ACS Data & Mapping

Susan E. Chen

Contents


```
knitr::opts_chunk$set(echo = TRUE)
#I load my packages as I use them so you can see what
#they are used with.
```

Quick review of extracting ACS data

I always like to point my working directory to a folder where I want to save my work. In the code chunk below I use the `setwd()` command in `R` to do this:

```
#Set your working directory to where you want to save your xlsx files
# Remember the path
setwd("C:/Users/ecsusan/Dropbox/2021VT-DSPG-FILES/Appalachia/")
```

Get a Census API key

Yesterday you should have gotten a Census API key. To use the tidycensus package in `R` you will need a Census API key. You can get a Census API key here. You should save this API key in the file `.Renviron` in your home directory.

Here are the steps to create your `.Renviron`. Open your text editor(In Windows this would be Notepad). Copy the statement `census_api_key=KEYGIVENBYCENSUS = XXXX`. Where you see XXXX please insert the long string of numbers you got when you clicked on the link above to get an API key. Save this in hte same working directory. You are now safe to pull data from the Census Bureau. To confirm you have read in your census key you can type `Sys.getenv("census_api_key")` after you have read in the API key.

```
readRenviron("~/.Renviron") #readin Census API key
Sys.getenv("census_api_key") #confirm you have read in your Census API key
```

```
## [1] "a99d2b0b6bc5ac909ed32e5c5fa7e48f05785989"
```

Setting up the list of counties to pull

Our objective in the first step is to put together a list of counties that we want to pull and map using ACS data. If you are interested here is a list of FIPS codes for all US counties database here.

Getting ACS data (variables)

We are now going to pull ACS data. First, you need to determine what variables you want to analyze for your project. I am interested in median population income so I will look at the ACS data tables to scope out what I need. I will then identify the "names" of the variables I want to pull from the Census archive. You can teach yourself here. There is also a pdf of the webinar. I prefer the pdf because I can scan it quickly but there is also a video explaining how to identify the name of the variable you want to pull.

Another alternative, and the one I will use here, is to read in a data table from Census and ask for the variable names. Either way, you have to have some clue about the naming convention so go back to the previous paragraph and link to learn the naming convention. Here is some code that reads in the 5 year estimates for the ACS data in 2019 and gives you the variable names in a file called `vars`. Scroll through the `vars` file to find the name of the variable you want. Note in the code chunk below I use the `tidycensus package`. You should install it if you do not already have it.

```
library(tidycensus)
# Set a year of interest
this.year = 2019

# This looks at the 5 year estimates
# You can also do "acs1"
```

```
vars <- load_variables(year = 2019,
                       dataset = "acs5",
                       cache = TRUE)

# There are 22711 possible variables
dim(vars)
```

```
## [1] 27040     3
```

Using the dataset of variable names `vars`, I have picked out a list of variables that I want. I give them names and they are:

```
popululation = "B02001_001",
median.gross.rent = "B25064_001",
median.household.income = "B19013_001",
rent.burden = "B25071_001",
white = "B03002_003",
af.am = "B03002_004",
hispanic = "B03002_012",
am.ind = "B03002_005",
asian = "B03002_006",
nh.pi = "B03002_007",
multiple = "B03002_009",
other = "B03002_008"
```

So now we can pull the data for one county for one year for `B19013_001`. In the code chunk below I pull the data for Powhatan county only and this is for the 5 year ACS in 2019. Note the use of `this.year`. Recall I set `this.year=2019` above. This means later on if I want to change in the year I only have to do it *once*.

You can look at the data object `VA_2019` and see the variables in the dataset. They are

```
GEOID
name - the census name
variable - the variable name
estimate - the estiamte of the variable from census table
geometry - stuff you need to make a map
```

Once again we will use the `tidycensus` package in R. This time with the `get_acs()` function.

```
## Names for variable types
# Gives five year estimates
# To get Goochland County you will need to change Powhatan to Goochland
VA_income <- get_acs(geography = "tract", year=this.year,
                     state = "VA", county = "Powhatan", geometry = TRUE,
                     variables = c(median.household.income ="B19013_001"))
```

```
## Getting data from the 2015-2019 5-year ACS
```

```
## Downloading feature geometry from the Census website.  To cache shapefiles for use in
##   |                                                                              |
```

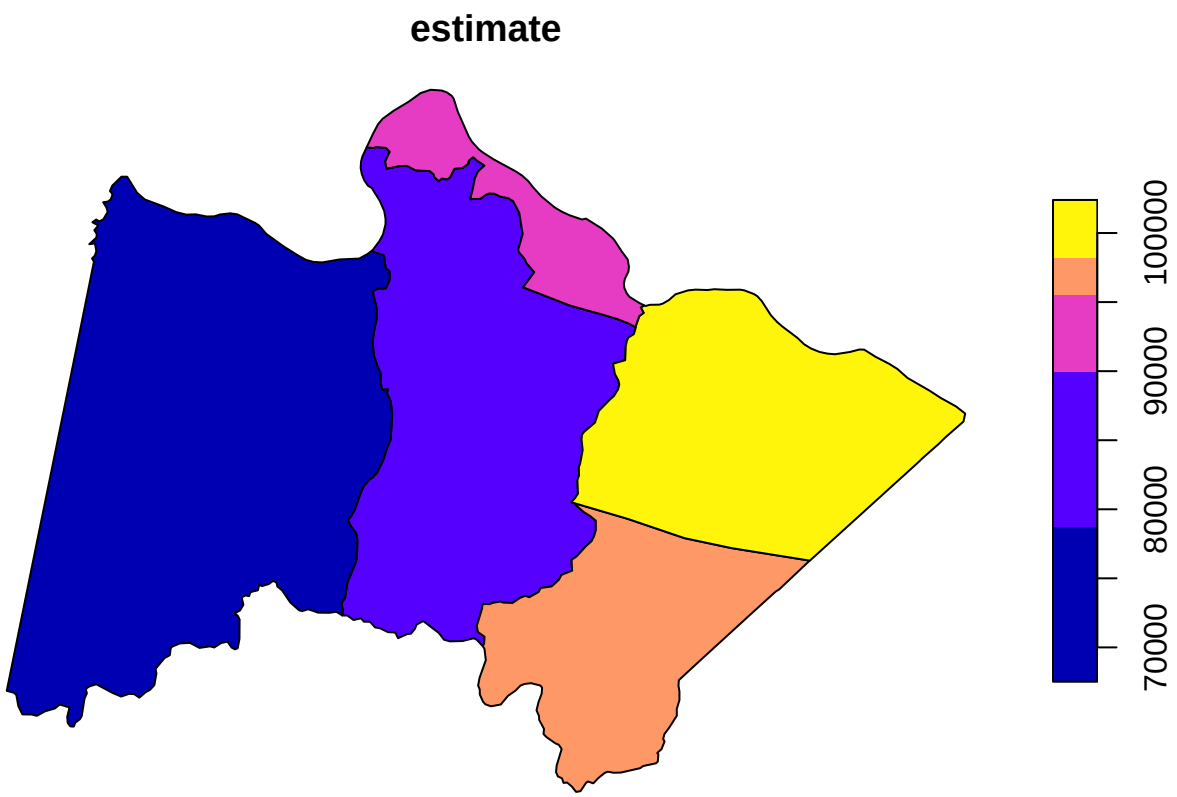
Yesterday Dr. Holmes and Dr. Stewart showed you how to pull data from the Census Bureau. In this example I am going to be pulling data from multiple counties in VA along with their geography. I will then explore how to map the variables you have pulled in a basic accessible map.

Once you have downloaded your ACS data, you can use the plot function to quickly map your variable of interest. Recall the data has one variable and the value is stored in the variable `estimate`.

```
head(VA_income)
```

```
## Simple feature collection with 5 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -78.13205 ymin: 37.41483 xmax: -77.6554 ymax: 37.69166
## Geodetic CRS:  NAD83
##         GEOID                                          NAME
## 1 51145500200    Census Tract 5002, Powhatan County, Virginia
## 2 51145500102 Census Tract 5001.02, Powhatan County, Virginia
## 3 51145500101 Census Tract 5001.01, Powhatan County, Virginia
## 4 51145500300    Census Tract 5003, Powhatan County, Virginia
## 5 51145500400    Census Tract 5004, Powhatan County, Virginia
##                   variable estimate   moe                       geometry
## 1 median.household.income    86469  9294 MULTIPOLYGON (((-77.96564 3...
## 2 median.household.income    97596  9668 MULTIPOLYGON (((-77.89829 3...
## 3 median.household.income    98906  7471 MULTIPOLYGON (((-77.8513 37...
## 4 median.household.income    93438 38203 MULTIPOLYGON (((-77.94981 3...
## 5 median.household.income    71004  9647 MULTIPOLYGON (((-78.13205 3...
```

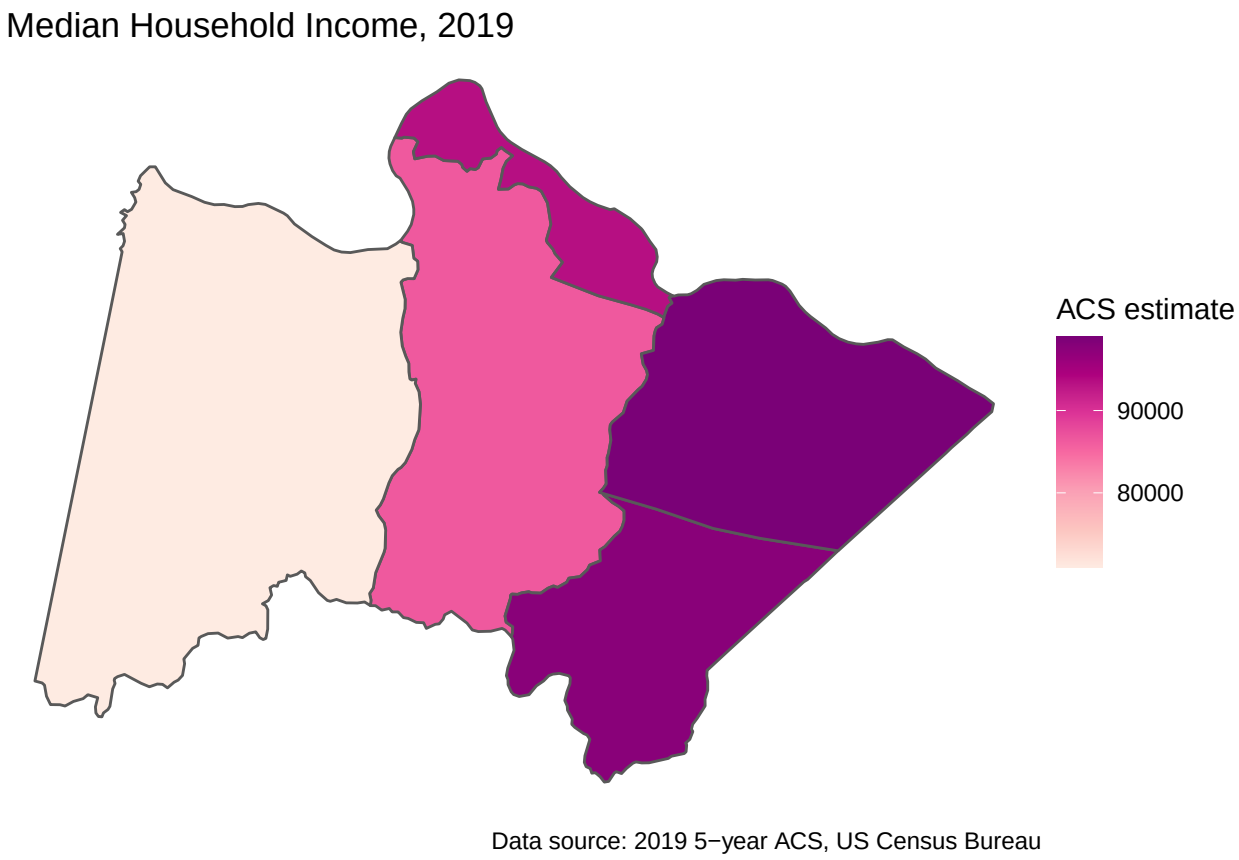
```
plot(VA_income["estimate"]) #quick plot
```



Using ggplot to map

The `ggplot2` package is also useful to drawing maps. In the code chunk below I use the `VA_income` dataset to show you how to draw a simple map with the `ggplot()` function in this package.

```
library(ggplot2)
ggplot(data = VA_income, aes(fill = estimate)) + geom_sf() + scale_fill_distiller(palett
       caption = "Data source: 2019 5-year ACS, US Census Bureau",
       fill = "ACS estimate") +
  theme_void()  #Better plot
```



Multiple ACS variables and maps

If you have a list of variables you need then you can modify the above code chunk to allow a list of variables that you place in a vector. Below you see my variable names are in the row vector:

```
c(population = "B02001_001",
median.gross.rent = "B25064_001",
median.household.income = "B19013_001",
rent.burden = "B25071_001",
white = "B03002_003",
af.am = "B03002_004",
hispanic = "B03002_012",
am.ind = "B03002_005",
asian = "B03002_006",
nh.pi = "B03002_007",
multiple = "B03002_009",
other = "B03002_008")
```

I feed this vector to `variables = c(...)` see below chunk.

```
## Names for variable types
# Gives five year estimates
# To get Goochland County you will need to change Powhatan to Goochland
ACS5.2019 <- get_acs(geography = "tract", year=this.year,
                  state = "VA", county = c("Powhatan","Goochland"), geometry = TRUE,
                  variables = c(population = "B02001_001",
                                median.gross.rent = "B25064_001",
                                median.household.income = "B19013_001",
                                rent.burden = "B25071_001",
                                white = "B03002_003",
                                af.am = "B03002_004",
                                hispanic = "B03002_012",
                                am.ind = "B03002_005",
                                asian = "B03002_006",
                                nh.pi = "B03002_007",
                                multiple = "B03002_009",
                                other = "B03002_008"))
```

```
## Getting data from the 2015-2019 5-year ACS
```

```
## Downloading feature geometry from the Census website.  To cache shapefiles for use in
```

```
head(ACS5.2019)
```

```
## Simple feature collection with 6 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -77.96564 ymin: 37.47227 xmax: -77.81921 ymax: 37.66908
## Geodetic CRS:  NAD83
##         GEOID                                                NAME   variable estimate
## 1 51145500200 Census Tract 5002, Powhatan County, Virginia population     8933
## 2 51145500200 Census Tract 5002, Powhatan County, Virginia      white     8288
## 3 51145500200 Census Tract 5002, Powhatan County, Virginia      af.am      317
## 4 51145500200 Census Tract 5002, Powhatan County, Virginia     am.ind       12
## 5 51145500200 Census Tract 5002, Powhatan County, Virginia      asian       29
## 6 51145500200 Census Tract 5002, Powhatan County, Virginia      nh.pi        0
##   moe                       geometry
## 1 405 MULTIPOLYGON (((-77.96564 3...
## 2 424 MULTIPOLYGON (((-77.96564 3...
## 3 157 MULTIPOLYGON (((-77.96564 3...
## 4  20 MULTIPOLYGON (((-77.96564 3...
## 5  29 MULTIPOLYGON (((-77.96564 3...
## 6  17 MULTIPOLYGON (((-77.96564 3...
```

Now that I have read in all the variables I will need to use the `dplyr` package to filter only the ones that I want to map.

Put ACS data on a map

Here is a simple way to draw a map. Better ways exist but this is a rough guide. Below I only want to map one variable: county level Median Household Income. I filter the data and create a new data object `myacs_map` which contains only the one variable I wish to draw the maps with. The `filter()` function is part of the `dplyr` package

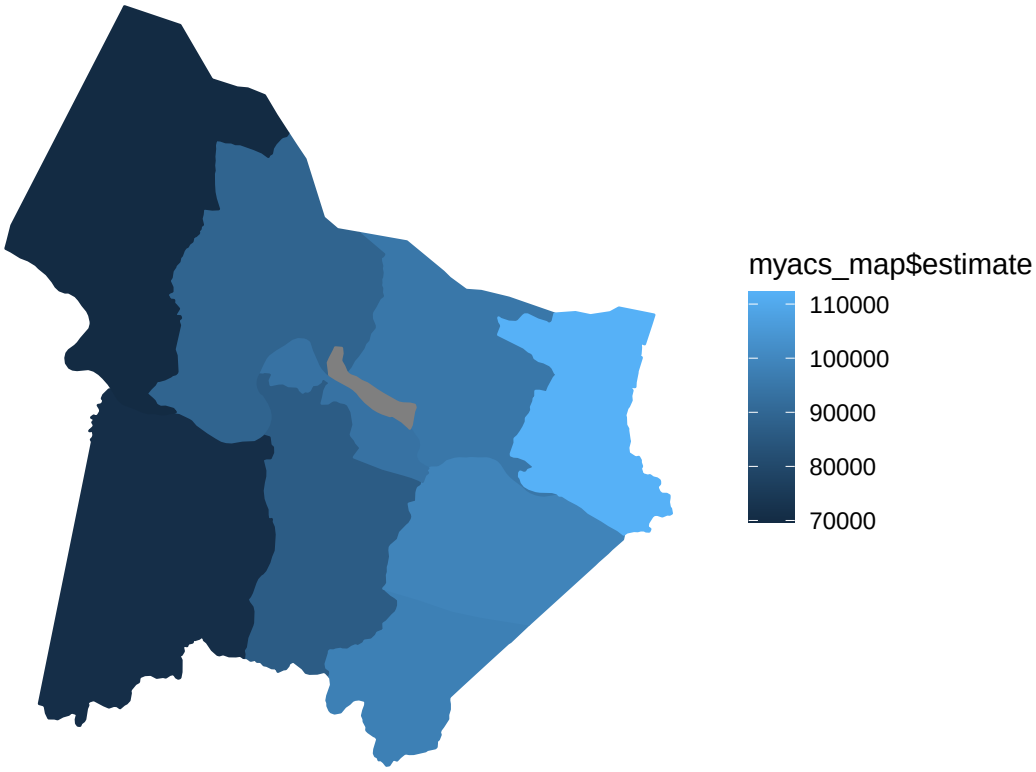
```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
myacs_map <- filter(ACS5.2019, variable == "median.household.income")

zap <- ggplot(myacs_map$geometry) +
    geom_sf(aes(fill=myacs_map$estimate, color = myacs_map$estimate)) +
    coord_sf(datum = NA) +
    theme_minimal()
zap
```



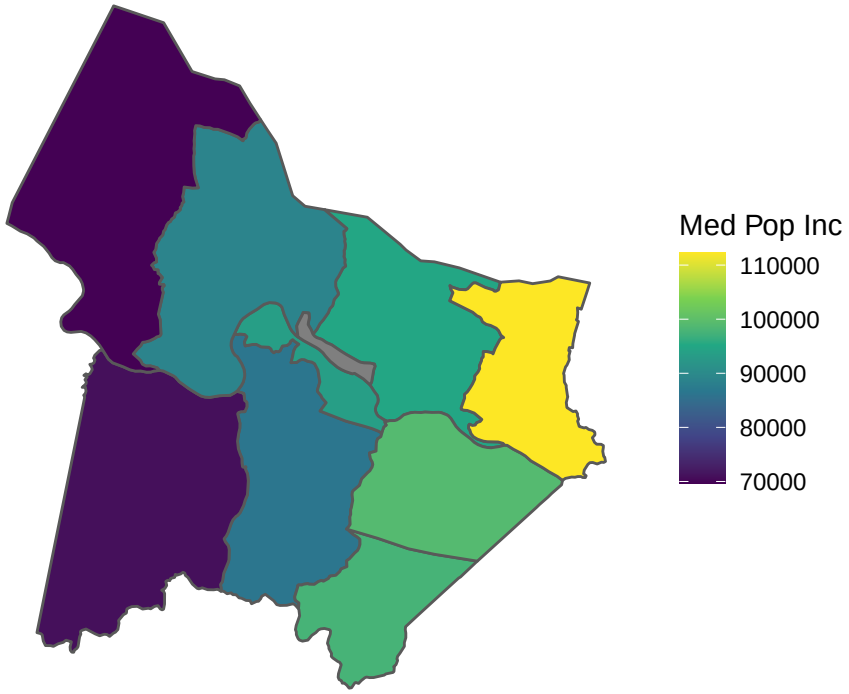
Colorblind Friendly packages

The `viridis` package is a color blind friendly palatte. The options for this package are" "viridis", "magma", "inferno", or "plasma".

```
# Create a colorblind friendly palette.
library(viridis)
```

```
## Loading required package: viridisLite
```

```
zap2 <- ggplot(myacs_map$geometry) + geom_sf(aes(fill=myacs_map$estimate)) +
    scale_color_viridis_c() + scale_fill_viridis_c()+
    coord_sf(datum = NA) +
    theme_minimal() +
  labs(fill = "Med Pop Inc",
       title = "Median Pop Income by Census Tracts: Powhatan & Goochland, VA",
       caption = "Note: I used get_acs() to get this data")
zap2
```

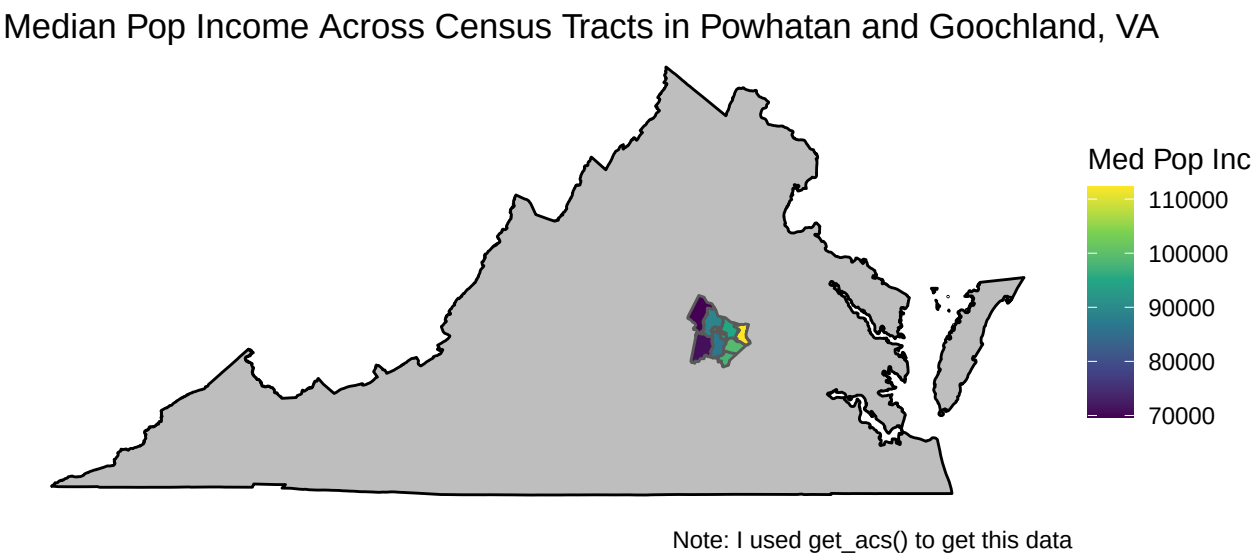


Note: I used get_acs() to get this data

State Borders

To put an overlay of the STATE on top of your counties you need to save the STATE geometry for making maps. You will use this in your maps below. This will change depending on your subsets of states. You can get state borders using the `states()` function in the `tigris` package. ## One Map

```
## To enable
## caching of data, set `options(tigris_use_cache = TRUE)` in your R script or .Rprofile

##
## Attaching package: 'tigris'

## The following object is masked from 'package:tidycensus':
##
##     fips_codes

##   |                                                                              |
```



These maps are UGLY! You can do better !

Saving and Exporting the R dataframe

At this point I would like to export my data to my hard drive. This is helpful if you plan to use it in RShiny apps. Here is how you save the data as an R dataframe.

```
save(ACS5.2019,file="ACS5.2019.Rda")
```

In another R script you can load the data that you saved above

```
load("ACS5.2019.Rda")
```

Leaflet Package for Making Interactive Maps in R

Now we are going to use the `leaflet` package to produce maps. This only works well and is interactive in an html file. In the code chunk below I first declare a color palette that will be defined using the colorNumeric() function. This function itself creates a function we are calling mypal(), which translates data values to color values for a given color palette. Our chosen color palette in this example is the viridis magma palette.

```r
# Creating a map with leaflet
library(leaflet)
```

```
## Warning: package 'leaflet' was built under R version 4.1.3
```

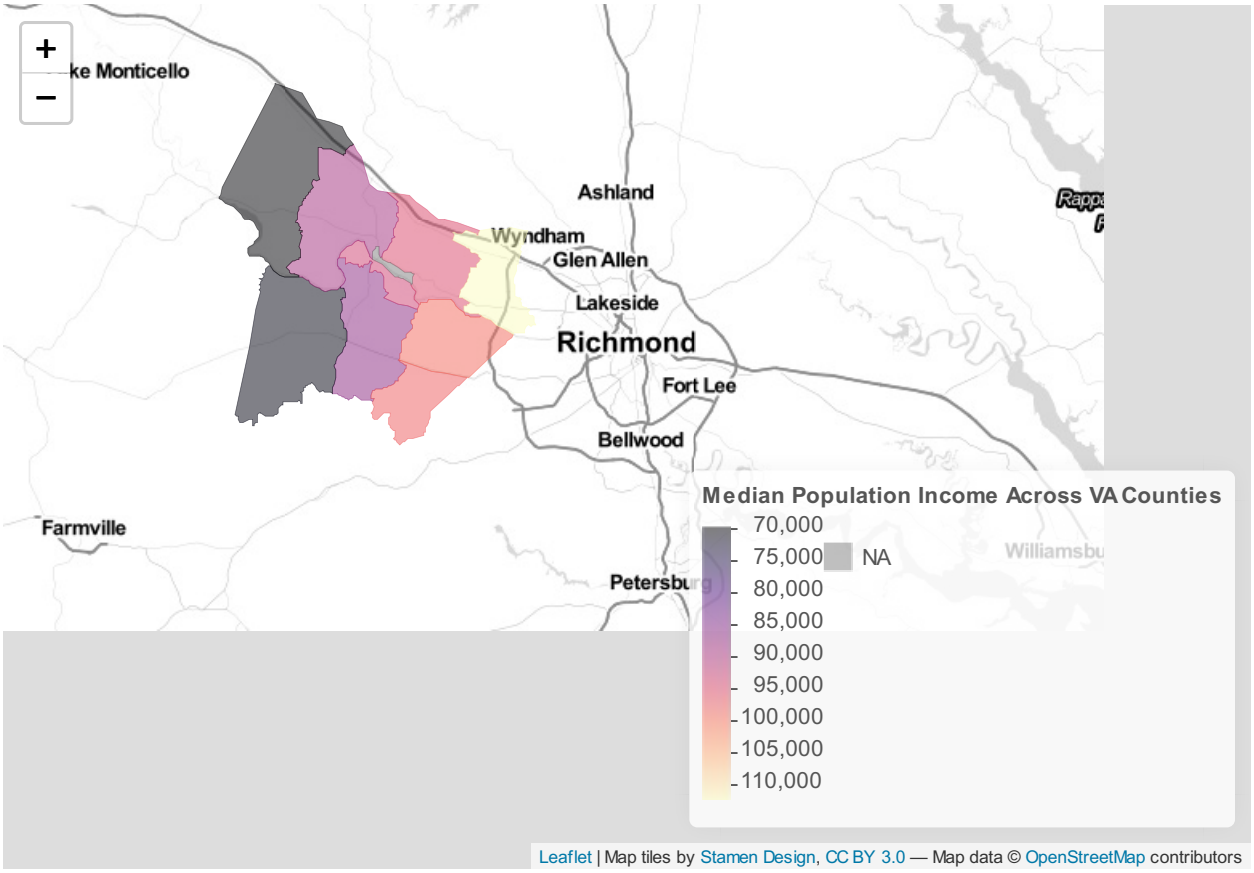
```r
mypal <- colorNumeric(
  palette = "magma",
  domain = myacs_map$estimate
)

mypal(c(10, 20, 30, 40, 50))
```

```
## Warning in mypal(c(10, 20, 30, 40, 50)): Some values were outside the color
## scale and will be treated as NA
```

```
## [1] "#808080" "#808080" "#808080" "#808080" "#808080"
```

```r
leaflet() %>%
  addProviderTiles(providers$Stamen.TonerLite) %>%
  addPolygons(data = myacs_map,
              color = ~mypal(estimate),
              weight = 0.5,
              smoothFactor = 0.2,
              fillOpacity = 0.5,
              label = ~estimate) %>%
  addLegend(
    position = "bottomright",
    pal = mypal,
    values = myacs_map$estimate,
    title = "Median Population Income Across VA Counties"
  )
```

```
## Warning: sf layer has inconsistent datum (+proj=longlat +datum=NAD83 +no_defs).
## Need '+proj=longlat +datum=WGS84'
```



I used this website as a reference document for my maps:

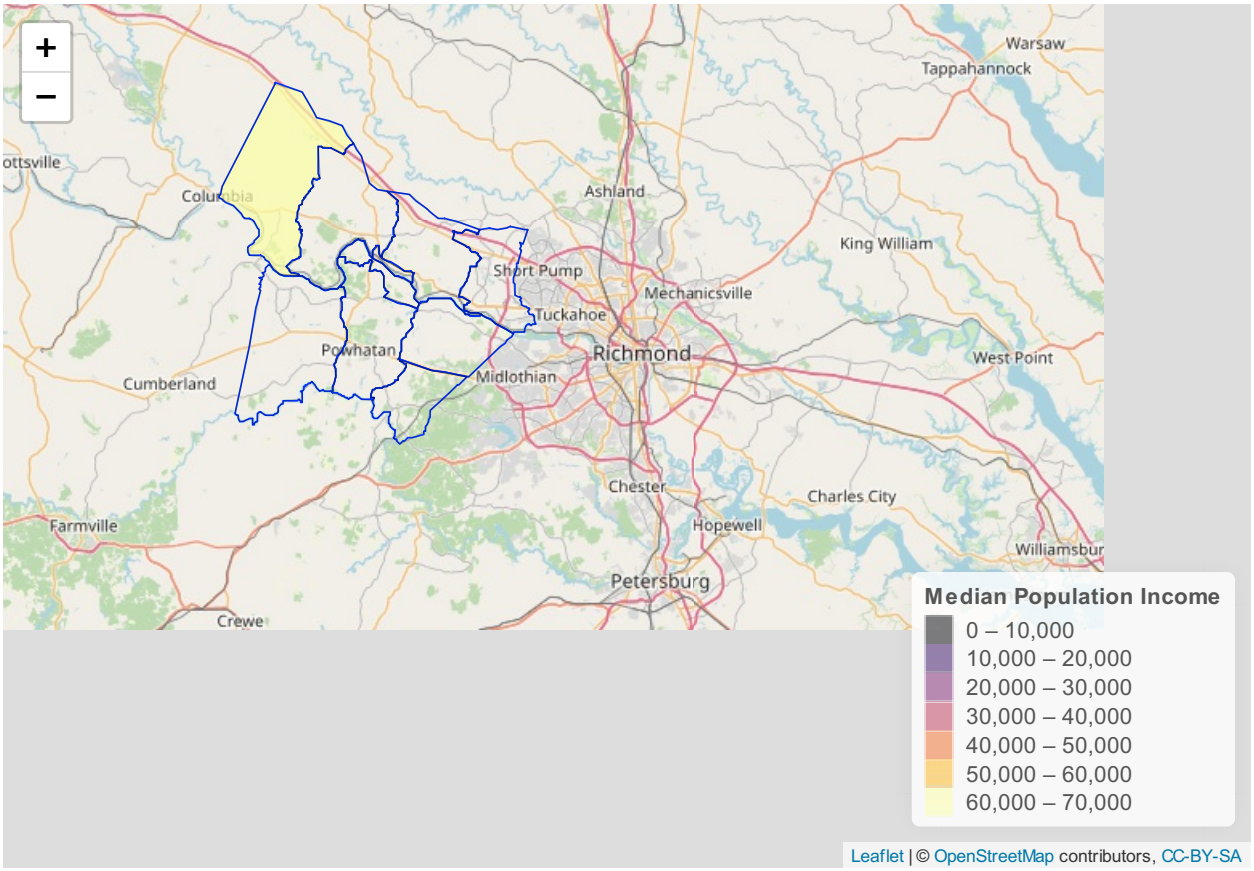
```
#specify the bin breaks
mybins <- c(0,10000,20000,30000,40000,50000,60000,70000)
#specify the default color
mypalette <- colorBin(palette="inferno", domain=myacs_map$estimate, na.color="transparen

leaflet(data = myacs_map) %>%
  addTiles() %>%
  addPolygons(
    fillColor = ~mypalette(myacs_map$`estimate`),
    stroke=TRUE,
    weight = 1,
    smoothFactor = 0.2,
    opacity = 1.0,
    fillOpacity = 0.7,
    label=paste("County: ",myacs_map$GEOID, ", Value: ",myacs_map$estimate),
    highlightOptions = highlightOptions(color = "white",
                                        weight = 2,
                                        bringToFront = TRUE)) %>%
  addLegend(pal=mypalette, position = "bottomright",
            values = ~myacs_map$estimate,
```

```
            opacity = 0.5, title = "Median Population Income") %>%
  addPolylines(data = myacs_map, color = "black",
               opacity = 0.2, weight = 1)
```

```
## Warning: sf layer has inconsistent datum (+proj=longlat +datum=NAD83 +no_defs).
## Need '+proj=longlat +datum=WGS84'
```

```
## Warning: sf layer has inconsistent datum (+proj=longlat +datum=NAD83 +no_defs).
## Need '+proj=longlat +datum=WGS84'
```

```
## Warning in mypalette(myacs_map$estimate): Some values were outside the color
## scale and will be treated as NA
```



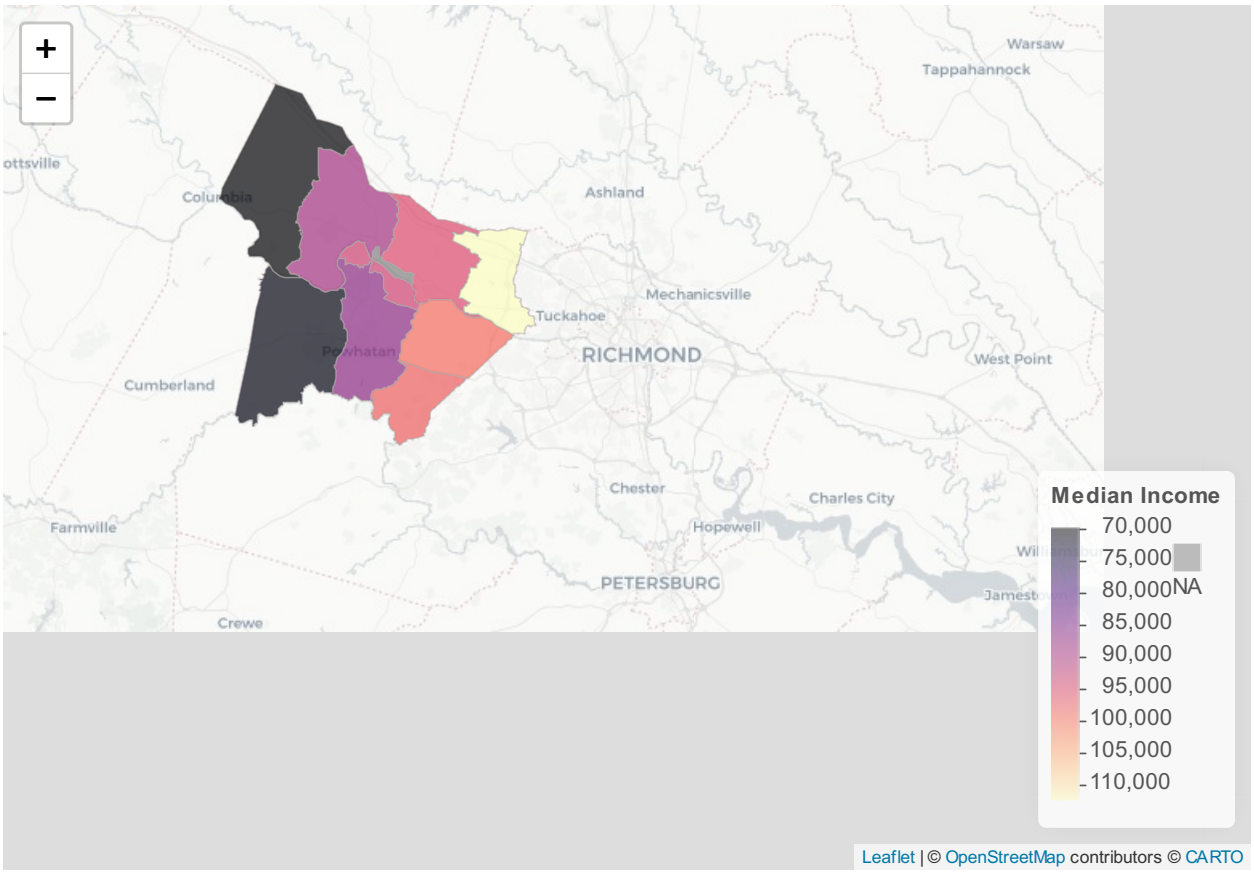
2. Or a different way to draw the map with a different baselayer.

```
pal <- colorNumeric(
  palette = "inferno",
  domain = myacs_map$estimate
)
map4<-leaflet() %>%
  addProviderTiles("CartoDB.Positron") %>%
  addPolygons(data = myacs_map,
              fillColor = ~mypal(estimate),
```

```
              color = "#b2aeae", # you need to use hex colors
              fillOpacity = 0.7,
              weight = 1,
              smoothFactor = 0.2) %>%
  addLegend(pal = mypal,
            values = myacs_map$estimate,
            position = "bottomright",
            title = "Median Income")
```

```
## Warning: sf layer has inconsistent datum (+proj=longlat +datum=NAD83 +no_defs).
## Need '+proj=longlat +datum=WGS84'
```

```
map4
```



What we learned today:

1. We reviewed how to get ACS data again
2. We learned how to map ACS data in R using plot
3. We learned how to map ACS data in R using ggplot
4. We learned how to map ACS data in R using leaflet