Introducing R

ML

1/9/2020

Getting Started

To begin, copy all the text on this screen. Then open RStudio, and in the "File" menu select "New File" then "R Markdown". Add a title like "SOCI 1001, Week One, Class Three", include your name in the "Author" box, select "PDF" as the Default Output Format, and click Ok. A default markdown file will open. Delete all the text in the default file below the header (everything after line 7) and paste in the text you copied to your clipboard. Make sure that "R Markdown" is selected in the file type drop down menu in the bottom right corner of this pane. Save this file to your desktop.

Welcome to R Studio!

Some navigation before we begin:

- The upper left pane is the *text editor* where you keep all of your commands.
- The lower left pane is the *console* where commands will be executed.
- The upper right pane's `Environment` window will show the data that you load.
- The lower right pane will show previews of your R Markdown documents, plots, help documents, packages, and other features.

This file is an R Notebook which is an R Markdown file that allows you to integrate text, R code, and R output into a single document. This is the recommended file type for our course, but you may want to use other file types in the future. All the possible file types are found when you click File>New File. For our purposes - and for producing transparent and reproducible research - the advantage of an R Notebook is that you can easily include text, notes, and comments that are not code (like this paragraph).

How does R know what is code and what is regular text? In an R Notebook, code goes in a fenced code block. To open the fence, type three backticks and an r in curly brackets on a single line; to close the fence, type three backticks on a single line like this:

Everything between those fences will be run as code (unless it is preceded by a hashtag).

Let's start by using R as a basic calculator. Since even the following calculation is considered code we need to place it within a code block. After the r, in the line where you open your fence, you can provide a short description of what the code in that chunk does. Those short descriptions are collected as bookmarks in the drop down menu on the bottom of this pane and can help you find pieces of your file.

```r
1000 + 1
```

```
## [1] 1001
```

To run this line of code, click the green arrow on the far right side of the first line of the code block.

For subtraction, use a -; for multiplication, use a *; for division, use a /.

What is 2002 divided by 2?

REPLACE THIS LINE WITH YOUR CODE

```r
2002 / 2
```

```
## [1] 1001
```

Our course number could be helpful to save so we do not have to calculate it each time we want to reference it. We save a value by creating an object. The syntax for doing so is to name our object first, then use the symbol "<-" to say what value our object takes. Let's call our new object `coursenumber`:

```
coursenumber <- 2002 / 2
```

Run this line of code. In the top right quadrant, we should see a new value saved. That's promising, but there's no output below the code chunk! That's because we have to separately recall the object for it to be displayed. The only line of code we need to do so is our object name:

```
coursenumber
```

```
## [1] 1001
```

Note that case matters in R code. See how by replacing the `c` in the previous chunk with a capital `C`.

We can now use the object we created in other calculations by referring to it by name:

```
coursenumber + 1019
```

```
## [1] 2020
```

Objects can take numerical or character values. The difference is that if we are using text we have to enclose the value in quotation marks:

```
coursename <- "Counting The US: The 2020 Census"
coursename
```

```
## [1] "Counting The US: The 2020 Census"
```

We can now reference these saved objects in our code as well as our text. To use *inline code*, wrap the code in single backticks with a preceding r:

> This class is called Counting The US: The 2020 Census and it is SOCI 1001.

The printed document will replace `coursename` and `coursenumber` with their saved values.

Introducing R Markdown

We will be using R Notebooks with R Markdown to write and run our code. R Markdown is a plain text-based form of writing with lots of neat tricks. For example, look at the heading for this paragraph. The two hashtags are how we start a new section in R Markdown. If we used one hashtag, the section heading would be bigger in the output. If we used three (or four) hashtags, the section headings would be smaller (or even smaller) in the output, which would be helpful if you have lots of sub- (or sub-sub-) headings

There's an R Markdown cheat sheet in the pages folder on Canvas that has much more info on how to use R Markdown. Here are a few first day features to get you started:

Wrapping a word in `single backticks` will highlight it when you print.

Wrapping text in one asterisk will italicize it when you print.

Wrapping text in two asterisks will bold it when you print.

Wrapping text in three asterisks will bold and italicize it when you print.

And three (or more) dashes on a single line will create a horizontal rule:

Getting Started With R

Today we will learn one way to import data into R and how to get simple summaries of different types of variables. We will use the Census Bureau's 2018 population estimates for every county to explore types of variables.

First, we need to install two packages. Packages are additional functions that supplement base R. We will use the `tidyverse` and `tidycensus` packages extensively this month.

Here's the code to install the package. You only have to install the package once on any computer you use. Once your computer has the package installed, you can load the package with the `library()` function.

Now that the package is installed, you can put a hashtag in front of the `install.packages()` line so it does not run again.

Next we'll get the data. Eventually we'll download data directly from the census servers using your APIs. Today we'll use data I already downloaded and cleaned up.

```
week_1_3 <- read.csv("https://raw.githubusercontent.com/mjclawrence/soci1001/master/data/week_01_03.csv
```

If the code above executed correctly, you should now see the data frame loaded and saved with the object name you created in the `Environment` panel in the top right quadrant of R Studio. There should be 3142 observations of 16 variables in the data frame named hurricanes.

Click the white arrow to the left of the object name to see all the variables (geoid, county, state, etc.). Click the spreadsheet icon on the far right side of the data name row to open the full data frame. Note that the *variables* are in the columns and the *values* are in the rows, with each row representating a separate *observation* or *case*.

Types of Variables

We have different types of variables in our data frame. Some have the `int` tag identifying them as interval variables, some have the `Factor` tag identifying them as categorical variables, and some have the `numeric` tag.

Let's look at the `state` variable to start. We can see in the `Environment` panel that this variable is a factor variable with 51 levels - or 51 different possible values. In this case, each value is a different name, making this an example of a *nominal variable*.

To see how many observations have each value - or how many counties are in each state - we will use the `table()` command. In R, we have to reference all variables by their data frame. So if we want a table of the `name` variable, we first have to tell R that the variable is in the `week_1_3` data frame. We separate the data frame and the variable name by a dollar sign:

```
table(week_1_3$state)
```

```
##
##              Alabama               Alaska              Arizona
##                   67                   29                   15
##             Arkansas           California             Colorado
##                   75                   58                   64
##          Connecticut             Delaware District of Columbia
##                    8                    3                    1
##              Florida              Georgia               Hawaii
##                   67                  159                    5
##                Idaho             Illinois              Indiana
##                   44                  102                   92
##                 Iowa               Kansas             Kentucky
##                   99                 105                 120
##            Louisiana               Maine            Maryland
##                   64                  16                  24
##        Massachusetts            Michigan           Minnesota
##                   14                  83                  87
##          Mississippi            Missouri             Montana
##                   82                 115                  56
##             Nebraska              Nevada       New Hampshire
##                   93                  17                  10
##           New Jersey          New Mexico            New York
##                   21                  33                  62
##       North Carolina        North Dakota                Ohio
##                  100                  53                  88
##             Oklahoma              Oregon        Pennsylvania
##                   77                  36                  67
##         Rhode Island      South Carolina        South Dakota
##                    5                  46                  66
##            Tennessee               Texas                Utah
##                   95                 254                  29
##              Vermont            Virginia          Washington
##                   14                 133                  39
##        West Virginia           Wisconsin             Wyoming
##                   55                  72                  23
```

When you run that line of code you should see a table with every possible value of the `state` variable and the number of observations with that value. There are 67 counties in Alabama, for example. How many counties are in Vermont? Which state has the most counties?

By default, R will sort factor variables alphabetically. An easier way to see which factor has the most observations - in this case, which state has the most counties - is to use the `sort()` function to order the names by the number of observations in each. Note that functions "wrap" code in parentheses:

```r
sort(table(week_1_3$state))
```

```
##
## District of Columbia            Delaware              Hawaii
##                    1                   3                   5
##         Rhode Island         Connecticut       New Hampshire
##                    5                   8                  10
##        Massachusetts             Vermont             Arizona
##                   14                  14                  15
##                Maine              Nevada          New Jersey
##                   16                  17                  21
##              Wyoming            Maryland              Alaska
##                   23                  24                  29
##                 Utah          New Mexico              Oregon
##                   29                  33                  36
##           Washington               Idaho      South Carolina
##                   39                  44                  46
##         North Dakota       West Virginia             Montana
##                   53                  55                  56
##           California            New York            Colorado
##                   58                  62                  64
##            Louisiana        South Dakota             Alabama
##                   64                  66                  67
##              Florida        Pennsylvania           Wisconsin
##                   67                   67                   72
##             Arkansas             Oklahoma          Mississippi
##                   75                   77                   82
##             Michigan            Minnesota                 Ohio
##                   83                   87                   88
##              Indiana             Nebraska            Tennessee
##                   92                   93                   95
##                 Iowa       North Carolina             Illinois
##                   99                  100                  102
##               Kansas             Missouri             Kentucky
##                  105                  115                  120
##             Virginia              Georgia                Texas
##                  133                  159                  254
```

To sort the table from highest to lowest value, add the `decreasing = TRUE` option:

```
sort(table(week_1_3$state), decreasing=TRUE)
```

```
## 
##                Texas              Georgia             Virginia
##                  254                  159                  133
##             Kentucky             Missouri               Kansas
##                  120                  115                  105
##             Illinois       North Carolina                 Iowa
##                  102                  100                   99
##            Tennessee             Nebraska              Indiana
##                   95                   93                   92
##                 Ohio            Minnesota             Michigan
##                   88                   87                   83
##          Mississippi             Oklahoma             Arkansas
##                   82                   77                   75
##            Wisconsin              Alabama              Florida
##                   72                   67                   67
##         Pennsylvania         South Dakota             Colorado
##                   67                   66                   64
##            Louisiana             New York           California
##                   64                   62                   58
##              Montana        West Virginia         North Dakota
##                   56                   55                   53
##       South Carolina                Idaho           Washington
##                   46                   44                   39
##               Oregon           New Mexico               Alaska
##                   36                   33                   29
##                 Utah             Maryland              Wyoming
##                   29                   24                   23
##           New Jersey               Nevada                Maine
##                   21                   17                   16
##              Arizona        Massachusetts              Vermont
##                   15                   14                   14
##        New Hampshire          Connecticut               Hawaii
##                   10                    8                    5
##         Rhode Island             Delaware District of Columbia
##                    5                    3                    1
```

Now let's look at the numeric variable `rnaturalinc`. This variable is the *rate of natural increase*. Positive values tell us there are more births than deaths per 1000 residents; negative values tell us there are more deaths than births per 1000 residents.

To see the distribution of a numeric variable, we use `summary()` instead of `table()`.

```
summary(week_1_3$rnaturalinc)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -17.1389  -1.8407   0.5182   0.9351   3.3653  27.0658
```

How would you describe this distribution?

Indexing and Filtering

How could we find the county with the highest rate of natural increase? Sometimes we only want to use a function for some observations rather than all observations. `R` calls this *indexing*. We use brackets to indicate the variable names and values we want to index.

In this next chunk, we want to pull the value for the `rnaturalinc` variable for the observation where the value for the `rnaturalinc` variable is the maximum (or `max`). Note that we will use two equal signs when referecing a specific value that exists in a data frame.

```
week_1_3$county[week_1_3$rnaturalinc==max(week_1_3$rnaturalinc)]
```

```
## [1] Madison County
## 1877 Levels: Abbeville County Acadia Parish Accomack County ... Ziebach County
```

```
week_1_3$state[week_1_3$rnaturalinc==max(week_1_3$rnaturalinc)]
```

```
## [1] Idaho
## 51 Levels: Alabama Alaska Arizona Arkansas California ... Wyoming
```

What is the rate of natural increase in this county?

REPLACE THIS LINE WITH YOUR CODE

```
week_1_3$rnaturalinc[week_1_3$county=="Madison County" & week_1_3$state=="Idaho"]
```

```
## [1] 27.06585
```

Find the county, state, and rate of natural increase for the county with the minimum value for that variable.

REPLACE THIS LINE WITH YOUR CODE

```
week_1_3$county[week_1_3$rnaturalinc==min(week_1_3$rnaturalinc)]
```

```
## [1] Covington city
## 1877 Levels: Abbeville County Acadia Parish Accomack County ... Ziebach County
```

```
week_1_3$state[week_1_3$rnaturalinc==min(week_1_3$rnaturalinc)]
```

```
## [1] Virginia
## 51 Levels: Alabama Alaska Arizona Arkansas California ... Wyoming
```

```
week_1_3$rnaturalinc[week_1_3$county=="Covington city" & week_1_3$state=="Virginia"]
```

```
## [1] -17.13886
```

We have seen how to pull values for specific observations using brackets. We can use the `filter()` function to pull observations into a new data frame based on values of a certain variable. For example, let's create a new data frame called `vermont` that only includes counties for which the value of the `state` variable is "Vermont".

We will do so using the basic structure of `tidyverse` functions. The first line assigns the name of our new object and asserts that this object will take values based on the existing `week_1_3` dataframe. The pipe (%>%) connects functions…read it as "don't execute the code until you get to the next line." The second line is the filter function.

```
vermont <- week_1_3 %>%
  filter(state == "Vermont")
```

How many observations are in this dataframe? What is the distribution of `rnaturalinc` across Vermont counties? Which county in Vermont has the highest rate of natural increase?

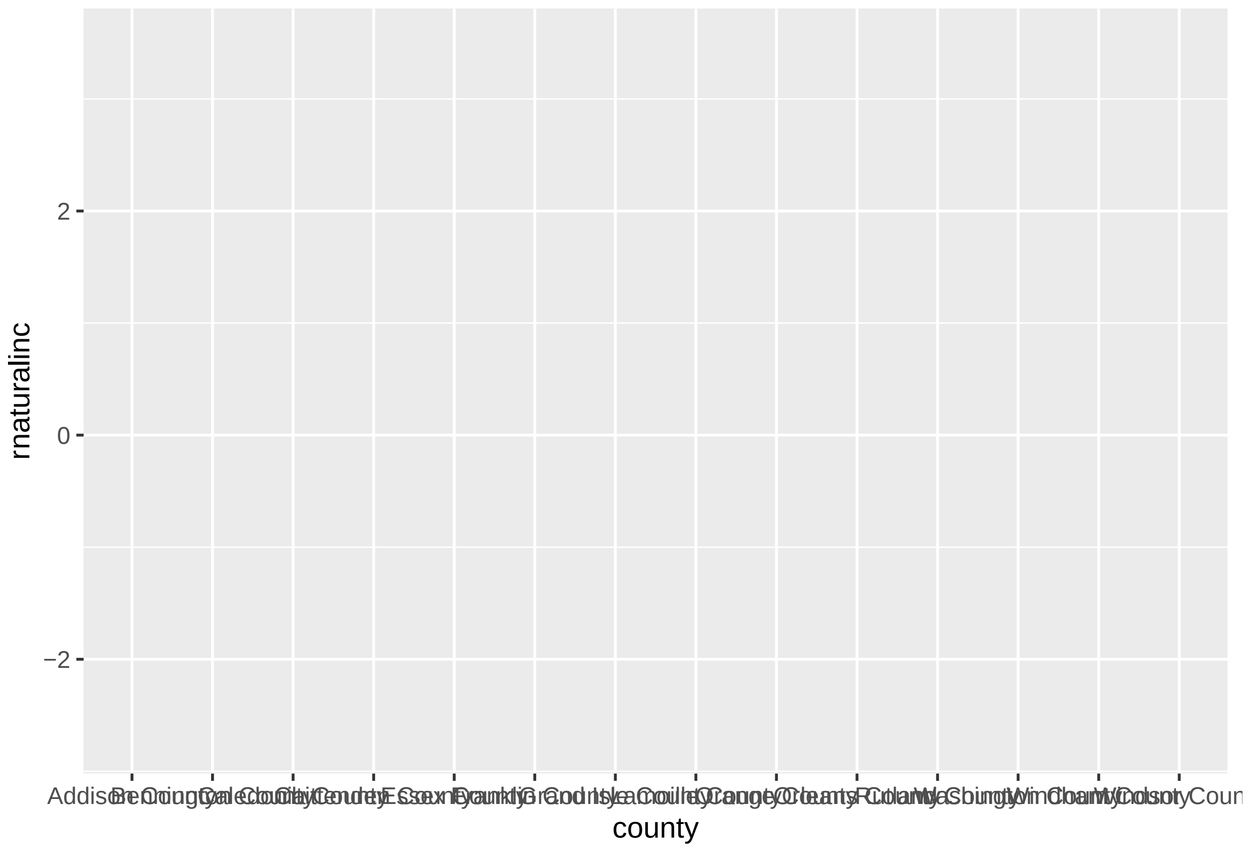
Introducing Graphics

The information we have seen so far might be better conveyed through a figure. We will use the `ggplot2` package - which loads with `tidyverse` for figures. The foundation for ggplot is to think of a figure as having multiple layers. The bottom layer is where we name the object, state the dataset we are using, and then give some aesthetic mapping information. In this example, we will use the Vermont dataframe created above, will use the values in the `county` column as our x variable, and will use the values in the `rnaturalinc` column for our y-axis.

```
vt_plot_rnaturalinc <- ggplot(vermont, aes(x = county, y = rnaturalinc))
```

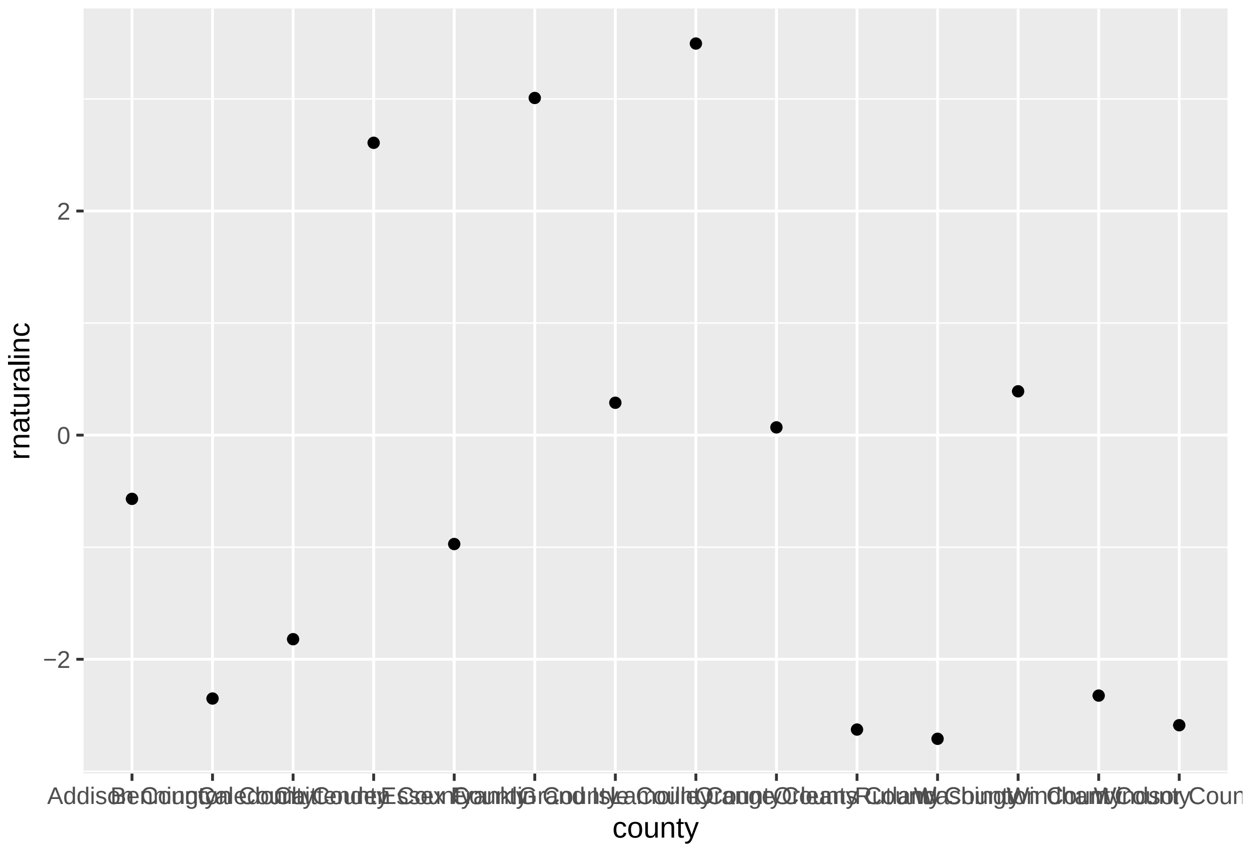
Recall the object to see what has been set up as the background layer.

```
vt_plot_rnaturalinc
```



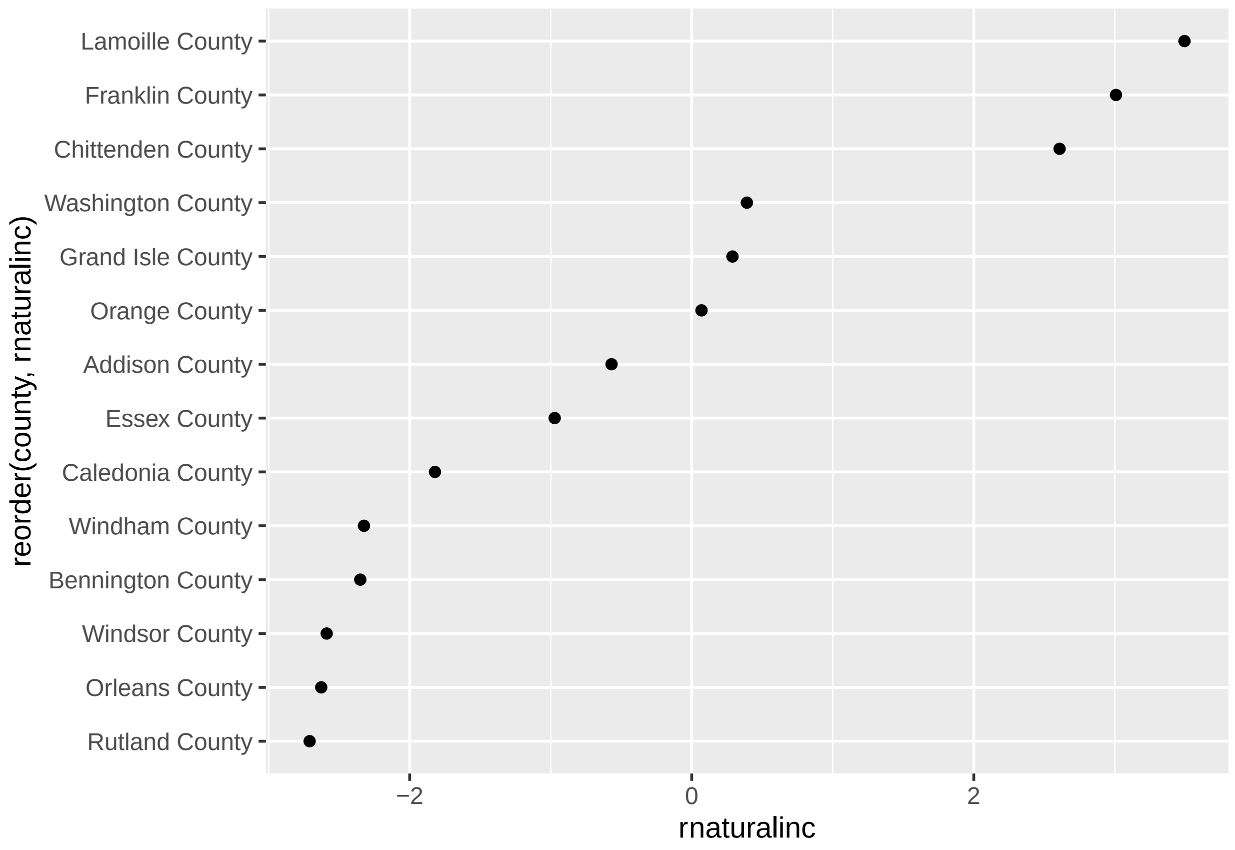
Next let's add the geometric shape as a new layer. We'll start with `geom_point()`.

```
vt_plot_rnaturalinc + geom_point()
```



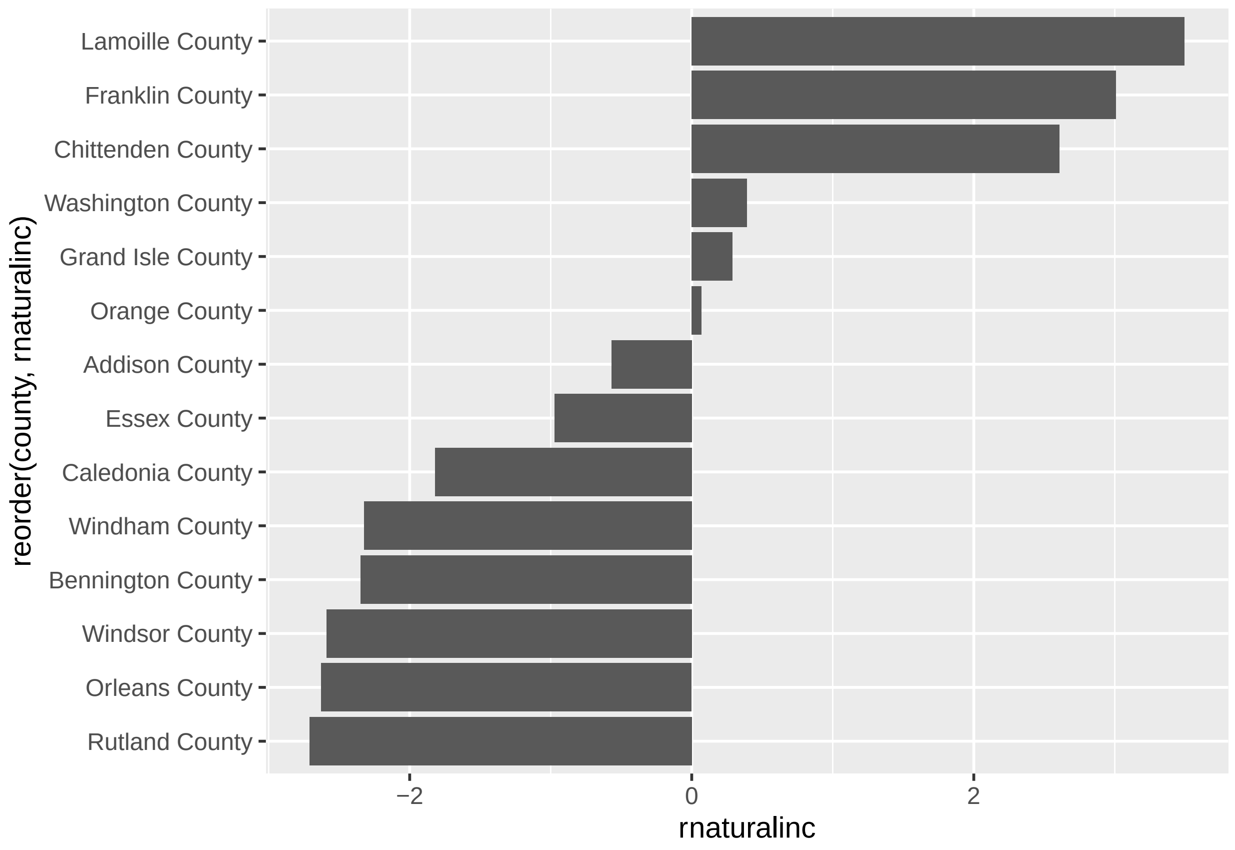
Let's fix two things. First, let's order the counties by value of `rnaturalinc` using `reorder()`. Second, let's flip the x and y axis using `coord_flip()`.

```
vt_plot_rnaturalinc <- ggplot(vermont, aes(x = reorder(county, rnaturalinc), y = rnaturalinc))

vt_plot_rnaturalinc + geom_point() + coord_flip()
```



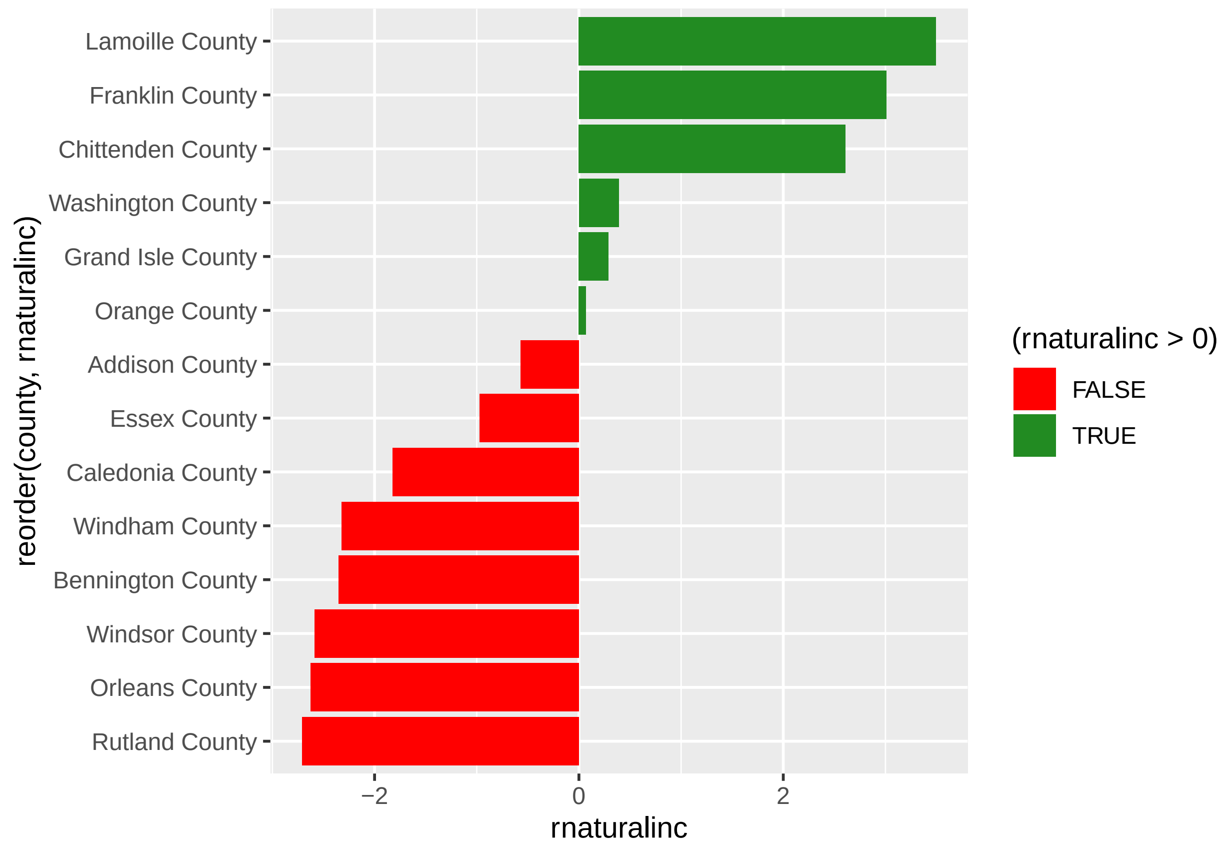
Let's look at one more geometric shape. Use `geom_col()` when you want a bar plot and have x and y variables. If you only have an x variable, use `geom_bar()`.

```
vt_plot_rnaturalinc + geom_col() + coord_flip()
```



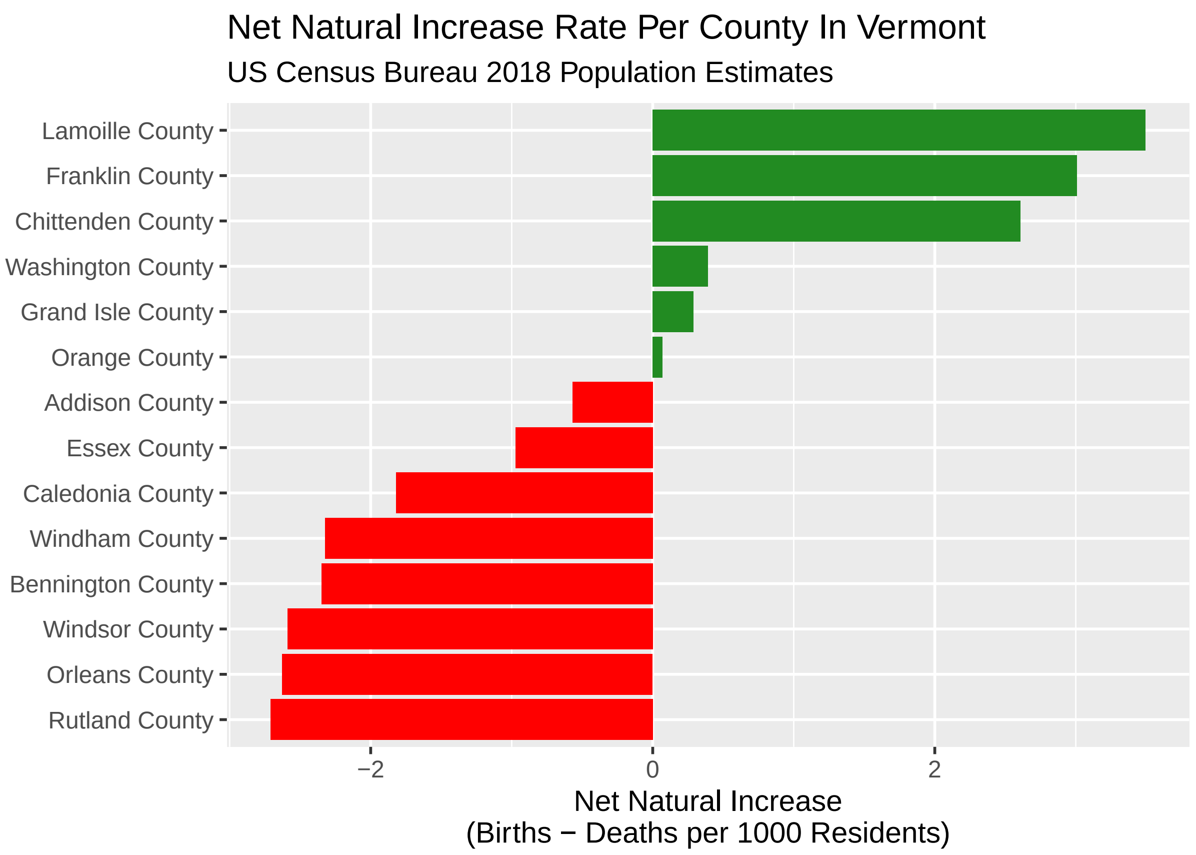
One way to improve this figure would be to `fill` the positive bars in one color and the negative bars in another color. We can add that in the aesthetic map and assert the colors we want to use with `scale_fill_manual()`.

```
vt_plot_rnaturalinc <- ggplot(vermont, aes(x = reorder(county, rnaturalinc),
                                          y = rnaturalinc,
                                          fill = (rnaturalinc>0))) +
  geom_col() + coord_flip() +
  scale_fill_manual(values = c("Red", "Forestgreen"))

vt_plot_rnaturalinc
```



We can make further adjustments to this figure like changing the axis labels, adding a title, moving (or deleting) the legend, etc.

```r
vt_plot_rnaturalinc <- ggplot(vermont, aes(x = reorder(county, rnaturalinc),
                                            y = rnaturalinc,
                                            fill = (rnaturalinc>0))) +
  geom_col() + coord_flip() +
  scale_fill_manual(values = c("Red", "Forestgreen")) +
  guides(fill = FALSE) +
  labs(y = "Net Natural Increase\n(Births - Deaths per 1000 Residents)",
       x = " ",
       title = "Net Natural Increase Rate Per County In Vermont",
       subtitle = "US Census Bureau 2018 Population Estimates")

vt_plot_rnaturalinc
```



EXTRA

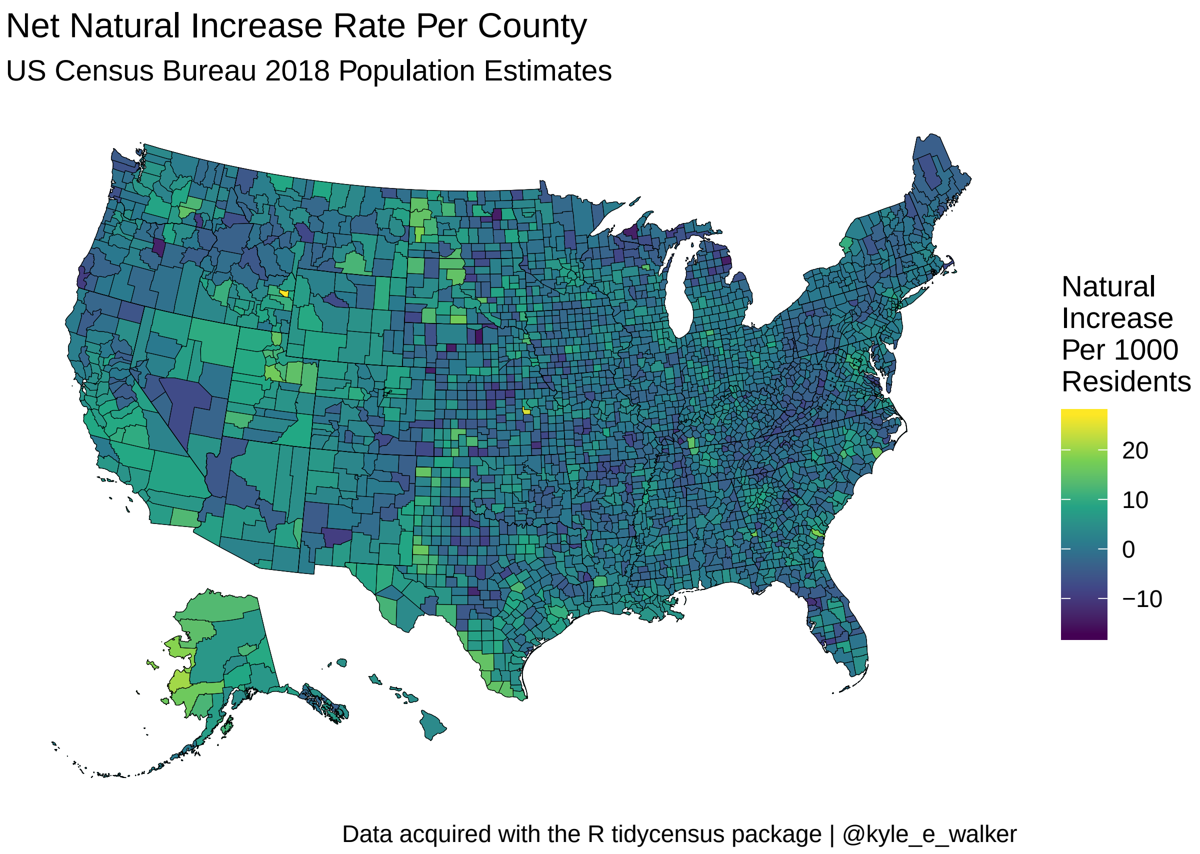
```
library(tidycensus)

pop_geo <- get_estimates(geography = "county",
                         product = "components",
                         year = 2018,
                         geometry = TRUE,
                         shift_geo = TRUE,
                         output = "wide")
```

```
## Please note: Alaska and Hawaii are being shifted and are not to scale.
```

```
vt_geo <- filter(pop_geo, grepl(NAME, " Vermont"))
```

```
## Warning in grepl(NAME, " Vermont"): argument 'pattern' has length > 1 and
## only the first element will be used
```

```
map_natural_increase <- ggplot() +
  geom_sf(data = pop_geo, aes(fill = RNATURALINC, color = RNATURALINC), lwd = 0.1) +
  geom_sf(data = tidycensus::state_laea, fill = NA, color = "black", lwd = 0.1) +
  geom_sf(data = tidycensus::county_laea, fill = NA, color = "black", lwd = 0.05) +
  coord_sf(datum = NA) + scale_fill_viridis_c() + scale_color_viridis_c() +
  guides(color = FALSE) +
  labs(title = "Net Natural Increase Rate Per County",
       subtitle = "US Census Bureau 2018 Population Estimates",
       fill = "Natural\nIncrease\nPer 1000\nResidents",
       caption = "Data acquired with the R tidycensus package | @kyle_e_walker") +
  theme_minimal()

map_natural_increase
```


Net Natural Increase Rate Per County
US Census Bureau 2018 Population Estimates
Natural Increase Per 1000 Residents
20
10
0
−10
Data acquired with the R tidycensus package | @kyle_e_walker