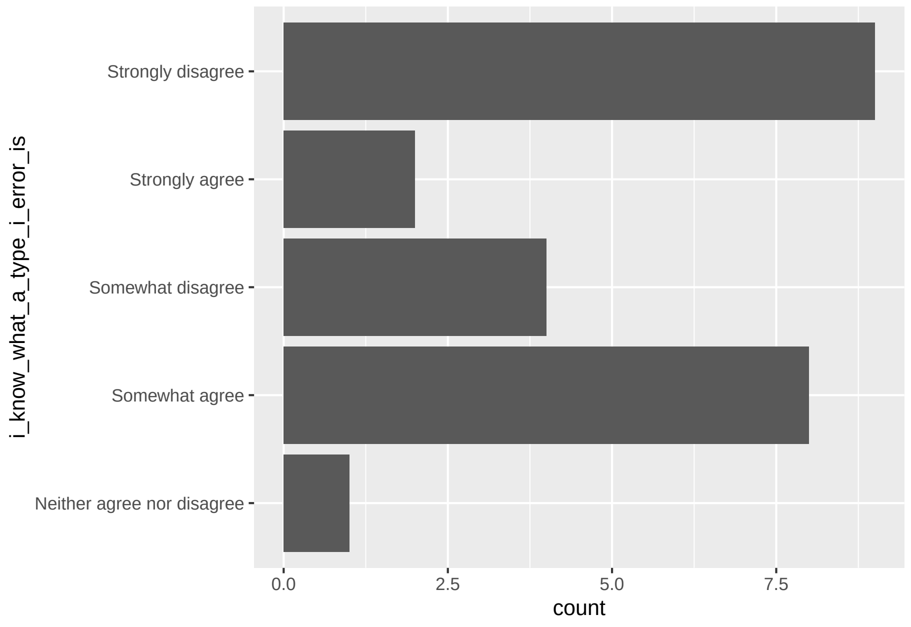
## 2.5 Plotting data

We will discuss plotting more next week, but here is a brief preview of what's to come...

There are multiple ways to plot data. Focusing on using ggplot, here are two.

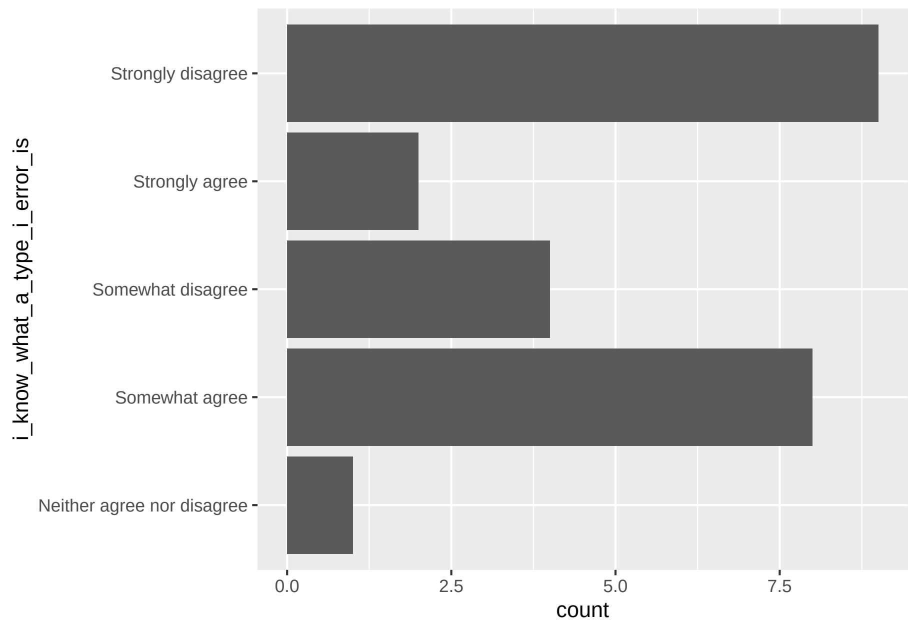
The first way passes the prior_survey dataframe explicitly to ggplot

```
ggplot(data = prior_survey, mapping = aes(x = i_know_what_a_type_i_error_is)) +
  geom_bar() +
  coord_flip()
```



The second way does this implicitly, using the pipe operator. Note that the results should be the same.

```
prior_survey %>%
  ggplot(mapping = aes(x = i_know_what_a_type_i_error_is)) +
  geom_bar() +
  coord_flip()
```



If we wanted to get extra fancy, we could first convert the data from a wide format to a long format and then start plotting all the items together.

Converting to long format would produce something like this:

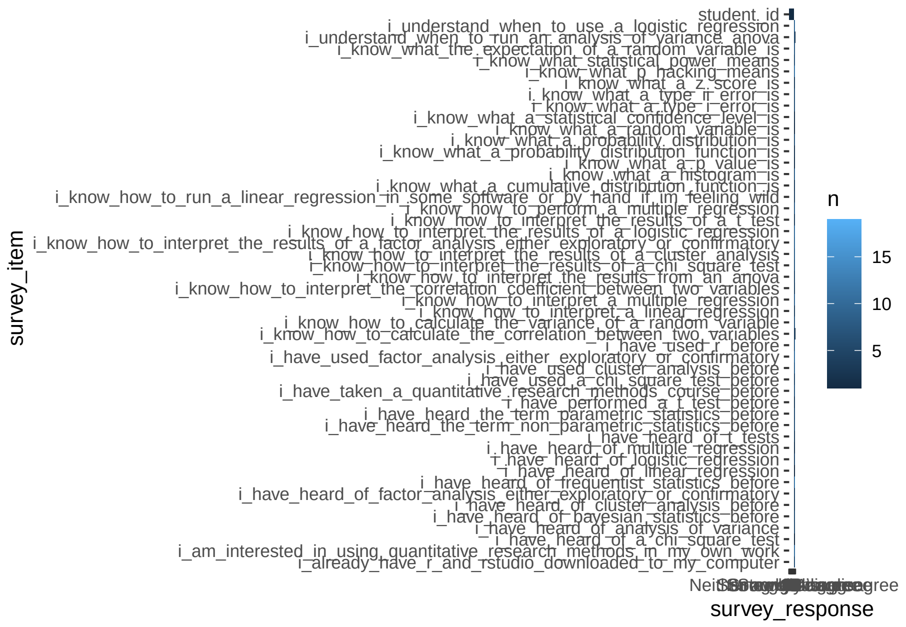
```
prior_survey %>%
  gather(key = "survey_item", value = "survey_response")
```

```
## # A tibble: 1,176 x 2
##    survey_item survey_response
##    <chr>       <chr>
##  1 student_id  1
##  2 student_id  2
##  3 student_id  3
##  4 student_id  4
##  5 student_id  5
##  6 student_id  6
##  7 student_id  7
##  8 student_id  8
##  9 student_id  9
## 10 student_id  10
## # ... with 1,166 more rows
```

Then we can combine that with the `group_by()` and `summarize()` functions and plot the results.

```r
prior_survey %>%
  gather(key = "survey_item", value = "survey_response") %>%
  group_by(survey_item, survey_response) %>%
  summarize(n = n()) %>%
  ggplot(mapping = aes(x = survey_response, y = survey_item, fill = n)) +
  geom_tile()
```

```
## `summarise()` regrouping output by 'survey_item' (override with `.groups` argument)
```



This plot is okay for giving a general sense of what is going on in these plots but there are a bunch of other ways to go about doing this.

First, maybe we want to rename the response categories to a numerical scale. We can accomplish this with a `mutate()` and `case_when()`.

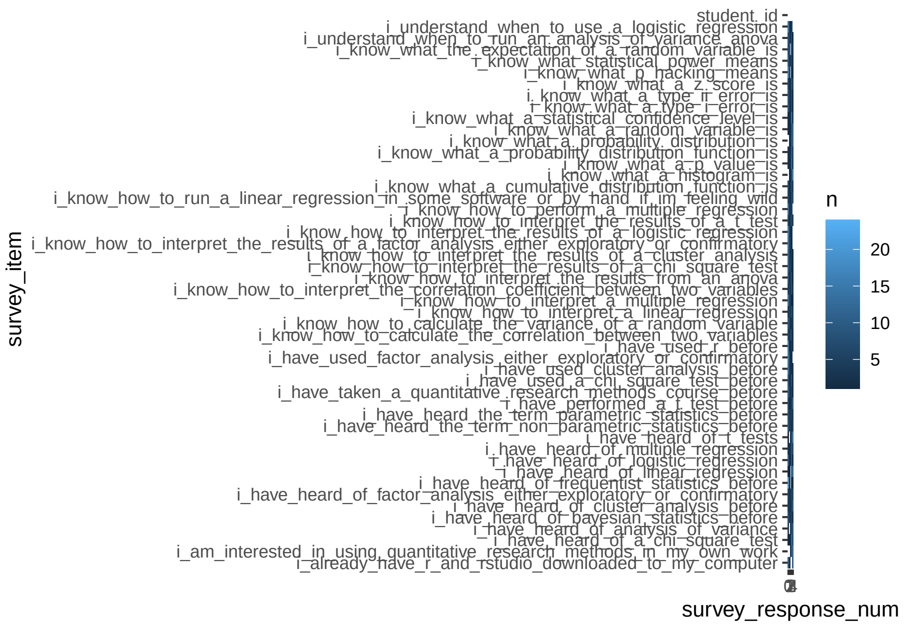
```r
prior_survey <- prior_survey %>%
  gather(key = "survey_item", value = "survey_response") %>%
  mutate(survey_response_num = case_when(survey_response == "Strongly disagree" ~ 0,
                                         survey_response == "Somewhat disagree" ~ 1,
                                         survey_response == "Neither agree nor disagree" ~ 2,
                                         survey_response == "Somewhat agree" ~ 3,
                                         survey_response == "Strongly agree" ~ 4,
                                         ))
```

Then we plot the same data but with the numerical scale along the x-axis.

```
prior_survey %>%
  group_by(survey_item, survey_response_num) %>%
  summarize(n = n()) %>%
  ggplot(mapping = aes(x = survey_response_num, y = survey_item, fill = n)) +
  geom_tile()
```

```
## `summarise()` regrouping output by 'survey_item' (override with `.groups` argument)
```

```
## Warning: Removed 3 rows containing missing values (geom_tile).
```

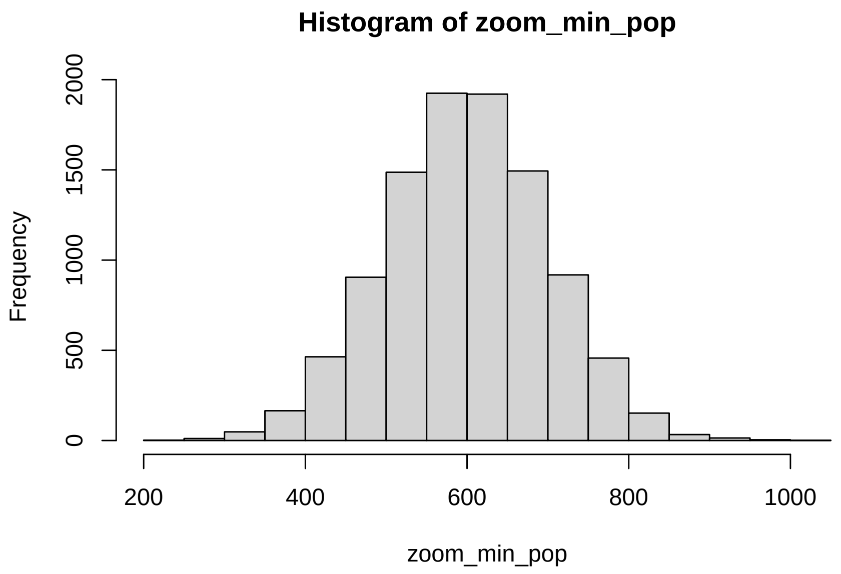


## 2.6 Some brief stats

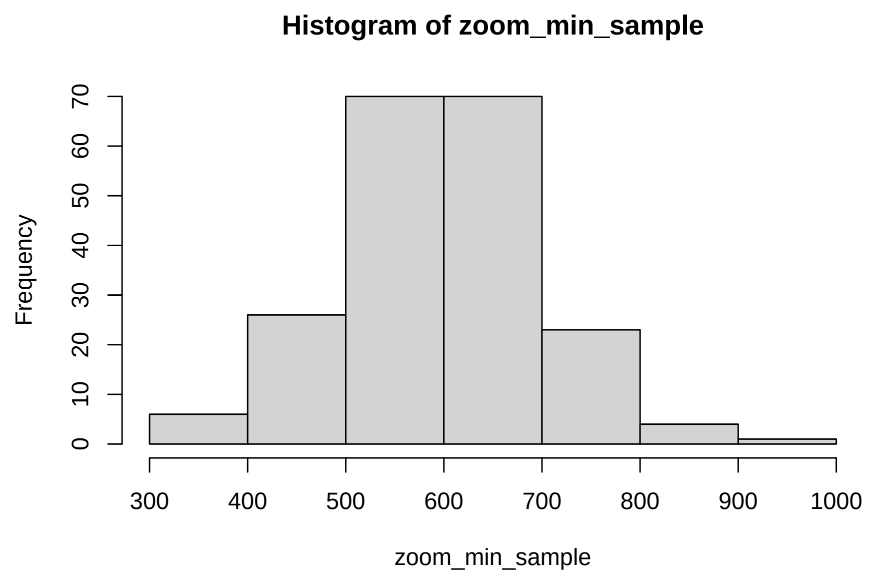
In this week's reading, there was also discussion about standard errors and the central limit theorem. These are fairly important theoretical concepts to grasp. To some extent they deal with the scenario where you go out and repeatedly sample from a population and calculate a statistic from each of those samples. The distributions *of that statistic* is what we will call the sampling distribution (as opposed to the sample distribution, which would more accurately describe the distribution of the data that we get in any one sample that we draw from the population).

### 2.6.1 Central Limit Theorem and Standard Error Demo —-

```
pop_students <- 10000

zoom_min_pop <- rnorm(n = pop_students, mean = 600, sd = 100)

hist(zoom_min_pop)
```



```
zoom_min_sample <- sample(x = zoom_min_pop,
                          size = 200,
                          replace = FALSE)


hist(zoom_min_sample)
```



```
mean(zoom_min_sample)
```

```
## [1] 599.6767
```

```
sd(zoom_min_sample)
```

```
## [1] 100.8203
```

As a brief aside, let's review the idea of a loop

```
num_reps <- 100

data_vec <- rep(NA, num_reps) # this creates an empty vector of size num_reps with NA

# this loops through the vector starting at position 1 and ending at the final positio

for (i in 1:num_reps){
  data_vec[i] <- i
}
```

Okay, so that's how we create an empty vector and how we loop through its different entries. For this demo, we will also need to remenber how to generate random numbers from a norm distribution with a specified mean and standard deviation.

```
rnorm(n = 10, mean = 5, sd = 2) # n is the number of random numbers we draw from this normal dis
```

```
##  [1] 6.087313 5.135112 4.883394 3.267231 5.729493 2.639171 4.271634 5.450025
##  [9] 0.766896 6.309798
```

Okay, so that's not bad. Now, that command will produce a vector with 10 random numbers. We can calculate the mean and standard deviation of those 10 numbers (which should be close to the values that we specified in `rnorm()` with the `mean()` and `sd()` functions.

```
mean(rnorm(n = 10, mean = 5, sd = 2))
```

```
## [1] 4.627293
```

```
sd(rnorm(n = 10, mean = 5, sd = 2))
```

```
## [1] 1.63701
```

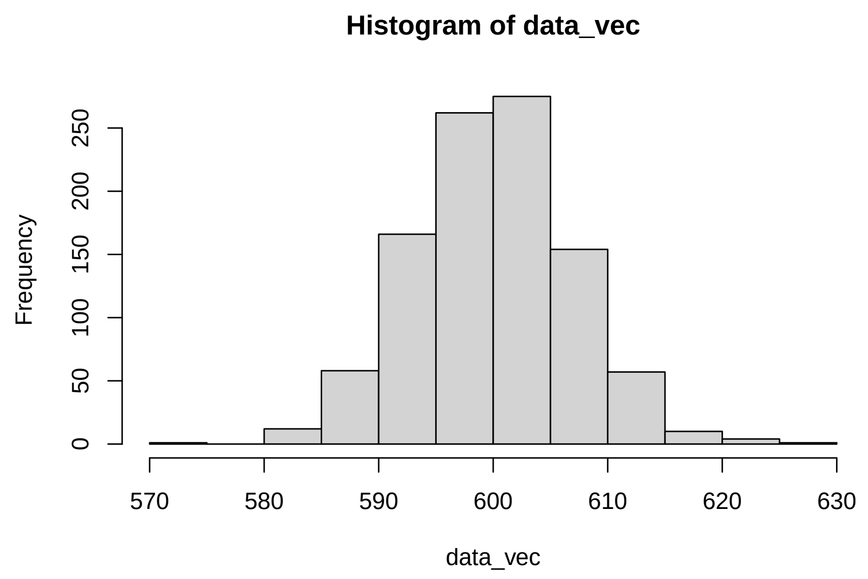
Next, let's act as if we are drawing a certain sample of size `samp_size` of data points for `num_reps` number of times. Keep in mind that, in practice, when we are collecting data ourselves in our own research, num_reps will almost always be 1. We are just demonstrating the underlying assumptions for how we can calculate some of the statistics that we use.

```
num_reps <- 1000 # specify how many times to take a sample
samp_size <- 200 # specify the size of each sample
data_vec <- rep(NA, num_reps) # create an empty vector of size num_reps with NA in each entry.
for (i in 1:num_reps){
  data_vec[i] <- mean(rnorm(n = samp_size, mean = 600, sd = 100)) # store the mean of each of the
}
```

With this, we have a vector `data_vec` of size `num_reps` with the mean of each of our samples that we drew. This vector contains our sampling distribution of our sample means. **NOTE**: The standard deviation of this sampling mean is what we are calling our *standard error*.

We can plot a histogram of this sampling distribution and calculate the standard deviation of the sampling mean.

```
hist(data_vec)
```



```
sd(data_vec)
```

```
## [1] 6.865372
```

On your own, try copying this code and changing the num_reps and sample_size variables to larger and smaller values. Focus on how the x-axis values in your histogram change when you change the num_reps and samp_size variables.

Hint: CLT will explain the normal distribution of the sampling mean (the shape you see in the histogram) while the Weak Law of Large Numbers will explain the concentration around the true mean as samp_size increases (i.e., when we draw a larger sample size from the population, our sample mean gets closer to the population mean).

```
## Quick note on the rep() function: notice what happens when you specify "each" vs "t
rep(c(1, 2), times = 5)
```

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

```
rep(c(1, 2), each = 5)
```

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

# Chapter 3

# Week 3: Data Cleaning, Organizing, Describing, and Communicating

This week we focus on different steps you will often take when you first start working with your data. These tend to fall under the umbrella of "data processing" and often need to happen before you can start doing any kind of analysis.

## 3.1 Visualizing your data

Once your data have been clearned, you are ready to start visualizing what you are working with. There is a huge range of what you can do with these plots. That's great! On the other hand, it can quickly start to feel overwhelming. To help get this under control and make it more manageable, it is convenient to think about the *types* of data that you have. In particular, are your variables nominal, ordinal, interval, or ratio variables?

### 3.1.1 One continuous variable (either predictor or outcome variable)

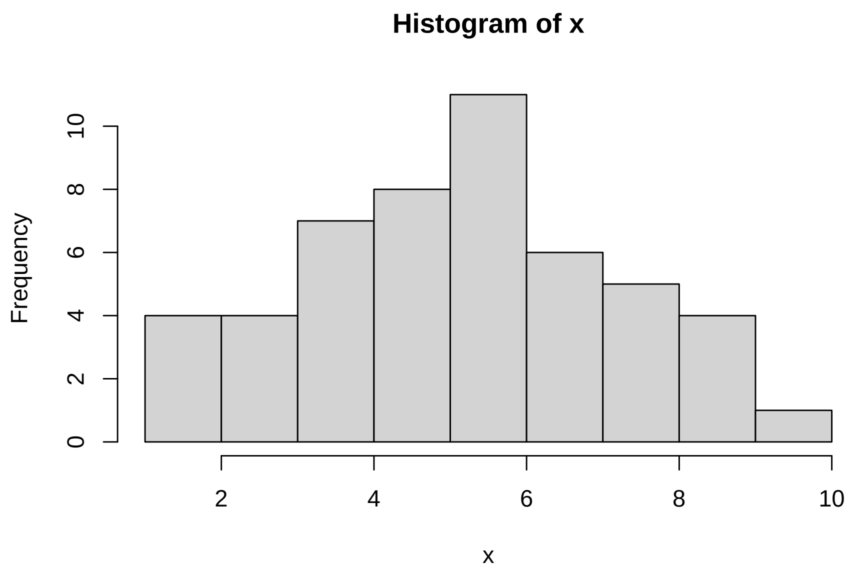
When you have one continuous variable, a standard option is to plot a histogram. These are plots that show the frequency of each of the values that the variable takes. Oftentimes it is helpful to create bins of values so that any number that falls in the 0-4 range counts in one bin, numbers from 5-9 are in a second bin, and so on.

For this example, we will start by generating some data using `rnorm()`, which generates a random number (or in our case, `num` numbers) from a normal distribution with mean `mu` and standard deviation `stdev`.

```
num <- 50
mu <- 5
stdev <- 2

x <- rnorm(n = num, mean = mu, sd = stdev)
```

With these data generated, we can then quickly plot the histogram with `hist()`. This will use base R graphics.

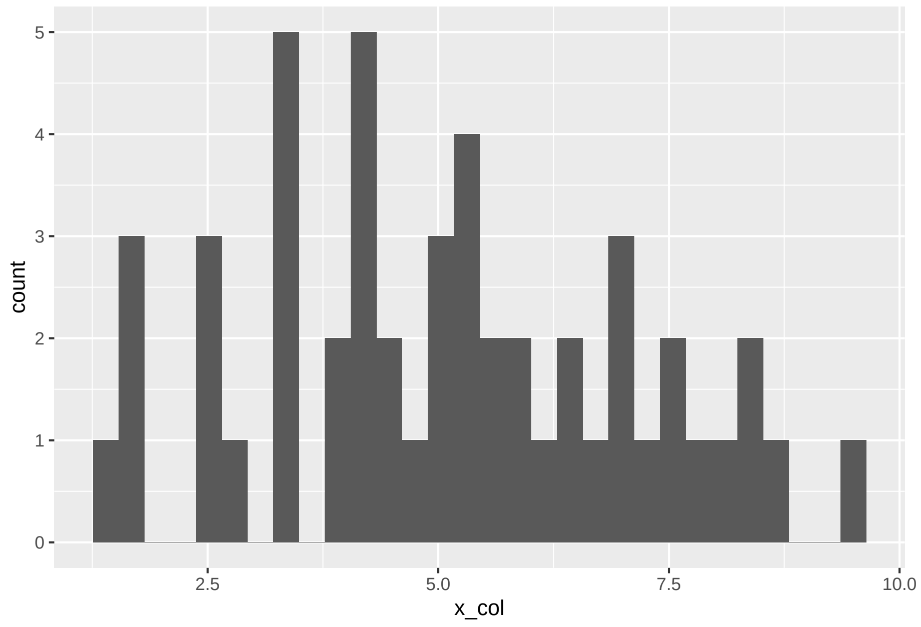
```
hist(x)
```



You can also do this using ggplot rather than base R graphics.

```
x_df <- tibble(x_col = x)

ggplot(data = x_df, mapping = aes(x = x_col)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

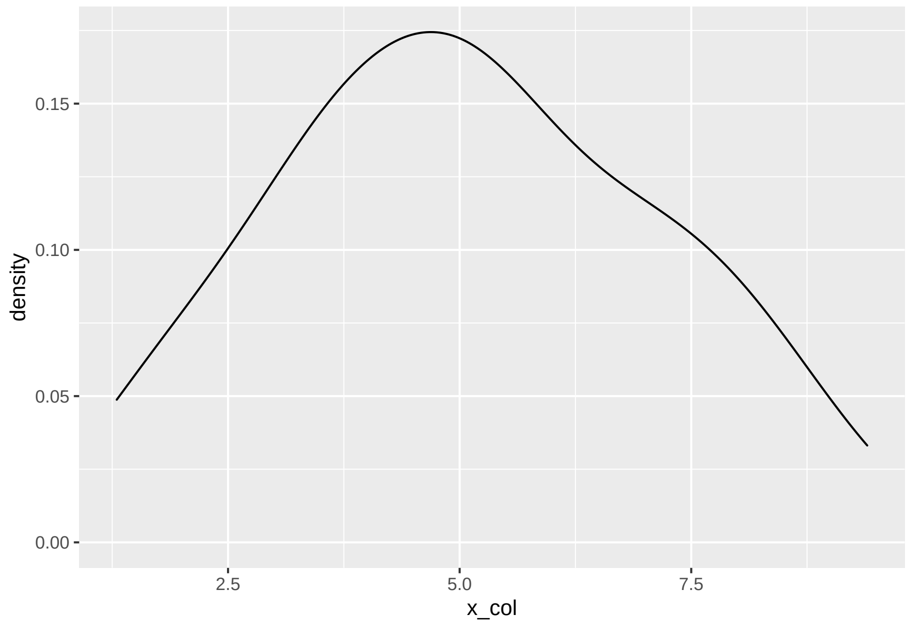


The histogram is a very standard plot, and you should consider it a go-to option in your toolkit. Alternatively, you can use geom_density() instead of geom_histogram() to get a smooth graph rather than one with discrete bins. We will use the same data that we generated before.

We will write this two ways to demnostrate how the pipe `%>%` operator works.

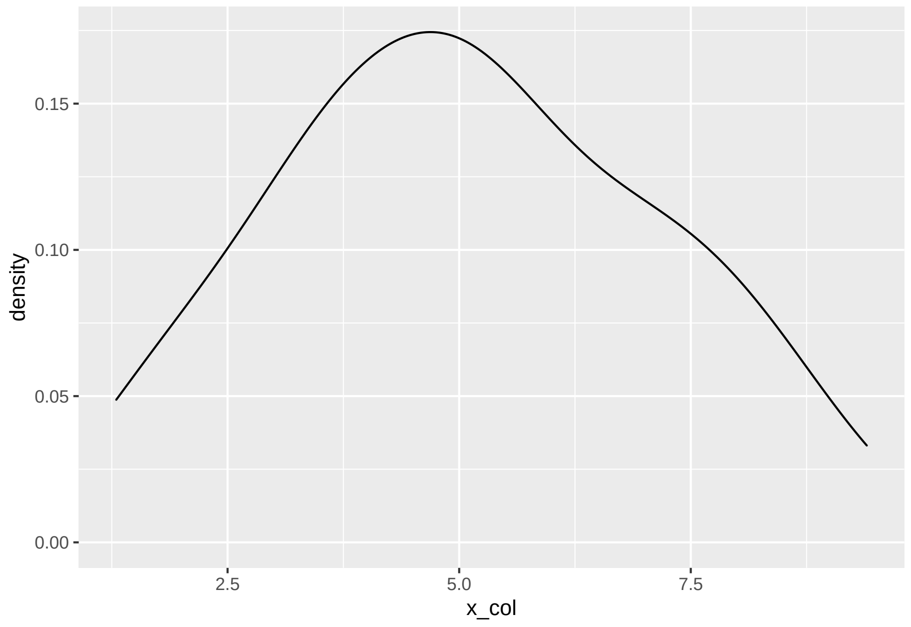
First way:

```
ggplot(data = x_df, mapping = aes(x = x_col)) +
  geom_density()
```



Second way:

```
x_df %>%
  ggplot(mapping = aes(x = x_col)) +
  geom_density()
```



Just for fun, look at what happens to the the plot if you increase the sample size

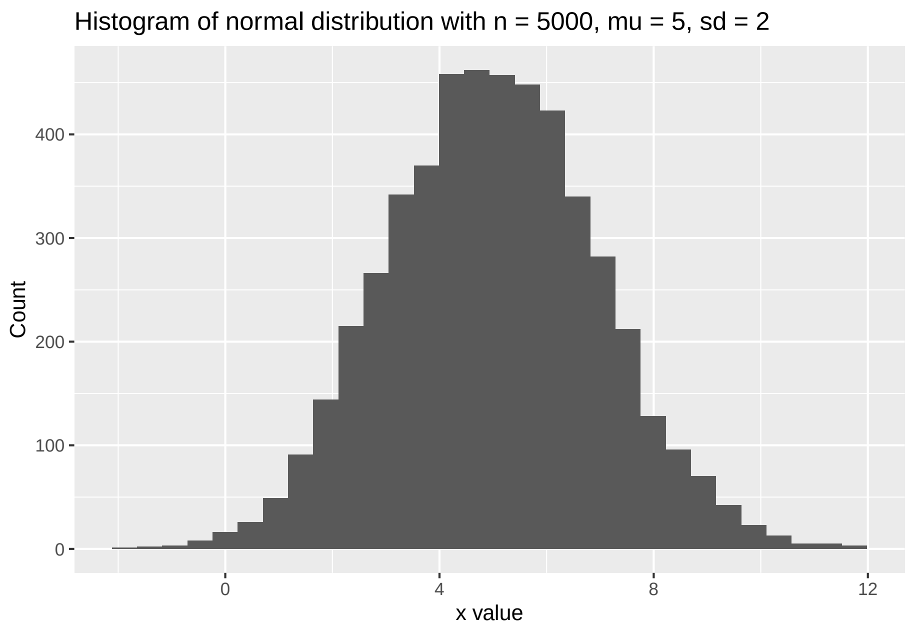
First, we will generate the data with a sample size of 5,000 rather than 50.

```
num <- 5000
mu <- 5
stdev <- 2

x <- rnorm(n = num, mean = mu, sd = stdev)
x_df <- tibble(x_col = x)
```

Then we will plot the histogram

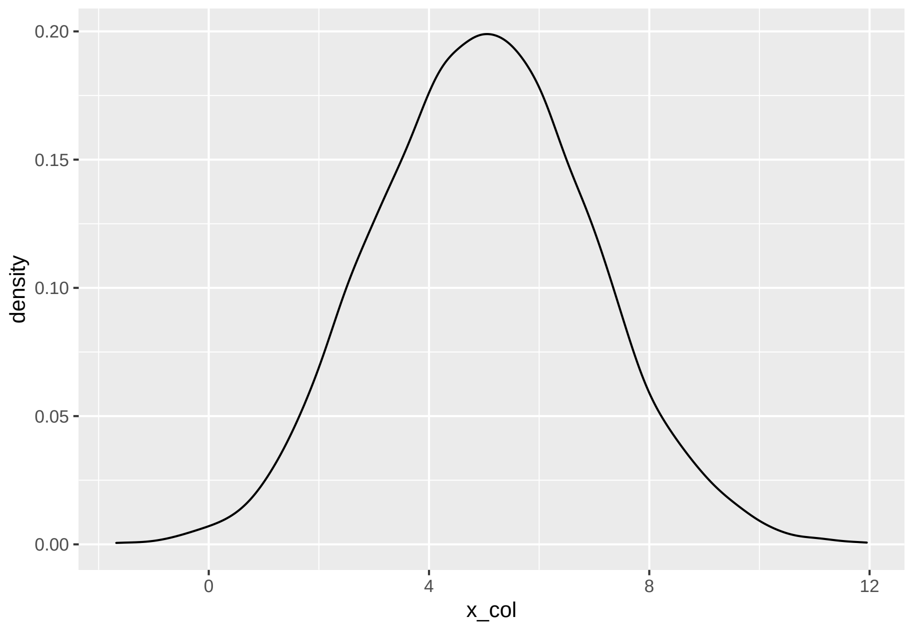
```
x_df %>%
  ggplot(aes(x = x_col)) +
  geom_histogram() +
  labs(x = "x value",
       y = "Count",
       title = "Histogram of normal distribution with n = 5000, mu = 5, sd = 2")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



And, finally, we can make the density plot instead of the histogram, if that's our jam.

```
ggplot(data = x_df, mapping = aes(x = x_col)) +
  geom_density()
```



### 3.1.2 One Discrete Variable (either predictor or outcome)

What if instead of a continuous (i.e., interval or ratio) variable we have a discrete variable such as a nominal (e.g., major, university) or ordinal (e.g., Likert scale item, level of education) variable? For that we can use something like `geom_bar()` or `geom_col()` to plot the counts of observations within each of those categories.

To demonstrate this, we first need some data to work with. We will use the pre-semester, prior knowledge survey that everyone took. I have combined this year's results with last year's results in order to increase the sample size. After reading in the data, I will also use the `clean_names()` function from the `janitor` package.

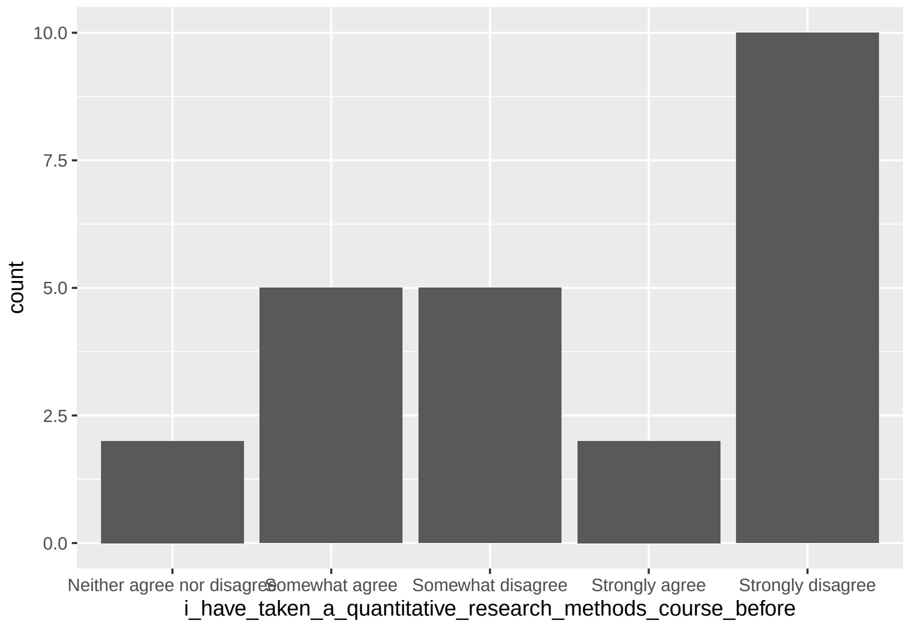
```
## load in the data
survey_df <- read_csv("ENGE_5714_2021_pre_survey.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   student_id = col_double()
## )

## See spec(...) for full column specifications.
```

```
survey_df <- survey_df %>% clean_names()
```

Next, we can go ahead and make a bar plot with the following code:

```
survey_df %>%
  ggplot(aes(x = i_have_taken_a_quantitative_research_methods_course_before)) +
  geom_bar()
```



Notice that the ordering is not quite what we would want. It is alphabetical. Let's try to fix this.

Here is one way: we first specify the levels of that variable (i.e., the different values that it could take) and store that in the variable `q_levels`. Then, we pass that to the `factor()` function, which will tell R that we want whichever variable is passed to `factor()` two things. First, it will say that we want to make that variable a factor variable with `levels = ...`. Second, we set `ordered = TRUE` to tell R that there is a specific ordering to that variable. This way, whenever there is something like a plot that we make, the ordering will persist in the labeling and R will not show the labels in alphabetical order.

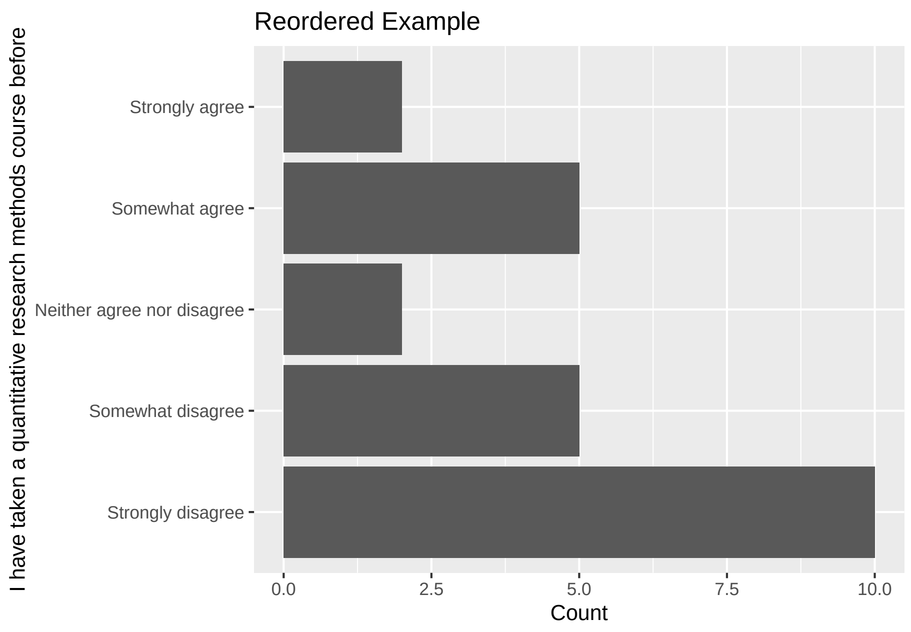
Here is an example of that in action:

```
q_levels <- c("Strongly disagree", "Somewhat disagree", "Neither agree nor disagree",
              "Somewhat agree", "Strongly agree")
```

```
survey_df$i_have_taken_a_quantitative_research_methods_course_before <- factor(survey_df$i_have_t
                                                                                levels = q_levels,
                                                                                ordered = TRUE)
```

Now try plotting these data. We will also add in a `coord_flip()` to plot the categories along the y-axis. This is a common move to avoid text from the different levels overlapping with each other. Finally, we will also change the x, y, and title labels with `labs()`.

```
survey_df %>%
  ggplot(aes(x = i_have_taken_a_quantitative_research_methods_course_before)) +
  geom_bar() +
  coord_flip() +
  labs(x = "I have taken a quantitative research methods course before",
       y = "Count",
       title = "Reordered Example")
```



## 3.2 Joining two datasets

Let's imagine that we have a separate dataset that has information about the students who completed the pre-course prior knowledge survey.

First, we will load in that dataset

```
survey_info_df <- read_csv("survey_student_info.csv")
```

```
## Parsed with column specification:
## cols(
##   student_id = col_double(),
##   standing = col_character(),
##   college = col_character(),
##   required = col_character()
## )
```

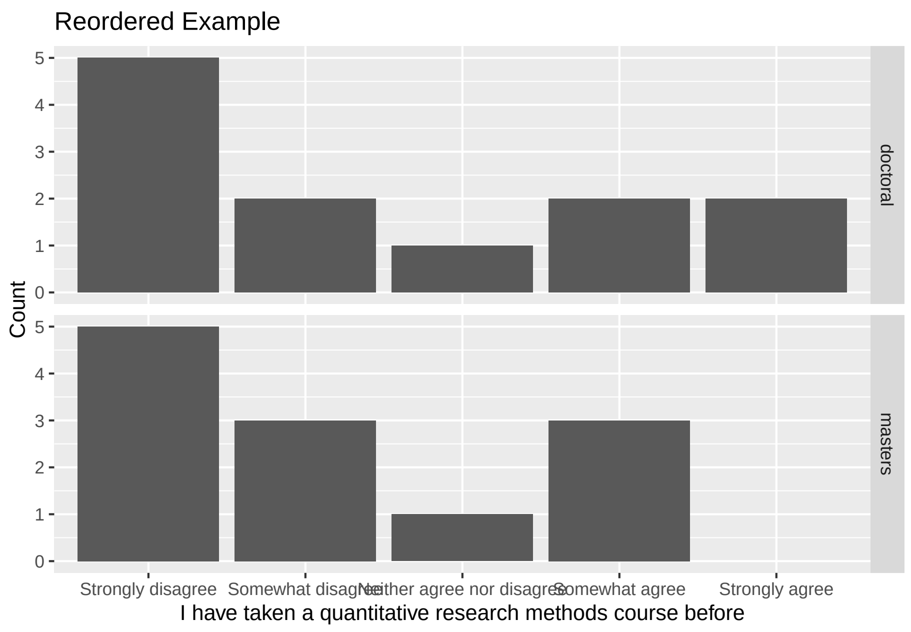
Next, let's join the two datasets based on the student id column, which is in each of the two dataframes.

```
survey_df <- survey_df %>% inner_join(survey_info_df, by = "student_id")
```

Now we should have both datasets joined into one and saved as survey_df.

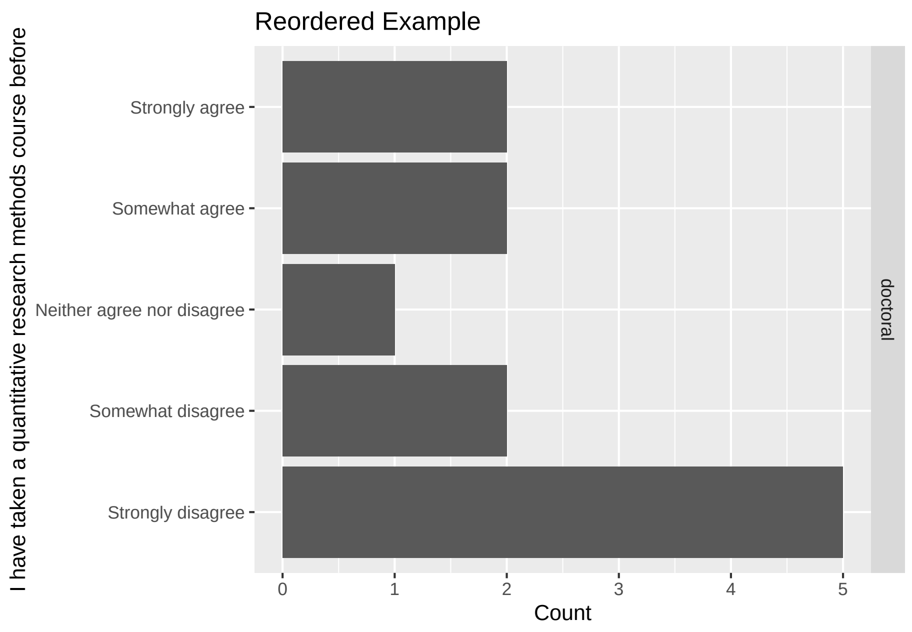
With this, we can make some nicer plots and do something like use facet_grid() to look at students who are masters and doctoral students, for example.

```
survey_df %>%
  ggplot(aes(x = i_have_taken_a_quantitative_research_methods_course_before)) +
  geom_bar() +
  facet_grid(standing ~.) +
  labs(x = "I have taken a quantitative research methods course before",
       y = "Count",
       title = "Reordered Example")
```



The x axis looks a little crowded. What if we try `coord_flip()`?

```
survey_df %>%
  filter(standing == "doctoral") %>%
  ggplot(aes(x = i_have_taken_a_quantitative_research_methods_course_before)) +
  geom_bar() +
  coord_flip() +
  facet_grid(standing ~.) +
  labs(x = "I have taken a quantitative research methods course before",
       y = "Count",
       title = "Reordered Example")
```



That looks much better.

### A quick note on filters

If you want to look at only a subset of your data, you will want to use the `filter()` function. The general idea is that you can look at observations (rows) that match a certain criteria. For example, you may want to only look at students from a certain region or year or major. In our case, with the prior knowledge survey, let's say we only want to look at student who have to take the course (i.e., there is a "yes" for them for the `required` variable). We can do that with the first line. The second line just stores the result as a new dataframe called `filtered_df`.

```
survey_df %>% filter(required == "yes")
```

```
## # A tibble: 12 x 52
##    student_id i_have_taken_a_~ i_am_interested~ i_know_what_a_t~
##         <dbl> <ord>            <chr>            <chr>
##  1          1 Somewhat disagr~ Somewhat agree   Strongly disagr~
##  2          2 Strongly disagr~ Neither agree n~ Somewhat agree
##  3          4 Somewhat disagr~ Strongly agree   Strongly disagr~
##  4          8 Somewhat agree   Somewhat agree   Somewhat agree
##  5          9 Strongly disagr~ Strongly agree   Somewhat agree
##  6         11 Strongly disagr~ Strongly agree   Strongly disagr~
##  7         16 Strongly agree   Strongly agree   Somewhat agree
##  8         17 Strongly disagr~ Strongly agree   Strongly disagr~
##  9         18 Somewhat disagr~ Somewhat agree   Somewhat disagr~
## 10         20 Strongly disagr~ Neither agree n~ Neither agree n~
## 11         22 Strongly disagr~ Strongly agree   Strongly disagr~
## 12         23 Somewhat agree   Strongly agree   Somewhat agree
## # ... with 48 more variables: i_know_what_a_type_ii_error_is <chr>,
## #   i_know_what_a_statistical_confidence_level_is <chr>,
## #   i_know_what_a_p_value_is <chr>, i_know_what_p_hacking_means <chr>,
## #   i_know_what_statistical_power_means <chr>,
## #   i_have_heard_of_frequentist_statistics_before <chr>,
## #   i_have_heard_of_bayesian_statistics_before <chr>,
## #   i_have_heard_the_term_parametric_statistics_before <chr>,
## #   i_have_heard_the_term_non_parametric_statistics_before <chr>,
## #   i_know_what_a_histogram_is <chr>,
## #   i_know_what_a_probability_distribution_is <chr>,
## #   i_know_what_a_random_variable_is <chr>,
## #   i_know_what_a_probability_distribution_function_is <chr>,
## #   i_know_what_a_cumulative_distribution_function_is <chr>,
## #   i_know_what_the_expectation_of_a_random_variable_is <chr>,
## #   i_know_how_to_calculate_the_variance_of_a_random_variable <chr>,
## #   i_know_what_a_z_score_is <chr>,
## #   i_know_how_to_calculate_the_correlation_between_two_variables <chr>,
## #   i_know_how_to_interpret_the_correlation_coefficient_between_two_variables <chr>,
## #   i_have_heard_of_linear_regression <chr>,
## #   i_know_how_to_run_a_linear_regression_in_some_software_or_by_hand_if_im_feeling_wild <chr
## #   i_know_how_to_interpret_a_linear_regression <chr>,
## #   i_have_heard_of_multiple_regression <chr>,
## #   i_know_how_to_perform_a_multiple_regression <chr>,
## #   i_know_how_to_interpret_a_multiple_regression <chr>,
## #   i_have_heard_of_logistic_regression <chr>,
## #   i_understand_when_to_use_a_logistic_regression <chr>,
## #   i_know_how_to_interpret_the_results_of_a_logistic_regression <chr>,
## #   i_have_heard_of_t_tests <chr>, i_have_performed_a_t_test_before <chr>,
## #   i_know_how_to_interpret_the_results_of_a_t_test <chr>,
## #   i_have_heard_of_analysis_of_variance <chr>,
## #   i_understand_when_to_run_an_analysis_of_variance_anova <chr>,
## #   i_know_how_to_interpret_the_results_from_an_anova <chr>,
## #   i_have_heard_of_a_chi_square_test <chr>,
## #   i_have_used_a_chi_square_test_before <chr>,
## #   i_know_how_to_interpret_the_results_of_a_chi_square_test <chr>,
## #   i_have_heard_of_cluster_analysis_before <chr>,
## #   i_have_used_cluster_analysis_before <chr>,
## #   i_know_how_to_interpret_the_results_of_a_cluster_analysis <chr>,
## #   i_have_heard_of_factor_analysis_either_exploratory_or_confirmatory <chr>,
## #   i_have_used_factor_analysis_either_exploratory_or_confirmatory <chr>,
## #   i_know_how_to_interpret_the_results_of_a_factor_analysis_either_exploratory_or_c
## #   i_already_have_r_and_rstudio_downloaded_to_my_computer <chr>,
## #   i_have_used_r_before <chr>, standing <chr>, college <chr>, required <chr>
```

```r
filtered_df <- survey_df %>% filter(required == "yes")
```

**A little more about plotting**

We are going to shift gears again and look at a few different kinds of plots. The main thing to remember here is that you want to think about whether the variables you have are nominal, ordinal, or continuous (that includes interval and ratio).

## 3.3 Discrete Predictor, Continuous Outcome

So far we have looked at plots for one variable, but of course we want to have ways to plot multiple variables simultaneously. We will start with the scneario where where we want to plot a continuous variable against a discrete variable. This can arise when you want to plot something like an assessment score and you think it may differ across groups in some way (maybe you intentionally introduced a difference by exposing the two groups to different interventions, for example).

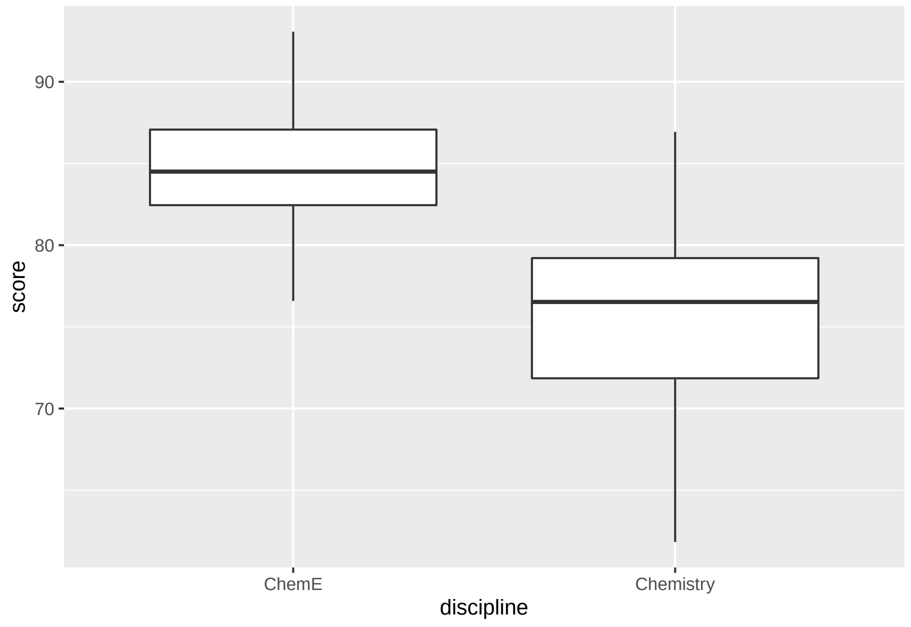
In these scenarios, a boxplot is a very standard way to go.

To demonstrate this, we will simulate a situation in which we want to look at differences on an assessment. We are specificially interested in differences between chemistry and chemical engineering students. Let's go ahead and create the data by creating two groups of 20 students each. The chemical engineering students will have scores generated from a normal distribution with $\mu = 85$ and $\sigma = 4$ (i.e., a mean of 85 and a standard deviation of 4). We will say the chemistry students have scores from a normal distribution with $\mu = 78$ anad $\sigma = 6$. This about what these distributions might look like in your head.

```r
group_size <- 20
chem_e_scores <- rnorm(n = group_size, mean = 85, sd = 4)
chem_scores <- rnorm(n = group_size, mean = 78, sd = 6)


data_df <- tibble(
  discipline = rep(c("ChemE", "Chemistry"), each = group_size),
  score = c(chem_e_scores, chem_scores)
)
```

With these data, we can then create a boxplot using `geom_boxplot()`

```
data_df %>%
  ggplot(aes(x = discipline, y = score)) +
  geom_boxplot()
```

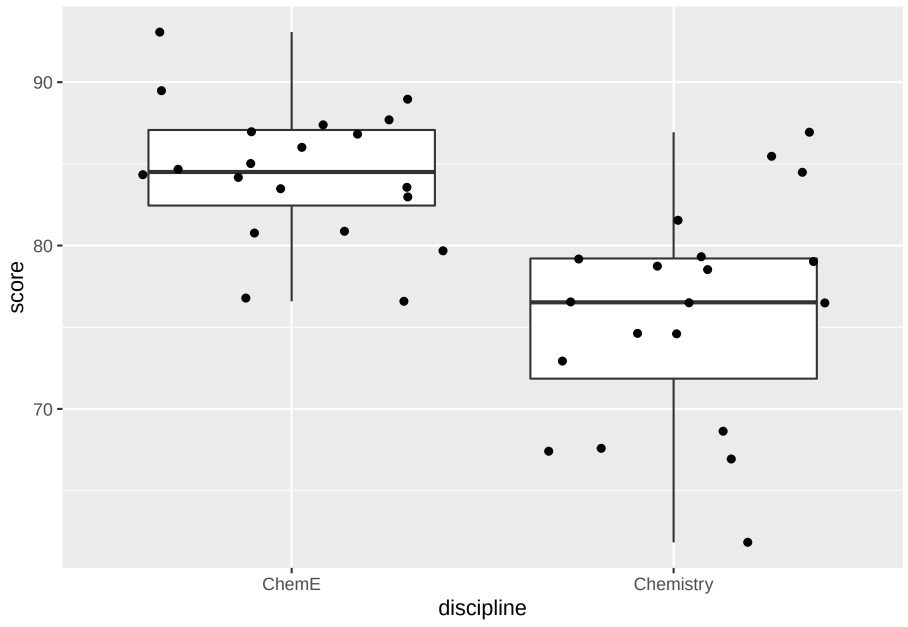


You can make a few modifications to possibly make this plot easier to read.

The first is to put the discrete category on the y axis instead of the x axis.

The second is to use geom_jitter() in addition to geom_boxplot() to show the individual points in each group.

```
data_df %>%
  ggplot(aes(y = score, x = discipline)) +
  geom_boxplot() +
  geom_jitter()
```



## 3.4 Continuous predictor and continuous outcome

First, let's re-do a lot of the steps in this week's script for reading in data and transforming it a little

```
mydata <- read_csv("Free Reduced Lunch by Schools and Grade Structures 2008-2017_final
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   div_num = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

Check the structure of the data (this output is a bit long).

```
str(mydata)
```

```
## tibble [2,101 x 137] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ sch_id        : chr [1:2101] "001-0070" "001-0080" "001-0530" "001-0540" ...
##  $ div_num       : num [1:2101] 1 1 1 1 1 1 1 1 1 1 ...
##  $ div_name      : chr [1:2101] "Accomack County" "Accomack County" "Accomack County" "Accomac
##  $ school_num    : chr [1:2101] "0070<U+00A0>" "0080<U+00A0>" "0530<U+00A0>" "0540<U+00A0>" ..
##  $ school_name   : chr [1:2101] "NANDUA HIGH" "CHINCOTEAGUE ELEM" "TANGIER COMBINED" "ARCADIA
##  $ school_name2  : chr [1:2101] NA NA NA NA ...
##  $ type0809      : chr [1:2101] "SCH-HIGH" "SCH-ELEM" "SCH-COMB" "SCH-HIGH" ...
##  $ lowgrade_2008 : chr [1:2101] "9" "PK" "KG" "9" ...
##  $ higrade_2008  : chr [1:2101] "12" "5" "12" "12" ...
##  $ totalFT_2008  : chr [1:2101] "731" "263" "80" "638" ...
##  $ total_2008    : chr [1:2101] "731" "263" "80" "638" ...
##  $ snp_0809      : chr [1:2101] "659" "257" "80" "622" ...
##  $ free_elig_0809: chr [1:2101] "306" "95" "38" "289" ...
##  $ free_per_0809 : chr [1:2101] "46.43%" "36.96%" "47.50%" "46.46%" ...
##  $ red_elig_0809 : chr [1:2101] "64" "8" "0" "56" ...
##  $ red_per_0809  : chr [1:2101] "9.71%" "3.11%" "0.00%" "9.00%" ...
##  $ totalFRL_0809 : chr [1:2101] "370" "103" "38" "345" ...
##  $ totalper_0809 : chr [1:2101] "56.15%" "40.08%" "47.50%" "55.47%" ...
##  $ type0910      : chr [1:2101] "SCH-HIGH" "SCH-ELEM" "SCH-COMB" "SCH-HIGH" ...
##  $ lowgrade_2009 : chr [1:2101] "9" "PK" "KG" "9" ...
##  $ higrade_2009  : chr [1:2101] "12" "5" "12" "12" ...
##  $ totalFT_2009  : chr [1:2101] "654" "266" "78" "634" ...
##  $ total_2009    : chr [1:2101] "654" "266" "78" "634" ...
##  $ snp_0910      : chr [1:2101] "655" "266" "78" "635" ...
##  $ free_elig_0910: chr [1:2101] "290" "99" "36" "286" ...
##  $ free_per_0910 : chr [1:2101] "44.27%" "37.22%" "46.15%" "45.04%" ...
##  $ red_elig_0910 : chr [1:2101] "37" "14" "0" "66" ...
##  $ red_per_0910  : chr [1:2101] "5.65%" "5.26%" "0.00%" "10.39%" ...
##  $ totalFRL_09010: chr [1:2101] "327" "113" "36" "352" ...
##  $ totalper_0910 : chr [1:2101] "49.92%" "42.48%" "46.15%" "55.43%" ...
##  $ type1011      : chr [1:2101] "SCH-HIGH" "SCH-ELEM" "SCH-COMB" "SCH-HIGH" ...
##  $ lowgrade_2010 : chr [1:2101] "9" "PK" "KG" "9" ...
##  $ higrade_2010  : chr [1:2101] "12" "5" "12" "12" ...
##  $ totalFT_2010  : chr [1:2101] "603" "268" "74" "614" ...
##  $ total_2010    : chr [1:2101] "603" "268" "74" "614" ...
##  $ snp_1011      : chr [1:2101] "603" "277" "74" "606" ...
##  $ free_elig_1011: chr [1:2101] "285" "108" "32" "308" ...
##  $ free_per_1011 : chr [1:2101] "47.26%" "38.99%" "43.24%" "50.83%" ...
##  $ red_elig_1011 : chr [1:2101] "46" "8" "0" "50" ...
##  $ red_per_1011  : chr [1:2101] "7.63%" "2.89%" "0.00%" "8.25%" ...
##  $ totalFRL_1011 : chr [1:2101] "331" "116" "32" "358" ...
##  $ totalper_1011 : chr [1:2101] "54.89%" "41.88%" "43.24%" "59.08%" ...
##  $ type1112      : chr [1:2101] "SCH-HIGH" "SCH-ELEM" "SCH-COMB" "SCH-HIGH" ...
##  $ lowgrade_2011 : chr [1:2101] "9" "PK" "KG" "9" ...
##  $ higrade_2011  : chr [1:2101] "12" "5" "12" "12" ...
##  $ totalFT_2011  : chr [1:2101] "593" "276" "73" "605" ...
##  $ total_2011    : chr [1:2101] "593" "276" "73" "605" ...
##  $ snp_1112      : chr [1:2101] "593" "281" "73" "611" ...
##  $ free_elig_1112: chr [1:2101] "289" "116" "31" "318" ...
##  $ free_per_1112 : chr [1:2101] "48.74%" "41.28%" "42.47%" "52.05%" ...
##  $ red_elig_1112 : chr [1:2101] "50" "14" "0" "44" ...
##  $ red_per_1112  : chr [1:2101] "8.43%" "4.98%" "0.00%" "7.20%" ...
##  $ totalFRL_1112 : chr [1:2101] "339" "130" "31" "362" ...
##  $ totalper_1112 : chr [1:2101] "57.17%" "46.26%" "42.47%" "59.25%" ...
##  $ type1213      : chr [1:2101] "SCH-HIGH" "SCH-ELEM" "SCH-COMB" "SCH-HIGH" ...
##  $ lowgrade_2012 : chr [1:2101] "9" "PK" "KG" "9" ...
##  $ higrade_2012  : chr [1:2101] "12" "5" "12" "12" ...
##  $ totalFT_2012  : chr [1:2101] "637" "258" "68" "579" ...
##  $ total_2012    : chr [1:2101] "637" "258" "68" "579" ...
##  $ snp_1213      : chr [1:2101] "633" "259" "68" "579" ...
##  $ free_elig_1213: chr [1:2101] "324" "117" "21" "348" ...
##  $ free_per_1213 : chr [1:2101] "51.18%" "45.17%" "30.88%" "60.10%" ...
##  $ red_elig_1213 : chr [1:2101] "42" "20" "5" "33" ...
##  $ red_per_1213  : chr [1:2101] "6.64%" "7.72%" "7.35%" "5.70%" ...
##  $ totalFRL_1213 : chr [1:2101] "366" "137" "26" "381" ...
##  $ totalper_1213 : chr [1:2101] "57.82%" "52.90%" "38.24%" "65.80%" ...
##  $ type1314      : chr [1:2101] "SCH-HIGH" "SCH-ELEM" "SCH-COMB" "SCH-HIGH" ...
##  $ lowgrade_2013 : chr [1:2101] "9" "PK" "KG" "9" ...
##  $ higrade_2013  : chr [1:2101] "12" "5" "12" "12" ...
##  $ totalFT_2013  : chr [1:2101] "670" "238" "66" "582" ...
##  $ total_2013    : chr [1:2101] "670" "238" "66" "582" ...
##  $ snp_1314      : chr [1:2101] "668" "239" "56" "589" ...
##  $ free_elig_1314: chr [1:2101] "346" "102" "12" "347" ...
##  $ free_per_1314 : chr [1:2101] "51.80%" "42.68%" "21.43%" "58.91%" ...
##  $ red_elig_1314 : chr [1:2101] "44" "19" "4" "54" ...
##  $ red_per_1314  : chr [1:2101] "6.59%" "7.95%" "7.14%" "9.17%" ...
##  $ totalFRL_1314 : chr [1:2101] "390" "121" "16" "401" ...
##  $ totalper_1314 : chr [1:2101] "58.38%" "50.63%" "28.57%" "68.08%" ...
##  $ type1415      : chr [1:2101] NA NA NA NA ...
##  $ lowgrade_2014 : chr [1:2101] "9" "PK" "KG" "9" ...
##  $ higrade_2014  : chr [1:2101] "12" "5" "12" "12" ...
##  $ totalFT_2014  : chr [1:2101] "685" "251" "65" "581" ...
##  $ total_2014    : chr [1:2101] "685" "251" "65" "581" ...
##  $ snp_1415      : chr [1:2101] "672" "239" "61" "586" ...
##  $ free_elig_1415: chr [1:2101] "361" "93" "14" "351" ...
##  $ free_per_1415 : chr [1:2101] "53.72%" "38.91%" "22.95%" "59.90%" ...
##  $ red_elig_1415 : chr [1:2101] "40" "17" "4" "40" ...
##  $ red_per_1415  : chr [1:2101] "5.95%" "7.11%" "6.56%" "6.83%" ...
##  $ totalFRL_1415 : chr [1:2101] "401" "110" "18" "391" ...
##  $ totalper_1415 : chr [1:2101] "59.67%" "46.03%" "29.51%" "66.72%" ...
##  $ CEP_1516      : chr [1:2101] "#NULL!" "#NULL!" "#NULL!" "#NULL!" ...
##  $ type1516      : chr [1:2101] "SCH-HIGH" "SCH-ELEM" "SCH-COMB" "SCH-HIGH" ...
##  $ lowgrade_2015 : chr [1:2101] "9" "PK" "KG" "9" ...
##  $ higrade_2015  : chr [1:2101] "12" "5" "12" "12" ...
##  $ totalFT_2015  : chr [1:2101] "737" "259" "65" "621" ...
##  $ total_2015    : chr [1:2101] "737" "259" "65" "621" ...
##  $ snp_1516      : chr [1:2101] "728" "268" "67" "608" ...
##  $ free_elig_1516: chr [1:2101] "362" "109" "12" "339" ...
##  $ free_per_1516 : chr [1:2101] "49.73%" "40.67%" "17.91%" "55.76%" ...
##   [list output truncated]
##  - attr(*, "spec")=
##   .. cols(
##   ..   sch_id = col_character(),
##   ..   div_num = col_double(),
##   ..   div_name = col_character(),
##   ..   school_num = col_character(),
##   ..   school_name = col_character(),
##   ..   school_name2 = col_character(),
##   ..   type0809 = col_character(),
##   ..   lowgrade_2008 = col_character(),
##   ..   higrade_2008 = col_character(),
##   ..   totalFT_2008 = col_character(),
##   ..   total_2008 = col_character(),
##   ..   snp_0809 = col_character(),
##   ..   free_elig_0809 = col_character(),
##   ..   free_per_0809 = col_character(),
##   ..   red_elig_0809 = col_character(),
##   ..   red_per_0809 = col_character(),
##   ..   totalFRL_0809 = col_character(),
##   ..   totalper_0809 = col_character(),
##   ..   type0910 = col_character(),
##   ..   lowgrade_2009 = col_character(),
##   ..   higrade_2009 = col_character(),
##   ..   totalFT_2009 = col_character(),
##   ..   total_2009 = col_character(),
##   ..   snp_0910 = col_character(),
##   ..   free_elig_0910 = col_character(),
##   ..   free_per_0910 = col_character(),
##   ..   red_elig_0910 = col_character(),
##   ..   red_per_0910 = col_character(),
##   ..   totalFRL_09010 = col_character(),
##   ..   totalper_0910 = col_character(),
##   ..   type1011 = col_character(),
##   ..   lowgrade_2010 = col_character(),
##   ..   higrade_2010 = col_character(),
##   ..   totalFT_2010 = col_character(),
##   ..   total_2010 = col_character(),
##   ..   snp_1011 = col_character(),
##   ..   free_elig_1011 = col_character(),
##   ..   free_per_1011 = col_character(),
##   ..   red_elig_1011 = col_character(),
##   ..   red_per_1011 = col_character(),
##   ..   totalFRL_1011 = col_character(),
##   ..   totalper_1011 = col_character(),
##   ..   type1112 = col_character(),
##   ..   lowgrade_2011 = col_character(),
##   ..   higrade_2011 = col_character(),
##   ..   totalFT_2011 = col_character(),
##   ..   total_2011 = col_character(),
##   ..   snp_1112 = col_character(),
##   ..   free_elig_1112 = col_character(),
##   ..   free_per_1112 = col_character(),
##   ..   red_elig_1112 = col_character(),
##   ..   red_per_1112 = col_character(),
##   ..   totalFRL_1112 = col_character(),
##   ..   totalper_1112 = col_character(),
##   ..   type1213 = col_character(),
##   ..   lowgrade_2012 = col_character(),
##   ..   higrade_2012 = col_character(),
##   ..   totalFT_2012 = col_character(),
##   ..   total_2012 = col_character(),
##   ..   snp_1213 = col_character(),
##   ..   free_elig_1213 = col_character(),
##   ..   free_per_1213 = col_character(),
##   ..   red_elig_1213 = col_character(),
##   ..   red_per_1213 = col_character(),
##   ..   totalFRL_1213 = col_character(),
##   ..   totalper_1213 = col_character(),
##   ..   type1314 = col_character(),
##   ..   lowgrade_2013 = col_character(),
##   ..   higrade_2013 = col_character(),
##   ..   totalFT_2013 = col_character(),
##   ..   total_2013 = col_character(),
##   ..   snp_1314 = col_character(),
##   ..   free_elig_1314 = col_character(),
##   ..   free_per_1314 = col_character(),
##   ..   red_elig_1314 = col_character(),
##   ..   red_per_1314 = col_character(),
##   ..   totalFRL_1314 = col_character(),
##   ..   totalper_1314 = col_character(),
##   ..   type1415 = col_character(),
##   ..   lowgrade_2014 = col_character(),
##   ..   higrade_2014 = col_character(),
##   ..   totalFT_2014 = col_character(),
##   ..   total_2014 = col_character(),
##   ..   snp_1415 = col_character(),
##   ..   free_elig_1415 = col_character(),
##   ..   free_per_1415 = col_character(),
##   ..   red_elig_1415 = col_character(),
##   ..   red_per_1415 = col_character(),
##   ..   totalFRL_1415 = col_character(),
##   ..   totalper_1415 = col_character(),
##   ..   CEP_1516 = col_character(),
##   ..   type1516 = col_character(),
##   ..   lowgrade_2015 = col_character(),
##   ..   higrade_2015 = col_character(),
##   ..   totalFT_2015 = col_character(),
##   ..   total_2015 = col_character(),
##   ..   snp_1516 = col_character(),
##   ..   free_elig_1516 = col_character(),
##   ..   free_per_1516 = col_character(),
##   ..   red_elig_1516 = col_character(),
##   ..   red_Per_1516 = col_character(),
##   ..   totalFRL_1516 = col_character(),
##   ..   totalper_1516 = col_character(),
##   ..   CEP_1617 = col_character(),
##   ..   type1617 = col_character(),
##   ..   lowgrade_2016 = col_character(),
##   ..   higrade_2016 = col_character(),
##   ..   totalFT_2016 = col_character(),
##   ..   total_2016 = col_character(),
##   ..   snp_2016 = col_character(),
##   ..   free_elig_1617 = col_character(),
##   ..   free_per_1617 = col_character(),
##   ..   red_elig_1617 = col_character(),
##   ..   red_per_1617 = col_character(),
##   ..   totalFRL_1617 = col_character(),
##   ..   totalper_1617 = col_character(),
##   ..   CEP_1718 = col_character(),
##   ..   type1718 = col_character(),
##   ..   lowgrade_2017 = col_character(),
##   ..   higrade_2017 = col_character(),
##   ..   totalFT_2017 = col_character(),
##   ..   total_2017 = col_character(),
##   ..   snp_1718 = col_character(),
##   ..   free_elig_1718 = col_character(),
##   ..   free_per_1718 = col_character(),
##   ..   red_elig_1718 = col_character(),
##   ..   red_per_1718 = col_character(),
##   ..   totalFRL_1718 = col_character(),
##   ..   totalper_1718 = col_character(),
##   ..   stable = col_character(),
##   ..   new = col_character(),
##   ..   closed = col_character(),
##   ..   close_yr = col_character(),
##   ..   reuseid = col_character(),
##   ..   gradechg = col_character(),
##   ..   gradechg_yr = col_character(),
##   ..   grchgyr_2 = col_character()
##   .. )
```

Or just check the structure of one specific variable.

```
str(mydata$total_2017)
```

```
##  chr [1:2101] "742" "236" "60" "624" "286" "485" "583" "550" "600" "514" ...
```

**NOTE:** When you have a lot of variables, running this str() function is not a great idea - the output is a little too cumbersome

## 3.5 Mutating Variables

Note that almost all of the data reads in as a "character" data type which are just strings, This can create issues.

We know that many of the columns are actually storing numbers or "numeric" values as R refers to them. We need to fix this.

Let's tell R that these columns (at least the two we are going to use) are numeric.

We are going to see two interchangeable ways to do this.

First, we use the $ operator which lets me specify a specific column within my data frame in combination with the as.numeric() function

```
mydata$total_2017<-as.numeric(mydata$total_2017)
mydata$totalFRL_1718<-as.numeric(mydata$totalFRL_1718)
```

Some columns have a percent symbol, which you will need to remove before coercing to numeric data type

```
mydata <- mydata %>%
  mutate(totalper_0809 = str_remove(totalper_0809, "%"))
```

Then we can change the column from character to numeric

```
mydata$totalper_0809 <- as.numeric(mydata$totalper_0809)
```

```
## Warning: NAs introduced by coercion
```

Check to make sure it converted the column type correctly using `str()`.

```
str(mydata$totalper_0809)
```

```
##  num [1:2101] 56.1 40.1 47.5 55.5 33.4 ...
```

Second, alternatively, we can do this for a whole set of variables at once. We just need to specify a matching criteria.

```
newdf <- mydata %>%
  mutate_at(vars(starts_with("total")), as.numeric)
```

```
## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion

## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
```

```
newdf <- newdf %>%
  mutate_at(vars(starts_with("totalFRL")), as.numeric)
```

Check whether the old and new variables are stored differently (old as a character, new as a numeric variable)

```
str(mydata$total_2008)
```

```
##  chr [1:2101] "731" "263" "80" "638" "333" "536" "610" "490" "585" "450" ...
```

```
str(newdf$total_2008)
```

```
##  num [1:2101] 731 263 80 638 333 536 610 490 585 450 ...
```

## 3.6 Filtering and Selecting

A basic operation we do a lot is to filter the data so that we are working with a subset of all that we have.

We can do this with the filter() function, part of the dplyr package (in the tidyverse collection of packages).

Let's say we want to look at the schools with div_num values less than 50.

```
newdf %>% filter(div_num < 50)
```

```
## # A tibble: 800 x 137
##    sch_id div_num div_name school_num school_name school_name2 type0809
##    <chr>    <dbl> <chr>    <chr>      <chr>       <chr>        <chr>
##  1 001-0~       1 Accomac~ "0070\xa0" NANDUA HIGH <NA>         SCH-HIGH
##  2 001-0~       1 Accomac~ "0080\xa0" CHINCOTEAG~ <NA>         SCH-ELEM
##  3 001-0~       1 Accomac~ "0530\xa0" TANGIER CO~ <NA>         SCH-COMB
##  4 001-0~       1 Accomac~ "0540\xa0" ARCADIA HI~ <NA>         SCH-HIGH
##  5 001-0~       1 Accomac~ "0580\xa0" CHINCOTEAG~ <NA>         SCH-COMB
##  6 001-0~       1 Accomac~ "0590\xa0" PUNGOTEAGU~ <NA>         SCH-ELEM
##  7 001-0~       1 Accomac~ "0600\xa0" KEGOTANK E~ <NA>         SCH-ELEM
##  8 001-0~       1 Accomac~ "0701\xa0" ACCAWMACKE~ <NA>         SCH-ELEM
##  9 001-0~       1 Accomac~ "0702\xa0" METOMPKIN ~ <NA>         SCH-ELEM
## 10 001-0~       1 Accomac~ "0703\xa0" NANDUA MID~ <NA>         SCH-MID
## # ... with 790 more rows, and 130 more variables: lowgrade_2008 <chr>,
## #   higrade_2008 <chr>, totalFT_2008 <dbl>, total_2008 <dbl>, snp_0809 <chr>,
## #   free_elig_0809 <chr>, free_per_0809 <chr>, red_elig_0809 <chr>,
## #   red_per_0809 <chr>, totalFRL_0809 <dbl>, totalper_0809 <dbl>,
## #   type0910 <chr>, lowgrade_2009 <chr>, higrade_2009 <chr>,
## #   totalFT_2009 <dbl>, total_2009 <dbl>, snp_0910 <chr>, free_elig_0910 <chr>,
## #   free_per_0910 <chr>, red_elig_0910 <chr>, red_per_0910 <chr>,
## #   totalFRL_09010 <dbl>, totalper_0910 <dbl>, type1011 <chr>,
## #   lowgrade_2010 <chr>, higrade_2010 <chr>, totalFT_2010 <dbl>,
## #   total_2010 <dbl>, snp_1011 <chr>, free_elig_1011 <chr>,
## #   free_per_1011 <chr>, red_elig_1011 <chr>, red_per_1011 <chr>,
## #   totalFRL_1011 <dbl>, totalper_1011 <dbl>, type1112 <chr>,
## #   lowgrade_2011 <chr>, higrade_2011 <chr>, totalFT_2011 <dbl>,
## #   total_2011 <dbl>, snp_1112 <chr>, free_elig_1112 <chr>,
## #   free_per_1112 <chr>, red_elig_1112 <chr>, red_per_1112 <chr>,
## #   totalFRL_1112 <dbl>, totalper_1112 <dbl>, type1213 <chr>,
## #   lowgrade_2012 <chr>, higrade_2012 <chr>, totalFT_2012 <dbl>,
## #   total_2012 <dbl>, snp_1213 <chr>, free_elig_1213 <chr>,
## #   free_per_1213 <chr>, red_elig_1213 <chr>, red_per_1213 <chr>,
## #   totalFRL_1213 <dbl>, totalper_1213 <dbl>, type1314 <chr>,
## #   lowgrade_2013 <chr>, higrade_2013 <chr>, totalFT_2013 <dbl>,
## #   total_2013 <dbl>, snp_1314 <chr>, free_elig_1314 <chr>,
## #   free_per_1314 <chr>, red_elig_1314 <chr>, red_per_1314 <chr>,
## #   totalFRL_1314 <dbl>, totalper_1314 <dbl>, type1415 <chr>,
## #   lowgrade_2014 <chr>, higrade_2014 <chr>, totalFT_2014 <dbl>,
## #   total_2014 <dbl>, snp_1415 <chr>, free_elig_1415 <chr>,
## #   free_per_1415 <chr>, red_elig_1415 <chr>, red_per_1415 <chr>,
## #   totalFRL_1415 <dbl>, totalper_1415 <dbl>, CEP_1516 <chr>, type1516 <chr>,
## #   lowgrade_2015 <chr>, higrade_2015 <chr>, totalFT_2015 <dbl>,
## #   total_2015 <dbl>, snp_1516 <chr>, free_elig_1516 <chr>,
## #   free_per_1516 <chr>, red_elig_1516 <chr>, red_Per_1516 <chr>,
## #   totalFRL_1516 <dbl>, totalper_1516 <dbl>, CEP_1617 <chr>, type1617 <chr>,
## #   lowgrade_2016 <chr>, higrade_2016 <chr>, ...
```

Or, if we want to look at schools where the highest grade in 2008 was grade five, we can try:

```
newdf %>% filter(higrade_2008 == "5") # this returns a subsetted dataframe with 878 ro
```

```
## # A tibble: 878 x 137
##    sch_id div_num div_name school_num school_name school_name2 type0809
##    <chr>    <dbl> <chr>    <chr>      <chr>       <chr>        <chr>
##  1 001-0~       1 Accomac~ "0080\xa0" CHINCOTEAG~ <NA>         SCH-ELEM
##  2 001-0~       1 Accomac~ "0590\xa0" PUNGOTEAGU~ <NA>         SCH-ELEM
##  3 001-0~       1 Accomac~ "0600\xa0" KEGOTANK E~ <NA>         SCH-ELEM
##  4 001-0~       1 Accomac~ "0701\xa0" ACCAWMACKE~ <NA>         SCH-ELEM
##  5 001-0~       1 Accomac~ "0702\xa0" METOMPKIN ~ <NA>         SCH-ELEM
##  6 002-0~       2 Albemar~ "0010\xa0" HOLLYMEAD ~ <NA>         SCH-ELEM
##  7 002-0~       2 Albemar~ "0030\xa0" SCOTTSVILL~ <NA>         SCH-ELEM
##  8 002-0~       2 Albemar~ "0040\xa0" MARY CARR ~ <NA>         SCH-ELEM
##  9 002-0~       2 Albemar~ "0100\xa0" BROADUS WO~ <NA>         SCH-ELEM
## 10 002-0~       2 Albemar~ "0150\xa0" PAUL H CAL~ <NA>         SCH-ELEM
## # ... with 868 more rows, and 130 more variables: lowgrade_2008 <chr>,
## #   higrade_2008 <chr>, totalFT_2008 <dbl>, total_2008 <dbl>, snp_0809 <chr>,
## #   free_elig_0809 <chr>, free_per_0809 <chr>, red_elig_0809 <chr>,
## #   red_per_0809 <chr>, totalFRL_0809 <dbl>, totalper_0809 <dbl>,
## #   type0910 <chr>, lowgrade_2009 <chr>, higrade_2009 <chr>,
## #   totalFT_2009 <dbl>, total_2009 <dbl>, snp_0910 <chr>, free_elig_0910 <chr>,
## #   free_per_0910 <chr>, red_elig_0910 <chr>, red_per_0910 <chr>,
## #   totalFRL_09010 <dbl>, totalper_0910 <dbl>, type1011 <chr>,
## #   lowgrade_2010 <chr>, higrade_2010 <chr>, totalFT_2010 <dbl>,
## #   total_2010 <dbl>, snp_1011 <chr>, free_elig_1011 <chr>,
## #   free_per_1011 <chr>, red_elig_1011 <chr>, red_per_1011 <chr>,
## #   totalFRL_1011 <dbl>, totalper_1011 <dbl>, type1112 <chr>,
## #   lowgrade_2011 <chr>, higrade_2011 <chr>, totalFT_2011 <dbl>,
## #   total_2011 <dbl>, snp_1112 <chr>, free_elig_1112 <chr>,
## #   free_per_1112 <chr>, red_elig_1112 <chr>, red_per_1112 <chr>,
## #   totalFRL_1112 <dbl>, totalper_1112 <dbl>, type1213 <chr>,
## #   lowgrade_2012 <chr>, higrade_2012 <chr>, totalFT_2012 <dbl>,
## #   total_2012 <dbl>, snp_1213 <chr>, free_elig_1213 <chr>,
## #   free_per_1213 <chr>, red_elig_1213 <chr>, red_per_1213 <chr>,
## #   totalFRL_1213 <dbl>, totalper_1213 <dbl>, type1314 <chr>,
## #   lowgrade_2013 <chr>, higrade_2013 <chr>, totalFT_2013 <dbl>,
## #   total_2013 <dbl>, snp_1314 <chr>, free_elig_1314 <chr>,
## #   free_per_1314 <chr>, red_elig_1314 <chr>, red_per_1314 <chr>,
## #   totalFRL_1314 <dbl>, totalper_1314 <dbl>, type1415 <chr>,
## #   lowgrade_2014 <chr>, higrade_2014 <chr>, totalFT_2014 <dbl>,
## #   total_2014 <dbl>, snp_1415 <chr>, free_elig_1415 <chr>,
## #   free_per_1415 <chr>, red_elig_1415 <chr>, red_per_1415 <chr>,
## #   totalFRL_1415 <dbl>, totalper_1415 <dbl>, CEP_1516 <chr>, type1516 <chr>,
## #   lowgrade_2015 <chr>, higrade_2015 <chr>, totalFT_2015 <dbl>,
## #   total_2015 <dbl>, snp_1516 <chr>, free_elig_1516 <chr>,
## #   free_per_1516 <chr>, red_elig_1516 <chr>, red_Per_1516 <chr>,
## #   totalFRL_1516 <dbl>, totalper_1516 <dbl>, CEP_1617 <chr>, type1617 <chr>,
## #   lowgrade_2016 <chr>, higrade_2016 <chr>, ...
```

Note that we had to set it equal to the character value “5” rather than the numeric value 5. Why?

If we wanted to filter on numeric values instead, we would want to do something like this:

```
newdf %>%
  mutate(higrade_2008 = as.numeric(higrade_2008)) %>%
  filter(higrade_2008 == 5) # again, this returns a subsetted dataframe with 878 rows
```

```
## Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
```

```
## # A tibble: 878 x 137
##    sch_id div_num div_name school_num school_name school_name2 type0809
##    <chr>    <dbl> <chr>    <chr>      <chr>       <chr>        <chr>
##  1 001-0~       1 Accomac~ "0080\xa0" CHINCOTEAG~ <NA>         SCH-ELEM
##  2 001-0~       1 Accomac~ "0590\xa0" PUNGOTEAGU~ <NA>         SCH-ELEM
##  3 001-0~       1 Accomac~ "0600\xa0" KEGOTANK E~ <NA>         SCH-ELEM
##  4 001-0~       1 Accomac~ "0701\xa0" ACCAWMACKE~ <NA>         SCH-ELEM
##  5 001-0~       1 Accomac~ "0702\xa0" METOMPKIN ~ <NA>         SCH-ELEM
##  6 002-0~       2 Albemar~ "0010\xa0" HOLLYMEAD ~ <NA>         SCH-ELEM
##  7 002-0~       2 Albemar~ "0030\xa0" SCOTTSVILL~ <NA>         SCH-ELEM
##  8 002-0~       2 Albemar~ "0040\xa0" MARY CARR ~ <NA>         SCH-ELEM
##  9 002-0~       2 Albemar~ "0100\xa0" BROADUS WO~ <NA>         SCH-ELEM
## 10 002-0~       2 Albemar~ "0150\xa0" PAUL H CAL~ <NA>         SCH-ELEM
## # ... with 868 more rows, and 130 more variables: lowgrade_2008 <chr>,
## #   higrade_2008 <dbl>, totalFT_2008 <dbl>, total_2008 <dbl>, snp_0809 <chr>,
## #   free_elig_0809 <chr>, free_per_0809 <chr>, red_elig_0809 <chr>,
## #   red_per_0809 <chr>, totalFRL_0809 <dbl>, totalper_0809 <dbl>,
## #   type0910 <chr>, lowgrade_2009 <chr>, higrade_2009 <chr>,
## #   totalFT_2009 <dbl>, total_2009 <dbl>, snp_0910 <chr>, free_elig_0910 <chr>,
## #   free_per_0910 <chr>, red_elig_0910 <chr>, red_per_0910 <chr>,
## #   totalFRL_09010 <dbl>, totalper_0910 <dbl>, type1011 <chr>,
## #   lowgrade_2010 <chr>, higrade_2010 <chr>, totalFT_2010 <dbl>,
## #   total_2010 <dbl>, snp_1011 <chr>, free_elig_1011 <chr>,
## #   free_per_1011 <chr>, red_elig_1011 <chr>, red_per_1011 <chr>,
## #   totalFRL_1011 <dbl>, totalper_1011 <dbl>, type1112 <chr>,
## #   lowgrade_2011 <chr>, higrade_2011 <chr>, totalFT_2011 <dbl>,
## #   total_2011 <dbl>, snp_1112 <chr>, free_elig_1112 <chr>,
## #   free_per_1112 <chr>, red_elig_1112 <chr>, red_per_1112 <chr>,
## #   totalFRL_1112 <dbl>, totalper_1112 <dbl>, type1213 <chr>,
## #   lowgrade_2012 <chr>, higrade_2012 <chr>, totalFT_2012 <dbl>,
## #   total_2012 <dbl>, snp_1213 <chr>, free_elig_1213 <chr>,
## #   free_per_1213 <chr>, red_elig_1213 <chr>, red_per_1213 <chr>,
## #   totalFRL_1213 <dbl>, totalper_1213 <dbl>, type1314 <chr>,
## #   lowgrade_2013 <chr>, higrade_2013 <chr>, totalFT_2013 <dbl>,
## #   total_2013 <dbl>, snp_1314 <chr>, free_elig_1314 <chr>,
## #   free_per_1314 <chr>, red_elig_1314 <chr>, red_per_1314 <chr>,
## #   totalFRL_1314 <dbl>, totalper_1314 <dbl>, type1415 <chr>,
## #   lowgrade_2014 <chr>, higrade_2014 <chr>, totalFT_2014 <dbl>,
## #   total_2014 <dbl>, snp_1415 <chr>, free_elig_1415 <chr>,
## #   free_per_1415 <chr>, red_elig_1415 <chr>, red_per_1415 <chr>,
## #   totalFRL_1415 <dbl>, totalper_1415 <dbl>, CEP_1516 <chr>, type1516 <chr>,
## #   lowgrade_2015 <chr>, higrade_2015 <chr>, totalFT_2015 <dbl>,
## #   total_2015 <dbl>, snp_1516 <chr>, free_elig_1516 <chr>,
## #   free_per_1516 <chr>, red_elig_1516 <chr>, red_Per_1516 <chr>,
## #   totalFRL_1516 <dbl>, totalper_1516 <dbl>, CEP_1617 <chr>, type1617 <chr>,
## #   lowgrade_2016 <chr>, higrade_2016 <chr>, ...
```

## 3.7 Grouping and Summarizing

Let's shift gears to a different combination of operations…

Let's go ahead and try using tidyverse to narrow to what we want. Imagine we want to see the county level aggregate numbers for FRL in the 2017-2018 school year.

We will start out with our entire data frame and then use pipes (the %>% operator) to work from there. The final result will be stored in our new data frame that we are creating, called county_level_aggregate.

First, select will pick columns Next, group_by and summarize work together to get us our aggregate totals.

```
county_level_aggregate <- newdf %>%
  select(div_name, total_2017, totalFRL_1718) %>%
  group_by(div_name) %>%
  summarize(totalstudents = sum(total_2017),
            totalFRL = sum(totalFRL_1718))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

Now, we can compute percentages if we like and we can specify a new column by referring to. One that doesn't exist yet but will after we run this code. We will do this two interchangeable ways.

First, the old school way:

```
county_level_aggregate$percent_FRL <- county_level_aggregate$totalFRL/county_level_aggregate$tota
```

Second, the tidyverse way:

```
county_level_aggregate <- county_level_aggregate %>%
  mutate(percent_frl = totalFRL / totalstudents * 100)
```

Just for fun, let's see how this could have been incorporated into our summarize call

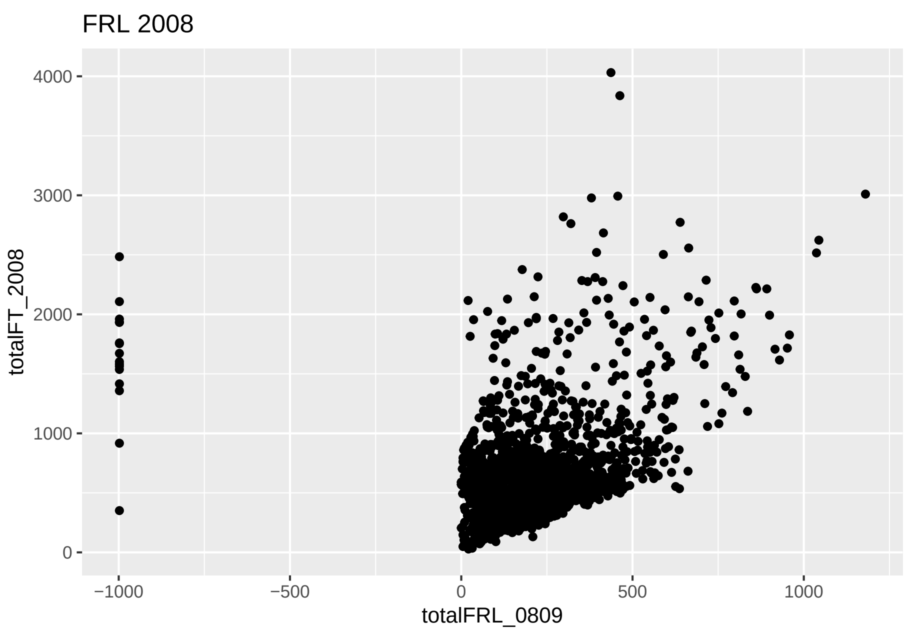
```
county_level_percents <- newdf %>%
  select(div_name, total_2017, totalFRL_1718) %>%
  group_by(div_name) %>%
  summarize(percentFRL=sum(totalFRL_1718)/sum(total_2017) * 100)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

Something is going to look weird with this plot

```
newdf %>%
  ggplot(aes(totalFRL_0809, totalFT_2008)) +
  geom_point() +
  labs(title = "FRL 2008", x = "totalFRL_0809")
```

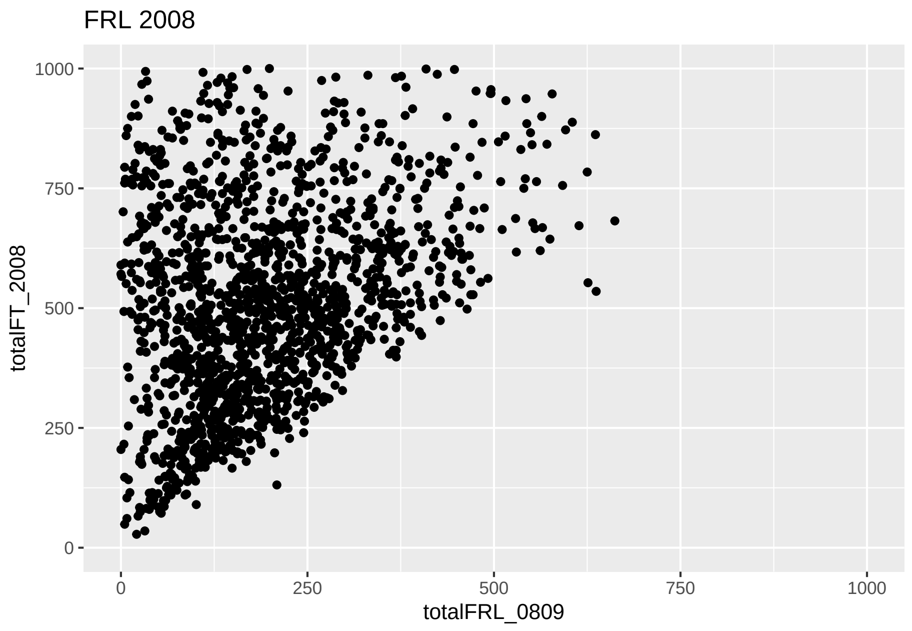
```
## Warning: Removed 236 rows containing missing values (geom_point).
```



Let's see if we can fix it

```
newdf %>%
  filter(!is.na(totalFRL_0809)) %>%
  ggplot(aes(totalFRL_0809, totalFT_2008)) +
  geom_point() +
  labs(title = "FRL 2008",
       x = "totalFRL_0809") +
  xlim(0, 1000) +
  ylim(0, 1000)
```

```
## Warning: Removed 324 rows containing missing values (geom_point).
```



# Chapter 4

# Week 4: Assumptions and Correlations

This week we will be discussing Chapters 5 and 6 from DSUR. These notes will pull out some of the important pieces from each chapter.

## 4.1 Assumptions

These assumptions that we are making are helpful when determining whether we should be using parametric vs non-parametric statistical tests. What does "parametric" mean here? It means that the data are from a parameterized distribution (i.e., parameters characterize the distribution that the data come from). An example of a parameterized distribution that we have already seen is the normal distribution. The two parameters for the normal distribution are $\mu$ for the mean and $\sigma$ for the standard deviation. We have seen this altogether with this kind of notation to denote that $x_i$ is from a normal distribution:

$x_i \sim \mathcal{N}(\mu, \sigma^2)$

### 4.1.1 Normally distributed data

This assumption is about the normality of the sampling distribution. The big idea here is that we tend to operate under the belief that if our *sampled* data are normally distributed then the underlying *sampling distribution* is also normally distributed. Also, keep in mind that this becomes less of a concern as our sample size increases (thank, Central Limit Theorem!).

There are several tests for normality that we will discuss, which include either (a) calculations or (b) visual examination. We will discuss both.

#### 4.1.1.1 Visual check of normality

You can accomplish this with a histogram (e.g., `hist()` or `geom_histogram()` or a q-q plot `qplot()` (which stands for quantile-quantile).

### 4.1.2 Homogeneity of variance

Here, you want to know whether the variance of a variable is the same across different groups. For example, if you are looking at test scores in chemistry and chemical engineering students, you want to know if the variances (spread) of the test scores in the chemistry group and the chemical engineering group are close to each other.

### 4.1.3 Interval data

This might be a little redundant given that we want normally distributed data, but you want at least interval data (ratio data are also fine, but in practice very few things we work with actually qualify as ratio variables). If you have ordinal or nominal variables, you might be in trouble with this assumption…

### 4.1.4 Independence

This assumption is about the observations not being related to each other or affecting each other in some way. In practice, this can also be a little tricky. For example, if you are sampling students from different classrooms, depending on the variables you are measuring, you might actually have reason to believe that students in one classroom are more related to each other than students in a different classroom. In practice, you can handle this with a multi-level model (aka hierarchical model), but that is beyond the scope of this class.

## 4.2 Correlation

### 4.2.1 Covariance

First, start with the observation that variance is calculated with: $Variance(s^2) = \frac{\sum(x_i - \overline{x})^2}{N-1} = \frac{\sum(x_i - \overline{x})(x_i - \overline{x})}{N-1}$

But now let's say that we want to know how, for each observation we have, how does the value of $x$ vary with the value of $y$ on average. For example, when the value of $x$ increases, does the value of $y$ also increase? This could happen when $x$ represents the number of hours of sleep you get each night and $y$ is your average grade on an exam you take the next day. The opposite could arise when $x$ increases but we expect $y$ to go down. An example of this might be when $x$ is the number of hilarious jokes that a teacher tells in class and $y$ is the number of students who fall asleep in class. As the number of jokes increases, we might expect/hope that it keeps students' attention and keeps them from dozing. This generally process of considering how one variable changes when another variable changes is where the notion of covariance comes in.

In practice, what we really want to know is: when $x_i$ is above its average value in a sample ($\overline{x}$), how does $y_i$ change? Does it also tend to be above the sample average for $y$ ($\overline{y}$)? This is expressed in the general formular for covariance:

$cov(x, y) = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{N-1}$

While covariance can be a helpful value to work with in many settings, for most of what we do in this class, we will be using correlation coefficients instead of covariance. This is because covariance is an unnormalized value, which can make comparisons across different ranges of values difficult.

### 4.2.2 Correlation coefficient

In order to standardize the covariance to a value the is easier to work with across ranges of values, we use the correlatoin coefficient. There are several version of this, depending on the type of data you are working with. The most basic version is the Pearson correlation coefficient. It is calculated by dividing the covariance by the standard deviations of your two variables of interest:

$r = \frac{cov_{xy}}{s_x s_y} = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{(N-1)s_x s_y}$

This is a *bivariate* correlation coefficient because it is looking at the correlation between *two variables*. There are also partial correlation coefficients, which look at the correlation between two variables whil controlling for other variables.

We can calculate the correlatoin between two variables using the `cor()` or `cor.test()` functions, which are part of base R.

## 4.3 Another worked example for cleaning and prelim analysis

This script takes an incomplete subset of senior data from a .csv file, cleans it, computes factor scores, and prepares it for analysis.

If you have not already done so, make sure that you have run `library(tidyverse)` and `library(psych)` since we will be using functions from both of those packages.

### 4.3.1 Loading in data

First, as usual, load in your data. We will use the file `seniorsurvey.csv` for this demo.

`file_path <- "YOUR PATH HERE"` `setwd(file_path)` use this command to change the working directory to the folder where you have your file `list.files()` run this to make sure that your file is in your current working directory

```
seniorSurvey_df <- read_csv("seniorsurvey.csv") # replace text in the parentheses with
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

### 4.3.2 Data prep and cleaning

After loading, it is always nice to just see how things loaded in. Functions like str() and describe() from the psych package are nice for this. For example, if we use describe(), we can see the following (we deleted some variables):

```
psych::describe(seniorSurvey_df)
```

```
##                                                    vars    n  mean    sd median
## What is your PRIMARY MAJOR?                           1 1849 31.79 20.43     29
## Internship, field experience, co-op, or practicum     2 1121  1.00  0.00      1
## ParticipateServiceL                                   3  489  1.00  0.00      1
## ParticipateCService                                   4 1296  1.00  0.00      1
## ParticipateStudyAbroadSemester                        5  142  1.00  0.00      1
## SJ1                                                   6 1733  2.32  1.03      2
## SJ2                                                   7 1732  2.08  0.96      2
## SJ3                                                   8 1731  2.77  0.88      3
## SJ4                                                   9 1726  2.27  1.01      2
## SJ5                                                  10 1728  3.27  0.92      3
## SJ6                                                  11 1719  3.50  0.83      4
## SJ7                                                  12 1719  4.01  0.79      4
## SJ8                                                  13 1719  4.15  0.83      4
## DA1                                                  14 1719  2.23  0.93      2
## DA2                                                  15 1719  2.86  0.95      3
## DA3                                                  16 1720  1.97  0.81      2
## DA4                                                17 1721  4.20  0.72      4
## DA5                                                18 1721  4.06  0.81      4
## LocalRole                                          19 1453  3.49  0.94      4
## LocalFinance                                       20 1453  3.22  0.90      3
## LocalTime                                          21 1453  3.58  0.89      4
## GlobalRole                                         22 1446  3.57  0.99      4
## GlobalFinance                                      23 1449  3.19  1.00      3
## GlobalTime                                         24 1449  3.42  0.98      3
## Your gender?                                       25 1678  1.49  0.50      1
##                                                  trimmed   mad min max range
## What is your PRIMARY MAJOR?                        31.25 28.17   1  70    69
## Internship, field experience, co-op, or practicum   1.00  0.00   1   1     0
## ParticipateServiceL                                 1.00  0.00   1   1     0
## ParticipateCService                                 1.00  0.00   1   1     0
## ParticipateStudyAbroadSemester                      1.00  0.00   1   1     0
## SJ1                                                 2.25  1.48   1   5     4
## SJ2                                                 1.98  1.48   1   5     4
## SJ3                                                 2.79  1.48   1   5     4
## SJ4                                                 2.20  1.48   1   5     4
## SJ5                                                 3.27  1.48   1   5     4
## SJ6                                                 3.54  1.48   1   5     4
## SJ7                                                 4.07  0.00   1   5     4
## SJ8                                                 4.23  1.48   1   5     4
## DA1                                                 2.15  1.48   1   5     4
## DA2                                                 2.89  1.48   1   5     4
## DA3                                                 1.90  0.00   1   5     4
## DA4                                                 4.27  1.48   1   5     4
## DA5                                                 4.11  1.48   1   5     4
## LocalRole                                           3.51  1.48   1   5     4
## LocalFinance                                        3.23  1.48   1   5     4
## LocalTime                                           3.63  1.48   1   5     4
## GlobalRole                                          3.62  1.48   1   5     4
## GlobalFinance                                       3.18  1.48   1   5     4
## GlobalTime                                          3.44  1.48   1   5     4
## Your gender?                                        1.49  0.00   1   2     1
##                                                   skew kurtosis   se
## What is your PRIMARY MAJOR?                        0.18    -1.33 0.48
## Internship, field experience, co-op, or practicum   NaN      NaN 0.00
## ParticipateServiceL                                 NaN      NaN 0.00
## ParticipateCService                                 NaN      NaN 0.00
## ParticipateStudyAbroadSemester                      NaN      NaN 0.00
## SJ1                                                0.46    -0.51 0.02
## SJ2                                                0.64    -0.16 0.02
## SJ3                                               -0.03    -0.14 0.02
## SJ4                                                0.52    -0.40 0.02
## SJ5                                               -0.19    -0.03 0.02
## SJ6                                                    -0.40     0.15 0.02
## SJ7                                                    -0.75     1.05 0.02
## SJ8                                                    -0.86     0.69 0.02
## DA1                                                     0.57    -0.11 0.02
## DA2                                                    -0.09    -0.41 0.02
## DA3                                                     0.70     0.31 0.02
## DA4                                                    -0.66     0.54 0.02
## DA5                                                    -0.59     0.16 0.02
## LocalRole                                              -0.44    -0.33 0.02
## LocalFinance                                           -0.19    -0.27 0.02
## LocalTime                                              -0.59     0.22 0.02
## GlobalRole                                             -0.47    -0.30 0.03
## GlobalFinance                                          -0.11    -0.33 0.03
## GlobalTime                                             -0.35    -0.17 0.03
## Your gender?                                            0.03    -2.00 0.01
```

Upon examining this, we can notice a few things: Primary Major variable is all messed up. We won't fix it here, but basically there is a numeric code needed (e.g., 13 = underwater basket weaving)

Columns 3 and 5 have lots of missing values (note the small N's) – this means that this was asked via checkbox so (1) is true and missing is not missing but False

SJ1-8 and DA1-5 all look essentially ok – about the same N (some survey fatigue or skips) but all values in range (1-5)

Now, we know that SJ and DA are scales from the literature and we want to compute scale scores for those. Typically for attitude scales like these we just report means across the items. So, we will use the "psych" package to use a built in function to help us with this. If you have not used psych yet, be sure it is installed using the command install.packages("psych") – you need only do this once and then in subsequent uses you only need `library(psych)` to tell R to look in that package for the functions you will be using.

```
library(psych)
```

Subset out only the SJ and DA items in their own dataframe and then use tools in the psych package to compute scale means

The first method to do this - use numbering of the columns:

```
seniorSurveyScales_df <- seniorSurvey_df[6:18]
```

A second method to do this - use select() from dplyr

```
seniorSurveyScales_df <- seniorSurvey_df %>% select(SJ1:DA5)
```

Use the make.keys() function from psych package to key-in how the scales are built (mapping items to scales, use - for reverse scored items)

```
my_keys <- make.keys(seniorSurveyScales_df, list(SJCa=c(-1,-2,-3,-4),SJCh=c(5,6,7),DA=c(-9,-10,-1
```

Use scoreItems function to score each respondent on the three scales of interest SJCa, SJCh, and DA – the default here in scoreItems is to takes the mean of the items (not additive though that is sometimes used) and also, it imputes missing values instead of dropping cases the scoreItems function calculates many things. At this stage, all we really want are the scores, so we include a line to only extract that info.

```
my_scales <- scoreItems(my_keys, seniorSurveyScales_df)
my_scores <- my_scales$scores
```

Now, if you view the first few rows of the my.scores vector using the header – head() command – it looks like we expect:

```
head(my_scores)
```

```
##      SJCa     SJCh  DA
## [1,] 2.75 3.000000 3.2
## [2,] 3.75 3.333333 4.2
## [3,] 3.00 3.000000 3.0
## [4,] 2.25 4.333333 3.6
## [5,] 3.00 3.333333 3.4
## [6,] 4.50 4.333333 3.4
```

Now, let's build a clean dataframe to prep for analysis - by clean in this case I mean that we have replaced item scores from the scales with their means and also that we have fixed the NAs that don't belong (for participation variables, in this dataset, the NAs should be 0s)

```
my_df <- data.frame(seniorSurvey_df[1:5],my_scores, seniorSurvey_df[19:25])
```

This is an old school method to replace NAs in specific columns

```
my_df$ParticipateServiceL[is.na(my_df$ParticipateServiceL)] <- 0
my_df$ParticipateCService[is.na(my_df$ParticipateCService)] <- 0
my_df$ParticipateStudyAbroadSemester[is.na(my_df$ParticipateStudyAbroadSemester)] <- 0
```

my_df*ParticipateInternCoop...*[*is.na*(*mydf*ParticipateInternCoop...)] <- 0 — —- this variable read in cumbersomely named and I don't care about it right now so I'll skip

Here is An alternative method to replace NAs in specific columns:

```
my_df <- my_df %>%
  replace_na(list(ParticipateCService = 0, ParticipateStudyAbroadSemester = 0, Particip
```

### 4.3.3 Preliminary analysis

At this point, we are ready for some analysis

Let's investigate correlations. What seems most obvious would just be to run cor() but, as we found out in class, this can cause us to run full speed ahead without considering assumptions

```
my_correlations <- my_df %>% select(SJCa,SJCh,DA) %>% cor()
print(my_correlations)
```

```
##           SJCa      SJCh        DA
## SJCa 1.0000000 0.2590211 0.3342276
## SJCh 0.2590211 1.0000000 0.2310703
## DA   0.3342276 0.2310703 1.0000000
```

Ok, so, it is important that we note that this ran correlations but R doesn't know that this was sample data and therefore that we are interested instatistical significance (or not) of these results AND that our data may need another method (e.g., non-parametric). cor() does have a way to run spearman instead.

```
my_spearman_correlations <- my_df %>% select(SJCa,SJCh,DA) %>% cor(method="spearman")
print(my_spearman_correlations)
```

```
##           SJCa      SJCh        DA
## SJCa 1.0000000 0.2727828 0.3148168
## SJCh 0.2727828 1.0000000 0.2340174
## DA   0.3148168 0.2340174 1.0000000
```

If we need p values though, we need to change to something else – corr.test

```
my_results <- corr.test(my_df$SJCa,my_df$DA)
```

Then we can pull out results from this list or print it. Let's do both.

```
print(my_results,short=FALSE)
```

```
## Call:corr.test(x = my_df$SJCa, y = my_df$DA)
## Correlation matrix
## [1] 0.33
## Sample Size
## [1] 1852
## Probability values  adjusted for multiple tests.
## [1] 0
##
##  Confidence intervals based upon normal theory.  To get bootstrapped values, try cor.ci
##       raw.lower raw.r raw.upper raw.p lower.adj upper.adj
## NA-NA      0.29  0.33      0.37     0      0.29      0.37
```

```
my_results$r # correlation coefficient
```

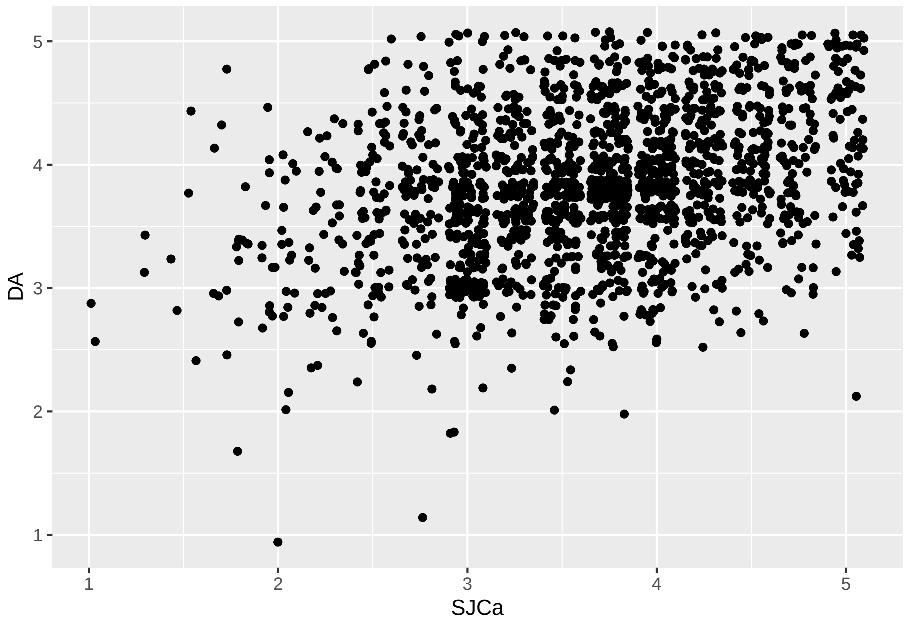
```
## [1] 0.3342276
```

```
my_results$p # p-value
```

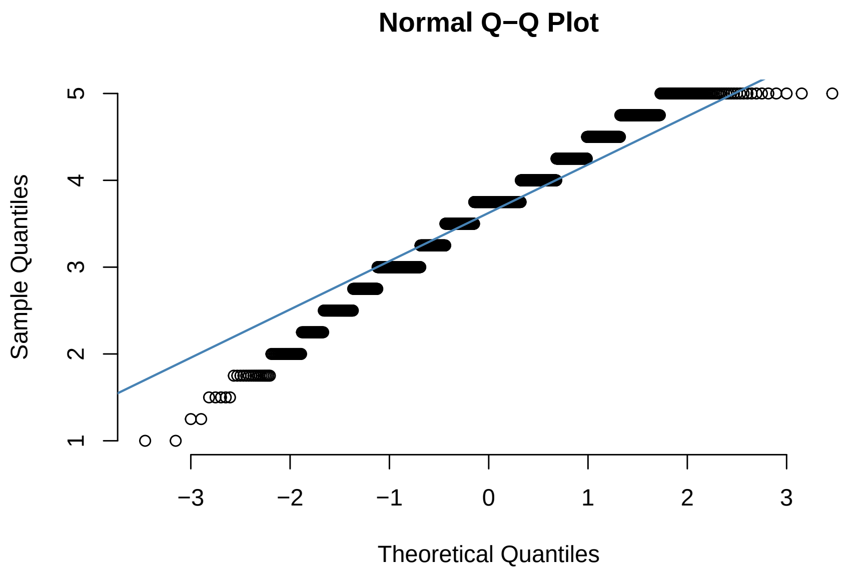
```
## [1] 1.433637e-49
```

Visually, we should be able to see this on a scatterplot. We are going to use qplot which stands for quickplot from within ggplot. It is useful and quicker for simple plotting than building up ggplot (though from the same package) we need to jitter my points (take geom="jitter" out if you want to see why)

```
qplot(SJCa,DA,data=my_df,geom="jitter")
```

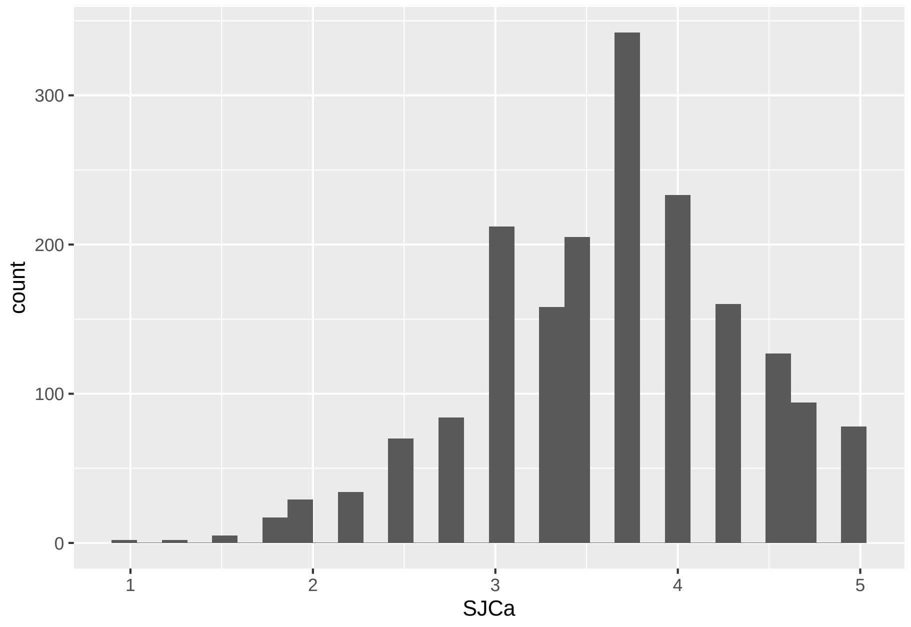


```
qqnorm(my_df$SJCa, frame = FALSE)
qqline(my_df$SJCa, col = "steelblue", lwd = 1.5)
```



```
my_df %>% ggplot(aes(x = SJCa)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Other functions we used today in class were describe() and also the q-q plot creation to investigate normality assumption copying syntax from the Field, Miles, & Field book

# Chapter 5

# Week 5: Simple Regression

```r
require(tidyverse)
require(psych)
require(kableExtra)
```

```
## Loading required package: kableExtra

##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```r
library(broom)
```

This week we will start learning about linear regression. In particular, we focus on simple regression. These kinds of models involve one predictor variable and one continuous outcome variable. Next week we will move to models with multiple regression, which involves - you guessed it - multiple predictor variables.

## 5.1 General Modeling Philosophy

The general approach is to model the outcome variable as a function of some predictor(s) plus an error term. Mathematically, this looks like:

$outcome_i = model + error_i$

where the $i$ subscript refers to the $i^{th}$ person in the sample.

##### 5.1.0.1 Review of the normal distribution and standardizing variables

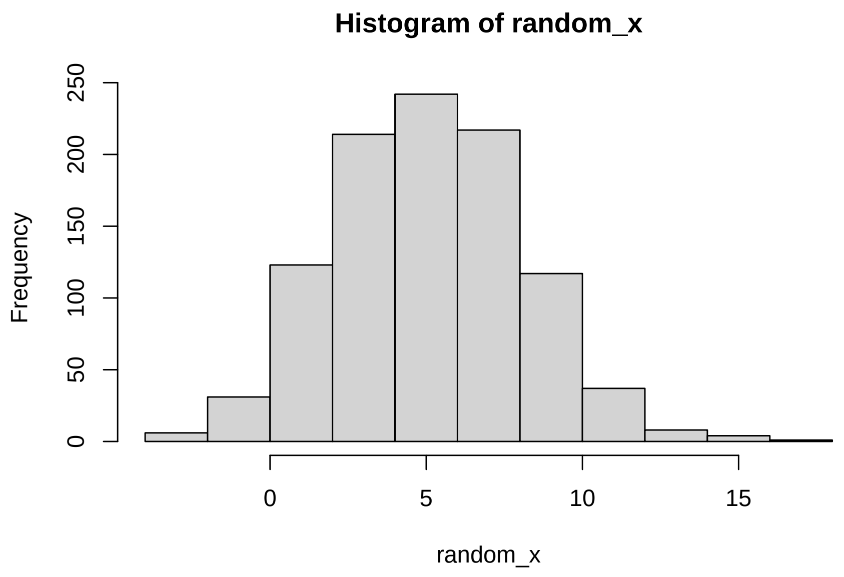
Just to review, let's think about normally distributed variables and the notion of centering and standardizing.

First, we will generate some data by drawing `n` random numbers from a normal distribution with a mean and standard deviation that we will specify.

```
mean <- 5
sd <- 3
n <- 1000

random_x <- rnorm(n = n, mean = mean, sd = sd)
```

We can then visualize those numbers

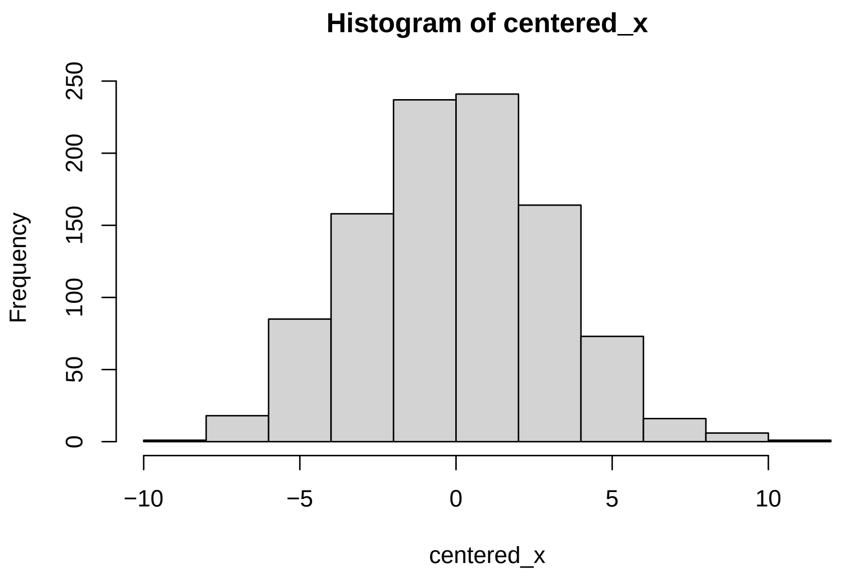
```
hist(random_x)
```



Now if we subtract the mean and plot the histogram, notice how the values have all basically shifted to the left along the x-axis.

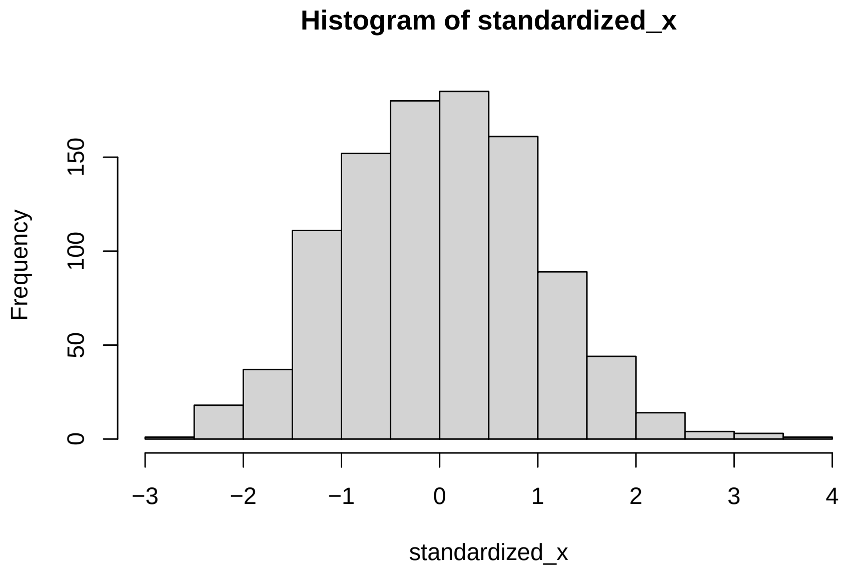
```
sample_mean <- mean(random_x)

centered_x <- random_x - sample_mean
```

```r
hist(centered_x)
```



Finally, we can divide by the sample standard deviation, which should have the effect of either stretching or squishing the values along the x-axis (without changing their mean). Pay attention again to the values along the x-axis.

```r
sample_sd <- sd(random_x)

standardized_x <- centered_x / sample_sd

hist(standardized_x)
```



This final plot should remind you have the standard normal plot (with mean 0 and standard deviation 1). This is noted as $x \sim \mathcal{N}(0,1)$ and is read as "x is distributed according to a normal distribution with a mean of 0 and variance of 1".

## 5.2 Data generation demo - one set sample size

The following is a demo from class, found in the week_5_demo.R file

```
#store the sample size that we want to use
samp_size <- 100
```

```
# uniformly sample X values (values for our predictor variable) from 0 to 20
x <- round(runif(n = samp_size, min = 0, max = 30), digits = 1) # this gives samp_size
```

Store the noise values for our different test models

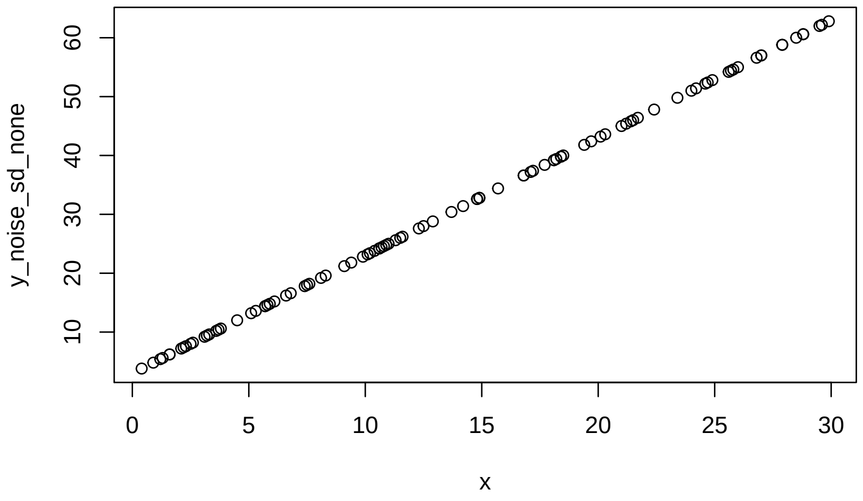
```
sd_min <- 2 # low noise
sd_med <- 6 # medium noise
sd_max <- 12 # high noise
```

Generate the outcome variable values under different amounts of noise (the rnorm() function is what is generating noise here)

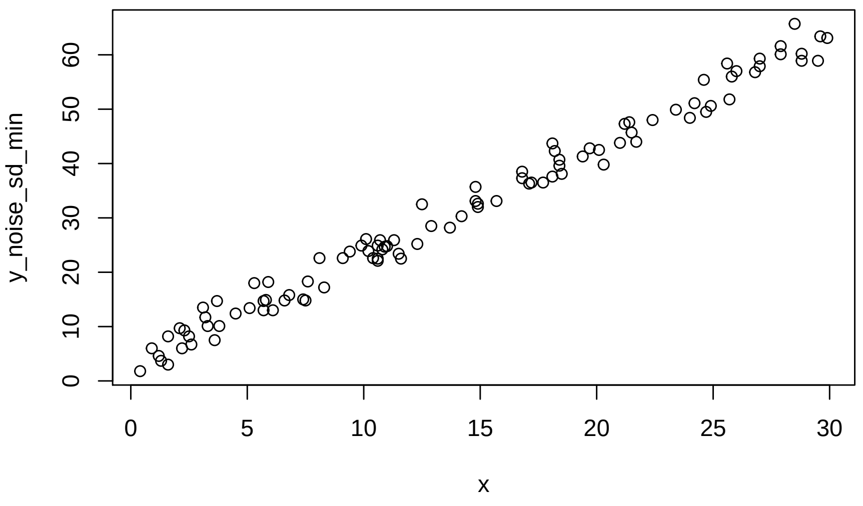
```
y_noise_sd_none <- 3 + 2*x # this is the true relationship without any noise
y_noise_sd_min <- 3 + 2*x + round(x = rnorm(n = samp_size, mean = 0, sd = sd_min), digits = 1)
y_noise_sd_med <- 3 + 2*x + round(x = rnorm(n = samp_size, mean = 0, sd = sd_med), digits = 1)
y_noise_sd_max <- 3 + 2*x + round(x = rnorm(n = samp_size, mean = 0, sd = sd_max), digits = 1)
```

Typical step 1: visualize! Let's plot each of these x values vs y

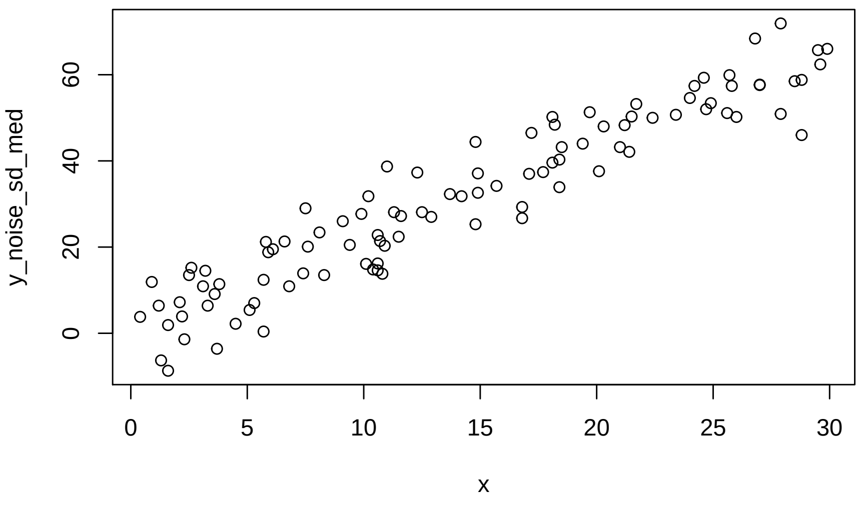
```
plot(x, y_noise_sd_none)
```



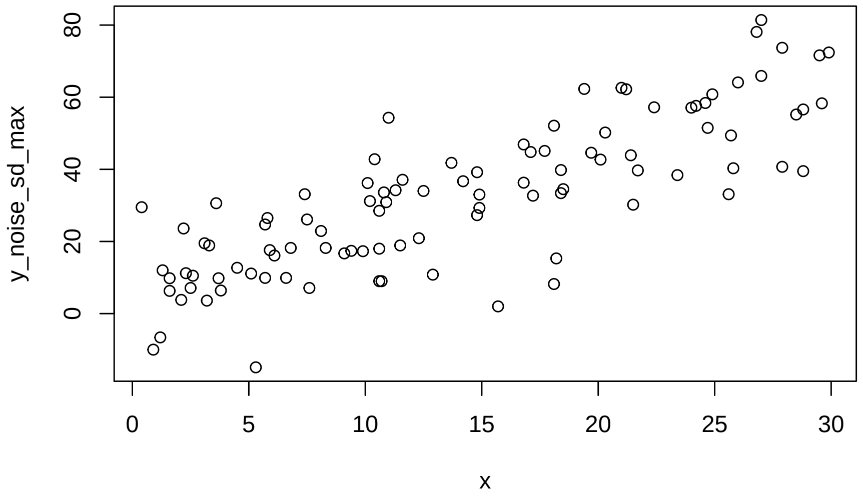
```
plot(x, y_noise_sd_min)
```



```
plot(x, y_noise_sd_med)
```



```
plot(x, y_noise_sd_max)
```



Let's put all of these vectors together into a data frame to make it easier to analyze later on Note, this is not a vital step for conducting the simple regression

```
demo_df <- tibble("x" = x,
                  "y_noise_sd_none"=y_noise_sd_none,
                  "y_noise_sd_min" = y_noise_sd_min,
                  "y_noise_sd_med" = y_noise_sd_med,
                  "y_noise_sd_max" = y_noise_sd_max)
```

Check out what demo_df looks like

```
head(demo_df)
```

```
## # A tibble: 6 x 5
##       x y_noise_sd_none y_noise_sd_min y_noise_sd_med y_noise_sd_max
##   <dbl>           <dbl>          <dbl>          <dbl>          <dbl>
## 1  14.9            32.8           32.6           32.6           33  
## 2   2.6             8.2            6.7           15.2           10.5
## 3   9.9            22.8           24.9           27.7           17.3
## 4  24.2            51.4           51.1           57.4           57.6
## 5  21              45             43.8           43.2           62.6
## 6  16.8            36.6           37.3           26.7           36.3
```

Order by increasing x value

```
demo_df <- demo_df %>%
  arrange(x)
```

Check out what the arrange() function did

```
head(demo_df)
```

```
## # A tibble: 6 x 5
##       x y_noise_sd_none y_noise_sd_min y_noise_sd_med y_noise_sd_max
##   <dbl>           <dbl>          <dbl>          <dbl>          <dbl>
## 1   0.4             3.8            1.8            3.8           29.5
## 2   0.9             4.8            6             11.9          -10
## 3   1.2             5.4            4.6            6.4           -6.6
## 4   1.3             5.6            3.7           -6.3           12
## 5   1.6             6.2            8.2           -8.7            6.3
## 6   1.6             6.2            3              1.9            9.8
```

Let's make this a long df so that we can plot multiple standard deviation values together

```
demo_df_long <- demo_df %>%
  pivot_longer(cols = starts_with("y_noise"),
               names_to = "y_col",
               values_to = "y_val"
  )
```

Again, check on what this did

```
head(demo_df_long)
```

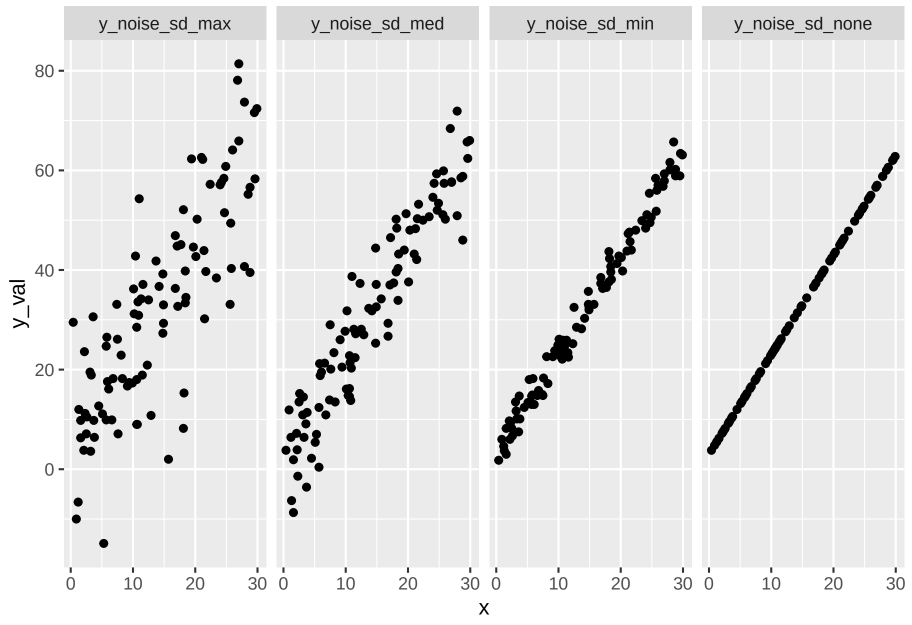
```
## # A tibble: 6 x 3
##       x y_col           y_val
##   <dbl> <chr>           <dbl>
## 1   0.4 y_noise_sd_none   3.8
## 2   0.4 y_noise_sd_min    1.8
## 3   0.4 y_noise_sd_med    3.8
## 4   0.4 y_noise_sd_max   29.5
## 5   0.9 y_noise_sd_none   4.8
## 6   0.9 y_noise_sd_min    6
```

Let's add in a column to note whether the value is from the min, med, max, or zero sd (noise) model

```
demo_df_long <- demo_df_long %>%
  mutate(sd_val = case_when(str_detect(y_col, "sd_none") ~ 0,
                            str_detect(y_col, "sd_min") ~ sd_min,
                            str_detect(y_col, "sd_med") ~ sd_med,
                            str_detect(y_col, "sd_max") ~ sd_max))
```

Use facet_grid to separate the plots out by

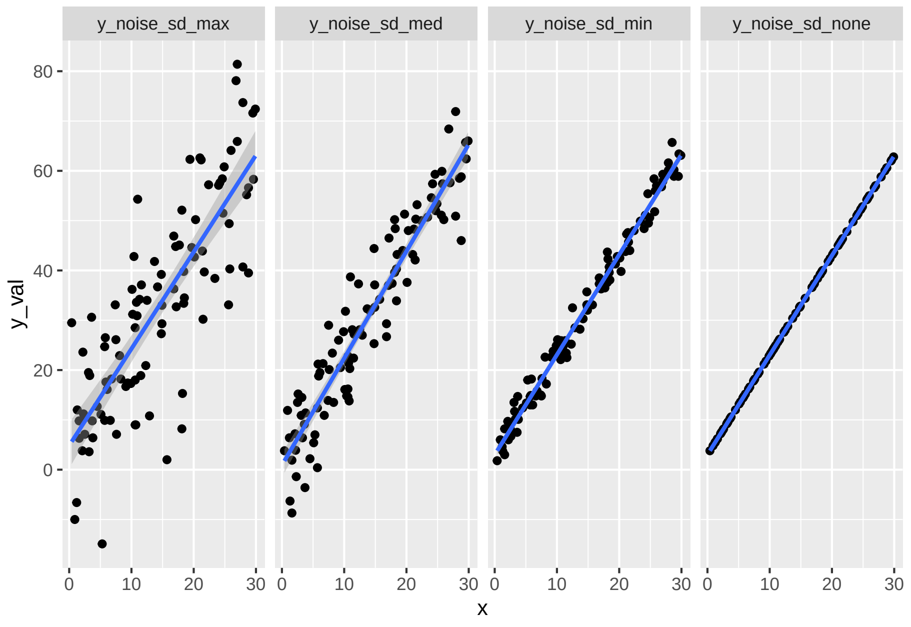
```
demo_df_long %>%
  ggplot(aes(x = x, y = y_val)) +
  geom_point() +
  facet_grid(.~y_col)
```



You can also automatically add in a line with the geom_smooth() function

```
demo_df_long %>%
  ggplot(aes(x = x, y = y_val)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_grid(.~y_col)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Now we can create a linear model for the data with minimum noise with the following command:

```
fit_demo_min <- lm(y_noise_sd_min ~ x)
```

…and we can look at the summary of the model with:

```
summary(fit_demo_min)
```

```
##
## Call:
## lm(formula = y_noise_sd_min ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9636 -1.5735 -0.1374  1.4690  5.4681
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.99463    0.41102   7.286 8.18e-11 ***
## x            2.00833    0.02483  80.892  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 2.155 on 98 degrees of freedom
## Multiple R-squared:  0.9852, Adjusted R-squared:  0.9851
## F-statistic:  6543 on 1 and 98 DF,  p-value: < 2.2e-16
```

We can also look at model results with the glance() function from the broom package

```
broom::glance(fit_demo_min)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.985         0.985  2.15     6543. 1.54e-91     1  -218.  441.  449.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

We can create models for the med and max sd values as well and take a look at those with the summary() function once again

```
fit_demo_med <- lm(y_noise_sd_med ~ x)
summary(fit_demo_med)
```

```
##
## Call:
## lm(formula = y_noise_sd_med ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.7568  -4.1599  -0.1476   4.2789  14.1738
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.90051    1.21652    0.74    0.461
## x            2.14779    0.07348   29.23   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.377 on 98 degrees of freedom
## Multiple R-squared:  0.8971, Adjusted R-squared:  0.896
## F-statistic: 854.3 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
fit_demo_max <- lm(y_noise_sd_max ~ x)
summary(fit_demo_max)
```

```
##
## Call:
## lm(formula = y_noise_sd_max ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -33.355  -6.300  -0.224   7.972  28.081
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.835      2.318   2.086   0.0396 *
## x              1.944      0.140  13.884   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.15 on 98 degrees of freedom
## Multiple R-squared:  0.663,  Adjusted R-squared:  0.6595
## F-statistic: 192.8 on 1 and 98 DF,  p-value: < 2.2e-16
```

Notice the increase in the standard error of the coefficient estimates as the noise in y values went up

From a programming perspective, this was not very efficient because I just copied, pasted, and corrected these values. There is a better way to do this using lists (see below)

Let's do some fancy stuff to make multiple models at once rather than having to write new lines for each model *Some of these ideas are taken from the R4DS book chapter 25

```
test_nest <- demo_df_long %>% nest(data = -sd_val)


linear_model <- function(df) {
  lm(y_val ~ x, data = df)
}


models <- map(test_nest$data, linear_model)
```

```
summary(models[[2]])
```

```
##
## Call:
## lm(formula = y_val ~ x, data = df)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9636 -1.5735 -0.1374  1.4690  5.4681
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.99463    0.41102   7.286 8.18e-11 ***
## x            2.00833    0.02483  80.892  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.155 on 98 degrees of freedom
## Multiple R-squared:  0.9852, Adjusted R-squared:  0.9851
## F-statistic:  6543 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
summary(models[[3]])
```

```
##
## Call:
## lm(formula = y_val ~ x, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.7568  -4.1599  -0.1476   4.2789  14.1738
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.90051    1.21652    0.74    0.461
## x            2.14779    0.07348   29.23   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.377 on 98 degrees of freedom
## Multiple R-squared:  0.8971, Adjusted R-squared:  0.896
## F-statistic: 854.3 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
summary(models[[4]])
```

```
##
## Call:
## lm(formula = y_val ~ x, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -33.355  -6.300  -0.224   7.972  28.081
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.835      2.318   2.086   0.0396 *
## x              1.944      0.140  13.884   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.15 on 98 degrees of freedom
## Multiple R-squared:  0.663,  Adjusted R-squared:  0.6595
## F-statistic: 192.8 on 1 and 98 DF,  p-value: < 2.2e-16
```

We can also store the models as new columns in the nested dataframe

```
test_nest <- test_nest %>%
  mutate(model = map(data, linear_model))
```

Finally, we can unnest the models to make it easier to compare them with each other in a data frame

```
test_nest <- test_nest %>%
  mutate(glance = map(model, broom::glance)) %>%
  unnest(glance)
```

```
## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable

## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable
```

## 5.3 Data generation demo - one set sample size;

The change from the past demo is that we are now sampling from integer values rather than continuous for the predictor

Store the sample size that we want to use

```
samp_size <- 200
```

Instead of sampling uniformly from 0 to 20, this is to sample integers from 40 to 100 uniformly. We take “samp_size” number of samples. Replace = TRUE means we can get the same x value multiple times

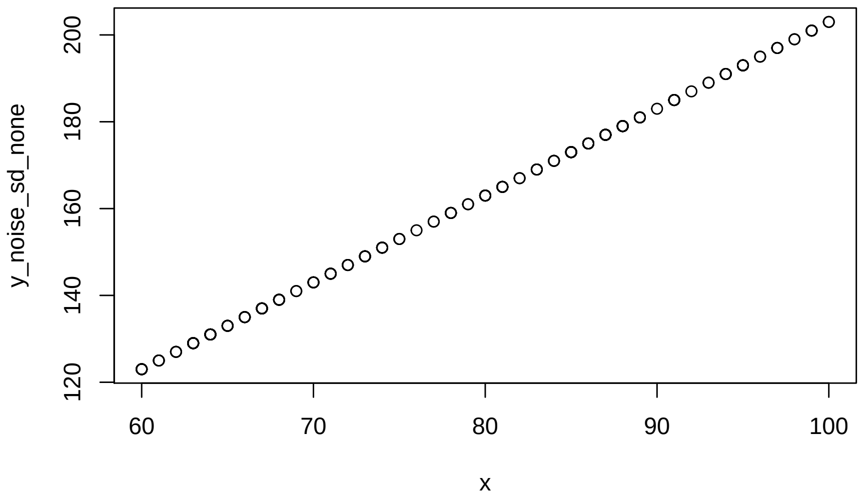
```
x <- sample(x = c(60:100), size = samp_size, replace = TRUE)
```

As before, store the noise values for our different test models

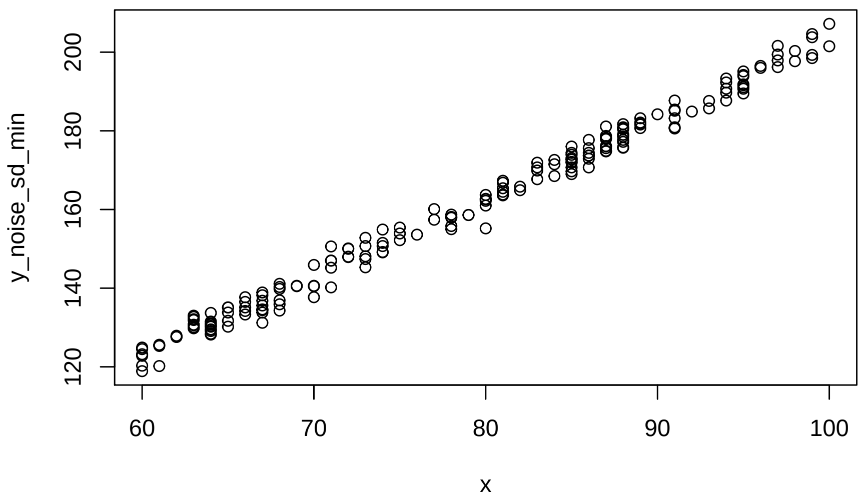
```
sd_min <- 2
sd_med <- 6
sd_max <- 12

y_noise_sd_none <- 3 + 2*x
y_noise_sd_min <- 3 + 2*x + round(x = rnorm(n = samp_size, mean = 0, sd = sd_min), digits = 1)
y_noise_sd_med <- 3 + 2*x + round(x = rnorm(n = samp_size, mean = 0, sd = sd_med), digits = 1)
y_noise_sd_max <- 3 + 2*x + round(x = rnorm(n = samp_size, mean = 0, sd = sd_max), digits = 1)
```

Typical step 1: visualize! Let's plot each of these x values vs y

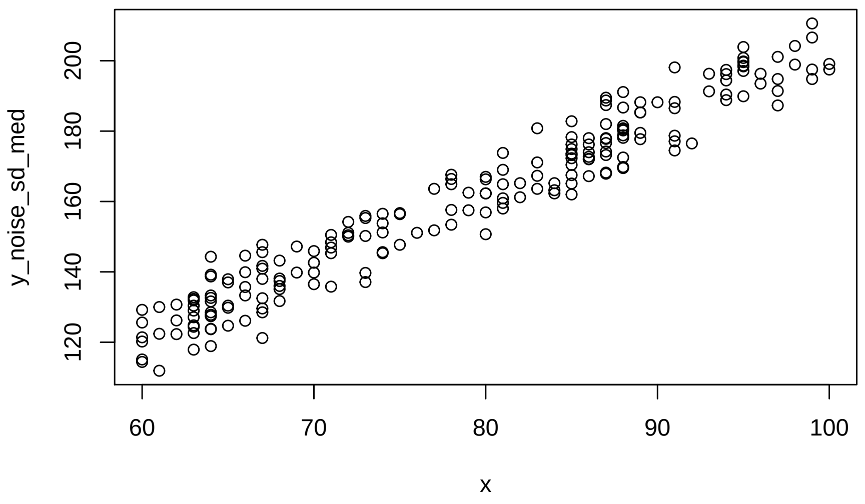
```
plot(x, y_noise_sd_none)
```



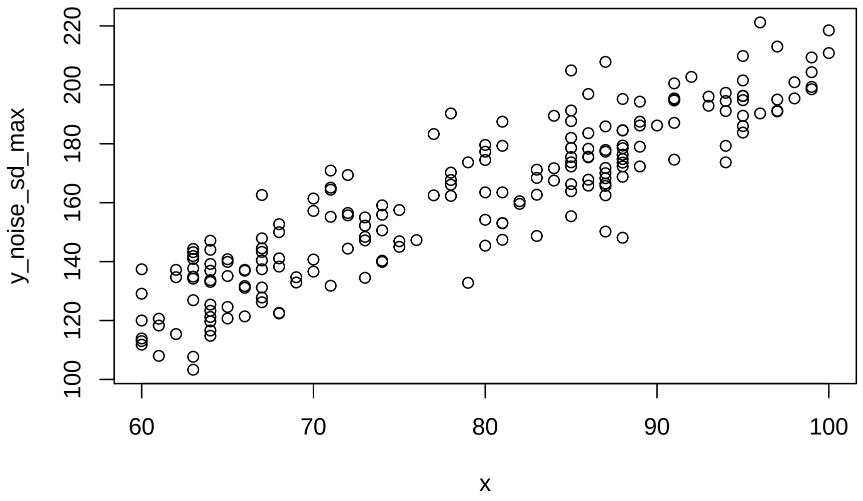
```
plot(x, y_noise_sd_min)
```



```
plot(x, y_noise_sd_med)
```



```
plot(x, y_noise_sd_max)
```



Let's put all of these vectors together into a data frame to make it easier to analyze later on. Note, this is not a vital step for conducting the simple regression

```
demo_df <- tibble("x" = x,
                  "y_noise_sd_none"=y_noise_sd_none,
                  "y_noise_sd_min" = y_noise_sd_min,
                  "y_noise_sd_med" = y_noise_sd_med,
                  "y_noise_sd_max" = y_noise_sd_max)
```

Order by increasing x value

```
demo_df <- demo_df %>%
  arrange(x)
```

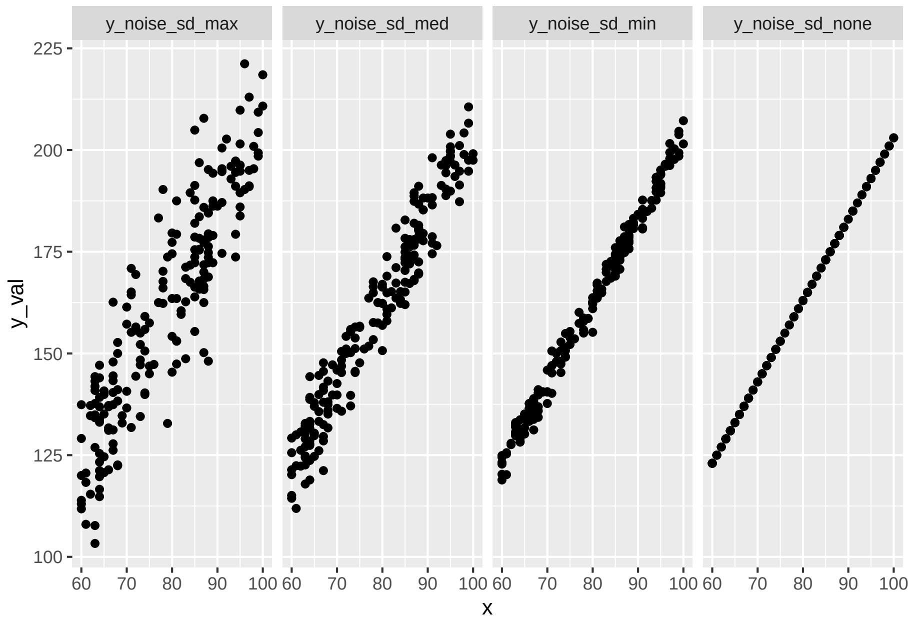
Let's make this a long df so that we can plot multiple standard deviation values together

```
demo_df_long <- demo_df %>%
  pivot_longer(cols = starts_with("y_noise"),
               names_to = "y_col",
               values_to = "y_val"
  )
```

```
demo_df_long <- demo_df_long %>%
  mutate(sd_val = case_when(str_detect(y_col, "sd_none") ~ 0,
                            str_detect(y_col, "sd_min") ~ sd_min,
                            str_detect(y_col, "sd_med") ~ sd_med,
                            str_detect(y_col, "sd_max") ~ sd_max))
```

And visualize the data, faceting by different noise

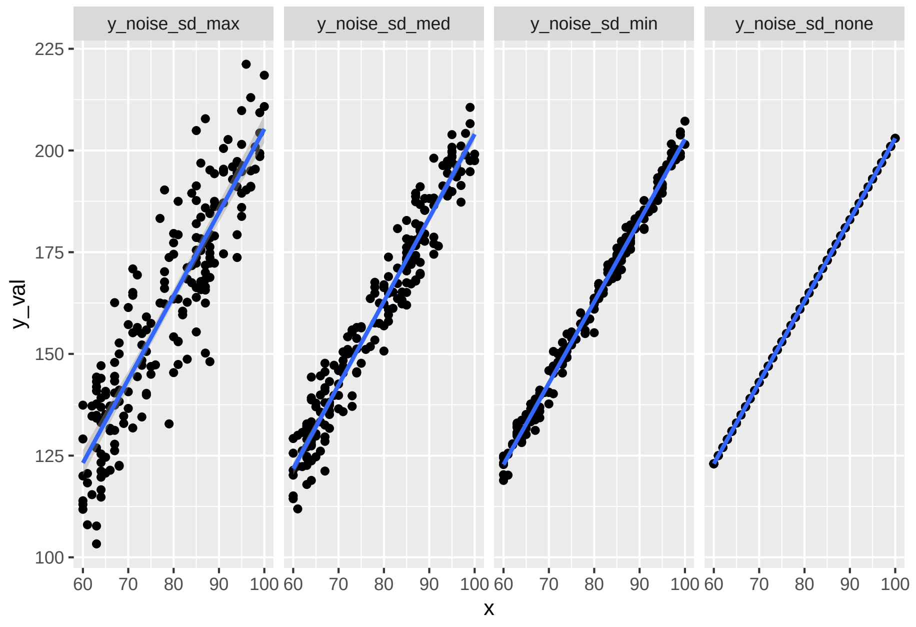
```
demo_df_long %>%
  ggplot(aes(x = x, y = y_val)) +
  geom_point() +
  facet_grid(.~y_col)
```



And add in a line with `geom_smooth(method = 'lm')`

```
demo_df_long %>%
  ggplot(aes(x = x, y = y_val)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_grid(.~y_col)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Create a linear model and look at the summary.

```
fit_demo_min <- lm(y_noise_sd_min ~ x)
summary(fit_demo_min)
```

```
##
## Call:
## lm(formula = y_noise_sd_min ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5887 -1.5825 -0.0405  1.6148  5.8059
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.83684    1.08534   2.614  0.00964 **
## x            1.99940    0.01365 146.470  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.256 on 198 degrees of freedom
## Multiple R-squared:  0.9909, Adjusted R-squared:  0.9908
## F-statistic: 2.145e+04 on 1 and 198 DF,  p-value: < 2.2e-16
```

We can also look at model results with the glance() function from the broom package

```
broom::glance(fit_demo_min)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic   p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>     <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.991         0.991  2.26    21454. 8.16e-204     1  -446.  897.  907.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

We can create models for the med and max sd values as well and take a look at those with the summary() function once again.

```
fit_demo_med <- lm(y_noise_sd_med ~ x)
summary(fit_demo_med)
```

```
##
## Call:
## lm(formula = y_noise_sd_med ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.9883  -4.5766   0.1295   4.5445  14.2781
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.52696    2.88759  -0.529    0.598
## x            2.05545    0.03632  56.596   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.003 on 198 degrees of freedom
## Multiple R-squared:  0.9418, Adjusted R-squared:  0.9415
## F-statistic:  3203 on 1 and 198 DF,  p-value: < 2.2e-16
```

```
fit_demo_max <- lm(y_noise_sd_max ~ x)
summary(fit_demo_max)
```

```
##
## Call:
## lm(formula = y_noise_sd_max ~ x)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -32.563  -8.642  -0.499   7.510  30.394
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05934    5.63131   0.011    0.992
## x            2.05232    0.07083  28.977   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.71 on 198 degrees of freedom
## Multiple R-squared:  0.8092, Adjusted R-squared:  0.8082
## F-statistic: 839.7 on 1 and 198 DF,  p-value: < 2.2e-16
```

Notice the increase in the standard error of the coefficient estimates as the noise in y values went up

From a programming perspective, this was not very efficient because I just copied, pasted, and corrected these values. There is a better way to do this using lists (see below)

Let's do some fancy stuff to make multiple models at once rather than having to write new lines for each model *Some of these ideas are taken from the R4DS book chapter 25

```r
test_nest <- demo_df_long %>% nest(data = -sd_val)


linear_model <- function(df) {
  lm(y_val ~ x, data = df)
}


models <- map(test_nest$data, linear_model)
```

```r
summary(models[[2]])
```

```
##
## Call:
## lm(formula = y_val ~ x, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5887 -1.5825 -0.0405  1.6148  5.8059
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.83684    1.08534   2.614  0.00964 **
## x            1.99940    0.01365 146.470  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.256 on 198 degrees of freedom
## Multiple R-squared:  0.9909, Adjusted R-squared:  0.9908
## F-statistic: 2.145e+04 on 1 and 198 DF,  p-value: < 2.2e-16
```

```
summary(models[[3]])
```

```
##
## Call:
## lm(formula = y_val ~ x, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.9883  -4.5766   0.1295   4.5445  14.2781
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.52696    2.88759  -0.529    0.598
## x            2.05545    0.03632  56.596   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.003 on 198 degrees of freedom
## Multiple R-squared:  0.9418, Adjusted R-squared:  0.9415
## F-statistic:  3203 on 1 and 198 DF,  p-value: < 2.2e-16
```

```
summary(models[[4]])
```

```
##
## Call:
## lm(formula = y_val ~ x, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.563  -8.642  -0.499   7.510  30.394
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05934    5.63131   0.011    0.992
## x            2.05232    0.07083  28.977   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.71 on 198 degrees of freedom
## Multiple R-squared:  0.8092, Adjusted R-squared:  0.8082
## F-statistic: 839.7 on 1 and 198 DF,  p-value: < 2.2e-16
```

We can also store the models as new columns in the nested dataframe

```
test_nest <- test_nest %>%
  mutate(model = map(data, linear_model))
```

Finally, we can unnest the models to make it easier to compare them with each other in a data frame

```
test_nest <- test_nest %>%
  mutate(glance = map(model, broom::glance)) %>%
  unnest(glance)
```

```
## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable

## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable
```

## 5.4 Data generation with three different sample sizes

Let's run the same demo but now have three different sample sizes - 10, 50, and 500

First, store the sample sizes we want to use

```
samp_sizes <- c(10, 50, 500)
```

Next, create a bookkeping column for ourselves to keep track of which sample size the future values will come from

```
samp_size_col <- rep(x = c(10,50, 500), times = samp_sizes)
```

Calculate the total number of values we will need from the three samples combined

```
tot_samp_size <- sum(samp_sizes)
```

Sample uniformly from 0 to 20

```
x <- round(x = runif(n = tot_samp_size, min = 0, max = 20), digits = 1)
```

Store the standard deviations for the min, med, and max models

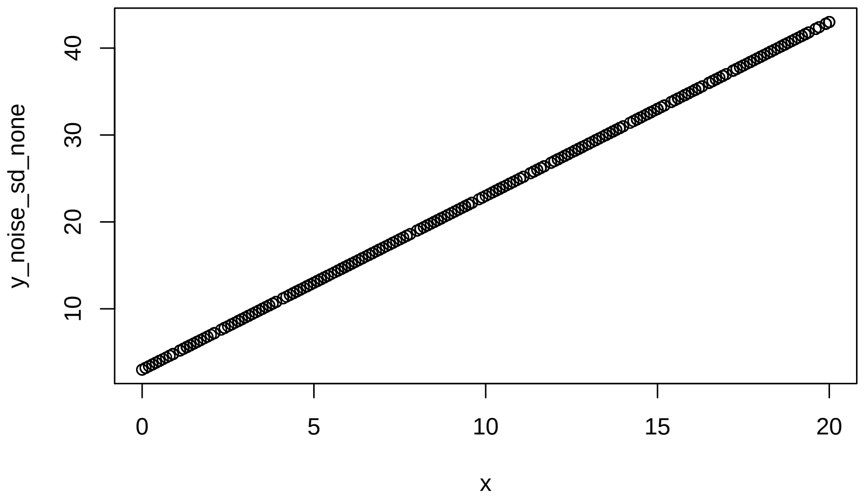
```
sd_min <- 2
sd_med <- 6
sd_max <- 12
```

Calculate the y values for the different scenarios where there is no noise up to max noise

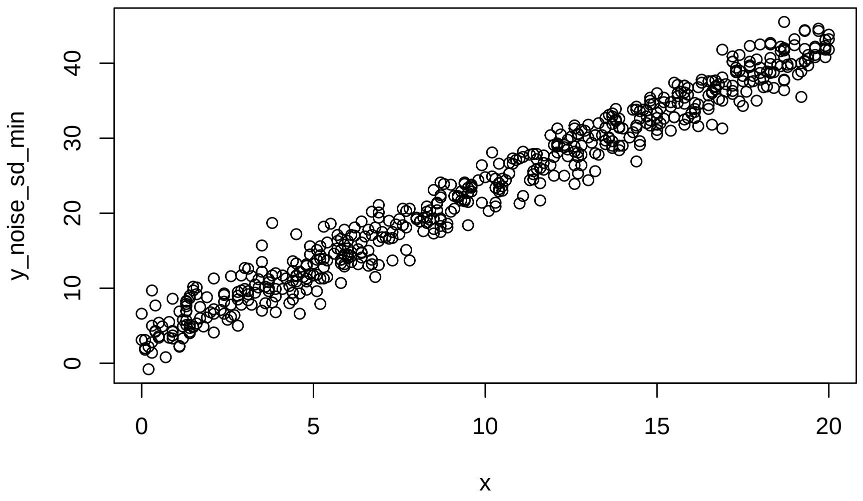
```
y_noise_sd_none <- 3 + 2*x
y_noise_sd_min <- 3 + 2*x + round(x = rnorm(n = tot_samp_size, mean = 0, sd = sd_min),
y_noise_sd_med <- 3 + 2*x + round(x = rnorm(n = tot_samp_size, mean = 0, sd = sd_med),
y_noise_sd_max <- 3 + 2*x + round(x = rnorm(n = tot_samp_size, mean = 0, sd = sd_max),
```

Typical step 1: visualize! Let's plot each of these x values vs y

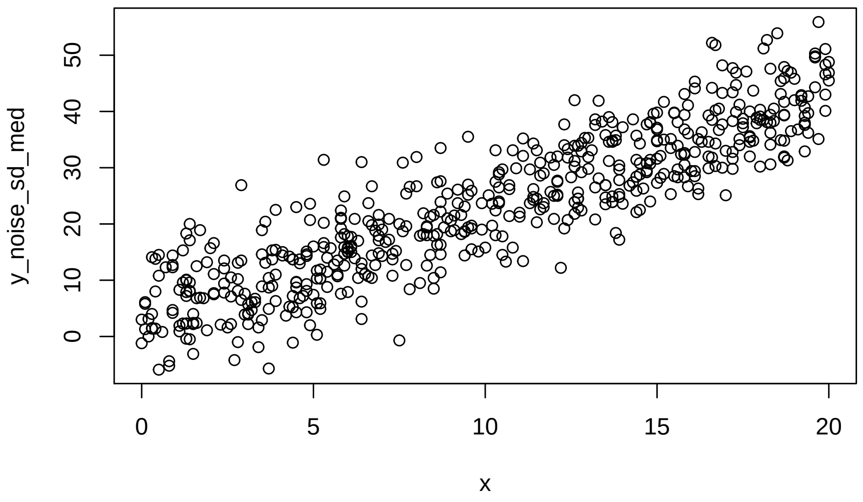
```
plot(x, y_noise_sd_none)
```



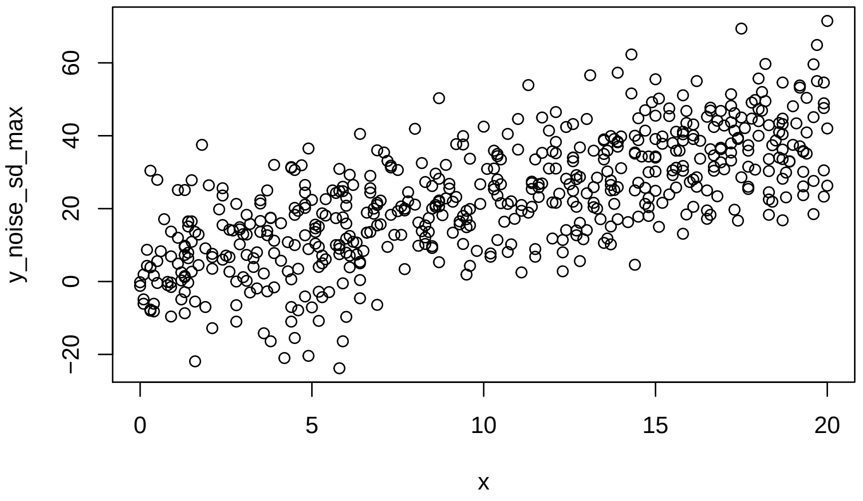
```
plot(x, y_noise_sd_min)
```



```
plot(x, y_noise_sd_med)
```



```
plot(x, y_noise_sd_max)
```



Can we calculate the correlations between x and these different y values? (pro tip: yes)

Let's put all of these vectors together into a data frame to make it easier to analyze later on Note, this is not a vital step for conducting the simple regression

```
demo_df <- tibble("n" = samp_size_col,
                  "x" = x,
                  "y_noise_sd_none"=y_noise_sd_none,
                  "y_noise_sd_min" = y_noise_sd_min,
                  "y_noise_sd_med" = y_noise_sd_med,
                  "y_noise_sd_max" = y_noise_sd_max)
```

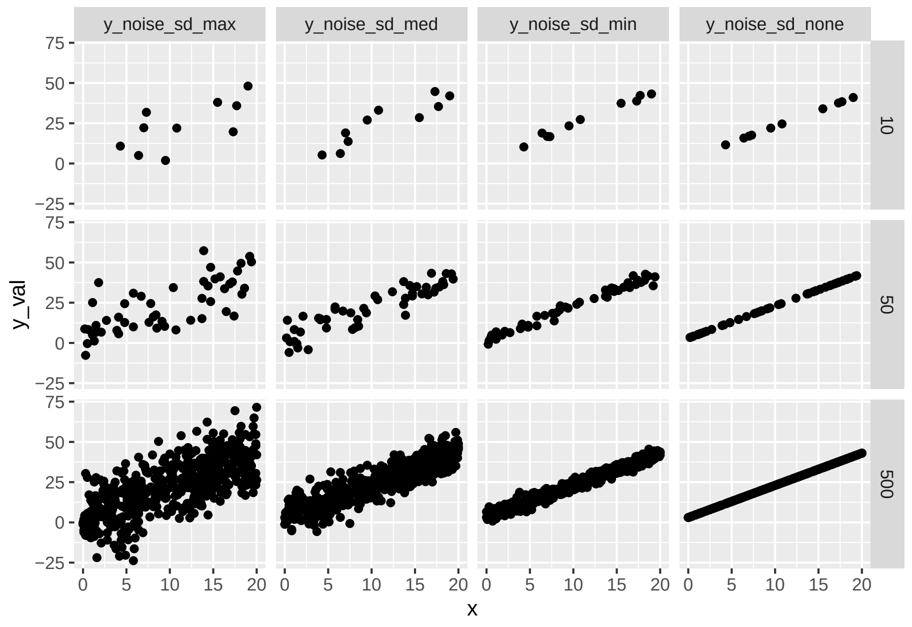
Order by increasing x value

```
demo_df <- demo_df %>%
  arrange(n, x)
```

Let's make this a long df so that we can plot multiple standard deviation values together

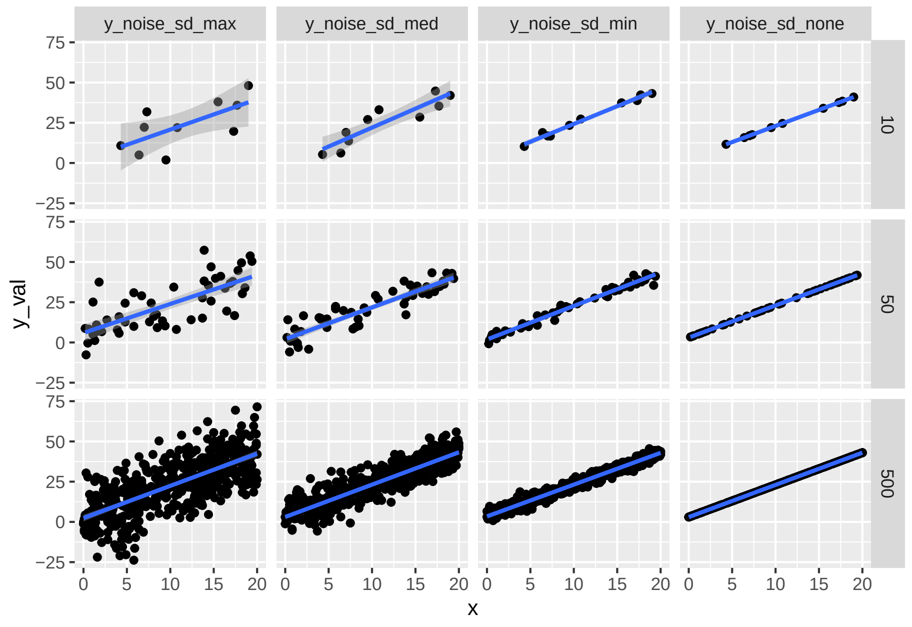
```
demo_df_long <- demo_df %>%
  pivot_longer(cols = starts_with("y_noise"),
               names_to = "y_col",
               values_to = "y_val"
  )

demo_df_long <- demo_df_long %>%
  mutate(sd_val = case_when(str_detect(y_col, "sd_none") ~ 0,
                            str_detect(y_col, "sd_min") ~ sd_min,
                            str_detect(y_col, "sd_med") ~ sd_med,
                            str_detect(y_col, "sd_max") ~ sd_max))
```

```
demo_df_long %>%
  ggplot(aes(x = x, y = y_val)) +
  geom_point() +
  facet_grid(n~y_col)
```



```
demo_df_long %>%
  ggplot(aes(x = x, y = y_val)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_grid(n~y_col)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
fit_demo_min <- lm(y_noise_sd_min ~ x)
summary(fit_demo_min)
```

```
##
## Call:
## lm(formula = y_noise_sd_min ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.9639 -1.2975  0.0287  1.4347  7.9107
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.22024    0.17470   18.43   <2e-16 ***
## x            1.99186    0.01498  132.96   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.07 on 558 degrees of freedom
## Multiple R-squared:  0.9694, Adjusted R-squared:  0.9693
## F-statistic: 1.768e+04 on 1 and 558 DF,  p-value: < 2.2e-16
```

We can also look at model results with the glance() function from the broom package

```
broom::glance(fit_demo_min)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.969         0.969  2.07    17677.       0     1 -1201. 2408. 2421.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

We can create models for the med and max sd values as well and take a look at those with the summary() function once again

```
fit_demo_med <- lm(y_noise_sd_med ~ x)
summary(fit_demo_med)
```

```
##
## Call:
## lm(formula = y_noise_sd_med ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.6897  -3.9551  -0.1918   3.9877  18.1683
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.89522    0.50178    5.77 1.32e-08 ***
## x            2.01259    0.04303   46.77  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.944 on 558 degrees of freedom
## Multiple R-squared:  0.7968, Adjusted R-squared:  0.7964
## F-statistic:  2188 on 1 and 558 DF,  p-value: < 2.2e-16
```

```
fit_demo_max <- lm(y_noise_sd_max ~ x)
summary(fit_demo_max)
```

```
##
## Call:
## lm(formula = y_noise_sd_max ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -38.020  -7.332   0.498   7.831  32.073
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.76569    0.97983   2.823  0.00493 **
## x            1.97494    0.08403  23.504  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.61 on 558 degrees of freedom
## Multiple R-squared:  0.4975, Adjusted R-squared:  0.4966
## F-statistic: 552.4 on 1 and 558 DF,  p-value: < 2.2e-16
```

Notice the increase in the standard error of the coefficient estimates as the noise in y values went up

From a programming perspective, this was not very efficient because I just copied, pasted, and corrected these values. There is a better way to do this using lists (see below)

Let's do some fancy stuff to make multiple models at once rather than having to write new lines for each model *Some of these ideas are taken from the R4DS book chapter 25

```
test_nest <- demo_df_long %>% nest(data = -c(sd_val, n))


linear_model <- function(df) {
  lm(y_val ~ x, data = df)
}


models <- map(test_nest$data, linear_model)

summary(models[[2]])
```

```
##
## Call:
## lm(formula = y_val ~ x, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5796 -1.2451 -0.2774  1.0443  2.6078
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.15945    1.19043   1.814    0.107
## x            2.20824    0.09463  23.335 1.21e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.539 on 8 degrees of freedom
## Multiple R-squared:  0.9855, Adjusted R-squared:  0.9837
## F-statistic: 544.5 on 1 and 8 DF,  p-value: 1.209e-08
```

```
summary(models[[3]])
```

```
##
## Call:
## lm(formula = y_val ~ x, data = df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -7.330 -4.372 -1.572  5.145  9.211
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.5375     4.7641  -0.323 0.755169
## x             2.3543     0.3787   6.217 0.000255 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.16 on 8 degrees of freedom
## Multiple R-squared:  0.8285, Adjusted R-squared:  0.8071
## F-statistic: 38.65 on 1 and 8 DF,  p-value: 0.0002548
```

```
summary(models[[4]])
```

```
##
## Call:
## lm(formula = y_val ~ x, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.9100  -6.7924   0.7142   7.0464  16.1344
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.9137     9.0180   0.212   0.8373
## x             1.8838     0.7169   2.628   0.0303 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.66 on 8 degrees of freedom
## Multiple R-squared:  0.4633, Adjusted R-squared:  0.3962
## F-statistic: 6.905 on 1 and 8 DF,  p-value: 0.03028
```

We can also store the models as new columns in the nested dataframe

```
test_nest <- test_nest %>%
  mutate(model = map(data, linear_model))
```

Finally, we can unnest the models to make it easier to compare them with each other in a data frame

```
test_nest <- test_nest %>%
  mutate(glance = map(model, broom::glance)) %>%
  unnest(glance)
```

```
## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable

## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable

## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable

## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable

## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable

## Warning in summary.lm(x): essentially perfect fit: summary may be unreliable
```

And look at the different models by just calling the data frame

```
test_nest
```

```
## # A tibble: 12 x 16
##         n sd_val data  model r.squared adj.r.squared    sigma statistic   p.value
##     <dbl>  <dbl> <lis> <lis>     <dbl>         <dbl>    <dbl>     <dbl>     <dbl>
##  1    10      0 <tib~ <lm>      1             1      2.04e-15   2.54e32 2.69e-127
##  2    10      2 <tib~ <lm>      0.986         0.984 1.54e+ 0   5.45e 2 1.21e-   8
##  3    10      6 <tib~ <lm>      0.828         0.807 6.16e+ 0   3.86e 1 2.55e-   4
##  4    10     12 <tib~ <lm>      0.463         0.396 1.17e+ 1   6.91e 0 3.03e-   2
##  5    50      0 <tib~ <lm>      1             1      4.20e-15   4.56e32 0.
##  6    50      2 <tib~ <lm>      0.978         0.977 2.04e+ 0   2.11e 3 2.52e- 41
## 7     50       6 <tib~ <lm>      0.827         0.823 5.85e+ 0   2.29e 2 6.75e- 20
## 8     50      12 <tib~ <lm>      0.536         0.526 1.09e+ 1   5.54e 1 1.55e-  9
## 9    500       0 <tib~ <lm>      1             1     2.55e-14   1.03e32 0.
## 10   500       2 <tib~ <lm>      0.969         0.969 2.06e+ 0   1.55e 4 0.
## 11   500       6 <tib~ <lm>      0.793         0.793 5.95e+ 0   1.91e 3 1.35e-172
## 12   500      12 <tib~ <lm>      0.496         0.495 1.17e+ 1   4.91e 2 3.61e- 76
## # ... with 7 more variables: df <dbl>, logLik <dbl>, AIC <dbl>, BIC <dbl>,
## #   deviance <dbl>, df.residual <int>, nobs <int>
```

# Chapter 6

# Week 6: Regression II

This is where the Week 6 magic happens...