# Regressions and Output in R

# Data Set Up

We have done some basic data cleanup so far; however, we have covered the minimum. I want to start off with shortcuts and key commands that we can use to help smooth the process of cleaning our data. Let’s start by setting up a new script for Homework 3.

#Economics 691 Classroom Example 3

#November 8, 2022

rm(list=ls())

library(tidyverse)

library(tidycensus)

votes<-read.csv("./Data/Vote Data.csv")

The data we are using is a download of the data from the MIT Election Lab data (https://electionlab.mit.edu/data) which was the easiest way to get the 2020 election data by county without the need to scrape a website. This data includes results back as far as 2000 so we need to limit the dataset down. Additionally, we do not want all of the states, rather we only want the states we want to focus on which are Indiana, Illinois, and Michigan. Additionally, the data includes any other presidential candidates that those states might have had, which can differ by state, among other factors in the data.

Let’s talk about this syntax for just a bit. We know that this data is in our Data subdirectory under our project folder; however, remember that we want anyone to be able to carry out this project. By using the version control and project tools in RStuido, we can lead our location with “**./**” which is telling R to go to the current working directory and start from there to look for a Data subdirectory and then the file in question. If we put the full directory address, such as “**c:/ta0jrg1/Documents/Projects/Data/…**”, and then you tried to run this on your system, it would fail because the location of your project files are someplace completely different. So using projects and the GIT tools, the current working directory on YOUR MACHINE is already known to YOUR MACHINE and so I do not have to know to make sure the file is where your computer can find it.

To clean this data, we want to complete the following tasks:

* Limit the data to 2020 and the three states of interest
* Find the total votes cast in the county.
* Find the percentages for only the two major candidates.
* Pivot the table from long to wide.

v.cast<-votes %>%

filter(year==2020) %>%

group\_by(county\_fips) %>%

summarize(cast = sum(totalvotes))

Breaking down this code chunk we see that we use the filter command to limit the data to the year 2020 and then we use the group\_by() command to tell R that we want to define groups. Here we use the *county\_fips* variable which is the county FIPS code and uniquely identifies each county. The summarize() command is creating a summary by those groups which can include means, medians, or in our case, sums. We want to sum the *totalvotes* variable using the sum() command.

When we run this code we get a new object with only two columns and this is why we did not put this in a nested code with other objects on our to-do list because the summarize part is going to create a new object with the IDs (here county fips codes) and the summary statistics we asked for. We can quickly join this with our core data easily and it is not a bad thing because when you use group\_by() once, it stays with that object and can cause problems with other commands later unless the data is ungrouped. By using it to create a completely different object, we do not have to worry about it impacting our core vote data.

far and that is the saving of the data. We can recall data in one of two ways. The first is to use the source() command to run another existing script from the current script. While this ensures we have the most recent version of the data and we have the right sequencing of the scripts, some scripts may take a longer time to run.

states <- c("IN", "IL", "MI")

main<-votes %>%

filter(year==2020) %>%

full\_join(., v.cast, by="county\_fips") %>%

filter(state\_po %in% states) %>%

filter(party == "DEMOCRAT" | party=="REPUBLICAN") %>%

mutate(cand = case\_when(party == "DEMOCRAT" ~ "Biden",

party == "REPUBLICAN" ~ "Trump"),

percent = candidatevotes/cast) %>%

select(state, county\_name, county\_fips, cand, candidatevotes, percent) %>%

pivot\_wider(names\_from = cand, values\_from = c(percent, candidatevotes)) %>%

rename(Biden = candidatevotes\_Biden,

Trump = candidatevotes\_Trump,

pctBiden = percent\_Biden,

pctTrump = percent\_Trump) %>%

left\_join(., v.cast, by="county\_fips")%>%

mutate(Other = cast - Biden - Trump,

pctOther = 1 - pctBiden – pctTrump,

GEOID = as.character(county\_fips)) %>%

select(-county\_fips)

The next code chunk will clean up our vote data and crate the variables we wanted to add. This chunk also highlights the advantage of piping commands together as we cover a lot of ground in one command. Since we are going back to core vote data, we must filter down to the 2020 data again and then we join in our total votes cast that we created before [NOTE: observant students will notice we did not need to county the total votes cast, however, I covered this here to show a simplified version of something that would have been a bigger problem with some states in this dataset that listed cast and absentee ballots separately so we would have needed to group by county and candidate to get their complete vote total].

Next, we will use our first shortcut to filter out the states. We could use a logic command in filter similar to what we do for party (line 5 in our pipped code). This can get cumbersome if we have a large list of things so we can use the strategy we use here and create an object **states** with our desired states listed. Then, in our filter() command, we use the command %in% to tell R to keep only those where the *state\_po* variable are listed in the object **states**. In our first mutate() command line we see another way to simplify our lives and that is using the case\_when() command. Many times, we will want to create a variable that takes on one value in some cases and another value in other cases. The best example of this is a dummy or identifier variable. The case\_when() command allows use to fill in one new variable, here *cand*, with the different possible options based on what the variable *party* is. We are doing this because the candidates’ full names are used, and this is cleaner than trying to break up the name string to get the last name only. Some key features to notice in this syntax: 1) we use double equal signs because the first part is a logical argument, 2) we use the squiggle ‘~’ to assign the value we want our new variable to take when the preceding logical arguments holds.

Next comes the select() command to take out some of the columns we no longer needed and then we use pivot() to turn the table into a wider version of the data meaning we have just one observation per county. We follow that with the rename() command to change the names for the created columns. Finally, we know we also want to create a variable for votes case for all other candidates, so we join our votes case data back in again, but this time we use the left\_join() command because the **v.cast** object has ALL of the states and we only want to keep those in our existing dataset and so anything not matching the left dataset (in the join list) will be dropped. We close out with the mutate() again to create the new variables we need and create the variable GEOID, which we will use later, by translating the *county\_fips* from an integer to a character and then remove the latter from our dataframe.

The one element we have failed to cover at this point, which given the way we are writing scripts is not as important, is the idea of saving our work. We can save either specific dataframe or the entire environment in R using either the save() or save.image() commands respectively.

save(df, file="./Data/df.RData")

save.image(file="./Data/df.RData")

When I use the save() command, I must specify the dataframe(s) I want to save, however, when I use the save.image() command, it will save everything currently in the environment. To get the files back, we simply call the load() command in our console or script. You should note; however, that when you use the load command, it will load the dataframe as the object it was saved as and replace any existing dataframe objects with that same name. Unless memory is a key constraint or if you have a cleaning operation that takes a great deal of time and you do not want to repeat often, you only need save at the end of your script.

#Census Data####

var<-c("B01001\_001", "B01001\_002", "B02001\_002", "B02001\_003", "B01002\_002", "B01002\_003")

fips <-c(17, 18, 26)

acs<-get\_acs(geography = "county",

variables = var,

year = 2020,

state = fips,

geometry = TRUE)

census<-acs %>%

select(GEOID, variable, estimate) %>%

pivot\_wider(names\_from = variable, values\_from = estimate) %>%

rename("TotPop" = "B01001\_001",

"Male" = "B01001\_002",

"White" = "B02001\_002",

"Black" = "B02001\_003",

"Age\_M" = "B01002\_002",

"Age\_F" = "B01002\_003") %>%

mutate(pMale = Male/TotPop,

pWhite = White/TotPop,

pBlack = Black/TotPop,

geometry = NULL)

Now we are going to pull in our census data and we are going to use our predefine trick here again to define our state FIPS codes we want and our variable lists. Before we do that, however, notice how we commented out the section we are working in and then followed the section title with four has marks. The four hash marks at the end will allow use to collapse this section of our code in our script to make it look nicer later. There is nothing new here that we did not already do in week 2. We notice that we have a GEOID variable here that is the county level FIPS code and so we can join on that to avoid our problem with county names we faced in last week’s example.

To finish off the first part of our core dataset, we want to create a map object so we keep our sf features to draw our maps and then bring all three of our objects together. This is in the code below.

core<-acs %>%

select(GEOID, geometry) %>%

distinct() %>%

full\_join(., census, by="GEOID") %>%

full\_join(., main, by="GEOID")

The very last piece of data we want to add to this is our COVID-19 data from our first day of class. Open your script from that class [In my case it is Class Example 1.R]. I have a few options with how to proceed from here. Option one is to make any corrections in this code and then, in my Class Example 3.R script, use the source() command to have R locate the Class Example 1.R script and run it as it runs the Class Example 3.R script. If I do that, I need to be careful to remove the rm() command from the start of my Class Example 1.R script or everything I have done in Class Example 3.R so far will be lost.

Option two is to copy and paste the necessary code from our first script into our current script. Option three is to fix what we want in the first script and then save the dataframe we create and then load it back into the environment as part of the Class Example 3.R script. Any method is fine; however, I think, given the shortness of the code, option two is likely best. So, we will copy and paste our code into our current example script.

#Census Data####

#COVID 19 Data from Class Example 1.R Script####

delta<-function(x){

temp<-((x-lag(x))/lag(x))

return(round(temp,4))

}

covid<-read.csv("https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-counties.csv")

You will notice that once I put the four hash marks after my new section title, there is a collapse arrow that shows up along the side of my script. If I go up to the Census Data section title and click it, that section of code, except for the title, collapses.

Upon examination of the covid data, we see that those at the New York Times must love us, or more likely are competent data scientists, and they have a variable called *fips* in the dataset. This means we, again, do not have to worry about name matching. We do; however, need to make sure it is of the same type as the *GEOID* in our data, which is a character type. We can use the command is.character() to see and it returns a FALSE meaning the *fips* in the covid data is not a character, so we know we need to fix that. We also know that we want the final number of cases and deaths for our use in each of our counties and by looking at the data we see that the cases and deaths variable are cumulative. This means we just need the last case of each of our counties in our data. Finally, if we have any counties with no cases or deaths or are not in our covid dataset, we need to control for that. With those factors in mind, let’s write the code to do all of that.

covid.2<- covid %>%

mutate(GEOID = as.character(fips),

Date=as.Date(date, "%Y-%m-%d")) %>%

filter(GEOID %in% main$GEOID) %>%

arrange(GEOID, Date) %>%

group\_by(GEOID) %>%

summarise\_all(last)

While that sounds daunting, the seven lines of code slims our dataset of more than two million observations down to just 277. First notice that I assign this to a different object because I know loading the covid data in the previous code chunk took some time. Secondly, I take care of my fixes in the first couple of lines as I create a *GEOID* variable to ensure it is of the correct type and then make sure that R knows that I am dealing with a date because remember I want the most recent observation from each county. The next line does a great deal of heavy lifting. Recall how I used the %in% command previously? I can do it again with an existing variable in an existing dataframe. Specifically, I want all the GEOIDs that I have in the **core** dataset I just finished building and so I will use the %in% command again to instruct R to keep everything in the covid dataset with a GEOID that can be found in the GEOIDs of the **core** object.

The last three lines of code set up the dataset to pull out the last report from each county. First, we use arrange(), which is the *tidyverse* version of sort, followed by the group\_by() command again. Finally, we create a summary, of sorts, and we want everything that is in the dataframe, so we use summarise\_all(last) to instruct R to take the last observation (which was sorted by date) for each group. The addition of \_all is just a way of saying we want ALL the variable in the dataset, not just a subset.

Running this chunk produces a new dataframe with 277 observations (so we can expect we have all the counties in our data present) and we also notice that the last date is May 13, 2022, which is the end point of the data collection and reporting by the New York Times. We could have just filtered by that date; however, there is no assurance that something did not drop out before that so our method ensures the most complete data.

To determine if we have more work ahead of us, lets join this to the *core* object using the full\_join() command and then summary(core) to see if any NAs are listed. In our case we do not see any cases of NAs so we had matches for everything. We do; however, notice that some column names are changed with either “.x” or “.y” added to the end. This indicates there were duplicated column names in the two datasets that contained different types. We can fix this by selecting those out before we join our covid.2 data to the core data. If we did encounter any NAs, we can fix those as well using the last command shown in the code chunk below.

core <- core %>%

select(-c(state, county\_name)) %>%

full\_join(., covid.2, by="GEOID") %>%

select(-c(Date, county, state, fips )) #%>%

#replace(is.na(.), 0)

You will notice that I commented out the last pipe mark and last line, because I did not need it. If I did, the command is.na() is a logical command to determine if an observation is an NA and replace() command is a way to replace specific observations with an alternative, in this case zero.

Congratulations, you have just created a master dataset pulling information from three different databases (MIT Election Lab, U.S. Census, and New York Times Covid) to create a master dataset upon which we can now run regression analysis.

# Simple Regressions

# Now that we have the core data, we can run the regressions. The command we use is lm() and the syntax has use list the equation that we wish to estimate using the squiggle in place of the equal’s sign. The lm() command will automatically assume a constant, but we can remove that by adding -1 to the end of our list of independent variables. The next element of the syntax is the location of data. So, to create mod1, we run the code below.

mod1<-lm(pctBiden ~ pMale + pBlack + Age\_M + Age\_F, data=core)

# When we run a regression, we want to assign it to an object because what is produced is a list object as you can see in the environment. To see what we generally think about when we think about regression results, we use the summary() command on that object which produces the output below in the console.

> summary(mod1)

Call:

lm(formula = pctBiden ~ pMale + pBlack + Age\_M + Age\_F, data = core)

Residuals:

Min 1Q Median 3Q Max

-0.278661 -0.070948 -0.006552 0.058211 0.286546

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 0.9754433 0.1654074 5.897 1.09e-08 \*\*\*

pMale -1.1734079 0.3410531 -3.441 0.000672 \*\*\*

pBlack 1.0301181 0.0973264 10.584 < 2e-16 \*\*\*

Age\_M -0.0004614 0.0042283 -0.109 0.913192

Age\_F -0.0017460 0.0044687 -0.391 0.696308

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.09023 on 272 degrees of freedom

Multiple R-squared: 0.3717, Adjusted R-squared: 0.3624

F-statistic: 40.23 on 4 and 272 DF, p-value: < 2.2e-16

# The first thing listed is the equation estimated as shown via the command we ran. Next, we see summary of the residuals followed by the coefficient estimates, standard errors, t-values, p-values, and significancy marks. Finally, we have some other diagnostic statistics listed as well. We see that the percentage of males in the county is negative and significant indicating that for each additional percentage point of males in a county, Biden would loose about 1.17 percentage points in support. The percentage of blacks in the county; however, increased support for Biden about one-for-one. Finally, the median age of mean and women seem to be insignificant meaning that counties with younger or older voters did not matter too much.

# What about the fact that in our sample we expect that Illinois is known as a blue (democrat) state while Indiana and Michigan are known as red (republican) states? We can add state fixed effects in one of two ways. The first would be to create a dummy variable for each state and the second would be to utilize the factor ability in R. We will start with using the factor tool.

core2 <- core %>%

mutate(s.fips = substr(GEOID, 1, 2),

state = case\_when(s.fips == "17" ~ "Indiana",

s.fips == "18" ~ "Illinois",

s.fips == "26" ~ "Michigan"))

core2$state<-factor(core2$state, levels = c("Indiana", "Illinois", "Michigan"))

mod2<-lm(pctBiden ~ pMale + pBlack + Age\_M + Age\_F + state, data=core2)

# In this code we use pull out the first two characters of the GEOID because we know that indicates the state and then we create a new variable called state and use our case\_when() command to create a character variable indicating the data. Finally, we use the factor() command to tell R that the variable *state* contains identifying factors and then we use the levels = option to denote the levels. This is important because when we run the regression, to avoid singularity, R will drop the level that is “first”. By listing the levels, we can dictate which gets drop. The output we get is shown below.

Call:

lm(formula = pctBiden ~ pMale + pBlack + Age\_M + Age\_F + state,

data = core2)

Residuals:

Min 1Q Median 3Q Max

-0.22283 -0.05194 -0.01302 0.04515 0.26886

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1.258848 0.144876 8.689 3.58e-16 \*\*\*

pMale -1.433251 0.292671 -4.897 1.68e-06 \*\*\*

pBlack 0.904517 0.085758 10.547 < 2e-16 \*\*\*

Age\_M -0.003248 0.003633 -0.894 0.3722

Age\_F -0.002934 0.003843 -0.764 0.4458

stateIllinois -0.029140 0.011606 -2.511 0.0126 \*

stateMichigan 0.094194 0.011786 7.992 3.89e-14 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.07713 on 270 degrees of freedom

Multiple R-squared: 0.5443, Adjusted R-squared: 0.5341

F-statistic: 53.74 on 6 and 270 DF, p-value: < 2.2e-16

# We see, surprisingly, that, relative to Indiana, Illinois slightly lowers the percentage of votes to Biden. This is likely caused by our previously included controls and the impact is small, both in magnitude and significance. Doing this with dummy variables is shown below.

core3<-core %>%

mutate(s.fips = substr(GEOID, 1, 2),

Indiana = case\_when(s.fips == "17" ~ 1,

TRUE ~ 0),

Illinois = case\_when(s.fips == "18" ~ 1,

TRUE ~ 0),

Michigan = case\_when(s.fips == "26" ~ 1,

TRUE ~ 0))

mod3<-lm(pctBiden ~ pMale + pBlack + Age\_M + Age\_F + Illinois + Michigan, data=core3)

# This methodology is a bit more intensive, but we see the results are the same

Call:

lm(formula = pctBiden ~ pMale + pBlack + Age\_M + Age\_F + Illinois +

Michigan, data = core3)

Residuals:

Min 1Q Median 3Q Max

-0.22283 -0.05194 -0.01302 0.04515 0.26886

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1.258848 0.144876 8.689 3.58e-16 \*\*\*

pMale -1.433251 0.292671 -4.897 1.68e-06 \*\*\*

pBlack 0.904517 0.085758 10.547 < 2e-16 \*\*\*

Age\_M -0.003248 0.003633 -0.894 0.3722

Age\_F -0.002934 0.003843 -0.764 0.4458

Illinois -0.029140 0.011606 -2.511 0.0126 \*

Michigan 0.094194 0.011786 7.992 3.89e-14 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.07713 on 270 degrees of freedom

Multiple R-squared: 0.5443, Adjusted R-squared: 0.5341

F-statistic: 53.74 on 6 and 270 DF, p-value: < 2.2e-16

# If we wanted to include all the states, we could do so and then remove the constant with the call below

Call:

lm(formula = pctBiden ~ pMale + pBlack + Age\_M + Age\_F + state -

1, data = core2)

Residuals:

Min 1Q Median 3Q Max

-0.22283 -0.05194 -0.01302 0.04515 0.26886

Coefficients:

Estimate Std. Error t value Pr(>|t|)

pMale -1.433251 0.292671 -4.897 1.68e-06 \*\*\*

pBlack 0.904517 0.085758 10.547 < 2e-16 \*\*\*

Age\_M -0.003248 0.003633 -0.894 0.372

Age\_F -0.002934 0.003843 -0.764 0.446

stateIndiana 1.258848 0.144876 8.689 3.58e-16 \*\*\*

stateIllinois 1.229709 0.143650 8.560 8.65e-16 \*\*\*

stateMichigan 1.353042 0.146425 9.241 < 2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.07713 on 270 degrees of freedom

Multiple R-squared: 0.9532, Adjusted R-squared: 0.952

F-statistic: 786.2 on 7 and 270 DF, p-value: < 2.2e-16

# By adding the -1 at the end, the constant is removed, and we see that Indiana is slightly higher than Illinois in the coefficient estimate.

# Let’s try one more where we regressive the number of cases of COVID-19 and the number of deaths on the votes for each candidate. First, we will modify our variables a bit and find the cases and deaths per 10,000 population by dividing both the cases and deaths variables by the total population and multiply by 10,000.

core4<-core2 %>%

mutate(case2 = (cases/TotPop)\*10000,

death2 = (deaths/TotPop)\*10000,

Biden2 = (Biden/TotPop)\*10000,

Trump2 = (Trump/TotPop)\*10000)

mod4<-lm(case2 ~ pMale + pBlack + Age\_M + Age\_F + Biden2 + Trump2 + state, data = core4)

# First notice we also modify the vote counts so that they are on the same scale and then run the model. The results are shown below.

Call:

lm(formula = case2 ~ pMale + pBlack + Age\_M + Age\_F + Biden2 +

Trump2 + state, data = core4)

Residuals:

Min 1Q Median 3Q Max

-1286.97 -220.24 4.46 193.76 869.34

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 2105.23421 720.38550 2.922 0.003770 \*\*

pMale 3230.90305 1357.66592 2.380 0.018024 \*

pBlack 1478.73412 469.81122 3.148 0.001833 \*\*

Age\_M -52.82068 16.22450 -3.256 0.001277 \*\*

Age\_F 17.96672 16.83531 1.067 0.286840

Biden2 -0.15082 0.05362 -2.813 0.005277 \*\*

Trump2 0.21208 0.06317 3.357 0.000902 \*\*\*

stateIllinois -304.91407 53.99495 -5.647 4.16e-08 \*\*\*

stateMichigan -359.35958 60.88585 -5.902 1.08e-08 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 337.4 on 268 degrees of freedom

Multiple R-squared: 0.438, Adjusted R-squared: 0.4212

F-statistic: 26.11 on 8 and 268 DF, p-value: < 2.2e-16

Notice that for every one vote per 10,000 that went for Biden, reduced the number of cases, per 10,000 by about 0.15 cases whereas more votes for Trump indicate higher cases.

Call:

lm(formula = death2 ~ pMale + pBlack + Age\_M + Age\_F + Biden2 +

Trump2 + state, data = core4)

Residuals:

Min 1Q Median 3Q Max

-29.781 -4.588 0.200 5.351 32.618

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 8.938e+01 2.099e+01 4.259 2.85e-05 \*\*\*

pMale -1.583e+02 3.955e+01 -4.001 8.17e-05 \*\*\*

pBlack 1.834e+01 1.369e+01 1.340 0.181401

Age\_M -4.449e-01 4.727e-01 -0.941 0.347400

Age\_F 1.526e+00 4.905e-01 3.112 0.002060 \*\*

Biden2 -9.678e-03 1.562e-03 -6.195 2.19e-09 \*\*\*

Trump2 -2.479e-03 1.841e-03 -1.347 0.179148

stateIllinois 2.857e+00 1.573e+00 1.816 0.070467 .

stateMichigan 6.758e+00 1.774e+00 3.810 0.000173 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 9.829 on 268 degrees of freedom

Multiple R-squared: 0.2947, Adjusted R-squared: 0.2736

F-statistic: 14 on 8 and 268 DF, p-value: < 2.2e-16

We get similar, albeit less significant, results for deaths.

## Displaying results

When publishing results, we need show our results together in a single table rather individual tables and we also need to show a table of our summary statistics. We could output the values to a text file and then modify the table inside a program such as Excel or we find a package to output our regression results and summary statistics in an organized way that allows us to still do some customization. Three packages I have found useful are *jtools [summ()]*, *stargazer*, and *finalfit*. Of the three, stargazer is the easiest to use; however, there are cases when stargazer does not support the model type (such as survival models) and you will need another package. You can google search each of these for documentation to use; we will focus on using the *stargazer* package.

The regression results can be output in one of three ways: (1) on screen in the console, (2) as an html file, (3) as a text file, or (4) latex script. These are identified with the type= option in the code and the out= tells R where to save the selected output. Typically, I will use html if I am going to load the table into Word or Excel. If (or rather when) you work with latex, you will be able to output the script and then just drop it into your latex document to produce the table.

Another nice feature of stargazer is that it can distinguish between summary statistics and regression model output and produce nice tables for the respective data. To highlight that fact that stargazer has issues with some types of data, it does not recognize *sf* objects so we need to get rid of our geometry if we want to create a table of summary stats.

core2t <- core2

core2t$geometry <- NULL

stargazer(core2t, type="html", out="./Analysis/Summary.html")

stargazer(mod1, mod2, mod3, type="html", out="./Analysis/Results.html")

Graphical user interface, text, application, Word

Description automatically generated

We can also report the results of a series of models using the stargazer package and command.

Graphical user interface, application

Description automatically generated

We can easily change any of the features of these tables using the options within the stargazer package or we can export them and then modify them further in whatever we are creating our presentation within.

## Visualizations

Sometimes the most powerful way to display something is with a graph or image. Consider the case of the number deaths per 10,000 people across my states. I can create a map showing that using the ggplot2 package and my geographic information from the census.

Map

Description automatically generated

With the ggplot2 package, we can clean this up using the options. This can get complicated so you will typically spend time doing Google Searches to see what you want to do and what commands to use, but we will try to keep it simple. Let’s get rid of the lat. and long., fix the scale, and change its color. The code is below

ggplot(corem) +

geom\_sf(aes(fill = death2)) +

scale\_fill\_continuous(limits = c(0, 100), type = "viridis", name = "Deaths per 10,000")+

theme\_bw()+

theme(axis.text.x=element\_blank(),

axis.ticks.x=element\_blank(),

axis.text.y=element\_blank(),

axis.ticks.y=element\_blank())

Breaking this code down, the ggplot() command calls the ggplot2 package and we can put any “common” elements here such as the dataframe from which our data will come. The style is like piping, but we use “+” instead of “%>%”. The geom\_sf() command is the geom command which tells *ggplot* which type of graph we are going to create. Here, sf indicates that we are drawing a map using an sf object. We could also have things like geom\_line, geom\_bar, geom\_ploygon, and any other number of geoms.

Either within this command or the previous ggplot() command, we must define the aesthetics for the graph. This includes things like the *x* and *y* variables and anything we are going to use to define the shape or color of the elements in the graph. Here we are doing a fill based on the deaths per 10,000 variables so we use fill =. Next, we want to make some changes to the default settings for that continuous fill which is the next line of code. We reset the limits, we change the color pallet use, and then we change the legend label. Finally, we set the overall theme of the graph. Commonly we use theme\_bw() but there are many other themes built in. You can also modify the theme using the last command shown which, in this case, removes the axis text and tick marks by assigning the values as blank. Running this creates the following.

Map

Description automatically generated with medium confidence

ggplot(corem) +

geom\_sf(aes(fill = pctTrump)) +

scale\_fill\_gradient2(limits = c(0, 1), low = "blue", high = "red", mid = "white",

midpoint = .5, name = "Percentage of Votes for Trump")+

theme\_bw()+

theme(axis.text.x=element\_blank(),

axis.ticks.x=element\_blank(),

axis.text.y=element\_blank(),

axis.ticks.y=element\_blank())+

labs(title = "2020 Presidential Election")

A picture containing map

Description automatically generated