Topic 3: Data Wrangling & Some Summarizing

Josh Clinton

September 6, 2021

NOTE: These notes are very, very crude and very, very rough. Note also that they are written to provide a gentle and gradual introduction. There are multiple ways to do things in R and the code that I use is not always the most elegant or the most efficient. However, I have tried to focus on providing code that is intuitive in the sense that beginning users can easily understand what is going on. This means that the code is often incremental, overly explicit, and ignoring potential short-cuts. I think it is important to minimize the hurdles to getting going as the excitement and power that you will get will incentivize you to push forward and learn more complicated techniques.

The first thing we want to do with data is to get a sense of what the data contains. Familiarizing yourself with the data is absolutely essential for knowing not only what information is contained in the data, but also what you have to do – if anything – to get you ready to use the data to answer the questions of interest. Data science should always be question-driven, not data-driven. If you are just poking around the data you will find weird relationships if you dig deep enough or look at the data in a particular way. One way to protect against over-interpreting relationships that are happenstance (spurious) in the particular data being looked at rather than a systemic relationship in the world at large is to start with a question and to be guided in your analysis and investigation by a question.

Now there are good and bad questions – What variables are most closely related to one another? is a bad question because there is nothing at stake. All you are doing is working with the data to find the pair of variables with the strongest relationship without any attention as to why we should care. You can judge how good a question you are asking by asking the follow-up question: why does anyone care about that answer? If you do not have a compelling answer to the second question, then think really hard about what you are doing.

At least 80% of data science is getting the data ready for analysis. The “fun" part of investigating and interrogating the data to answer the questions of interest is actually only a very small part of the larger data-science workflow. As a result, it is critically important that you get some skills to be able to read in data from a bunch of different formats and also to manipulate the data being read in. You do not want to be entirely reliant on the data that others have collected and the measures that others have defined. An important part of data science is the power to see how things change if you analyze different data, or if you analyze the data in a different way using different measures or methods. Determining how robust results are to such changes is essential for knowing whether the answers you get depend on the decisions that must inevitably be made in the process of analyzing the data.

We use the term *data wrangling* to describe all of the tasks that are involved in getting the data ready for analysis – wrangling the data if you will. We are going to cover some basics, but realize that we are not really going to dive deeply into this. For the most part we are giving you relatively clean data – we are not going to have you scrape data from the web, have data arranged in different formats (e.g., some with observations on the "rows" and some with observations on the "columns"), extracting data from a relational database or other complicated situations. Instead, we are going to focus on tools that are useful once you have a "flat-file" – that is a dataset that is organized where each row is a unique observation in the data (e.g., the opinions of an individual, the votes in a county, the amount of statewide education spending in a year, the economic growth in a country for a certain year) and each column is a variable that describes a characteristic of relevance for that observation. So if we are working with data on public opinion, for example, each row will be associated with a specific respondent and each column will be everything collected about that individual with each column containing a different characteristic (e.g., a column containing age, a column contained education level, a column containing their current opinion about the president, etc.). We call this a flat-file because in principle, we could kill a lot of trees by printing out the data and rebuilding what we can see in a spreadsheet program on the ground. If you use a spreadsheet program like Excel, Numbers, or Google Sheets, a flat-file is equivalent to a worksheet.

Where in your computer are you?

For every project you do, the first thing you need to do is to figure out where you are in your computer. R does not know where your data is located and you want to make sure that all of your data and code are in places that you can easily access.

You can figure out where R thinks you are by getting the working directory that R is looking at using `getwd()`. This will print to the screen the location that R thinks it is located at. So for me, this would produce:

```r
getwd()
```

```
## [1] "/Users/clintojd/GitHub/vandy_ds_1000_materials/Lectures/LectureNotes"
```

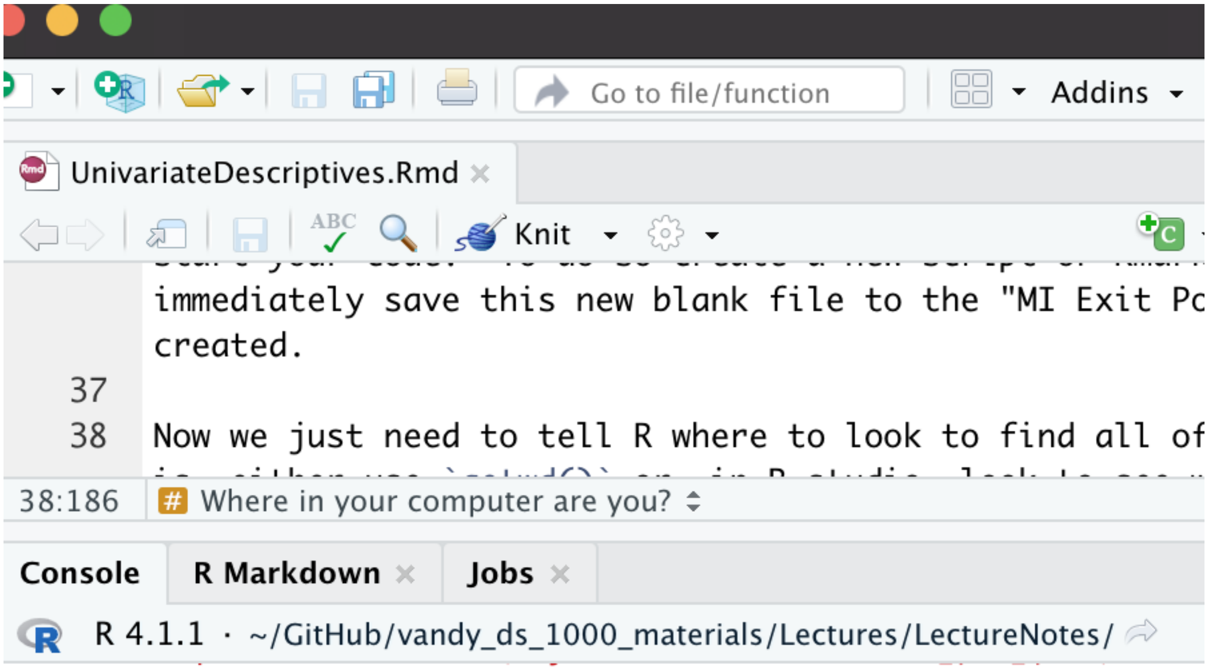
This is super important because if I am trying to load any files or data into R it will be looking in this folder! If my data is located in another folder, R will fail to find it. If so, I either need to change my directory (using the `setwd()` function) or make sure to include the directory location when loading the file.

Staring out, it makes sense to create a new folder for each project that you are working on. So for the work we are going to do in this chapter analyzing the 2020 Michigan Exit Poll you should create a new folder on your computer and call it "MI Exit Poll 2020".

Copy the data file we are going to work with `Final MI subset.Rdata` either directly into this folder or else create a new folder within the “MI Exit Poll 2020” folder called “data” and save it there. Starting out it is easier to keep everything in the same folder, but if you want to take a baby step at navigating folders feel free to create the data folder and save it there.

Ok, so now you have your folder and your data located in that folder. Now you want to start your code. To do so create a new script or Rmarkdown document and then immediately save this new blank file to the “MI Exit Poll 2020” directory you just created.

Now we just need to tell R where to look to find all of this! To see where R thinks it is, either use `getwd()` or, in R studio, look to see what the directory is in the Console window.



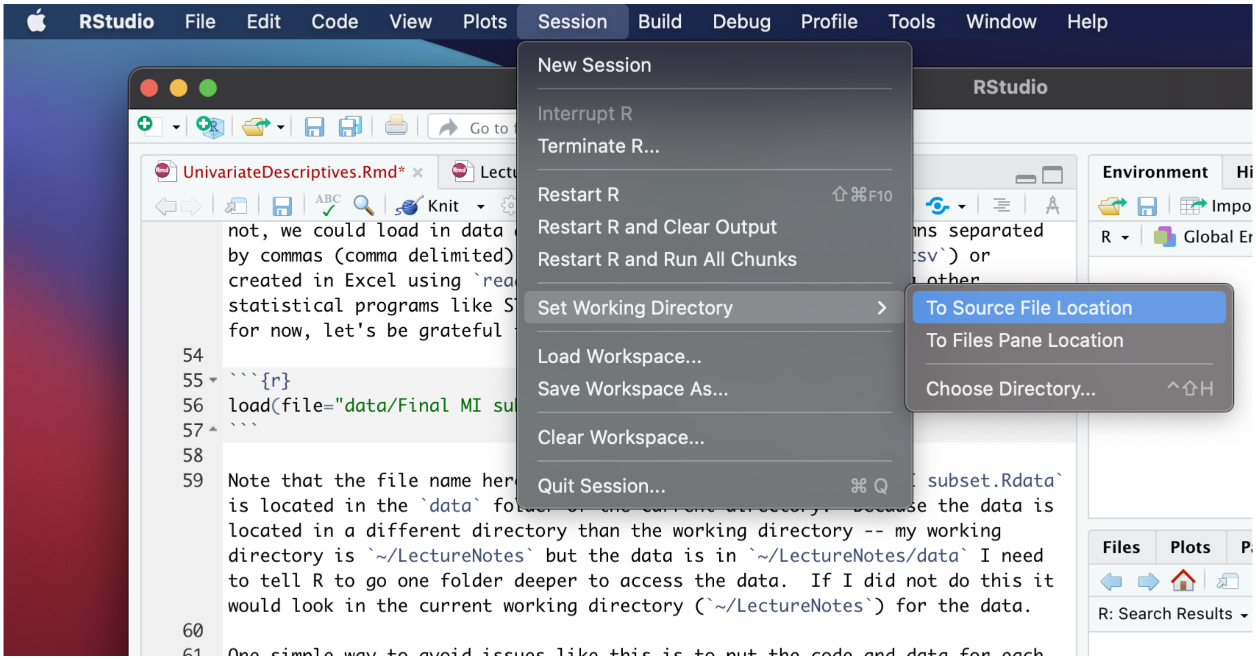
```
immediately save this new blank file to the "MI Exit Pc
created.

Now we just need to tell R where to look to find all of
```

R 4.1.1 · ~/GitHub/vandy_ds_1000_materials/Lectures/LectureNotes/

The directory listed in the Console window tells you where R is looking. Is this where your code and data are located?

If not, can get R to the right place in two ways. First, make sure that your R code is active in R studio (i.e., “File -> Open File” to load the code into R Studio if it is not already loaded and active). If you are comfortable with directories, you can start your code with `setwd()` to tell R to set the working data to where your code is located. If so, when you run your code it will redefine the working directory to be that directory and you should be OK.

If you are not yet comfortable with directories, you can always use R studio to do the same thing for you. Using R Studio, select the following menus: “Session -> Set Working Directory -> To Source File Location”.



~~~
not, we could load in data                                ns separated
by commas (comma delimited)                               sv`) or
created in Excel using `rea                               other
statistical programs like S
for now, let's be grateful

```{r}
load(file="data/Final MI su
```

Note that the file name her                                subset.Rdata`
is located in the `data` fo                           se the data is
located in a different directory than the working directory -- my working
directory is `~/LectureNotes` but the data is in `~/LectureNotes/data` I need
to tell R to go one folder deeper to access the data.  If I did not do this it
would look in the current working directory (`~/LectureNotes`) for the data.

One simple way to avoid issues like this is to put the code and data for each
~~~

This will tell R to look to where your code is when looking for additional files to load. Note that selecting this option thru the menus will result in the code being pasted into the Console window. You can copy and paste this snippet into your code to have your script set the working directory at the beginning. Note that this is also a really good way to figure out where you are if you are not really comfortable with how computers are organized and where things are located!

So now you should be in a good spot – R is looking in the same place as your data and your code. Just remember to create a different folder for each project and to do the same process for each project to keep things separate.

There are two important caveats. First, if you are working with data that you are loading directly from the internet then the local directory structure is less relevant because the data of interest is not stored on your computer. We will see this shortly.

Second, if you are working with RMarkdown it is important to realize that what the RMarkdown code "sees" is different from what you see in your Global Environment. For example, if I were to "knit" this completed document right now all of the code would run, but nothing would be placed into my Global Environment! Similarly, even if the objects you want to work with are in your Global Environment, that does not mean that your RMarkdown code knows where to find them! Your RMarkdown code needs to contain the specific code to load the data in.

This can sometimes make writing code in RMarkdown difficult, but this is where the "Run All Chunks Above" and "Run Current Chunk" to run the code and and load the objects being wrangled into the Global Environment to allow you to make sure your code is doing what you think it is doing.

Loading & Inspecting Data

R can load lots of different types of data. It can read directly from a spreadsheet worksheet such as Excel or it can read in data that is organized in a way such that the variables are separates by a comma (*.csv) or tab-delimited. It can also read in data from other statistical programs like STATA or SPSS. There are lots of ways to get data into R. Recall also that R can keep several different datasets in the environment at the same time. Some programs only allow you to work with one dataset at a time. R does not. R lets you work with as many data sets at the same time as you want. This makes it easy to load in lots of data from lots of places and then work with the data. The downside is that you need to ensure that your environment is clean and that the variables you are working with are the ones you think you are working with. It is a good idea to always start with a clean environment when you are getting started to ensure that the only objects that exist are ones that you intentionally create for the task at hand.

In the work that follows we are going to use the tools of the `tidyverse` package. We can do everything in a slightly different way using the tools of "base" R, but most modern data scientists who use R rely on the tools of tidyverse so it makes sense to just jump right in. We will talk about some useful base R tools along the way as well.

To start, let's load `tidyverse` into active memory so R will have access to all of the functions associated with it.

```
library(tidyverse)
```

R has a built in help, so if you want to access it you can call it up using:

```
??tidyverse
```

Note that this will populate the "Help" tab in R-Studio with a bunch of hyperlinks you can follow that provide an overview of the package. In full disclosure, it is not very helpful for beginners.

The data we are going to wrangle and then analyze is data from the actual 2020 Exit Pool that was done in the state of Michigan for the National Election Pool consisting of NBC, CBS, ABC, and CNN by Edison Research. By way of backgrounds, Exit Pools are never used to actually project a winner by the networks, but we do use them on Election Night to help tell the story of the election by better understanding who is voting and what those voters were thinking when they voted. Both of these are important because elections can only determine who wins and loses. The final tally of certified votes can never tell us *why* the outcome was what it was. But the *why* is critically important. A candidate who wins because voters are voting for the candidate because of her policies has a much different mandate for action than a candidate who wins only because voters were voting against the other candidate. The reasons why voters choose to support a candidate – and what they think the priorities for the winner should be going forward – can be very important for helping to ensure the accountability and responsiveness of the political system.

So let's get going! To start we will load the data into our Environment so we can work with it. Because we have done you a solid and already cleaned it for you and created a dataset in the R-format we can use the `load` command. If not, we could load in data arranged in other formats (e.g., columns separated by commas (comma delimited) or tabs (tab-delimited) using `read.csv`) or created in Excel using `read.xls` or `read.xlsx` or created using other statistical programs like STATA or SPSS using `read.dta` and `read.sav`. But for now, let's be grateful to ignore those and load in our data using:

```
load(file="data/Final MI subset.Rdata")
```

Note that the file name here is telling R that the file `Final MI subset.Rdata` is located in the `data` folder of the current directory. Because the data is located in a different directory than the working directory – my working directory is `~/LectureNotes` but the data is in `~/LectureNotes/data` I need to tell R to go one folder deeper to access the data. If I did not do this it would look in the current working directory (`~/LectureNotes`) for the data.

For those that just saved the data in the same directory as your code you would load the data using just the file name as follows:

```
load(file="Final MI subset.Rdata")
```

Now you should see the results of this load in your Global Environment window – the window that lists all of the objects you have to work with. Note that in R, everything you create and work with can be an object. In this case, we have a tibble object that consists of the data we are going to work with. If you want to see a list of objects in R (rather than RStudio), we can use the objects() function to list what objects are currently available to us.

```
objects()
```

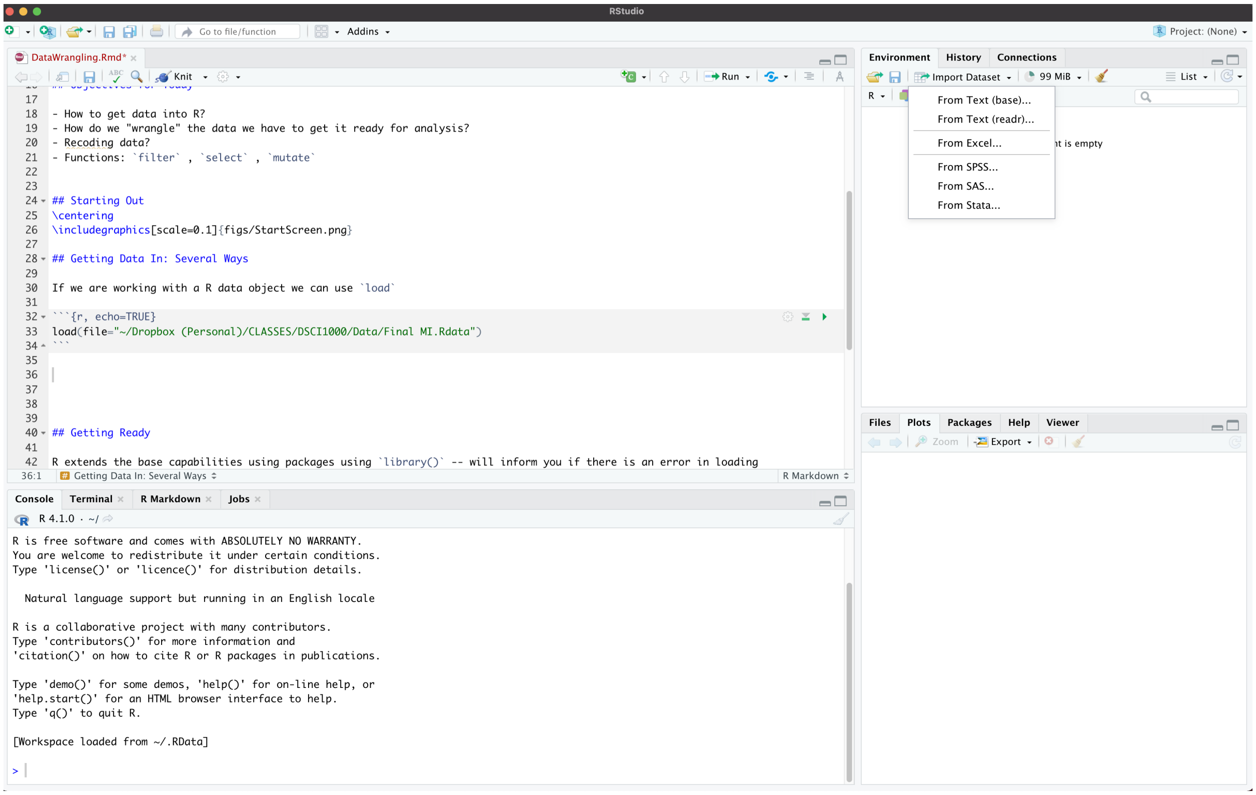
```
## [1] "MI_final"
```

It is important to realize that when you load a R data object into memory the name of the object you load may be different from the file name. Here, for example, the name of the file we loaded was `Final MI.Rdata` but the tibble that was saved in that file name is named `MI_final`.

We can also load in data directly from the web. If we wanted to access the data from the class GitHub page from the last topic we could use a function to load the data – here we are using another function to read in R-formatted data `readRDS` – by telling R that the file location is a web address using the `url()` function:

```
df <- readRDS(url("https://github.com/wdoyle42/vandy_ds_1000/raw/main/Lectures/Lecture2_
```

You can also use RStudio to read in your data. Using the Rstudio menus we can find and select the appropriate type of data from your computer.



~~~

- How to get data into R?
- How do we "wrangle" the data we have to get it ready for analysis?
- Recoding data?
- Functions: `filter` , `select` , `mutate`

Starting Out
\centering
\includegraphics[scale=0.1]{figs/StartScreen.png}

Getting Data In: Several Ways

If we are working with a R data object we can use `load`

```{r, echo=TRUE}
load(file="~/Dropbox (Personal)/CLASSES/DSCI1000/Data/Final MI.Rdata")
~~~

## Getting Ready

R extends the base capabilities using packages using `library()` -- will inform you if there is an error in loading
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

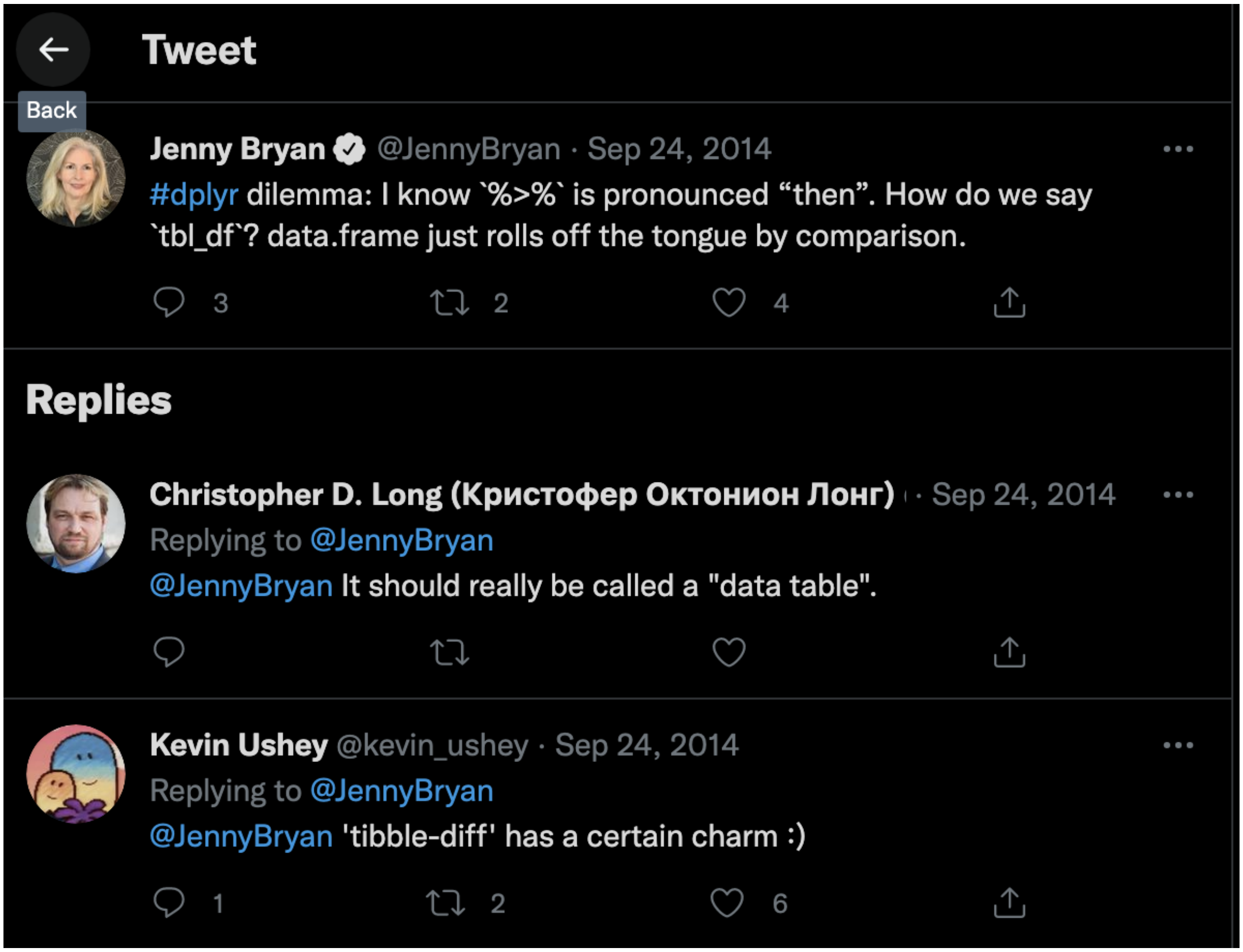
  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

When you use RStudio to import the data the code that replicates what you did will appear in the Console window of Rstudio. You should copy and paste that snippet into your code. Now the next time you run your code – assuming you do not move your data! – you can load the data in using a function instead of RStudio!

When data is loaded in base R, the basic data structure is called a `dataframe`. When data is loaded using `tidyverse` the basic data structure is called a `tibble`. Moreover, the output of a base R function is a `dataframe` object, but the output of a tidyverse function is a `tibble` object.
Why is it called a `tibble`? I am not sure there is an official origin story, but according to the internet (always a very dangerous source) it may be because of a question that was asked about how to pronounce an earlier version of the data format.
```



Jenny Bryan @JennyBryan · Sep 24, 2014

#dplyr dilemma: I know `%>%` is pronounced “then”. How do we say `tbl_df`? data.frame just rolls off the tongue by comparison.

Christopher D. Long (Кристофер Октонион Лонг) · Sep 24, 2014

Replying to @JennyBryan

@JennyBryan It should really be called a "data table".

Kevin Ushey @kevin_ushey · Sep 24, 2014

Replying to @JennyBryan

@JennyBryan 'tibble-diff' has a certain charm :)

What is the largest difference between the older `dataframe` object and the more modern `tibble` object? One seemingly trivial difference is in how R displays the two types of objects. When inspecting a dataframe object, R simply tries to dump as much as it can to the screen. In contrast, when inspecting a tibble object R cleans things up for us and only prints a “readibly small” portion band lets us know what type of data we are working with (e.g, double, characterm factor). To see this, pay attention to how `table(MI_final$SEX)` compares to `count(MI_final,SEX)` in the work we will do shortly!

There are other important differences in terms of the allowable variable names, what happens when we combine variables, and what variable types can be contained in the object (e.g., tibbles can include a `list` object whereas dataframes cannot) but most of those will be beyond what we are going to work with. For the most part, everything we do could be done using either a dataframe or a tibble, but because most current methods rely on tibbles we will use tibbles as much as possible – taking only brief excursions into dataframes.

# Data Types

So what type of data can we work with in R? What types of variables can a tibble contain. Let's focus on the simplest types to start. There are other types (e.g., lists), but we will introduce them when needed.

When we think of data science we think of numbers and statistics. Variables containing numeric values are obviously central to what we are going to do and variables containing "numbers as numbers" are of a type known as a "double." A variable that is a "double" (<dbl>) is simply a numeric variable with high degree of scientific precision.[1] This means that every observation in the variable is a number (or a recognized missing character – more on that later). As such, we can use mathematical operations on these variables (e.g., addition, subtraction, multiplication, division and mathematical functions such as mean, median) directly to the variable. A double is simply a "a number as a number."

Note that if you are working in base R you will sometimes see variables of type <int>. This designates an integer variable. For our purposes there is no real distinction between an integer and a double – you can do the same operations on each. An integer is also "a number as a number." (For more advanced work the distinction can matter in terms of the ability for R to distinguish between two values – doubles are more precise than integers – but this is not going to matter for the type of data and analysis we will be doing.)

A variable that is a "character" (<chr>) – sometimes also called a "string" – is a variable where R treats the variable as if it were consisting of letters rather than numbers. As a result, mathematical operations are not defined for these types of variables. However, there are other functions that are (e.g., the stringr library). Typically character variables have at least one observation consisting of letters rather than numbers - e.g., a variable consisting of all words – e.g., the County variable consisting of the county name for each Exit Poll respondent – is a character as is a variable that has every value being a number except for one.[2]

Consider creating the following new variable object called test consisting of 4 observations – 3 numbers and a set of initials. To do so we are going to create a new object called test and assign it (<-) a collection (c()) of the values 1, 2, 3, and the set of letters "ADC" as follows:

```
test <- c(1,2,3,"ADC")
test
```

```
## [1] "1"   "2"   "3"   "ADC"
```

R attempts to auto-recognize the format of every new object we create. Here is sees that test contains both numbers (1,2,3) and letters ("ADC"). So when determining the object type it will use the type that can accommodate all of those values. Because numbers can be converted to characters but characters cannot be converted to numbers, when an object contains both letters and numbers R will default to a character object.

[1] 'double' refers to double floating point precision a technicality that is only relevant once you get really deep into data science and scientific computing.

[2] It is also possible to have all numbers be saved as a character.

Note that when we call this variable to the screen every value is in quotes – the tell-tale sign of a character variable.

To illustrate the autorecognition of variable types based on the elements in the object let's create two variables with identical elements. So for the `test2` object R will see 3 numbers and treat it as a double. But since we enclosed the values being collected into `test3` in quotes, R will read them in as characters! So despite being defined by the same set of values we can do math using `test2` but not `test3` because the meaning of the values of the two objects differ.

```
test2 <- c(1,2,3)
test3 <- c("1","2","3")
test2
```

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

```
test3
```

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

The third variable type that we are going to initially discuss is a variable that denotes membership in a group. Such a variable is called a "factor" (`<fct>`). The values of a factor variable can be defined by numbers, letters, or a combination of both so it may not be obvious that the variable is a factor by looking at its' values. Mathematical operations are *not* defined for factor variables because the purpose of a factor variable is to indicate how observations are grouped together. If these seems semi-mysterious, don't worry, we will see why this is a useful datatype to have in a bit.

R has a set of built-in functions that tell us how it interprets objects: `is.numeric`, `is.character`, and `is.factor` tells us whether the object being evaluated is a numeric, character or factor respectively. For example:

```
is.numeric(test)
```

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

```
is.character(test)
```

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

When we ask R if it is a numeric variable (using the command `is.numeric()`) it tells us it is not. When we ask if it is a character using `is.character()` it answers that it is. Note that a single observation containing letters (or a letter) will cause the variable to be treated as a character!

There are also an associated set of functions `as.character`, `as.numeric`, and `as.factor` that will attempt to convert a variable from one type to another. Note that you cannot convert characters to numerics, but you can convert numerics to characters.

So if we were to revisit our `test2` variable, conform that is is indeed a numeric variable, convert it to a character-type variable called `test5`, print the newly-created `test5` to the screen to confirm that the values are in quotes (indicating a character value), and then confirm that it is indeed a character, we would see:

```
is.numeric(test2)
```

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

```
test5 <- as.character(test2)
test5
```

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

```
is.character(test5)
```

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

Data Inspection

The first thing you should always do after confirming that the data has been loaded is to get a sense of what the data looks like. This is essential as if you do now know what it is you are analyzing, you should be doing any analysis! It also lets you know if something got messed up. You can have the best code and analysis plan, but if your data is messed up those will all be for naught! So we want to soak and poke our data a bit to get familiar with what we have and to make sure it makes sense in terms of things looking like you expected at a basic level (i.e., the rows and columns are as expected, it has all the data you were expecting, etc.).

There are a few ways to get a basic familiarity with the data. At this point we just want to see what the data literally looks like. There are two useful functions available in the base R package: `head` and `foot`. If I want to see the first 5 rows (observations) or every column (variable) in my tibble I can use the `head` function to print as much information as my screen will display using:

```
head(MI_final)
```

```
## # A tibble: 6 x 13
##     SEX AGE10 PRSMI20 PARTYID WEIGHT QRACEAI EDUC18  LGBT BRNAGAIN LATINOS
##   <dbl> <dbl>   <dbl>   <dbl>  <dbl>   <dbl>  <dbl> <dbl>    <dbl>   <dbl>
## 1     2     2       1       3  0.405       1      4    NA       NA       2
## 2     2    10       1       1  1.81        2      1     2        1       2
## 3     2     7       1       1  0.860       1      5     2        2       2
## 4     1     9       1       3  0.199       1      4    NA       NA       2
## 5     2     8       1       3  0.177       1      5    NA       NA       2
## 6     2     7       1       3  0.492       1      3     2        2       2
## # ... with 3 more variables: RACISM20 <dbl>, QLT20 <fct>, preschoice <chr>
```

As previously mentioned, note that R is letting us know that the results of the function `head()` is a `tibble` (because we have `tidyverse` loaded) with 6 rows and 13 columns. Because it is a tibble object, R only prints the information that can be displayed on the screen and lets us know what is excluded (i.e., `... with 3 more variables...`). In addition to the name of each variable and the first six observations of the displayed varaibles, we are also told the type of each variable (double `<dbl>`, factor `<fct>`, character `<chr>`).

In looking at the results, we can see the letters "NA" in the `LGBT` and `BRNAGAIN` variables even though they are numeric (double) variables. What is going on? R uses the value `NA` to denote missing data and so what we learn is that we are missing the responses for those questions for respondents/observations 1, 4 and 5. (This is because those questions were only asked of a random half of the the sample in the Exit Poll.)

Visually inspecting the first few observations and variables can be helpful for making sure that there wasn't something messed up when reading in the data (e.g., variable names that were mistakenly treated as "data", etc.) You would be surprised how many times take a look at the first few observations can reveal that something went wrong.

Similarly, if you want to look at the last 5 observations of the tibble we can use the function `foot`. This again is useful for making sure that you have all of the data you expected to. If your data was sorted by state postal code, for example, and the last entry in your dataset is "PA" you might want to double-check your data (and code)!

While `head` and `foot` are available in base R, `tidyverse` extends the tools available to us by providing us with the ability to `glimpse` at our data. Taking a `glimpse` will print to the screen not only the size of the data – number of rows and columns – but also every variable in the tibble (each row is a variable), the characteristics of each variable (more on that later), and also the first few observations so you can still get a sense of what the data literally looks like.

```
glimpse(MI_final)
```

```
## Rows: 1,231
## Columns: 13
## $ SEX        <dbl> 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2,~
## $ AGE10      <dbl> 2, 10, 7, 9, 8, 7, 9, 8, 6, 8, 9, 10, 1, 5, 9, 10, 8, 4, 1,~
## $ PRSMI20    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1,~
## $ PARTYID    <dbl> 3, 1, 1, 3, 3, 3, 1, 1, 2, 1, 3, 2, 4, 4, 1, 1, 3, 3, 3, 1,~
## $ WEIGHT     <dbl> 0.4045421, 1.8052619, 0.8601966, 0.1991648, 0.1772090, 0.49~
## $ QRACEAI    <dbl> 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 9, 1, 1, 1, 1, 1, 3, 1, 1, 1,~
## $ EDUC18     <dbl> 4, 1, 5, 4, 5, 3, 3, 3, 4, 4, 5, 5, 4, 1, 1, 1, 5, 2, 4, 2,~
## $ LGBT       <dbl> NA, 2, 2, NA, NA, 2, 2, 2, NA, NA, NA, NA, NA, 2, NA, 2, 2,~
## $ BRNAGAIN   <dbl> NA, 1, 2, NA, NA, 2, 1, 2, NA, NA, NA, NA, NA, 2, NA, 2, 1,~
## $ LATINOS    <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2,~
## $ RACISM20   <dbl> NA, 2, 2, NA, NA, 2, 2, 2, NA, NA, NA, NA, NA, 2, NA, 9, 4,~
## $ QLT20      <fct> Has good judgment, NA, NA, Has good judgment, Cares about p~
## $ preschoice <chr> "Joe Biden, the Democrat", "Joe Biden, the Democrat", "Joe ~
```

What we see is that there are 1231 observations in the dataset and 13 variables. Each variable in the data set is listed as a separate row following the \$ character. Why? This helps emphasize that these variables are all part of the MI_final tibble. If I want to work with the sex variable (SEX) from the MI_final tibble I would reference it as follows:

```
MI_final$SEX
```

Note that R Studio recognizes that you are referencing a variable within MI_final and it suggests ways to "complete" the command by listing all of the variables that match what has been typed when you type in the above code. If I just typed in SEX then R would look in my Environment for a standalone variable called SEX. It is always a good idea to keep variables related to a tibble nested within that tibble to prevent confusion and user-induced errors!

After the variable names we then have a label that defines what the characteristics of each variable is. Here we see: <dbl> , <chr> , and <fct>. As discussed above, these types are very important because they describe the format of each variable and what types of manipulations and analysis make sense given the variable type.

How we can inspect our data depends on the type of data that it is. A simple, but not always useful way to inspect the data is to simply print the variable of interest to the screen as we did above. This is not always useful when we have a large number of observations as not every observation will be printed. Moreover, if you are interested in questions such as "What kind of values do I have?" then looking at a list of every observations will not be terribly useful.

One way to summarize your data is to look at how many unique values you have in the variable. The table function is a function in base R that reports how many observations of each unique value there are in the object of interest.

For example, if we look at the numeric variable SEX we get the number of 1's and 2's in the data. (We will return to what that means in this context shortly!)

```
table(MI_final$SEX)
```

```
##
##   1   2
## 579 652
```

In tidyverse we have the option of count. The syntax is slightly different, as is the output.

```
count(MI_final,SEX)
```

```
## # A tibble: 2 x 2
##     SEX     n
##   <dbl> <int>
## 1     1   579
## 2     2   652
```

When we ask for a table we receive an object that is a vector where each element is the number of observations associated with each value and the length of the returned vector is the number of unique values. So in the code we just ran table(MI_final$SEX), the output is a vector of length 2, with the first observation containing the number of observations with value 1 and the second observation containing the number of observations with value 2.

The `count` function gives us something more and something different. From `count` we get a `tibble` (i.e., a `tidyverse` object) containing a row for each unique value of the variable being counted and 2 columns –

If we look at a character variable we also get a meaningful report using `table`:

```
table(MI_final$preschoice)
```

```
##
##               Another candidate    Donald Trump, the Republican
##                              25                             459
##         Joe Biden, the Democrat                         Refused
##                             723                              14
##            Undecided/Don’t know Will/Did not vote for president
##                               4                               6
```

… and `count`:

```
count(MI_final,preschoice)
```

```
## # A tibble: 6 x 2
##   preschoice                          n
##   <chr>                           <int>
## 1 Another candidate                  25
## 2 Donald Trump, the Republican      459
## 3 Joe Biden, the Democrat           723
## 4 Refused                            14
## 5 Undecided/Don’t know                4
## 6 Will/Did not vote for president     6
```

And similarly with a factor variable using `table`…

```
table(MI_final$QLT20)
```

```
##
## [DON'T READ] Don’t know/refused           Can unite the country
##                              26                             125
##      Cares about people like me                Has good judgment
##                             121                             205
##              Is a strong leader
##                             138
```

…and `count`

```
count(MI_final, QLT20)
```

```
## # A tibble: 6 x 2
##   QLT20                              n
##   <fct>                          <int>
## 1 [DON'T READ] Don’t know/refused    26
## 2 Can unite the country            125
## 3 Cares about people like me       121
## 4 Has good judgment                205
## 5 Is a strong leader               138
## 6 <NA>                             616
```

So `table` will always work as a way of inspecting our data. Sometimes, however, there are so many unique values that the outcome is nearly identical to simply printing the variable to the screen. This is often the case for numeric variables that are nearly continuous (e.g., age). If we look a table of respondent weights (more on what that is later!) we get uninformative output because nearly every value occurs only once in the data!

To illustrate, considser this ill-advised use of count on a variable that has nearly as many unique values as observations.

```
count(MI_final, WEIGHT)
```

```
## # A tibble: 411 x 2
##    WEIGHT     n
##     <dbl> <int>
##  1  0.100     1
##  2  0.113     1
##  3  0.119     1
##  4  0.133     2
##  5  0.141     1
##  6  0.142     1
##  7  0.144     1
##  8  0.146     1
##  9  0.147     1
## 10  0.149     5
## # ... with 401 more rows
```

When looking at numeric variables it therefore makes sense to use mathematical functions to try to summarize the variables. In particular, we can use the `summary` function to identify the maximum (`Max.`) and minimum (`Min.`) value of the variabkle is, as well as things like the average value (`Mean`), and what the 25th (`1st Qu.`), 50th (`Median`), and 75th (`3rd Qu.`) values would be if we arranged the data from the lowest value to the highest value.

```
summary(MI_final$WEIGHT)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1003  0.3775  0.8020  1.0000  1.4498  5.0853
```

Note that in addition to inspecting individual variables, we can also `summary` to the entire tibble to summarize everything. Note that this will produce the equivalent of `table` for factor variables (e.g., `QLT20`) and let you know which variables are characters (e.g., `preschoice`).

```
summary(MI_final)
```

```
##       SEX           AGE10           PRSMI20        PARTYID     
##  Min.   :1.00   Min.   : 1.000   Min.   :0.00   Min.   :1.000  
##  1st Qu.:1.00   1st Qu.: 6.000   1st Qu.:1.00   1st Qu.:1.000  
##  Median :2.00   Median : 8.000   Median :1.00   Median :2.000  
##  Mean   :1.53   Mean   : 8.476   Mean   :1.63   Mean   :2.236  
##  3rd Qu.:2.00   3rd Qu.: 9.000   3rd Qu.:2.00   3rd Qu.:3.000  
##  Max.   :2.00   Max.   :99.000   Max.   :9.00   Max.   :9.000  
##                                                                
##      WEIGHT          QRACEAI          EDUC18           LGBT      
##  Min.   :0.1003   Min.   :1.000   Min.   :1.000   Min.   :1.000  
##  1st Qu.:0.3775   1st Qu.:1.000   1st Qu.:2.000   1st Qu.:2.000  
##  Median :0.8020   Median :1.000   Median :3.000   Median :2.000  
##  Mean   :1.0000   Mean   :1.572   Mean   :3.288   Mean   :2.224  
##  3rd Qu.:1.4498   3rd Qu.:1.000   3rd Qu.:5.000   3rd Qu.:2.000  
##  Max.   :5.0853   Max.   :9.000   Max.   :9.000   Max.   :9.000  
##                                                   NA's   :615    
##     BRNAGAIN        LATINOS         RACISM20    
##  Min.   :1.000   Min.   :1.000   Min.   :1.000  
##  1st Qu.:1.000   1st Qu.:2.000   1st Qu.:2.000  
##  Median :2.000   Median :2.000   Median :2.000  
##  Mean   :1.907   Mean   :2.175   Mean   :2.325  
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:3.000  
##  Max.   :9.000   Max.   :9.000   Max.   :9.000  
##  NA's   :615                     NA's   :615    
##                               QLT20      preschoice       
##  [DON'T READ] Don’t know/refused: 26   Length:1231       
##  Can unite the country          :125   Class :character  
##  Cares about people like me     :121   Mode  :character  
##  Has good judgment              :205                     
##  Is a strong leader             :138                     
##  NA's                           :616                     
## 
```

Using these tools you can (and should!) inspect every new dataset you work with. I cannot stress how essential it is that you know what your data looks like before you start to do anything. If you are missing data, or if the data is oddly organized and classified now is the time to figure that out and fix it. You cannot easily fix in the analysis what you screw up in the data and there are (unfortunately) far too many examples of bad data leading to erroneous conclusions and real-world consequences!

Working with tibbles as Matrices

Another way that we can check our data is to use the fact that R sees the tibble as a matrix with rows and columns and it is possible to select individuals rows and columns. We can check the number of dimensions using `dim()`, the number of rows using `nrow()` and the number of columns using `ncol()`.

```
dim(MI_final)
```

```
## [1] 1231   13
```

```
nrow(MI_final)
```

```
## [1] 1231
```

```
ncol(MI_final)
```

```
## [1] 13
```

As a result, if I want to see all of the values associated with the first row, I can use:

```
MI_final[1,]
```

```
## # A tibble: 1 x 13
##     SEX AGE10 PRSMI20 PARTYID WEIGHT QRACEAI EDUC18  LGBT BRNAGAIN LATINOS
##   <dbl> <dbl>   <dbl>   <dbl>  <dbl>   <dbl>  <dbl> <dbl>    <dbl>   <dbl>
## 1     2     2       1       3  0.405       1      4    NA       NA       2
## # ... with 3 more variables: RACISM20 <dbl>, QLT20 <fct>, preschoice <chr>
```

If I wanted to see the first column (variable), I would simply use `MI_final[,1]`.

I can use the ability to extract elements of the tibble "matrix" to extract particular rows and columns of interest, For example, if I wanted to replicate the `head` function and print the first 5 rows of the tibble to the screen we would use `MI_final[1:5,]`.

If we wanted to get fancy and look at the first 3 rows for the 1st, 2nd, and 6th variables we would use `1:3` to select the sequence of rows 1,2,3 and then "combine" ('c()') the values 1,2,6 together to identify the columns we want:

```
MI_final[1:3,c(1,2,6)]
```

```
## # A tibble: 3 x 3
##     SEX AGE10 QRACEAI
##   <dbl> <dbl>   <dbl>
## 1     2     2       1
## 2     2    10       2
## 3     2     7       1
```

To test yourself, what does each of the following code return?

```
MI_final[c(1:3),c(1:4)]
MI_final[c(1:3,4,567),c(1,2:6)]
MI_final[c((nrow(MI_final)-100):nrow(MI_final)),c(7:ncol(MI_final))]
```

The ability to use a tibble as a matrix is useful for data manipulation and extracting elements of interest.

Recall that a powerful aspect of R is that everything can be an object! So the `count` objects we have been printing to the screen in the prior code can actually be converted into objects that we can manipulate later.

For example, if we want to see how many respondents chose either Biden or Trump, we can save the output of our `count` and then print just the 2nd and 3rd rows corresponding to Biden and Trump to the screen.

```
prescount.count <- count(MI_final,preschoice)
prescount.count[2:3,]
```

```
## # A tibble: 2 x 2
##   preschoice                        n
##   <chr>                         <int>
## 1 Donald Trump, the Republican    459
## 2 Joe Biden, the Democrat         723
```

Wrangling your data

Stepping off the soap box, once you have the data loaded and you have inspected it so that you get a sense of what it contains, we can now start the process of wrangling it to create the variables you are interested in. This is when we perform all of the coding and recoding that we need to do to create the variables we need for the analysis we need to do to answer the question we want to try to answer.

There are three basic functions that we are going to work with: `mutate`, `select`, and `filter`. To create a new variable from an existing variable we will `mutate`, to create a new tibble containing only a few of the variables we will `select` and to create a new tibble containing only a few observatations we will `filter`.

As a matter of practice it often makes sense to do the manipulations at the largest level where it makes sense. For example, if you create subsets of the data and then create new variables that is less efficient than creating the new variables and then subsetting because the former would require you writing code to do the variable creation for each subset. Typically the order of operations is : `mutate`, `select`, and `filter` as this corresponds to: creating the variables of interest, creating a new tibble containing the desired variables, and then selecting the observations of interest (perhaps using those newly defined variables).

That said, we are going to discuss the functions in a different order to start with the easier-to-undertand functions first. So even though in your analysis workflow you should typically start with `mutate`, we will start with selecting variables using `select`, selecting observations using `filter`, and then defining new variables using `mutate`.

Working with a subset of variables: `select`

Suppose you have a lot of variables in your dataset and you only want to work with a subset of them. It is here that the select function can be useful. One thing you can do is to supply a list of all of the variables you want to select. To do so we want to concatenate or combine (`c()`) the variable names into a listing that R will use. The syntax to do so is:

```
library(dplyr)
MI_small <- select(MI_final, c(SEX,AGE10,PRSMI20,PARTYID))
glimpse(MI_small)
```

```
## Rows: 1,231
## Columns: 4
## $ SEX     <dbl> 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1,~
## $ AGE10   <dbl> 2, 10, 7, 9, 8, 7, 9, 8, 6, 8, 9, 10, 1, 5, 9, 10, 8, 4, 1, 8,~
## $ PRSMI20 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1,~
## $ PARTYID <dbl> 3, 1, 1, 3, 3, 3, 1, 1, 2, 1, 3, 2, 4, 4, 1, 1, 3, 3, 3, 1, 1,~
```

Note that we now only have 4 variables – just the ones we selected.

We can also drop variables by using "-" to eliminate them. So if we decided we did not want `AGE10` we could rewrite `MI_small` dropping `AGE10`:

```
MI_small <- select(MI_small, -AGE10)
glimpse(MI_small)
```

```
## Rows: 1,231
## Columns: 3
## $ SEX     <dbl> 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1,~
## $ PRSMI20 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1,~
## $ PARTYID <dbl> 3, 1, 1, 3, 3, 3, 1, 1, 2, 1, 3, 2, 4, 4, 1, 1, 3, 3, 3, 1, 1,~
```

Now we are down to 3! There are also functions where we can select variables that start with or ends with a particular set of letters. So if we wanted to select just the variables that start with the letter "P" (capitalization matters!) we could use:

```
MI_small <- select(MI_small, starts_with("P"))
glimpse(MI_small)
```

```
## Rows: 1,231
## Columns: 2
## $ PRSMI20 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1,~
## $ PARTYID <dbl> 3, 1, 1, 3, 3, 3, 1, 1, 2, 1, 3, 2, 4, 4, 1, 1, 3, 3, 3, 1, 1,~
```

And we can always select by removing variables. So if we wanted to select variables that did not end in a 0 – which is the same as selecting cases that end in a 0 in this case as we only have 2 variables left – we could use:

```
MI_small <- select(MI_small, !ends_with("0"))
glimpse(MI_small)
```

```
## Rows: 1,231
## Columns: 1
## $ PARTYID <dbl> 3, 1, 1, 3, 3, 3, 1, 1, 2, 1, 3, 2, 4, 4, 1, 1, 3, 3, 3, 1, 1,~
```

You can also select all variables that lie between two variable to include or exclude, but that is often less useful as it depends on the precise arrangement of your variables. But to illustrate this recreate our small tibble and then lets select all variables between SEX and PRSMI20 (so SEX, AGE10, PRSMI20 given the ordering of the columns:

```
MI_small <- select(MI_final, c(SEX,AGE10,PRSMI20,PARTYID))
glimpse(MI_small)
```

```
## Rows: 1,231
## Columns: 4
## $ SEX     <dbl> 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1,~
## $ AGE10   <dbl> 2, 10, 7, 9, 8, 7, 9, 8, 6, 8, 9, 10, 1, 5, 9, 10, 8, 4, 1, 8,~
## $ PRSMI20 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1,~
## $ PARTYID <dbl> 3, 1, 1, 3, 3, 3, 1, 1, 2, 1, 3, 2, 4, 4, 1, 1, 3, 3, 3, 1, 1,~
```

```
# Now select the range of variables
MI_small <-  select(MI_small, SEX:PRSMI20)
glimpse(MI_small)
```

```
## Rows: 1,231
## Columns: 3
## $ SEX     <dbl> 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1,~
## $ AGE10   <dbl> 2, 10, 7, 9, 8, 7, 9, 8, 6, 8, 9, 10, 1, 5, 9, 10, 8, 4, 1, 8,~
## $ PRSMI20 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1,~
```

And, of course, we can combine conditionals to select variables. For example if we wanted to select SEX and the variables that start with a "P" we would use:

```
MI_small <- select(MI_final, SEX & starts_with("P"))
glimpse(MI_small)
```

```
## Rows: 1,231
## Columns: 0
```

Here you see that we have the curious situation of 1231 rows and 0 columns because the set of variables that we have asked R to select does not exist – i.e, SEX does not start with the letter P! As a result, we get a glimpse of nothingness. (There may be a larger philosophical possibility/reality here, but I leave it to others to think what it means to have 1231 rows and `0` columns!)

Working with a subet of rows: `filter`

Instead of using mindless examples and manipulations, let's consider the impact of `filter` and `mutate` using a more sustained example making use of the 2020 Exit Poll that also demonstrates the importance of non-data considerations when conducting and intepreting the results.

A question of historical interest has been the so-called "Gender Gap" in politics. The extent to which men and women participate in politics differently (in terms of how likely they are to participate (e.g., vote, give money to a candidate, run for office) as well as the issues and candidates that men and women support has been of interest throughout history given that the right to vote in the United States was originally given only to some males. The question of whether men and women have similar political beliefs and activities was central to debates over the importance of women's suffrage and it is of continuing interest to scholars of political representation and accountability interested in analyzing the extent to which the political system is "working."

There is a deep and important literature on the topic, but we are going focus on a very small question – how much did the opinions of men and women of Michigan differ in the 2020 presidential election?

At one level, this question seems extremely straightforward to analyze. We simply need to see how the opinions of men and women differ. Unfortunately, this simplicity masks some deeper issues that require some consideration.

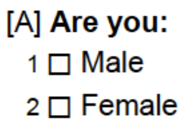
The first rule of data science is that *data never exists in a vacuum* and when analyzing data we *always* need to think of where it came from, how it was collected, and how those processes might affect how we interpret the results. Let's just identify a few of these issues as it relates to using the 2020 Exit Poll to characterize the gender gap to illustrate some of this complexity.

First, who is included in this data? The Exit Poll is the result of calling registered voters in Michigan and the data is collected from those registered voters that were able to be reached and who also agreed to take the survey. So right off the bat we might worry that some voters might be excluded because they became registered after the survey was done (Michigan allows same-day voter registration so voters could register to vote on Election Day). We may also worry that the ability to call a registered voter depended on the ability of the voter file company to attach a phone number to a registered voter. Because you do not supply your phone number when you register to vote, polls making use of voter files rely on companies that try to match voters to phone numbers. How so? Well when you order a pizza for delivery and you give your address and phone number the pizza company may sell that information and allow the voter file to link your address in the voter file to the phone number. (Credit card companies are also a large source of data.) But this means that not every registered voter has an assocaited phone number so not every registered voter can be called. If voters with phone numbers differ from those that do not, this can cause real problems! Even if the pollster gets a voter on the phone, we may also worry that people who chose to participate may differ from those who do not! If a voter thinks that the polls is a "voter suppression poll" or "fake news," for example, they may choose not to participate.

All of these factors may affect how good our data is and whether the voters we have data on are similar to the voters that we do not. If there are differences, this may make it hard to know how to interpret our data! Realize that these are all questions and issues that occur prior to loading the data into R! Data science starts with data collection and thinking about all of the ways that things can go sideways. Moreover, it highlights that data science often requires an argument as to why things are OK! In particular, *why* should we trust the data? This is related to *internal validity* versus *external validity.* Internal validity is whether the analysis makes sense given the data we have. In data science, almost always have internal validity – the analysis makes sense given the data we have. But we often want to extrapolate beyond the data we have. Here, we are interested not just in the differences among the people we interview, but we want to use that to make a statement about Michigan voters in general – including the people we did not interview. But to be confident that our results generalize – i.e.,that we have *external validity* – we need to be able to have a persuasive argument argue that these concerns are not likely to be impactful.

Those are just questions about the nature of the data we have, but there are also difficult questions about the question of interest. While the "gender gap" may have been relatively simply to conceptualize in the past, recent work has highlighted the importance of thinking through what it is that we are really interested in (and why!). Many now draw a distinction between sex – the biological assignment of male and female – and gender – defined to be the self-conception of self that is more complicated than simply "man" or "woman." So when we talk about "gender gap" are we interested in differences by gender or sex or both?

This matters because it has implications for how we might measure this characteristic. Some pollsters using human interviewers attempt to save costs by having the interviewer determine the sex of the respondent based on the respondent's voice. If so, this could cause issues if you are interested in gender. The other way is ask the respondent directly, and here pollsters are faced with the question of what to do. The National Exit Poll, for example, asked simply: "Are you: Male, Female." But what is that measuring? Is that gender? Sex?



[A] **Are you:**

1 □ Male

2 □ Female

Moreover, is it an issue that there are only two choices for the respondent to choose? Lots of potential considerations instantly arise that are worth thinking thru. How many respondents might be impacted by this? With only 1000 respondents, even if more choices were added there are unlikely to be enough respondents to say anything about those individuals. Moreover, given that this was the first question and the definition of gender is hotly contested, perhaps providing multiple responses would result in some conservative voters being less likely to participate and therefore undermine the ability to say much using the poll? The point is that whatever you decide has consequences and not only do you need an argument for what you did, but you also need to think seriously about how that decision may affect your data and what you can learn from it. For example, given the question we analyze are we going to identify the gender gap or the sex gap? We will call it the gender gap for consistency, but it is worth thinking through whether you agree that we are actually measuring the gender gap given this measure!

Finally, it is important to be question driven and to realize *why* that question matters! Demographic variables are rarely a *cause* of political opinions and they are often used to approximate lived experiences that are common among that group. As a result, do we really care about the differences between all men and all women? Or should be more interested in how other considerations intersect with gender? For example, whether younger men and women are more or less similar than older men and women? Or differences by gender may differ for voters of different races and ethnicities? By considering only differences based on gender we are ignoring the potential for important differences related to other characteristics. But if we want to consider other characteristics, we may be limited by the available data! With only 1000 respondents, it will be nearly impossible to look at differences by gender, by age and by race except perhaps for the largest group. But the fact that we cannot because of the data does not mean that we should not. Acknowledging what the data can and cannot examine is critically important for putting the results in context. Moreover, the lack of data is never an excuse for doing bad analysis – if the analysis cannot be justified for what it is, do not do it. In particular, if you cannot make an argument as to why the overall gender gap is interesting, if not also limited in consequential ways that you acknowledge, think very hard about doing that analysis. Data Science is as much critical thinking as it is statistics and if you cannot make an argument as to why someone should care about the answer to the question you are asking then the analysis is likely not worth doing.

Those are all deep and important questions. For now, however, we are going to acknowledge the inherent complexity and move on assuming that the voters we have data on are representative of those that we do not (although more on this in later chapters!) and also that the measure of sex (male and female) is similar enough to gender so as to use `SEX` to measure gender. In terms of whether the gender-gap varies by age, race, or other considerations is a complexity we will dive into once we get underway.

So let's get going!

Our (initial) motivating question is: how large is the gender gap in Michigan among younger voters? And how does that compare to the gender gap among older voters?

Given this question it is immediately clear that we only need to work with a subset of the data. To start, let's focus on female voters in Michigan. To do this we want to "filter" our tibble to include only respondents who indicate that they are "female" when asked. To do so we want to use the `dplyr` package to get access to a function called `filter`. Note – if you forget to use the `dplyr` library this will produce the wrong result because the `filter` command is already defined (in the `stats` package that comes in the base R installation).

Loading (or requiring) the `dplyr` package will mask the original definition of the funciton

Once we load in the `dplyr` command we can then use it. The following code will filter the `MI_final` tibble to select only female respondents (i.e., the observations where the value of `SEX` is 2) and print the filtered tibble to the screen.

```
require(dplyr)
filter(MI_final,SEX==2)
```

```
## # A tibble: 652 x 13
##      SEX AGE10 PRSMI20 PARTYID WEIGHT QRACEAI EDUC18  LGBT BRNAGAIN LATINOS
##    <dbl> <dbl>   <dbl>   <dbl>  <dbl>   <dbl>  <dbl> <dbl>    <dbl>   <dbl>
##  1     2     2       1       3  0.405       1      4    NA       NA       2
##  2     2    10       1       1  1.81        2      1     2        1       2
##  3     2     7       1       1  0.860       1      5     2        2       2
##  4     2     8       1       3  0.177       1      5    NA       NA       2
##  5     2     7       1       3  0.492       1      3     2        2       2
##  6     2     6       2       2  1.50        1      4    NA       NA       2
##  7     2     1       1       4  0.593       1      4    NA       NA       2
##  8     2    10       1       1  1.59        1      1     2        2       2
##  9     2     8       1       1  1.44        1      2     2        2       2
## 10     2     9       2       3  0.438       1      2    NA       NA       2
## # ... with 642 more rows, and 3 more variables: RACISM20 <dbl>, QLT20 <fct>,
## #   preschoice <chr>
```

You can tell that the tibble is filtered by the fact that the resulting dimensions are reported as having 652 observations and 13 variables – roughly half the size of the original dataset.

But what we just did is a no-no. We have have done a filter-and-forget because we failed to define a new tibble object using the filter command. As a result, what we just did is not available for future analysis. You can tell this by looking in your Environment window and noticing that there nothing changes when we fun the code to filter the tibble.

To fix this we just need to create a new tibble object – say `MI_final_female` – and assign it the to be the results of the filter we just used. That is:

```
MI_female <- filter(MI_final,SEX==2)
```

If wanted to create a tibble of voters under the age of 24 and another consisting of voters over the age of 75 we can create these using:

```
MI_under24 <- filter(MI_final,AGE10==1)
MI_over75 <- filter(MI_final,AGE10==10)
```

Unlike before, nothing is printed to the Console window from running these two lines of code because we have chosen to create new objects in our Global Environment that we can use in future analyses. You can tell this by the fact that `MI_female` and `MI_under24` now appear as objects in the Environment window of R Studio.

Note that we can filter by as many variables and conditions as we want. If we want to select observations if and only if the observations satisfy multiple conditions we want to use the character `&` indicating that both conditions must be true. This is known as filtering based on the intersection of the conditions (i.e., "AND")

For example, if we wanted to create a tibble of voters who were female AND under the age of 24,we would use `SEX` and `AGE10` to filter as such:

```
MI_female_under24 <- filter(MI_final,SEX == 2 & AGE10 == 1)
```

Because we have already defined a tibble based on `SEX`, we could also have also simply filtered the tibble we just created using:

```
MI_female_under24 <- filter(MI_female, AGE10 == 1)
```

There is no limit to the number of conditions that we can use (e.g., if we were to filter on `SEX == 2 & AGE10 == 1 & EDUC18 ==1 * QRACEAI==1` selects voters who identify as female AND who are 24 and under AND who report having a High School Education or less AND who self-identify as "White"). Filtering all-at-once using the intersection of a series of conditional statements is obviously the same as applying the filter in a sequentially filtered fashion.

For later, we will do the same for male voters using:

```
MI_male_under24 <- filter(MI_final,SEX == 1 & AGE10 == 1)
```

Instead of requiring multiple conditions to be true, we can also require that either condition X OR condition Y is true. This is the equivalent of the union of the two events. Suppose, for example, that we wanted to filter cases to identify women voters who are either under the age of 24 (i.e., `AGE10==1`) OR over the age of 74 (i.e, `AGE10=10`). We could create a tibble of this group of women voters using the previously filtered data frame `MI_final_female` using:

```
MI_female_under24over74 <- filter(MI_female, AGE10 == 1 | AGE10 == 10)
```

There is no limit to the types of conditions we can use to filter our data and we can easily combine both `&` (AND) and `|` (OR) selections. To illustrate this, consider that another way to select voters who are female and either under the age of 24 or over the age of 74 from the original, unfiltered, data set is by combining the conditions as follows:

```
MI_female_under24over74 <- filter(MI_final,SEX == 2 & (AGE10 == 1 | AGE10 == 10))
```

Note how the second condition involving `|` is grouped using parentheses. This is important for defining how R interpreted what is being requested. Filtering based on `SEX == 2 & (AGE10 == 1 | AGE10 == 10)` is very different than filtering based on `(SEX == 2 & AGE10 == 1) | AGE10 == 10)`! Can you figure out how they differ?[3]

It often helps by thinking about what it is that you are trying to do, putting that into a

[3] The filter `SEX == 2 & (AGE10 == 1 | AGE10 == 10)` selects women who are either under 24 or over 74. The filter `(SEX == 2 & AGE10 == 1) | AGE10 == 10)` selects women under the age of 24 and everyone (male and females) over the age of 74. Big differences!

sentence and then just translating that sentence into code replacing "AND" with `&`, replacing "OR" with `&` and using parentheses to keep conditions together.

Creating new variables and using `mutate`

It is always good practice to have variables stored in ways that make sense. Unfortunately this is not true of our current Michigan Exit Poll data. In particular, `SEX` is defined as a numeric, but what does that mean? In particular, what does it mean to take the average of the `SEX` variable?

```
mean(MI_under24$SEX)
```

```
## [1] 1.515152
```

Huh? For variables that are not numeric and which the scale of the variables is meaningless – why is Male 1 and Female 2? What does a 1-unit difference between males and females even mean?

It often makes sense to use variable types that recognizes that the numeric values are meaningless as a number and that they only serve to designate different categories or to transform variables into variables that indicate whether a condition is true or not so that the values have inherent meaning. Put differently, when working with variables that have values that are not cardinally meaningful (i.e., the distance between values is meaningless as it is here), it is good practice to turn them into variables that minimize the potential for mistakes. This often means either redefining numeric variables that designate group membership into either a character variable with meaningful values (e.g., "male" and "female"), a factor variable to acknowledge that the values have no cardinal meaning, or creating new numeric variables that indicate whether the observation is a member of a specific group or not (so-called indicator variables). We will talk through the creation of each.

One way to create new variables is to use the assignment operator (`<-`) to define a new variable object directly. This allows us to create new variables, but those new variables will be created either outside a tibble or within a tibble.

The following code, for example, creates a new object called `sex.recode` that is a copy of the `SEX` variable in the `MI_final` tibble. This object is created outside of the `MI_final` tibble which means that we can work with the separately and any manipulations done to `sex.recode` will not impact `MI_final$SEX` and visa-versa.

```
sex.recode <- MI_final$SEX
```

If we wanted to create a copy of the variable *within* the `MI_Final` tibble we would want to use the `$` operator to indicate that we want to create the object within an existing tibble. In particular, we would use:

```
MI_final$sex.recode <- MI_final$SEX
```

Using similar technology, we can recode the values of variables into more sensible values. Using the fact that we can access individual elements of the tibble, we can use the following code to recode the `sex.recode` variable we just created to contain more sensible values.

```
MI_final$sex.recode[MI_final$sex.recode==1] <- "Male"
MI_final$sex.recode[MI_final$sex.recode==2] <- "Female"
```

So what we have just done is to recode the sex.recode object we created in the `MI_final` tibble and replace all values of 1 with "Male" and all values of 2 with "Female". In particular, we are using the `[ ]` to select/filter to select the observations for the condition is TRUE and then we assign those observations the appropriate text. But we have only done this for the `sex.recode` object that we defined within the `MI_final` tibble!

So let's take a look at what we just did!

```
table(sex.recode)
```

```
## sex.recode
##   1   2
## 579 652
```

```
count(MI_final,sex.recode)
```

```
## # A tibble: 2 x 2
##   sex.recode     n
##   <chr>      <int>
## 1 Female       652
## 2 Male         579
```

So why did we use `table` for one and `count` for the other? The answer is that because `sex.recode` is an object unattached to a tibble, the `count` function cannot be used – it can only be used to count objects attached to a tibble. So if you have an object unattached to a tibble, you will want to use `table` to inspect the distribution of values.

Before we proceed, to prevent future confusion, let's remove `sex.recode` from our environment using:

```
rm(sex.recode)
```

Note that it is good practice to do such recoding at the highest level possible so that they apply to any dataset that is created by filtering that tibble. For example, we could certainly have defined a new variable in the `MI_under24` tibble, but if we wanted to so similar analyses for other age groups (or other filters) we would have then have to redo the code to apply it to the other tibbles of interest. Having to "copy and paste" code greatly increases the chance of user error by requiring you to apply the same code snippet multiple times and increasing the chances of forgetting to do so if you ever have to change the code later. To minimize user error it is best to perform all of the recodings on the original tibble first. This will ensure that the changes will be reflected in every additional filtered tibble that may be created (e.g., if we wanted to filter to only voters who are over the age of 74 to compare how the gender gap among younger and older voters compares.)

So we can use the assignment operator `<-` to create new variables and recode variables both in and out of tibbles but it may be that we want to create a new tibble with multiple new variables. Rather than creating a *variable* using the assignment operator `<-` we may want a way to create an entirely new (or different) *tibble*. This is where `mutate` from `tidyverse` comes in. Turns out mutations can be more than just the source of superpowers, but it can also be used to create new variables using `tidyverse` in R.

In R, we can use `mutate` to create a new tibble that is defined by the old tibble plus various mutations of variables in that tibble. While provocatively named,the `mutate` command will create a new tibble that "mutates" an existing variable into a new variable while keeping the original variable being mutated intact. The basic function call takes two objects – the tibble being used and the mutation being requested.

So if we wanted to use the variable `SEX` to create a new factor variable called `FEMALE` that takes on the value of 1 if "female" and 0 if "male", we would use:

```
mutate(MI_final, FEMALE = SEX - 1)
```

This is semi-clever..Recall that the variable `SEX` takes on the value of 2 for females and 1 for males and it is a numeric variable so mathematical operations are defined for it. As a result, if we subtract off 1, that will lead to a value of `1-1=0` for males and `2-1=1` for females. Thus, the new variable will be what we call and "indicator" variable that indicates if the respondent self-identifies as a female (1) or not (0)!

Unfortunately, we were not clever enough because what we just did was not saved. If you were to run that code it would mutate the tibble in the console window but not save it for future use. To make the mutation "stick" we need to make sure to define the new object containing the mutated variable.

Usually we can just replace the old dataset with the new one. So:

```
MI_final <- mutate(MI_final, FEMALE = SEX - 1)
```

Now let us see how all the variables we have defined compare: `SEX`, `sex.recode`, and `FEMALE` in terms of our ability to interpret their meaning.

```
count(MI_final,SEX)
```

```
## # A tibble: 2 x 2
##     SEX     n
##   <dbl> <int>
## 1     1   579
## 2     2   652
```

```
count(MI_final,sex.recode)
```

```
## # A tibble: 2 x 2
##   sex.recode     n
##   <chr>      <int>
## 1 Female       652
## 2 Male         579
```

```
count(MI_final,FEMALE)
```

```
## # A tibble: 2 x 2
##   FEMALE     n
##    <dbl> <int>
## 1      0   579
## 2      1   652
```

The original variable `SEX` is clearly the worst because it is not immediately obvious what the values mean. The variable we created `sex.recode` solves this issue by creating a character-variable with informative labels, but the indicator variable `FEMALE` is equally informative by indicating which observations are female. (Note that the variable name is critical for `FEMALE` because it defines what the meaing of a value of `1` means! If we instead to call the variable "SEX2" (or something similarly ambiguous) we would have no idea what a 1 or 0 would refer to. Always be informative!

While both sex.recode and FEMALE are equally informative, the FEMALE variable is arguably of more use because of its numeric properties. In particular, now the mean function has a reasonable intepretation – it is the fraction of respondents who self-identified as female!

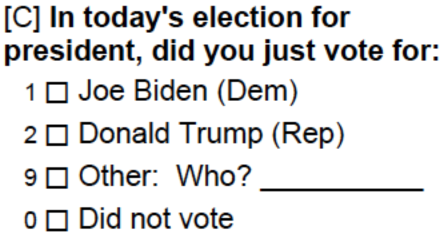
```
mean(MI_final$FEMALE)
```

```
## [1] 0.5296507
```

So this suggests that 53 percent of the voters in Michigan were female!

Note that we can do a bunch of mutations all at once, but we often want to be mindful of the readability and interpretability of what we are doing – hence it can be useful to use separate lines.

So far we have focused on how we can identify males and females using the data. Now let's turn to the ways in which they may differ. Since we are dealing with an Exit Poll concerning the 2020 presidential election it makes sense to focus on how the differ in their support for the most powerful elected official in the United States, if not arguably the world, the president. To get this information, the Exit Poll asked respondents the following question:



[C] **In today's election for president, did you just vote for:**

1 □ Joe Biden (Dem)
2 □ Donald Trump (Rep)
9 □ Other: Who? ___________
0 □ Did not vote

Of primary interest is the extent to which males and females report supporting President

Biden or President Trump. Did males and females have different preferences in terms of who should win? If so, do those differences vary by age? If so, is the gender gap larger among older or younger voters? And is the difference by gender greater than the difference by age? Lots of interesting questions! Let's get cranking!

To start we need to create the outcome measure of interest – the support for President Trump and President Biden. We have two variables in our tibble: `PRSMI20` and `preschoice` to work with. To start, we want to set the `table` and get a sense of what we have to work with.

```
count(MI_final,PRSMI20)
```

```
## # A tibble: 6 x 2
##   PRSMI20     n
##     <dbl> <int>
## 1       0     6
## 2       1   723
## 3       2   459
## 4       7     4
## 5       8    14
## 6       9    25
```

```
count(MI_final,preschoice)
```

```
## # A tibble: 6 x 2
##   preschoice                          n
##   <chr>                           <int>
## 1 Another candidate                  25
## 2 Donald Trump, the Republican      459
## 3 Joe Biden, the Democrat           723
## 4 Refused                            14
## 5 Undecided/Don't know                4
## 6 Will/Did not vote for president     6
```

Can you tell that these variables contain the same information with `preschoice` being a more informative version of `PRSMI20`? `PRSMI20` can be interpreted using the value labels in the Exit Poll questionnaire, but unless you have that at the ready when doing your analysis you are likely to wonder whether voters supporting Biden were coded as a 1 or a 2 and what values of 7, 8 and 9 correspond to!

Here again it makes sense to think about what we want to know before we get to work. Ultimately we want to compare how the percentage of males who support each of the presidential candidates compares to the percentage of females. How can we create a variable that lets us easily calculate these percentages? One way is to use a similar approach we took when defining `FEMALE` which is to create indicator variables that identify whether the voter supports President Biden or President Trump.

There are several different ways to create an indicator for whether an individual voted for Biden (`BidenVoter`) or Trump (`TrumpVoter`). We can use the `ifelse()` function that takes on a value of 1 if the condition is TRUE and 0 otherwise.

```
MI_final <- mutate(MI_final,
                   BidenVoter = ifelse(preschoice=="Joe Biden, the Democrat", 1 , 0),
                   TrumpVoter = ifelse(preschoice=="Donald Trump, the Republican", 1, 0)
```

Alternatively, we can use the `as.logical()` function that determines whether the statement being evaluated is TRUE (1) or FALSE.

```
MI_final <- mutate(MI_final,
                   BidenVoter1 = as.logical(preschoice=="Joe Biden, the Democrat"),
                   TrumpVoter1 = as.logical(preschoice=="Donald Trump, the Republican"))
```

Note that because we are doing this sequentially, the `MI_final` tibble being mutated in the second set of commands already contains `BidenVoter` and `TrumpVoter`! Note also that there is no reason why we could not have done a single mutation creating all four variables at once using:

```
MI_final <- mutate(MI_final,
                   BidenVoter = ifelse(preschoice=="Joe Biden, the Democrat", 1 , 0),
                   TrumpVoter = ifelse(preschoice=="Donald Trump, the Republican", 1, 0)
                   TrumpVoter1 = as.logical(preschoice=="Donald Trump, the Republican"))
```

To confirm that they are the same, lets just take a cross-tabulation of `BidenVoter` against `BidenVote1` using the `table` function to see how the values of `MI_final$BidenVoter` vary depending on the values of `MI_final$BidenVoter`:

```
table(MI_final$BidenVoter,MI_final$BidenVoter1)
```

```
##
##      FALSE TRUE
##   0    508    0
##   1      0  723
```

From this you can confirm that they are identical. We could also check this using the `as.logical()` function to see how many observations of `BidenVoter` and `BidenVoter1` differ from one another. This uses the fact that testing for equality uses `==` and tests for non-equality `!=`:

```
sum(as.logical(MI_final$BidenVoter != MI_final$BidenVoter1))
```

```
## [1] 0
```

Note how powerful `mutate` is. We can create an entirely new tibble containing new variables using whatever functions we want! We can use pre-defined functions (e.g., mean, sd) and transformations (e.g., log) or we can do whatever transformations we fancy.

There is another function called `transmutate` has the same functionality as `mutate` except it replaces the original variable with the mutations. Speaking personally, I avoid transmutations because I always like the ability to compare the final variable I am working with to what I started with. This allows me to do multiple mutations if I am concerned about the robustness of the results. Plus I hate to lose data, but if you are working with very large datasets `transmutate` can help you be more efficient in your memory use.

So why would we want to do all of these mutations? Well now we have a numeric variable that makes sense as it is defined. If we take the mean of the variable we just created, that will tell us what fraction of voters said they voted for each candidate.

So the proportion of repondents who report voting for Biden and Trump seems like it could be calculated using the `mean` function!

```
mean(MI_final$BidenVoter)
```

```
## [1] 0.5873274
```

```
mean(MI_final$TrumpVoter)
```

```
## [1] 0.3728676
```

The above code prints the calculation to the screen but it does not save anything to the Environment. If we wanted to save that value – perhaps because we wanted to use it in a future calculation then we could define a new object and assign the calculation to that object. As with most things in R, there are multiple ways to do this. We can do things in a piecemeal fashion using "base" R as follows by simply taking the mean:

```
PctBiden <- mean(MI_final$BidenVoter)
PctTrump <- mean(MI_final$TrumpVoter)
```

Note that if there is missing data – which there is not because we are kind to you – then this will not compute. (More on this shortly!)

Suppose that we want to display the number of voters who select each option. There are at least two ways to do this – one using the `table` command of base R and one using the `count` function in tidyverse.

```
table(MI_final$preschoice)
```

```
## 
##               Another candidate    Donald Trump, the Republican 
##                              25                             459 
##          Joe Biden, the Democrat                         Refused 
##                             723                              14 
##             Undecided/Don't know Will/Did not vote for president 
##                               4                               6
```

```
count(MI_final, preschoice)
```

```
## # A tibble: 6 x 2
##   preschoice                          n
##   <chr>                           <int>
## 1 Another candidate                  25
## 2 Donald Trump, the Republican      459
## 3 Joe Biden, the Democrat           723
## 4 Refused                            14
## 5 Undecided/Don't know                4
## 6 Will/Did not vote for president     6
```

Note that both of these are simply printing the results to the screen, but the wonderful thing about R is that we can save the resulting table as an object and then manipulate it directly.

Thus, we can do the following:

```
A <- count(MI_final,preschoice)
B <- sum(A$n)
A$prop <- A$n/B
```

Can you figure out what each step does? (No peeking!) Let's see by calling the objects.

```
A # The Table object
```

```
## # A tibble: 6 x 3
##   preschoice                          n    prop
##   <chr>                           <int>   <dbl>
## 1 Another candidate                  25 0.0203
## 2 Donald Trump, the Republican      459 0.373
## 3 Joe Biden, the Democrat           723 0.587
## 4 Refused                            14 0.0114
## 5 Undecided/Don't know                4 0.00325
## 6 Will/Did not vote for president     6 0.00487
```

```
B # The total number of observations being tabulated
```

```
## [1] 1231
```

```
A$prop # the fraction of observations in each category over the total: i.e., the % in
```

```
## [1] 0.020308692 0.372867587 0.587327376 0.011372868 0.003249391 0.004874086
```

Cool. So even though preschoice was a factor variable where the value for each observation was non-numeric, we can use R to compute a table of the categories (in base R) or count the number of observations using tidyverse and then do analysis on those counts! We can then use the indexing ability of R to extract specific values. For example if we wanted the percentage of voters choosing Biden we can do:

```
A[2,]
```

```
## # A tibble: 1 x 3
##   preschoice                       n  prop
##   <chr>                        <int> <dbl>
## 1 Donald Trump, the Republican   459 0.373
```

```
A$prop[2]
```

```
## [1] 0.3728676
```

We can get to the same outcome in another way using the summarize function in tidyverse. To do so, we are going to use what is called a "pipe"

Because the `summarize()` function only works with numeric variables, we need to use the numeric variables associated with vote choice that we created earlier. Summarize allows us to apply a pre-defined function to our data and create a new variable based on the outcome of that application.

```
summarize(MI_final,
          BidenPct = mean(BidenVoter),
          TrumpPct = mean(TrumpVoter))
```

```
## # A tibble: 1 x 2
##   BidenPct TrumpPct
##      <dbl>    <dbl>
## 1    0.587    0.373
```

In fact, we can get even fancier and skip the intermediate step of creating the logical variable by taking the mean of what is inherently a logical condition.Taking the mean of whether `preschoice=="Joe Biden, the Democrat"` is the same as taking the mean of `BidenVoter` given that `BidenVoter` was defined by the same logical. So we can actually skip a step.

```
summarize(MI_final,
          BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
          TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

```
## # A tibble: 1 x 2
##   BidenPct TrumpPct
##      <dbl>    <dbl>
## 1    0.587    0.373
```

Note how powerful summarize is. We can apply a function to create a new object as a consequence of that call. Moreover, we can use pre-defined functions (e.g., `mean`, `sd`) to provide a detailed summary. In fact, we can use summarize to create a new tibble containing whatever fuctions of an existing tibble that we want!

For example, we can count the number of observations (using the length of a variable), calculate means (using `mean` for variables without and with missing data (using `na.rm=TRUE`)), standard deviations (using `sd`) and medians (using `median`) of different variables in a tibble all at once.

```
summarize(MI_final,
          N_Obs = length(BidenVoter),
          BidenPct = mean(BidenVoter),
          TrumpPct = mean(TrumpVoter),
          WeightSD = sd(WEIGHT),
          medianED = median(EDUC18),
          ImpRacism = mean(RACISM20, na.rm=TRUE))
```

```
## # A tibble: 1 x 6
##   N_Obs BidenPct TrumpPct WeightSD medianED ImpRacism
##   <int>    <dbl>    <dbl>    <dbl>    <dbl>     <dbl>
## 1  1231    0.587    0.373    0.779        3      2.32
```

Now we have not yet actually saved the tibble being produced by summarize as an object that can be used later. To do that we need to assign the resulting tibble to a new object. Let's do that by creating a new object called `StateVote`. Can you figure out how what the dimensions of `StateVote` will be?

```
StateVote <- summarize(MI_final,
             BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
             TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

Let's see if you were right by calling it to the screen:

```
StateVote
```

```
## # A tibble: 1 x 2
##   BidenPct TrumpPct
##      <dbl>    <dbl>
## 1    0.587    0.373
```

So `StateVote` is an object with two elements and we can access each of them independently using the indexing ability of R. In particular, we can extract the first and second elements of the object using:

```
StateVote[1]  # Extract the 1st observation (BidenPct)
```

```
## # A tibble: 1 x 1
##   BidenPct
##      <dbl>
## 1    0.587
```

```
StateVote[2]  # Extract the 2nd observation (TrumpPct)
```

```
## # A tibble: 1 x 1
##   TrumpPct
##      <dbl>
## 1    0.373
```

So does this give us the percentage of voters who report supporting President Biden and President Trump? But why do the percentages not sum to 100%? In fact, if we do the calculation using using the `StateVote` object we just created we are missing some voters. How many? Well, lets see what fraction of voters are included?

```
sum(StateVote)
```

```
## [1] 0.960195
```

So this raises the question – how should we treat the 4 percent of the respondents who chose a different answer (i.e., they said they were voting for "Another candidate" or they "Refused" to say)? Should we include them or exclude them? Here again is a question that cannot be answered by R or using statistics. You must make an argument as to whether you think they should or should not be included based on the question you are interested in! Perhaps you think that people who were undecided as who they were going to vote for when contacted right before a presidential election are unlikely to vote. If so, maybe it makes sense to exclude them if you are interested in voters. At the very least, you should think through the consequences of your decision. Better yet, if you are truly uncertain, do the calculation both ways and see how much it matters.

Given the tools in our toolbox, we can do this in this case by `summarizing` after filtering our data to include only self-reported Biden and Trump voters using:

```
StateVote2 <- summarize(filter(MI_final, preschoice=="Joe Biden, the Democrat" |
                        preschoice=="Donald Trump, the Republican"),
                   BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
                   TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

To see what the impact is we can compare the two quantities:

```
StateVote
```

```
## # A tibble: 1 x 2
##   BidenPct TrumpPct
##      <dbl>    <dbl>
## 1    0.587    0.373
```

```
StateVote2
```

```
## # A tibble: 1 x 2
##   BidenPct TrumpPct
##      <dbl>    <dbl>
## 1    0.612    0.388
```

So 61.2 percent of the voters in Michigan voted for President Biden in 2020? And only 38.8 percent voted for President Trump? So this implies that Biden won Michigan by 22.4 percentage points based on the Exit Poll? Hold that thought.

So now let's put all the pieces together starting from the top to keep things clean and tidy. Check yourself that you can follow and understand each and every step.

```
load(file="data/Final MI subset.Rdata")
MI_final <- mutate(MI_final,
                   FEMALE = SEX - 1,
                   BidenVoter = ifelse(preschoice=="Joe Biden, the Democrat", 1 , 0),
                      TrumpVoter = ifelse(preschoice=="Donald Trump, the Republican", 1,0))

MI_under24 <- filter(MI_final,AGE10==1 & (BidenVoter==1 | TrumpVoter==1))
MI_over75 <- filter(MI_final,AGE10==10 & (BidenVoter==1 | TrumpVoter==1))
```

From this there are several ways to calculate what we are interested in!

One way is to use filter and summarize to calculate the percentage of younger voters supporting Biden vs. Trump by gender.

```
PresVoteFemaleUnder24 <- summarize(filter(MI_under24, FEMALE==1),
    BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
    TrumpPct = mean(preschoice=="Donald Trump, the Republican"))

PresVoteMaleUnder24 <- summarize(filter(MI_under24, FEMALE==0),
    BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
    TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

To see the fraction of females under 24 supprting each candidate we can inspect:

```
PresVoteFemaleUnder24
```

```
## # A tibble: 1 x 2
##   BidenPct TrumpPct
##      <dbl>    <dbl>
## 1    0.938   0.0625
```

to see that young females overwhelmingly – and nearly unanimously? – support President Biden.

In contrast, among younger males President Biden has strong support, but not as much as he has among the younger female respondents to the Exit Poll!

```
PresVoteMaleUnder24
```

```
## # A tibble: 1 x 2
##   BidenPct TrumpPct
##      <dbl>    <dbl>
## 1    0.562    0.438
```

To see how these translate into differences by gender we can calculate the difference:

```
PresVoteFemaleUnder24 - PresVoteMaleUnder24
```

```
##   BidenPct TrumpPct
## 1    0.375   -0.375
```

This reveals that the support for President Biden was 37.5 percentage points higher among young females than young males of similar age (and the support for President Trump was 37.5 percentage points higher among young men than young women)!

If we don't want to use pipes, we can also just use the mean applied to certain subsets. Because we have focused on respondents voting for either Biden or Trump, it is enough to consider the percentage supporting Biden. We can define those percentages as two objects using:

```
male24 <- mean(MI_under24$BidenVoter[MI_under24$FEMALE==0])
female24 <- mean(MI_under24$BidenVoter[MI_under24$FEMALE==1])
```

Where we use that to determine the support for President Biden among males (`male24`), females (`female24`) and the difference in support.

```
male24
```

```
## [1] 0.5625
```

```
female24
```

```
## [1] 0.9375
```

```
female24 - male24
```

```
## [1] 0.375
```

Reassuringly, we get the same difference of 37.5 percentage points!

How does that compare the the gender gap among older voters?

```
PresVoteFemaleOver75 <- summarize(filter(MI_over75, FEMALE==1),
    BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
    TrumpPct = mean(preschoice=="Donald Trump, the Republican"))

PresVoteMaleOver75 <- summarize(filter(MI_over75, FEMALE==0),
    BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
    TrumpPct = mean(preschoice=="Donald Trump, the Republican"))

PresVoteFemaleOver75 - PresVoteMaleOver75
```

```
##    BidenPct   TrumpPct
## 1 0.0384949 -0.0384949
```

So nearly nothing! What is going on? Let's start by comparing how young and old males and females vote to figure out what is causing the dramatic gap we observe among younger voters.

Let's start by comparing the support for Biden among women over the age of 75 relative to the support among women under the age of 24 reveals a massive gap.

```
PresVoteFemaleOver75[1] - PresVoteFemaleUnder24[1]
```

```
##     BidenPct
## 1 -0.3242925
```

The -32.4 percentage point gap is nearly the same as the gap between males and females.

So we see a massive gap – nearly the same size as the gap between young males and females. In contrast, comparing the difference in support for President Biden between older and younger males reveals almost no difference at all!

```
PresVoteMaleOver75[1] - PresVoteMaleUnder24[1]
```

```
##     BidenPct
## 1 0.01221264
```

Stepping back, the largest difference is among young females. But how much confidence should we have in that? Do we really think that 93.8 percent of the female voters in Michigan under the age of 24 voted for President Biden? Seems suspicious.

One possibility is that we may just have a small sample. As we shall see, when the number of data points is low it is possible that our data may provide very imprecise estimates. So how many data points are we talking about?

```
count(MI_under24,FEMALE)
```

```
## # A tibble: 2 x 2
##   FEMALE     n
##    <dbl> <int>
## 1      0    16
## 2      1    16
```

So 16 males and 16 females. That is not a lot of data! And how did that data breakout?

To see how the values of two variables compare we can use `table` to see how the values of each variable relate to one another – with the first variable defining the rows of the resulting table object and the second variable defining the columns. To help read the results, we are also going to explicitly label the rows using `Female =` and the columns using `Vote =`.

```
table(Female = MI_under24$FEMALE, Vote = MI_under24$preschoice)
```

```
##       Vote
## Female Donald Trump, the Republican Joe Biden, the Democrat
##      0                            7                       9
##      1                            1                      15
```

So only 1 female out of the 16 indicated that they were voting for President Trump! That would do it!

We can also do this breakout using tools and functions from `tidyverse`:

```
MI_female_under24 <- filter(MI_final, FEMALE == 1 & AGE10 == 1)
MI_male_under24 <- filter(MI_final, FEMALE == 0 & AGE10 == 1)

count(MI_male_under24, preschoice)
```

```
## # A tibble: 2 x 2
##   preschoice                       n
##   <chr>                          <int>
## 1 Donald Trump, the Republican     7
## 2 Joe Biden, the Democrat          9
```

```
count(MI_female_under24, preschoice)
```

```
## # A tibble: 3 x 2
##   preschoice                         n
##   <chr>                          <int>
## 1 Another candidate                  1
## 2 Donald Trump, the Republican     1
## 3 Joe Biden, the Democrat         15
```

This reveals a few things. First, there are not very many voters under the age of 24 in the Michigan Exit Poll – there are only 17 females and 16 males! It is hard to say anything very precise about young voters as a result. It is especially hard given how unbalanced the results are. The younger female voters are nearly unanimous in their support for Biden – only a single female under the age of 24 in the Michigan Exit Poll reported voting for Trump! (If that seems suspiciously low, hold that thought…)

Piping

So far we have called functions of variables using both the tibble and the variable name of interest. For example, `count(MI_male_under24, preschoice)`. This works fine and is easily understood, but there is also a more advanced way of working with tibbles that is useful – especially when we want to apply the same function/manipulation to several subsets of the data. Or if we want to doing several manipulations in sequence using a single code snippet.

To do so we are going to use the "pipe" operator `%>%` that tells R how we want to apply certain functions. To understand what this does, think of `%>%` as meaning "then". To start though, let's see how we can do what we have already done using this new technology. To produce a count of how males under the age of 24 votes, we can "pipe" the count function through the `MI_male_under24` tibble as follows:

```
MI_male_under24 %>%
  count(preschoice)
```

```
## # A tibble: 2 x 2
##   preschoice                         n
##   <chr>                          <int>
## 1 Donald Trump, the Republican     7
## 2 Joe Biden, the Democrat          9
```

In words, this tells R to "use `MI_male_under24` then `count` the `preschoice` variable." So we start with the name of the tibble object we want to work with and *then* we tell R what we want to do. Here, the code doesn't seem much different from simply `count(MI_male_under24,preschoice)` so you may wonder what the point is.

But the value of piping is that it allows us to do several steps in ways that are arguably easier to follow. So if we wanted to summarize the percentage of males under the age of 24 who support Biden, we can use pipes to do the following: use the tibble `MI_under24` *then* `filter` to select male respondents *then* `summarize` those respondents to compute the percentage supporting Biden and Trump in objects called `BidenPct` and `TrumpPct` respectively.

```
MI_under24 %>%
  filter(FEMALE==0)  %>%
    summarize(BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
    TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

```
## # A tibble: 1 x 2
##   BidenPct TrumpPct
##      <dbl>    <dbl>
## 1    0.562    0.438
```

But note that what we have done has not been saved as an object – note that we know this because there is no assignment operator in the code! – but we can easily do that.

```
MaleUnder24Vote <- MI_under24 %>%
  filter(FEMALE==0)  %>%
    summarize(BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
    TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

Conditional Relationships and `group_by`

So far we have done the analysis using separate tibbles – using the same code snippets applied to the two tibbles filtered by the characteristics of interest. But `tidyverse` has ways of working that do not require us to create so many different objects. Recall that the idea is to minimize the potential for user error. The more we have to "copy and paste" code the more likely it is that we are going to make a mistake. And the more of a problem it may be to change the analysis being applied to multiple objects.

As much as possible, we therefore want to try to minimize how much we have to do that. Because so much of what we do in data science involves the comparison of observations with different characteristics, `tidyverse` has a very useful function that allows us apply our analysis to different groups without having to first create separate filtered tibbles. Not only is this efficient – especially if we are working with large amounts of data when creating new objects can be very resource intensive – but it also minimizes the possibility for error by eliminating the need to apply identical code to the different groups of data, and it is also arguably easier to follow (once you get familiar with it!).

Let's see this! Recall that we were interesting in how many males and females under the age of 24 indicated voting for Biden and Trump. Preciously we either had to count each gender-specific tibble individually or else we had to use table to break out how preschoice varied by FEMALE for the MI_under24 tibble. The former requires us to create gender-specific tibbles and the latter produces a dataframe object that is rather clunky.

Using pipes and group_by we can do the same in a much more efficient fashion. So what do we want to do? We want to take the MI_under24 tibble, group the observations by FEMALE and then count how many in each group support Biden or Trump. To do this we simply translate these sentences into code! In particular, we want to pipe the group_by() command thru the MI_under24 tibble to have R internally separate the observations for which FEMALE==0 and FEMALE==1 and then pipe the count function thru to each of these groups to produce separate counts for each group.

```
MI_under24 %>%
  group_by(FEMALE) %>%
  count(preschoice)
```

```
## # A tibble: 4 x 3
## # Groups:   FEMALE [2]
##   FEMALE preschoice                      n
##    <dbl> <chr>                       <int>
## 1      0 Donald Trump, the Republican     7
## 2      0 Joe Biden, the Democrat          9
## 3      1 Donald Trump, the Republican     1
## 4      1 Joe Biden, the Democrat         15
```

Note that this produces a single table with a variable that denotes whether the number of responses is associated with FEMALE==0 or FEMALE==1. So now we have in a single tibble what we had previously using two separate tibbles!

Of course, we can easily turn this into the gender-specific tibbles using our well-loved filter command. To do so we would need to rerun the code and assign the resulting tibble object (as GenderVoteChoice).

```
GenderVoteChoice <- MI_under24 %>%
  group_by(FEMALE) %>%
  count(preschoice)
```

Once we create that tibble, we can manipulate it however we want – here filtering by FEMALE to pull out the counts for males and females separately.

```
filter(GenderVoteChoice, FEMALE==0)
```

```
## # A tibble: 2 x 3
## # Groups:   FEMALE [1]
##   FEMALE preschoice                      n
##    <dbl> <chr>                       <int>
## 1      0 Donald Trump, the Republican     7
## 2      0 Joe Biden, the Democrat         9
```

```
filter(GenderVoteChoice, FEMALE==1)
```

```
## # A tibble: 2 x 3
## # Groups:   FEMALE [1]
##   FEMALE preschoice                        n
##    <dbl> <chr>                         <int>
## 1      1 Donald Trump, the Republican     1
## 2      1 Joe Biden, the Democrat         15
```

Amazing!

Note that there is no limit to how many pipes we can apply. So, if we really wanted to be resource efficient and minimize the number of tibbles we create, we can incorporate multiple pipes to produce what we want while working with the original tibble to get the breakout for males and females under the age of 24. For example:

```
MI_final %>%
  filter(AGE10=="1") %>%
  group_by(FEMALE) %>%
  count(preschoice)
```

```
## # A tibble: 5 x 3
## # Groups:   FEMALE [2]
##   FEMALE preschoice                        n
##    <dbl> <chr>                         <int>
## 1      0 Donald Trump, the Republican     7
## 2      0 Joe Biden, the Democrat          9
## 3      1 Another candidate                1
## 4      1 Donald Trump, the Republican     1
## 5      1 Joe Biden, the Democrat         15
```

Note how group by is very similar to a filter except it returns an object that has observations grouped by the `group_by()` variable instead of having separate tibbles defined by the `group_by()` variable as is produced below where we instead `filter` by `FEMALE`.

```
MI_final %>%
  filter(FEMALE==0 & AGE10=="1") %>%
  count(preschoice)
```

```
## # A tibble: 2 x 2
##   preschoice                        n
##   <chr>                         <int>
## 1 Donald Trump, the Republican     7
## 2 Joe Biden, the Democrat          9
```

```
MI_final %>%
  filter(FEMALE==1 & AGE10=="1") %>%
  count(preschoice)
```

```
## # A tibble: 3 x 2
##   preschoice                        n
##   <chr>                         <int>
## 1 Another candidate                 1
## 2 Donald Trump, the Republican      1
## 3 Joe Biden, the Democrat          15
```

Note also that we had to use a lot of similar code to do what we could above. This is where the potential for error creeps in (especially if we were doing more than taking a `count`).

Note that we `group_by()` multiple variables to provide us even more detailed break-outs.

```
AgeGenderVote <- MI_final %>%
  group_by(AGE10,FEMALE) %>%
  count(preschoice)

AgeGenderVote
```

```
## # A tibble: 75 x 4
## # Groups:   AGE10, FEMALE [22]
##    AGE10 FEMALE preschoice                        n
##    <dbl>  <dbl> <chr>                         <int>
##  1     1      0 Donald Trump, the Republican      7
##  2     1      0 Joe Biden, the Democrat           9
##  3     1      1 Another candidate                 1
##  4     1      1 Donald Trump, the Republican      1
##  5     1      1 Joe Biden, the Democrat          15
##  6     2      0 Another candidate                 1
##  7     2      0 Donald Trump, the Republican      5
##  8     2      0 Joe Biden, the Democrat           7
##  9     2      0 Refused                           1
## 10     2      1 Donald Trump, the Republican      1
## # ... with 65 more rows
```

Note that this will return a tibble of counts organized in the order of the variables listed in `group_by` – so arranged according to the value of `AGE10` (lowest to highest) and then, within each value of `AGE10` by `FEMALE` (lowest to highest). The resulting tibble also contains variables identifying the value associated with each count. This is useful because we can then work with the resulting tibble itself.

So if we wanted to see the breakout for males and females under the age of 24 we could use:

```
filter(AgeGenderVote, AGE10==1)
```

```
## # A tibble: 5 x 4
## # Groups:   AGE10, FEMALE [2]
##   AGE10 FEMALE preschoice                       n
##   <dbl>  <dbl> <chr>                        <int>
## 1     1      0 Donald Trump, the Republican     7
## 2     1      0 Joe Biden, the Democrat          9
## 3     1      1 Another candidate                1
## 4     1      1 Donald Trump, the Republican     1
## 5     1      1 Joe Biden, the Democrat         15
```

Or if we wanted the counts for males, by age, regardless of age, we could use:

```
filter(AgeGenderVote, FEMALE==0)
```

```
## # A tibble: 37 x 4
## # Groups:   AGE10, FEMALE [11]
##    AGE10 FEMALE preschoice                       n
##    <dbl>  <dbl> <chr>                        <int>
##  1     1      0 Donald Trump, the Republican     7
##  2     1      0 Joe Biden, the Democrat          9
##  3     2      0 Another candidate                1
##  4     2      0 Donald Trump, the Republican     5
##  5     2      0 Joe Biden, the Democrat          7
##  6     2      0 Refused                          1
##  7     3      0 Another candidate                1
##  8     3      0 Donald Trump, the Republican    11
##  9     3      0 Joe Biden, the Democrat         15
## 10     4      0 Donald Trump, the Republican     7
## # ... with 27 more rows
```

Or if we wanted the count for females under the age of 24:

```
filter(AgeGenderVote, AGE10==1 & FEMALE==1)
```

```
## # A tibble: 3 x 4
## # Groups:   AGE10, FEMALE [1]
##   AGE10 FEMALE preschoice                       n
##   <dbl>  <dbl> <chr>                        <int>
## 1     1      1 Another candidate                1
## 2     1      1 Donald Trump, the Republican     1
## 3     1      1 Joe Biden, the Democrat         15
```

But this is only the start of what we can do. Because what we are piping thru is creating a tibble, we can also `mutate` to create new variables.

When doing something complicated it is always good to break the task into a series of steps that you can describe and then use R to do each.

So what we are going to do is to work with our original tibble `MI_final` and `filter` the data frame to select voters under the age of 24 (yes, we could `group_by` AGE10 to do the same thing, but since we are only interested in younger voters it makes sense to focus on the data we are interested in). Then we want to group voters by FEMALE and count the number of voters of each gender who select each presidential candidate (preschoice). Since this gives us the *number* of respondents and we want the *proportion* of female and male voters who select each candidate we want to mutate the resulting tibble to create a variable measuring the proportion of respondents selecting each category. To do so we want to divide the number of respondents in each group choosing each response over the total number of respondents in each group.

So let's do this!

```
MI_final %>%
  filter(AGE10 == 1) %>%
  group_by(FEMALE) %>%
  count(preschoice) %>%
  mutate(Prop = n / sum(n))
```

```
## # A tibble: 5 x 4
## # Groups:   FEMALE [2]
##   FEMALE preschoice                        n   Prop
##    <dbl> <chr>                         <int>  <dbl>
## 1      0 Donald Trump, the Republican      7 0.438
## 2      0 Joe Biden, the Democrat           9 0.562
## 3      1 Another candidate                 1 0.0588
## 4      1 Donald Trump, the Republican      1 0.0588
## 5      1 Joe Biden, the Democrat          15 0.882
```

Recall that pipes are applied in order. This is very important because it means that when we mutate at the end we are mutating the tibble that has already been grouped and counted. This means that we are taking the sum *within* the group FEMALE==0 and FEMALE==1. This is important because it means that the proportions are relative to the size of the group!

We can also clean things up. Maybe you want to trim the output and remove the column of actual counts. To do so we can use the select command to select everything BUT n. To do so we use the fact that - indicates removing a variable. So we could either use select to include all of the variables we want to include, or we can do the opposite and list all of the variables we want to remove.

```
MI_final %>%
  filter(AGE10 == 1) %>%
  group_by(FEMALE) %>%
  count(preschoice) %>%
  mutate(Prop = n / sum(n))  %>%
  select(-n)
```

```
## # A tibble: 5 x 3
## # Groups:   FEMALE [2]
##   FEMALE preschoice                      Prop
##    <dbl> <chr>                          <dbl>
## 1      0 Donald Trump, the Republican 0.438
## 2      0 Joe Biden, the Democrat      0.562
## 3      1 Another candidate            0.0588
## 4      1 Donald Trump, the Republican 0.0588
## 5      1 Joe Biden, the Democrat      0.882
```

Importantly, note that we can `ungroup` at any time to perform the calculations on all observations regardless of group status. So if we wanted to know the proportion of voters under the age of 24 who were: males who voted for Biden, males who voted for Trump, females who voted for Biden, and females who voted for Trump we would ungroup the analysis before the mutation. What this means is that we are taking the proportion in terms of *all* younger voters – not in terms of younger voters by gender as above.

```
MI_final %>%
  filter(AGE10 == 1) %>%
  group_by(FEMALE) %>%
  count(preschoice) %>%
  ungroup() %>%
  mutate(Prop = n / sum(n)) %>%
  select(-n)
```

```
## # A tibble: 5 x 3
##   FEMALE preschoice                     Prop
##    <dbl> <chr>                         <dbl>
## 1      0 Donald Trump, the Republican 0.212
## 2      0 Joe Biden, the Democrat      0.273
## 3      1 Another candidate            0.0303
## 4      1 Donald Trump, the Republican 0.0303
## 5      1 Joe Biden, the Democrat      0.455
```

From this you can see that 45 percent of the young voters in the sample are female Biden voters. Across all of those who are under 24, only 3% are female Trump voters. Note that by ungrouping what we are effectively calculation is the joint probability of gender *and* vote choice. In contrast, previously we were calculating the probability of vote choice conditional on gender.

Gender Gap: Any Age

We just looked at the gender gap for voters under the age of 24 and found that there were very few voters. So let us consider the relationship between male and female voters regardless of age:

```
MI_final %>%
  group_by(FEMALE) %>%
  summarize(BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
            TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

```
## # A tibble: 2 x 3
##   FEMALE BidenPct TrumpPct
##    <dbl>    <dbl>    <dbl>
## 1      0    0.525    0.427
## 2      1    0.643    0.325
```

Interesting! Note that the results are reported by the group_by variable. Thus, this is asking "What percentage of Male voters voted for Biden and what percentage voted for Trump?" and "What percentage of female voters voted for Biden and what percentage of female voters voted for Trump." It IS NOT asking the percentage of Biden or Trump voters that are male. Instead, it is asking what fraction voted for Biden and Trump *conditional* on being a male (or female). Thus, we can see that 52.5% of males voted for Biden and 64.3% of females voted for Biden. 42.7% of males voted for Trump and only 32.5% of females. (If it seems odd that Biden won a majority of male voters and a large majority of female voters despite winning the state in 2020 50.6% to 47.8%, hold that thought!)

If we want to flip the question and ask the faction of Trump voters who are male and female, we need to change the variable we are grouping by. In particular:

```
MI_final %>%
  group_by(preschoice) %>%
  summarize(MalePct = mean(FEMALE==0),
            FemalePct = mean(FEMALE==1))
```

```
## # A tibble: 6 x 3
##   preschoice                       MalePct FemalePct
##   <chr>                              <dbl>     <dbl>
## 1 Another candidate                  0.68      0.32
## 2 Donald Trump, the Republican       0.538     0.462
## 3 Joe Biden, the Democrat            0.420     0.580
## 4 Refused                            0.5       0.5
## 5 Undecided/Don’t know               0.75      0.25
## 6 Will/Did not vote for president    0.167     0.833
```

This outputs the gender breakdown of each response category. So 58% of the voters for Biden were female (and 42% were male) whereas 53.8% of Trump voters were male and 46.2% were female.

Conditional probability can be tricky and people get this wrong all of the time. The important thing to remember is that the variable we are grouping by is the variable we are conditioning on. So before we were looking at how vote choice varies by gender. Conditional on being male (or female) what fraction supported Trump or Biden? Now we have switched to looking at how gender varies by vote choice. Conditional on voting for Biden (or Trump), what percentage were male and female. While we are using similar words in a slightly different order, the meaning of what we are estimated is very, very different.

But now we can do all kinds of interesting comparisons. How did self-identified hispanics vote compared to non-hispanics:

```
MI_final %>%
  group_by(LATINOS) %>%
  summarize(BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
            TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

```
## # A tibble: 3 x 3
##   LATINOS BidenPct TrumpPct
##     <dbl>    <dbl>    <dbl>
## 1       1    0.611    0.306
## 2       2    0.592    0.372
## 3       9    0.417    0.472
```

Both supported Biden at equal rates, but non-hispanics were slightly more likely to vote for Trump than hispanics and hispanics were more likely to choose a residual category (which is why the "Yes" row associated with hispanics only adds up to .917).

How about LGBT individuals?

```
MI_final %>%
  group_by(LGBT) %>%
  summarize(BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
            TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

```
## # A tibble: 4 x 3
##    LGBT BidenPct TrumpPct
##   <dbl>    <dbl>    <dbl>
## 1     1    0.609    0.304
## 2     2    0.591    0.382
## 3     9    0.478    0.435
## 4    NA    0.587    0.364
```

Similar, but how many LGBT individuals are we talking about in this sample? Let's see…

```
MI_final %>%
  count(LGBT)
```

```
## # A tibble: 4 x 2
##    LGBT     n
##   <dbl> <int>
## 1     1    23
## 2     2   570
## 3     9    23
## 4    NA   615
```

Umm…only 23 respondents. So it may be really hard to know what to make of that result with so few cases!

How about self-identified race?

```
MI_final %>%
  group_by(QRACEAI) %>%
  summarize(BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
            TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

```
## # A tibble: 7 x 3
##   QRACEAI BidenPct TrumpPct
##     <dbl>    <dbl>    <dbl>
## 1       1    0.566   0.400
## 2       2    0.892   0.0784
## 3       3    0.6     0.35
## 4       4    0.692   0.308
## 5       5    0.556   0.278
## 6       6    0.435   0.435
## 7       9    0.429   0.476
```

Massive differences, but every group was again curiously more likely to support Biden. (So why was this race so close? More on this later).

What if we look at the gender gap among whites only? Let's `filter`, `group_by` and `summarize` to see!

```
MI_final %>%
  filter(QRACEAI == 1) %>%
  group_by(FEMALE) %>%
  summarize(BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
            TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
```

```
## # A tibble: 2 x 3
##   FEMALE BidenPct TrumpPct
##    <dbl>    <dbl>    <dbl>
## 1      0    0.512    0.449
## 2      1    0.614    0.356
```

Missing Data – and what characteristics do you associate with a candidate?

Although we do not try to predict who will win using the Exit Poll – as some of the results from above make clear! – we do use it to try to interpret what voters are thinking. One question asks voters to identify the characteristics that they associate with each candidate.

One thing to note is that this question was not asked of every voter. because there are more interesting questions than there is room on the Exit Poll survey, pollsters sometimes ask the questions to a randomly selected subset of respondents to increase the number of questions at the expense of a smaller sample size on those questions. This is what was done on this question.

To begin, let's see the number of respondents selecting each response. Among Biden voters, 362 were not asked the question and they are therefore irrelevant to the analysis that follows.

```
MI_final %>%
  group_by(preschoice) %>%
  count(QLT20)
```

```
## # A tibble: 30 x 3
## # Groups:   preschoice [6]
##    preschoice                    QLT20                                n
##    <chr>                         <fct>                            <int>
##  1 Another candidate             [DON'T READ] Don't know/refused      1
##  2 Another candidate             Can unite the country                2
##  3 Another candidate             Cares about people like me           2
##  4 Another candidate             Has good judgment                    8
##  5 Another candidate             Is a strong leader                   1
##  6 Another candidate             <NA>                                11
##  7 Donald Trump, the Republican [DON'T READ] Don't know/refused     15
##  8 Donald Trump, the Republican Can unite the country                9
##  9 Donald Trump, the Republican Cares about people like me          47
## 10 Donald Trump, the Republican Has good judgment                   44
## # ... with 20 more rows
```

Becasue they have no useful data for us, let's use the filter command to remove voters who did not answer the question. To do so we use the logical `is.na()` which determines "is the data missing (i.e., `na`)" and because we want to analyze only the cases that are not missing we are going to select the not missing cases – i.e., those for which `!is.na()` is `TRUE`.

```
#  Drop Missing data in candidate quality (only asked of half sample)
MI_final %>%
  filter(!is.na(QLT20)) %>%
  group_by(preschoice) %>%
  count(QLT20)
```

```
## # A tibble: 24 x 3
## # Groups:   preschoice [6]
##    preschoice                    QLT20                                n
##    <chr>                         <fct>                            <int>
##  1 Another candidate             [DON'T READ] Don't know/refused      1
##  2 Another candidate             Can unite the country                2
##  3 Another candidate             Cares about people like me           2
##  4 Another candidate             Has good judgment                    8
##  5 Another candidate             Is a strong leader                   1
##  6 Donald Trump, the Republican [DON'T READ] Don't know/refused     15
##  7 Donald Trump, the Republican Can unite the country                9
##  8 Donald Trump, the Republican Cares about people like me          47
##  9 Donald Trump, the Republican Has good judgment                   44
## 10 Donald Trump, the Republican Is a strong leader                 109
## # ... with 14 more rows
```

Bam. There it is. Now we have the table without the missing data to distract us. So one thing we might be interested is how the candidate traits differ between Biden voters and Trump voters. There are several ways to do this, but lets do it slowly and methodically.

First let’s use the filter to identify how Biden and Trump voters describe the characteristics of their candidate. To do so we want to `filter` to the set of Biden and Trump voters, remove any voters who were not asked the quality questions, group the remaining respondents by their vote choice, count the number of each response, and calculate the proportion who give each response within each group.

```
CandQuality <- MI_final %>%
  filter(preschoice=="Joe Biden, the Democrat" | preschoice=="Donald Trump, the Republic
  filter(!is.na(QLT20)) %>%
  group_by(preschoice) %>%
  count(QLT20) %>%
  mutate(Prop = n / sum(n))
```

So let’s see what we just created!

```
CandQuality
```

```
## # A tibble: 10 x 4
## # Groups:   preschoice [2]
##    preschoice                   QLT20                                n   Prop
##    <chr>                        <fct>                            <int>  <dbl>
##  1 Donald Trump, the Republican [DON'T READ] Don’t know/refused    15 0.0670
##  2 Donald Trump, the Republican Can unite the country                9 0.0402
##  3 Donald Trump, the Republican Cares about people like me          47 0.210
##  4 Donald Trump, the Republican Has good judgment                   44 0.196
##  5 Donald Trump, the Republican Is a strong leader                 109 0.487
##  6 Joe Biden, the Democrat      [DON'T READ] Don’t know/refused     8 0.0222
##  7 Joe Biden, the Democrat      Can unite the country              109 0.302
##  8 Joe Biden, the Democrat      Cares about people like me          70 0.194
##  9 Joe Biden, the Democrat      Has good judgment                  147 0.407
## 10 Joe Biden, the Democrat      Is a strong leader                  27 0.0748
```

With this object, we can now filter to look at the reasons being given by each candidate’s supporters using `filter`. For example,

```
filter(CandQuality, preschoice == "Joe Biden, the Democrat")
```

```
## # A tibble: 5 x 4
## # Groups:   preschoice [1]
##   preschoice              QLT20                                n   Prop
##   <chr>                   <fct>                            <int>  <dbl>
## 1 Joe Biden, the Democrat [DON'T READ] Don’t know/refused     8 0.0222
## 2 Joe Biden, the Democrat Can unite the country             109 0.302
## 3 Joe Biden, the Democrat Cares about people like me         70 0.194
## 4 Joe Biden, the Democrat Has good judgment                 147 0.407
## 5 Joe Biden, the Democrat Is a strong leader                 27 0.0748
```

We can use this to compare how the percentage of supporters choosing each reason compares across candidates by extracting the filtered proportions and taking the difference between them!

```
BidenQuality <- filter(CandQuality, preschoice == "Joe Biden, the Democrat")
TrumpQuality <- filter(CandQuality, preschoice == "Donald Trump, the Republican")

BidenQuality$Prop - TrumpQuality$Prop
```

```
## [1] -0.04480362  0.26176049 -0.01591561  0.21077364 -0.41181490
```

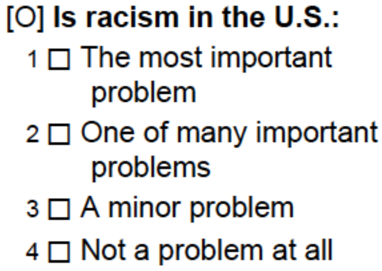
Here positive values indicate traits that Biden supporters are more likely to associate with Biden than Trump supporters are likely to associate with Trump and negative values are traits that Trump voters are more likely to associate with Trump than Biden voters are likely to associate with Biden. So the biggest value is -.41 for “Is a strong leader” meaning that there is a 40 percentage point difference in the fraction of Trump voters who see Trump in those terms compared to Biden voters seeing Biden in those terms. In contrast, the values of .26 indicate that Biden voters are 26 percentage points more likely to think Biden “Can unite the country” relative to how Trump voters think about Trump. Interestingly, supporters of Trump and Biden do not differ much in whether they think their candidate “Cares about people like me”.

So this is useful information because it reveals how differently the supporters of Trump and Biden saw their candidates. Trump voters overwhelmingly described Trump in terms of being a strong leader. That was his most noticeable character trait. In contrast, Biden voters descibed Biden as having “good judgment” and “Can unite the country”. It is clear that the supporters saw their candidates as being very different from one another.

Categorical Variables as continuous variables? How Opinions about the importance of race Varies by Partisanship

Sometimes we treat categorical variables as continuous variables. This can be dangerous, as it requires some strong assumptions about the meaning of the values. Nonetheless, it is sometimes done to help summarize support across the different values. Consider for example the answers to a question about the state of race relations in the United States in 2020. Given the enduring history and legacy of race and race relations in the United States – likely exacerbated and highlighted by the events in the summer of 2020 – the relationship between views towards racism in the US and the support for presidential candidates became a point of emphasis.

The question that was asked of a random half of Exit Poll participants was:



[O] **Is racism in the U.S.:**

1 □ The most important problem

2 □ One of many important problems

3 □ A minor problem

4 □ Not a problem at all

So let's see what we have.

```
MI_final %>%
  count(RACISM20)
```

```
## # A tibble: 6 x 2
##   RACISM20     n
##      <dbl> <int>
## 1        1    41
## 2        2   420
## 3        3   109
## 4        4    38
## 5        9     8
## 6       NA   615
```

Here we see we have missing data that is already recognized by R caused by the question not being asked – i.e., the 615 observations for which `RACISM20` has a value of `NA` – as we as some missing data caused by respondents choosing not to answer the question – coded as a `9` in the original data. As a result, we can either filter the tibble to remove the missing cases or else we can recode those values to be a missing value recognized by R that will be dropped in later analysis. It typically makes sense to recode such cases to missing data rather than dropping the cases. The reason is that if we drop the case and then continue to analyze the tibble using different variables we may have dropped cases that are useful for that analysis. Put differently, filtering the tibble for missing data changes the number of observations that are available for other analyses even though the observations may contain all the information required for those analyses.

```
MI_final$RACISM20[MI_final$RACISM20==9] <- NA
```

To check this worked, let's take the `count` again.

```
count(MI_final,RACISM20)
```

```
## # A tibble: 5 x 2
##   RACISM20     n
##      <dbl> <int>
## 1        1    41
## 2        2   420
## 3        3   109
## 4        4    38
## 5       NA   623
```

So now that we have fixed that, lets think about the way the variable is coded. This scale is OK in that the values are ordered, but it is confusing in that higher values are associated with less concern about racism. A good principle of measurement is to have values and orderings that make intuitive sense. Since we tend to think of an increased concern with higher levels it makes sense to have larger values be associated with more concern. To do this we need to flip the scale being used. Here the mutate command is useful. Since the maximum value is 4 (associated with not a problem at all), if we subtract 4 (so that the maximum value is now 0) and multiple by -1 (so that the lowest value of -3 associated with the most important problem becomes a 3) we can flip the scale so that higher values are associated with more concern and a value of 0 indicates no concern at all.

So let's flip the values and see what we find.

```
MI_final <- mutate(MI_final,
                                   tRACISM20 = -1*(RACISM20-4))

count(MI_final,tRACISM20)
```

```
## # A tibble: 5 x 2
##   tRACISM20     n
##       <dbl> <int>
## 1         0    38
## 2         1   109
## 3         2   420
## 4         3    41
## 5        NA   623
```

Better – now those who think that the issue of racism is the "most important" problem have the highest value ($) and those who think racism is "not a problem at all" have the lowest value (0). It is very tempting to simply use the value labels to measure differences in opinions about the impact of racism in the United States – which is what we are to do – but let's first think very hard about what that assumes about opinions towards racism.

When we measure opinions towards racism using the value labels – and we then do calculations like take the average opinion of various groups of voters – we are explicitly saying that the value differences have meaning. That is, the difference in the importance of racism between those who think it is "not a problem at all" (1) and those who say it is a "minor problem" (2) is the same as the difference in the importance of racism between those who say it is "the most important" (4) problem and those who say it is "One of many important problems" (3). Why does our measurement assume this? Because `2-1 =1` and so does `4-3=1`. Relatedly, the importance of racism to those who say it is the "most important" issue is twice as large as those saying that it is a "minor problem"! This is where the critical thinking part of data science comes into play! It is tempting to convert responses into numeric values and analyze those values, but the values you assign to the responses has important implications for how the opinions compare to one another!

So what do we do if we disagree with the assumptions implied by using those numeric value labels? Here again we can use *indicator variables* to help. In particular, we could create new variables indicating whether a respondent thought that the issue of racism was the "most important" problem – or perhaps whether the respondent thought it was either the "most important" or "one of many important problems" given that there are relatively few respondents who say it is the "most important" problem.

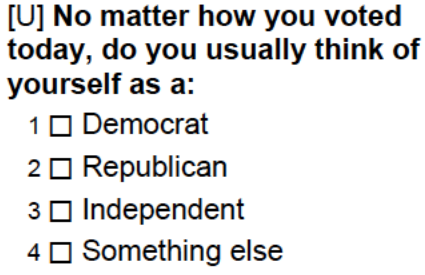
One way to do this is to create a "holding" variable called `ImpRacism` in our `MI_final` tibble and then replace the `NA` values to be a 1 if the respondent provides a (recoded) value of 3 or more and a value of 0 of the value is 2 or less

```
MI_final$ImpRacism <- NA
MI_final$ImpRacism[MI_final$tRACISM20 >= 3] <- 1
MI_final$ImpRacism[MI_final$tRACISM20 <= 2] <- 0
```

So now that we have discussed the outcome of interest – opinions towards the importance of racism in 2020 – we can now consider whose opinions on that issue we want to compare.

In the United States, partisanship is increasingly important for how and what people think and act. It is also the main way in which political conflict is organized – debates about policy issues almost always divide along party lines with the opinions of Democrats and Republicans defining the political cleavages. Given that, lets see how opinions towards the importance of racism vary by partisanship.

In particular, how how do perceptions of the importance of racism in the US vary between Michigan voters who self-identify as Democrats and Republicans? Here we want to summarize the average concern about racism by party. To do so we can group_by() and then summarize the transformed variable. But first we want to make sure that our variable of partisanship makes sense. The question that the exit poll asked about partisanship was as follows"



[U] No matter how you voted today, do you usually think of yourself as a:

1 □ Democrat

2 □ Republican

3 □ Independent

4 □ Something else

To see how people responded to this question we can take a look at the resulting `count`:

```
count(MI_final,PARTYID)
```

```
## # A tibble: 5 x 2
##   PARTYID     n
##     <dbl> <int>
## 1       1   425
## 2       2   280
## 3       3   416
## 4       4    94
## 5       9    16
```

It is not exactly immediately obvious what these values mean, which means we should do some work to get them into a more easily interpretable output. A good place to start is to think about what we are interested in. It seems that we want to compare the opinions of respondents who self-idnetify as Democrats to the opinions of those who self-identify as Republicans and how that also compared to the opinions of those who self-identify as independents. If so, we should create a variable that lets us easily identify each group

```
MI_final$party3 <- NA
MI_final$party3[MI_final$PARTYID==1] <- "Democrat"
MI_final$party3[MI_final$PARTYID==2] <- "Republican"
MI_final$party3[MI_final$PARTYID==3] <- "Independent"
```

Note that we have left those who have a `PARTYID==4` as missing. That is, we are choosing to exclude those who self-identify with a party other than the two major parties. So let's focus on the set of respondents who self-identify as a Democrat, Republican, or independent by filtering out those respondents with a newly-created `party3` variable that is missing.

```
MI_final %>%
  filter(!is.na(party3)) %>%
  group_by(party3) %>%
  summarize(ImpRacism = mean(tRACISM20))
```

```
## # A tibble: 3 x 2
##   party3      ImpRacism
##   <chr>           <dbl>
## 1 Democrat           NA
## 2 Independent        NA
## 3 Republican         NA
```

What happened? Let's take a look at whether there is missing data.

```
sum(is.na(MI_final$tRACISM20))
```

```
## [1] 623
```

There's the problem! If there is any missing data, R will crash when taking a mean unless we tell it explicitly to remove the missing data before taking the mean. We do this by adding `na.rm=TRUE` to the mean function call. Thus, we need to do:

```
MI_final %>%
  filter(!is.na(party3)) %>%
  group_by(party3) %>%
  summarize(ImpRacism = mean(tRACISM20, na.rm=TRUE))
```

```
## # A tibble: 3 x 2
##   party3      ImpRacism
##   <chr>           <dbl>
## 1 Democrat         2.05
## 2 Independent      1.79
## 3 Republican       1.37
```

But now we can rock and roll and we can see how perceptions of the importance of racism varies between various demographic groups.

By self-identified race:

```
MI_final %>%
  group_by(QRACEAI) %>%
  summarize(ImpRacism = mean(tRACISM20, na.rm=TRUE))
```

```
## # A tibble: 7 x 2
##   QRACEAI ImpRacism
##     <dbl>     <dbl>
## 1       1      1.76
## 2       2      2.04
## 3       3      1.22
## 4       4      2
## 5       5      1.62
## 6       6      1.8
## 7       9      1.38
```

Education level:

```
MI_final %>%
  group_by(EDUC18) %>%
  summarize(ImpRacism = mean(tRACISM20, na.rm=TRUE))
```

```
## # A tibble: 6 x 2
##   EDUC18 ImpRacism
##    <dbl>     <dbl>
## 1      1      1.62
## 2      2      1.73
## 3      3      1.62
## 4      4      1.86
## 5      5      1.86
## 6      9      2
```

Among white voters depending on whether or not they are “born again”:

```
filter(MI_final, QRACEAI == 1) %>%
  group_by(BRNAGAIN) %>%
  summarize(ImpRacism = mean(tRACISM20, na.rm=TRUE))
```

```
## # A tibble: 4 x 2
##   BRNAGAIN ImpRacism
##      <dbl>     <dbl>
## 1        1      1.61
## 2        2      1.84
## 3        9      1.85
## 4       NA    NaN
```

Among white voters by various education levels:

```
filter(MI_final, QRACEAI == 1) %>%
  group_by(EDUC18) %>%
  summarize(ImpRacism = mean(tRACISM20, na.rm=TRUE))
```

```
## # A tibble: 6 x 2
##   EDUC18 ImpRacism
##    <dbl>     <dbl>
## 1      1      1.59
## 2      2      1.75
## 3      3      1.59
## 4      4      1.88
## 5      5      1.87
## 6      9      2
```

Using Summaries as Objects

While these are all interesting, we can also use this – as well as the resampling we talked about earlier – to determine just how different these average concerns are. In contemporary politics you will often hear discussions about how race and education levels structure opinions. So let’s see how much opinions towards the issue of racism in the United States differs between white and Black voters based on their education level.

To begin we define the average concern with racism for white Michigan voters by education level WhiteEdRacism and the same for Black Michigan voters (BlackEdRacism).

```
WhiteEdRacism <- filter(MI_final, QRACEAI == 1) %>%
  group_by(EDUC18) %>%
  summarize(ImpRacism = mean(tRACISM20, na.rm=TRUE))

BlackEdRacism <- filter(MI_final, QRACEAI == 2) %>%
  group_by(EDUC18) %>%
  summarize(ImpRacism = mean(tRACISM20, na.rm=TRUE))
```

If we want to see how they differ, we can look at the difference between these two objects:

```
BlackEdRacism - WhiteEdRacism
```

```
##   EDUC18 ImpRacism
## 1      0 0.4142857
## 2      0 0.2477876
## 3      0 0.4096386
## 4      0 0.3035573
## 5      0 0.1282051
## 6      0 0.0000000
```

So that is how much the average concern varies (interpret), but how certain are we of that difference? We know that some of those comparisons have relative few respondents in them and we always want to quantify how certain we are of the relationships we report. Numbers without an attempt to quantify uncertainty are not scientific because you have no idea what they mean.

So let's use the resampling to get this!

CI for Biden and Trump

It is always important to quantify our uncertainty. Let's use the tools we talked about to quantify how much our results might change based on random resampling. That is, *if* our sample of respondents is random sample of Michigan voters, how much woud the results change if we were to take another random sample?

```
NObs <- nrow(MI_final)  # HAS TO BE THE NUMBER OF VALID OBSERVATIONS -- ASSUMES NO MISS
n.samples <- 1000
Est.Pct <- NULL

# tidyverse creates list() objects

for(i in 1:n.samples){
  dat.sample <- sample_n(MI_final,NObs,replace= TRUE)
  Est.Pct[[i]] <- dat.sample %>%
    summarize(BidenPct = mean(preschoice=="Joe Biden, the Democrat"),
              TrumpPct = mean(preschoice=="Donald Trump, the Republican"))
}
```

```
# Better way
Est.Pct.Matrix <- bind_rows(Est.Pct)

unlist(Est.Pct)[1:10]
```

```
##  BidenPct  TrumpPct  BidenPct  TrumpPct  BidenPct  TrumpPct  BidenPct  TrumpPct
## 0.5840780 0.3777417 0.5946385 0.3720552 0.5987002 0.3712429 0.6011373 0.3614947
##  BidenPct  TrumpPct
## 0.5751422 0.3874898
```

```
Est.Pct.Matrix <- matrix(unlist(Est.Pct),
                         ncol=2,
                         byrow=TRUE)

quantile(Est.Pct.Matrix[,1],c(.025,.975))
```

```
##      2.5%     97.5%
## 0.5597076 0.6149472
```

```
quantile(Est.Pct.Matrix[,2],c(.025,.975))
```

```
##      2.5%     97.5%
## 0.3452478 0.3980504
```

```
# Prob Biden > 60
mean(Est.Pct.Matrix[,1] > .6)
```

```
## [1] 0.186
```

```
# Prob Biden > Trump
mean(Est.Pct.Matrix[,1] > Est.Pct.Matrix[,2])
```

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

Average Difference in Importance of Racism?

Once we talk about resampling, we can use those tools to examine how much variation there is to be expected in those characterizations based on random sampling alone. That is, if the respondents are a random sample of Michigan voters, how much might our calculations change if we were to take another random sample of the same size?

```
n.samples <- 1000
Pty.Racism.Mean <- NULL
NObs <- sum(!is.na(MI_final$tRACISM20))

for(i in 1:n.samples){
  dat.sample <- sample_n(MI_final,NObs,replace= TRUE)
  Pty.Racism.Mean[[i]] <- dat.sample %>%
    summarize(DemAvgRacismImp = mean(tRACISM20[PARTYID == 1],
                                     na.rm=TRUE),
              RepAvgRacismImp = mean(tRACISM20[PARTYID == 2],
                                     na.rm=TRUE))
}
```

Now let's work with the output to analyze it.

```
unlist(Pty.Racism.Mean)[1:10]
```

```
## DemAvgRacismImp RepAvgRacismImp DemAvgRacismImp RepAvgRacismImp DemAvgRacismImp
##        2.035714        1.343284        2.093750        1.500000        2.037383
## RepAvgRacismImp DemAvgRacismImp RepAvgRacismImp DemAvgRacismImp RepAvgRacismImp
##        1.322581        2.093458        1.474576        1.918699        1.390625
```

```
Pty.Racism.Mean.Matrix <- matrix(unlist(Pty.Racism.Mean),
                         ncol=2,
                         byrow=TRUE)

quantile(Pty.Racism.Mean.Matrix[,1],c(.025,.975))
```

```
##     2.5%    97.5%
## 1.947826 2.145846
```

```
quantile(Pty.Racism.Mean.Matrix[,2],c(.025,.975))
```

```
##     2.5%    97.5%
## 1.214286 1.532489
```

```
DemRepDiff <- Pty.Racism.Mean.Matrix[,1] - Pty.Racism.Mean.Matrix[,2]
mean(DemRepDiff)
```

```
## [1] 0.6741454
```

```
quantile(DemRepDiff,c(.025,.975))
```

```
##      2.5%     97.5%
## 0.4781833 0.8612428
```

Now we can also do other analyses to answer other closely related questions. What is the probability that the average gap between the importance of racism among Democrats and Republicans is .5 on our scale running from 1 ("not important at all") to 4 ("most important").

To do so we can simply calulate the proportion of resampled average differences that are larger than .5!

```
# Probability that Avg Diff > .5?
mean(as.logical(DemRepDiff > .5))
```

```
## [1] 0.964
```

How about a difference of .75?

```
# Probability that Avg Diff > .75?
mean(as.logical(DemRepDiff > .75))
```

```
## [1] 0.221
```