

# Everyday Statistics CODE-ONLY

*Marieke Jones*

*3/18/2020*

## Goal

The goal of this session is to teach you how to conduct basic statistical analyses in R. We will cover descriptive statistics, exploratory data analysis, t-tests, ANOVA, and methods for discrete variable analysis along with hypothesis testing and assessing the assumptions of the tests.

## Setting up

Hopefully you got here by double-clicking the .Rproj file in the Instructional Materials. If not, exit out of RStudio and enter in through the .Rproj file so that you are in the proper folder.

Remember that an RStudio project (specifically the .Rproj file) makes it straightforward to place your work into it's own working directory. Creating a project takes away some of the stress of navigating through file directories and file paths. A project creates an encapsulation for source files, images, and anything else created during your R Session.

The next step of setting up will be to load packages we will need today.

```
# install.packages("inspectdf")  
# install.packages("broom")  
  
library(tidyverse)  
library(inspectdf)  
library(broom)
```

Remember that if your console says “Error package name not found”, you will need to install the package using `install.packages("packagename")`. I like to put the install lines in my script, run them to install the package, and then comment the line out so that installations will not get in the way of rendering the document (in case you want to do that).

As our final set-up step, let's load in the homes dataset that we have been using for the past few weeks. This is the version after all of the cleaning steps we did with David in the Data Prep lesson.

We will use the `read_csv()` function from the tidyverse to get our data into R. Once we've loaded it, we can type the name of the object itself (`homes`) to see it printed to the screen.

```
homes <- read_csv("albemarle_homes.csv")  
homes
```

Take a look at that output. The nice thing about loading the tidyverse and reading data with the `read_csv()` from readr is that data are displayed in a much more friendly way. This dataset has 31K+ rows and 27 columns. When you import/convert data this way and try to display the object in the console, instead of trying to display all of the rows, you'll only see 10 by default. The printing behavior of `read_csv()` causes 1000s of lines of output that are seldom helpful. Also, if you have so many columns that the data would wrap off the edge of your screen, those columns will not be displayed, but you'll see at the bottom of the output which, if any, columns were hidden from view.

Remember that if you want to see the whole dataset, there are two ways to do this. First, you can click on the name of the data.frame in the **Environment** panel in RStudio. Or you could use the `View()` function (*with a capital V*).

```
# View(homes)
```

Note: I have commented out the `View()` function because its behavior when rendering an Rmd is not desirable. You could also change the Rmd chunk options to prevent this chunk from being evaluated.

Recall several built-in functions that are useful for working with data frames.

- Content:
  - `head()`: shows the first few rows
  - `tail()`: shows the last few rows
- Size:
  - `dim()`: returns a 2-element vector with the number of rows in the first element, and the number of columns as the second element (the dimensions of the object)
  - `nrow()`: returns the number of rows
  - `ncol()`: returns the number of columns
- Summary:
  - `names()`: returns the column names
  - `glimpse()` (from **dplyr**): Returns a glimpse of your data, telling you the structure of the dataset and information about the class, length and content of each column
  - `summary()`: Returns a frequency count for factor variables and a 6 number summary for numeric variables

Let's try a few

```
glimpse(homes)
names(homes)
summary(homes)
```

## Descriptive statistics

We can access individual variables within a data frame using the `$` operator, e.g., `mydataframe$specificVariable`.

### Descriptive statistics for categorical variables

Printing out all the `esdistrict` values in the data will not be useful. It will print 1000 rows and then stop, trying to warn us that we didn't really want to do that.

```
# Display all esdistrict values
homes$esdistrict
```

Now, let's see what are the `unique` values of `esdistrict` using base R and then using a dplyr pipeline

```
# Get the unique values of Race
unique(homes$esdistrict)

# Do the same thing the dplyr way
homes %>% distinct(esdistrict)
```

Think about the difference between these two types of output produced by the base r function or by the dplyr equivalent. Sometimes one format is vastly preferred over another so it is good to note the difference.

```
#create a table of frequencies
table(homes$esdistrict) #base r
homes %>% count(esdistrict) #this returns a tibble

#2-factor table or cross tabulation
table(homes$hsdistrict, homes$esdistrict)

homes %>% count(hsdistrict, esdistrict)
```

Stony point elementary is the only ES that feeds into 2 different HS

## Descriptive statistics for categorical variables

Now let's calculate some descriptive stats for the **totalvalue** variable.

```
# measures of the middle: mean, median
mean(homes$totalvalue)
median(homes$totalvalue)

# measures of overall spread
range(homes$totalvalue)
sd(homes$totalvalue)
# the Interquartile Range (difference between 75th and 25th quartiles)
IQR(homes$totalvalue)

# quartiles (points that divide data into quarters)
quantile(homes$totalvalue)

# 90th percentile of total value
quantile(homes$totalvalue, probs = .90)

# a distributional summary
summary(homes$totalvalue)
```

What are your conclusions about this variable?

Recall the following about the mean and the median: - The median is the middle of the sorted data - The mean is the “balance point” of the data - Symmetric data have similar means and medians

You can calculate these same descriptive statistics using dplyr, but remember, this returns a single-row, single-column tibble, *not* a single scalar value like the above. This is only really useful in the context of grouping and summarizing.

```

# Compute the median totalvalue
homes %>%
  summarize(median(totalvalue))

# Now grouped by other variables
homes %>%
  group_by(esdistrict) %>%
  summarize(med = median(totalvalue))

# let's order that output
homes %>%
  group_by(esdistrict) %>%
  summarize(med = median(totalvalue)) %>%
  arrange(desc(med))

```

## Missing data

Let's try taking the mean of a different variable, either the dplyr way or the simpler \$ way.

```

# the dplyr way: returns a single-row single-column tibble/dataframe
homes %>% summarize(mean(totalrooms))

# returns a single value
mean(homes$totalrooms)

```

What happened there? NA indicates *missing data*. Take a look at a summary of the totalrooms variable.

```
summary(homes$totalrooms)
```

Notice that there are 9 missing values for totalrooms. Trying to get the mean a bunch of observations with some missing data returns a missing value by default. This is almost universally the case with all summary statistics – a single NA will cause the summary to return NA. Now look at the help for `?mean`. Notice the `na.rm` argument. This is a logical (i.e., TRUE or FALSE) value indicating whether or not missing values should be removed prior to computing the mean. By default, it's set to FALSE. Now try it again.

```
mean(homes$totalrooms, na.rm = TRUE)
```

The `is.na()` function tells you if a value is missing. Get the `sum()` of that vector, which adds up all the TRUEs to tell you how many of the values are missing.

```
is.na(homes$totalrooms)
```

```

sum(is.na(homes$totalrooms))

# proportion missing
sum(is.na(homes$totalrooms)) / length(homes$totalrooms)

```

What if we want to know the amount of missing data in each column? There's a package for that! Let's use the `inspect_na()` function in the `inspectdf` package to help. Remember that if you get an error message that the function cannot be found, go back to the top of the script and try the `library(inspectdf)` line. If that fails, `install.packages("inspectdf")`

```
inspect_na(homes) %>% print(n = Inf)
```

This function provides the count of missings and the proportion missing for each variable, sorted by amount missing.

### What to do about missing data?

This is a tough question, but an important one. If you have missing data and are wondering what to do about it, please request a consult with one of the PhD+ instructors, since our advice will depend a lot on the situation (how much is missing, is it missing at random or not, what are you trying to do with the data, etc.)

## EDA

Now, let's talk about exploratory data analysis (EDA).

It's always worth examining your data visually before you start any statistical analysis or hypothesis testing. We already spent an entire day on data visualizations, so I'll just remind us of a few of the big ones here: **histograms** and **scatterplots**. We will also re-familiarize ourselves with **density plots** when we go through some stats in a few minutes.

### Histograms

We can learn a lot from the data just looking at the distributions of particular variables. Let's make some basic histograms with ggplot2.

```
ggplot(homes, aes(finsqft)) + geom_histogram()
```

How would you describe the shape of this distribution?

What about age?

```
ggplot(homes, aes(age)) + geom_histogram()
```

What do you notice?

What is going on with age around 260

```
filter(homes, age < 270 & age > 250)
```

They do seem to be real cases. I would love to know what happened in Charlottesville in 1754 to prompt so many houses to be built (that are still around today)!

### Scatterplots

Let's look at how a few different variables relate to each other. E.g., lotsize and finsqft:

```
ggplot(homes, aes(lotsize, finsqft)) + geom_point()
```

Some very large homes on small lots. But (I think) lot size is in acres, so really we are looking at few properties that are > 300 acres.

## Stats with Continuous variables

A very well-known group of basic statistical tests is called the t-test. A t-test is used when we want to know the difference between 2 group means.

T-tests assume a few things about the data:

1. Data were randomly sampled from their populations
2. two groups are independent from one another
3. groups were sampled from populations with normal distributions (symmetrical, bell-shaped)
4. variance is equal for the two populations

As we go, we will learn what to do if assumptions do not hold, but for now, here is a brief flowchart: - Random Sampling – if not met, no statistics will help you - Independent Samples – if not met, use a paired t-test - Normality – if not met, use a Wilcoxon-Mann-Whitney U test - Equal variance – if not met, use a Welch's t-test - If independent samples and normality are both not met – use Wilcoxon signed rank test

### T-tests

Let's do a two-sample t-tests to assess the *difference in means between two groups*. The function for a t-test is `t.test()`. See the help using `?t.test`.

We'll investigate whether the condition of the home impacts its total value for homes that are either in Fair or Substandard condition.

Right now we have a census of all of the homes in Albemarle. Let's create a sample of homes that are either condition == "Fair" or condition == "Substandard" to use.

```
set.seed(317)
lessgood <- homes %>%
  sample_frac(.2) %>%
  filter(condition == "Fair" | condition == "Substandard")
```

Great!

To answer our questions about the assumptions, let's create some exploratory plots. I like density plots colored by group. I think these help you see the distribution of the variable to assess equal variance. This plot also informs you somewhat on normality, and helps you see if there is a noticeable difference in groups.

```
ggplot(lessgood, aes(totalvalue, fill = condition)) + geom_density()
```

```
# hmm, definitely not normally distributed, but lots of $ variables are log transformed, so let's try t.
ggplot(lessgood, aes(log(totalvalue), fill = condition, color = condition)) +
  geom_density(alpha = .5)
```

First we make sure that a t-test is the correct type of analysis. A t-test tests the difference in 2 means - yes that is what we want to do. Next we need to decide what type of t-test we need to perform by thinking through the assumptions. Domain specific knowledge and exploratory data analyses will help here.

Random sampling – YES, we sampled a random (pseudo-random, but good enough) subset of rows from our homes dataset

Independent samples – YES (fair and substandard are different homes - unrelated). It could be a paired t-test if we were assessing fair and substandard pairs where the two homes were matched by street or something

Equal variance. Also called homoscedasticity of variances. ?? we could think about the populations of fair and substandard homes and what we know about how those designations impact the variability of value... if we were an appraiser. I don't feel like I have the knowledge of these variables, so I will allow my EDA plots to help. Looking on the width of the bases of the density plots, they don't look too different. Maybe the substandard one is a bit wider, but I would likely call them the same in this case.

Normality – Our density plots don't look too bad, but there are better ways to assess this.

Normality can be assessed graphically or via hypothesis tests. There are pros and cons to either approach.

Graphically, we could look at a histogram, boxplot, or a more specialized plot to assess normality called a QQ plot (quantile-quantile plot or quantile comparison plot or normal probability plot). A QQ plot graphs the expected data value given a normal distribution on the X axis against the observed data value on the y axis. If the data is normally distributed, we should see a 1:1 ratio between the expected values and the observed values. Let's have a look for `log(totalvalue)`:

```
ggplot(lessgood, aes(sample = log(totalvalue))) +  
  geom_qq() +  
  geom_qq_line() +  
  facet_wrap(~condition)
```

```
# both look good, but look at that class imbalance
```

```
# count by condition
```

```
lessgood %>%  
  group_by(condition) %>%  
  count()
```

Learning what is a normal QQ plot looks like is a process.

Certain fields love hypothesis tests of normality and sometimes reviewers will specifically request one. There is a theoretical problem with trying to *prove* a null hypothesis and they are known to reject the null when sample sizes are large. My best advice is to use your brain, subject matter expertise, and graphical assessments as much as possible, but in case you are forced to do a hypothesis test for normality check out `shapiro.test()`, since it seems to be the least awful choice (<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3693611/>).

Ok. We've checked our assumptions and are ready to perform a two-sample, pooled variance t-test

```
t.test(log(totalvalue) ~ condition, data=lessgood, var.equal = TRUE)
```

```
# what if we had assumed unequal variance to be safe
```

```
t.test(log(totalvalue) ~ condition, data=lessgood)
```

In either case, we conclude that the condition groups are different in terms of their `log(totalvalue)`. We also can see a 95% confidence interval for the difference, meaning that we are 95% confident that the difference in *population* `log(totalvalue)` is between 0.39 and 0.89 (pooled) or between 0.31 and 0.97 (Welch's).

What if we felt uncomfortable with the assumption of normality? t-tests are robust to departures in normality, so in this case I think a t-test is the best option, but for demonstration purposes, the non-parametric alternative is the Wilcoxon-Mann-Whitney U test

```
wilcox.test(log(totalvalue) ~ condition, data=lessgood)
```

Notice that the output does not provide confidence intervals or summary statistics, but we can add a confidence interval for the difference and calculate the summary statistics ourselves.

```
# with confidence interval for the pseudo-median difference
wilcox.test(log(totalvalue) ~ condition, conf.int = TRUE, data=lessgood)

# calculate means and medians by group
lessgood %>%
  group_by(condition) %>%
  summarize(med = median(log(totalvalue)),
            mean = mean(log(totalvalue)))
```

**A note on one-tailed versus two-tailed tests:** A two-tailed test is usually more appropriate. The hypothesis you're testing here is spelled out in the results ("alternative hypothesis: true difference in means is not equal to 0"). If the p-value is very low, you can reject the null hypothesis that there's no difference in means. Because you may not know *a priori* whether the difference in means will be positive or negative, you want to do the two-tailed test. However, if we *only* wanted to test a very specific directionality of effect, we could use a one-tailed test and specify which direction we expect. This is more powerful if we "get it right", but much less powerful for the opposite effect. The p-value of a one-tailed test will be half of that of a two-tailed hypothesis test. BUT again, the **two-tailed test is almost always conducted** to be conservative.

**A note on paired versus unpaired t-tests:** The t-tests we performed here were unpaired tests. Fair and substandard homes are different and not related. In these cases, an *unpaired* test is appropriate. An alternative design might be when data is derived from samples who have been measured at two different time points or locations, e.g., before versus after treatment, left versus right hand, etc. In this case, a *paired t-test* would be more appropriate. A paired test takes into consideration the intra and inter-subject variability, and is more powerful than the unpaired test. There is a paired = TRUE option for both the t-test and the Wilcoxon test.

## Relationship between t-test ANOVA and Linear models

Analysis of variance (ANOVA) and linear modeling are complex topics that deserve an entire semester dedicated to theory, design, and interpretation. Luckily, next week we will be focused on linear modeling. What follows is a necessary over-simplification with more focus on implementation, and less on theory and design.

Where t-tests and their nonparametric substitutes are used for assessing the differences in means between two groups, ANOVA is used to assess the significance of differences in means between multiple groups. In fact, a t-test is just a specific case of ANOVA when you only have two groups. And both t-tests and ANOVA are just specific cases of linear regression, where you're trying to fit a model describing how a continuous outcome (e.g., log(totalvalue)) changes with some predictor variable (e.g., condition, hsdistrict, etc.). The distinction is that ANOVA has a categorical predictor while linear modeling has a continuous predictor.

Let's examine the same question we just used as a t-test, but run it as an ANOVA to prove that t-test and ANOVA are really the same test. Because ANOVA has an assumption of equal variance, let's run a t-test specifying that the variances between groups *are* equal.

```
t.test(log(totalvalue) ~ condition, data=lessgood, var.equal=TRUE)
```

It looks like fair homes have a higher log(totalvalue) than those in the substandard condition. Now, let's do the same test in a linear modeling framework. First, let's create the fitted model and store it in an object called `fit`.



```
fit <- lm(log(totalvalue) ~ condition, data=lessgood)
```

You can display the linear model object itself, but that isn't too interesting. You can get the more familiar ANOVA table by calling the `anova()` function on the `fit` object. More generally, the `summary()` function on a linear model object will tell you much more. (Note this is different from dplyr's `summarize` function).

```
fit
anova(fit)
summary(fit)
```

Notice from these pieces of output that:

1. The p-values from all three tests (t-test, ANOVA, and linear regression) are all identical ( $p=4.79e-07$ ). This is because the tests are all identical: a t-test is a specific case of ANOVA, which is a specific case of linear regression.
2. The test statistics are all related. The  $t$  statistic from the t-test is **5.1591**, which is the same as the  $t$ -statistic from the linear regression. If you square that, you get **26.62**, the  $F$  statistic from the ANOVA.
3. The `t.test()` output shows you the means for the two groups, Fair and Substandard. Just displaying the `fit` object itself or running `summary(fit)` shows you the coefficients as a linear model. Here, the model assumes the “baseline” condition level is *Fair* (first alphabetically), and that the *intercept* in a regression model (e.g.,  $\beta_0$  in the model  $Y = \beta_0 + \beta_1 X$ ) is the mean of the Fair group (12.25 is the mean  $\log(\text{totalvalue})$  for Fair homes). Being *Substandard* results in a decrease in  $\log(\text{totalvalue})$  of 0.64. This is the  $\beta_1$  coefficient in the model.

**A note on dummy coding:** If you have a  $k$ -level factor, R creates  $k - 1$  dummy variables, or indicator variables, by default, using the alphabetically first level as the reference group. For example, the levels of condition in our lessgood dataset are “Fair” and “Substandard”. R creates a dummy variable called “conditionSubstandard” that's **0** if the home is Fair, and **1** if the home is Substandard. The linear model is saying for every unit increase in conditionSubstandard, i.e., going from Fair to Substandard, results in a 0.64-unit decrease in  $\log(\text{totalvalue})$ . You can change the ordering of the factors to change the interpretation of the model (e.g., treating Substandard as baseline and going from Substandard up to Fair). We'll do this in the next section.

## ANOVA

Recap: t-tests are for assessing the differences in means between *two* groups. A t-test is a specific case of ANOVA, which is a specific case of a linear model. Let's run ANOVA, but this time looking for differences in means between more than two groups.

Let's look at the relationship between `hsdistrict`, and  $\log(\text{totalvalue})$

We'll use the same technique as before to create a sample

```
set.seed(317)
hs <- homes %>%
  sample_frac(.005)

hs %>% count(hsdistrict)
```

We have pretty balanced sample sizes. This is good for ANOVA. If you can choose to design your study with balanced sample sizes, ANOVA will perform better (better power, higher probability of equal variances assumption being met, etc).

First let's run a grouped summary to see what we have in terms of differences in means

```
hs %>%
  group_by(hsdistrict) %>%
  summarize(logmean = mean(log(totalvalue)),
            mean = exp(logmean))
```

Ok, so Western Albemarle seems higher than the other two hsdistricts.

Now that we understand these data a bit, let's proceed to fit the model

```
fit <- lm(log(totalvalue) ~ hsdistrict, data=hs)
anova(fit)
```

The F-test on the ANOVA table tells us that there *is* a significant difference in mean log(totalvalue) between the high school districts ( $p=2.015 \times 10^{-6}$ ).

For more details, let's take a look at the linear model output

```
summary(fit)
exp(12.63487) # $307K
exp(12.63487-0.07278) # $285K
exp(12.63487+0.50269) # $508K
```

Interpretation: - the mean log(totalvalue) for the Albemarle hsdistrict = 12.63 or  $3.0708197 \times 10^5$  in actual dollars - the Monticello hsdistrict has a mean log(totalvalue) -0.07 less than that of Albemarle.  $2.8552646 \times 10^5$  in real dollars. That difference is not statistically significant - the Western Albemarle district has a mean log(totalvalue) 0.50 more than that of Albemarle.  $5.0765634 \times 10^5$  in real dollars. That difference is statistically significant

The bottom of the output has the ANOVA information and model summary statistics.

Because the default handling of categorical variables is to treat the alphabetical first level as the baseline, "Albemarle" high school district is treated the baseline group (the intercept row) and the coefficients for "Monticello" and "Western Albemarle" describe how those groups' mean log(totalvalue) differs from that of the Albemarle hsdistrict.

What if we wanted "Western Albemarle" to be the reference category, followed by Albemarle, then Monticello? Let's use `mutate()` and `factor()` to change the factor levels.

```
hs <- hs %>%
  mutate(hsdistrict = factor(hsdistrict, levels = c("Western Albemarle", "Albemarle", "Monticello")))
```

Re-run the model to see how re-leveling the variable changes the output

```
fit <- lm(log(totalvalue) ~ hsdistrict, data=hs)
anova(fit)
```

Same F statistic and p-value overall. Let's have a look at the summary

```
summary(fit)
```

Now Western Albemarle is the reference group and the comparisons are to that hsdistrict.

So far, we have been looking at model output as a large paragraph. If you need to do something downstream with the output from these models, the `tidy()` and `glance()` functions in the broom package may help. For example, these functions can help tremendously if you'd like to output the results of a model into a table. These functions work for t-test, `wilcox.test`, ANOVA, and linear models (and maybe other types of tests too).

```
# coefficients section
tidy(fit)

# model summary section
glance(fit)
```

If you wanted to remain in the ANOVA framework, you can run the typical post-hoc ANOVA procedures on the fit object. For example, the `TukeyHSD()` function will run *Tukey's test*. Tukey's test computes all pairwise mean difference calculation, comparing each group to each other group, identifying any difference between two groups that's greater than the standard error, while controlling the type I error for all multiple comparisons. First run `aov()` (**not** `anova()`) on the fitted linear model object, then run `TukeyHSD()` on the resulting analysis of variance fit.

```
TukeyHSD(aov(fit))
tidy(TukeyHSD(aov(fit)))

plot(TukeyHSD(aov(fit)))
```

This shows that there isn't much of a difference between Albemarle and Monticello, but that both of these differ significantly from Western Albemarle that has a higher `log(totalvalue)`.

Finally, let's visualize the differences between these groups.

```
# plot results
hs %>%
  ggplot(aes(hsdistrict, log(totalvalue))) + geom_boxplot()
```

```
# a fancier plot using a grouped summary
hssum <- hs %>%
  group_by(hsdistrict) %>%
  summarise(mean = mean(log(totalvalue)),
            sd = sd(log(totalvalue)),
            lower = mean - sd,
            upper = mean + sd)

ggplot() +
  geom_jitter(data = hs, aes(x= hsdistrict,
                             y = log(totalvalue),
                             col = hsdistrict),
              width = .1, alpha = .25) +
  geom_pointrange(data = hssum, aes(x = hsdistrict,
                                    y = mean,
                                    ymin = lower,
```

```

                                ymax = upper,
                                col = hsdistrict),
                                size = 1, fatten = 2) +
theme_classic() +
labs(y = "Log-transformed Total Home Value",
     x = "",
     col = "High School District")

```

## Discrete variable Statistics

Until now we've only discussed analyzing *continuous* outcomes (dependent variables). We've tested for differences in means between two groups with t-tests, differences among means between  $n$  groups with ANOVA, and more general relationships using linear regression. In all of these cases, the dependent variable, i.e., the outcome, or  $Y$  variable, was *continuous*. What if our outcome variable is *discrete*, e.g., "Yes/No", "Excellent/Fair/Substandard", etc.? Here we use a different set of procedures for assessing significant associations.

So far we have covered: 1. T-tests – analyzing differences in one continuous variable between 2 groups 2. ANOVA – analyzing differences in one continuous variable between 3+ groups 3. LM – Next week Clay will go through analyzing the impact of a continuous variable on another continuous variable, but we did an introduction to the functions we will need.

### Test of proportion

One of the most basic questions we could have is what proportion of homes are above half a million dollars? We'll generate a sample to see the proportion

```

homes %>%
  sample_frac(0.01) %>%
  summarize(prop = mean(totalvalue > 500000))

```

Run the above code again and we will see that a different sample of homes will lead to a different sample proportion. One of our questions should be what is the variability in that sample proportion? (Remember we usually do not have access to the whole population like we do here)

```

# generate a sample that is the same for all of us
set.seed(317)
samp <- homes %>%
  sample_frac(0.01)

# see the sample proportion and the raw number
samp %>%
  summarize(prop = mean(totalvalue > 500000),
            n = sum(totalvalue > 500000))

```

Let's create a confidence interval around this proportion

```
prop.test(x = sum(samp$totalvalue > 500000), n = length(samp$totalvalue))
```

We are 95% confident that the population proportion of homes > .5M in Albemarle county is between 0.19 and .29

The hypothesis test this function performs simply tests if the proportion = .5 (not helpful for us)

Let's increase the sample size and see what happens to our sample proportion and to our confidence interval

```
# generate a sample that is the same for all of us
set.seed(317)
bigcamp <- homes %>%
  sample_frac(0.1)

# calculate the confidence interval
prop.test(x = sum(bigcamp$totalvalue > 500000), n = length(bigcamp$totalvalue))
```

Notice that the point estimate of our sample stayed similar but that the confidence interval shrank considerably. This is the wonderful effect of increasing your sample size!

Since we have access to our population, let's see what the true proportion is

```
mean(homes$totalvalue > 500000)
```

## Testing 2 proportions

Often we are curious to compare 2 proportions. For example, is there is difference in the proportion of homes rated as "Substandard" or "Poor" condition with and without central air?

Let's use the `bigcamp` object we created above to test our question

```
bigcamp %>%
  group_by(cooling) %>%
  summarise(prop = mean(condition == "Poor" | condition == "Substandard"),
            n = sum(condition == "Poor" | condition == "Substandard"),
            totN = n())
```

So 0.3% of homes with cooling are classified as poor or substandard while 8.9% of homes without cooling are poor/substandard. Let's do the test of proportions to see the confidence interval for the difference in proportions

The `prop.test()` function's first argument is called `x` and it is the number of "successes" for each group. The second argument, `n`, is the total number of homes in each group

```
prop.test(x = c(10,30), n = c(2785, 338))
```

```
## Warning in prop.test(x = c(10, 30), n = c(2785, 338)): Chi-squared approximation
## may be incorrect
```

The difference between our sample proportions is likely between -.117 and -.053, a pretty large difference.

Let's see what the population difference in proportions is

```
homes %>%
  group_by(cooling) %>%
  summarise(prop = mean(condition == "Poor" | condition == "Substandard"),
            n = sum(condition == "Poor" | condition == "Substandard"),
            totN = n())

0.00352-0.103
```

## Chi Square Contingency tables

Many times we are interested in how two discrete variables interact with each other. When ever you have count data with more than one variable, you should be thinking about contingency tables. This type of analysis is also called chi square test of independence.

The `xtabs()` function is useful for creating contingency tables from categorical variables. Let's create a cross tabulation showing `hdsdistrict` and `condition2` of the home, and assign it to an object called `xt`. After making the assignment, type the name of the object to view it.

```
xt <- xtabs(~hdsdistrict + condition2, data = homes)
xt
```

Ok, so our data is a 4 by 7 table. Looking at this table, it is clear that we don't have a lot of homes in the Unassigned `hdsdistrict`, so we should likely drop that prior to analysis. Let's also drop `condition2 == None` since we don't know what that means for the home.

Let's also use a function from the `forcats` package to combine levels of condition together. The syntax for `fct_collapse()` is `new = c("old", "Old")`

```
homes_chi <- homes %>%
  filter(hdsdistrict != "Unassigned" & condition2 != "None") %>%
  mutate(condition3 = fct_collapse(condition2,
                                    Good = c("Excellent", "Good"),
                                    Average = c("Average", "Fair"),
                                    Poor = c("Poor", "Substandard")
                                    ))

homes_chi %>%
  count(condition3)
```

Great. We dropped ~1000 cases and are ready to re-make our `xt` object with our `homes_chi` dataset and the `condition3` variable

```
xt <- xtabs(~hdsdistrict + condition3, data = homes_chi)
xt
```

Nice. Now we have a 3x3 table.

There are two useful functions, `addmargins()` and `prop.table()` that add more information or manipulate how the data is displayed. `addmargins()` adds totals for rows and columns. By default, `prop.table()` will divide the number of observations in each cell by the total.

```
# Add marginal totals
addmargins(xt)

# Get the proportional table
prop.table(xt)
#each cell divided by grand total
# That isn't really what we want
```

We want to see what proportion of homes in each `hdsdistrict` fall into each condition, so we want proportions by rows. Rows is the first dimension in R, so we specify `margin = 1`.

```
# Calculate proportions over the first margin (rows)
prop.table(xt, margin=1)
```

Looks like Monticello has fewer homes in “Good” condition and Albemarle has fewer homes in “Poor” condition. But are these differences significant?

The chi-square test is used to assess the independence of these two factors (hsdistrict and condition3). That is, if the null hypothesis that hsdistrict and condition3 are independent is true, then we would expect a proportionally equal number of homes within each condition across each hsdistrict.

Let’s do the test and see.

```
chisq.test(xt)
```

The p-value is small suggesting that we found changes from an equal distribution.

The last thing we will do is to compare where the differences are by looking at our observed table, xt, and the values expected if there was an even distribution of conditions across hsdistricts.

```
xt
chisq.test(xt)$expected
```

- We expected 9619 Average homes in Albemarle, and found fewer
- We expected 7888 Average homes in Monticello, and found **more**
- We expected 6910 Average homes in WA, and found fewer
- We expected 2113 Good homes in Albemarle, and found **more**
- We expected 1732 Good homes in Monticello, and found fewer
- We expected 1518 Good homes in WA, and found **more**
- We expected 187 Poor homes in Albemarle, and found fewer
- We expected 153 Poor homes in Monticello, and found **more**
- We expected 134 Poor homes in WA, and found **more**

## Feedback

Thank you all so much for sticking through to the end with a new format and mode of learning. Because this is our first time teaching like this, if you have any feedback about how it went or improvements we could make I would be *very* grateful. David and I are teaching a full curricular coding course starting in the next few weeks, so knowing what works and what presents challenges with this virtual format would help us become better educators.

Here is a link to a Google Form where you can add your anonymous thoughts: <https://forms.gle/MLKnyw36CzMkLtZQ6>

Thank you in advance!