Week 4: HW - MapData (Visualizing Median Income on a Map)

[Name]

[Date]

# Instructions

Use the dataset named “MedianZIP.xlsx” that has median income by zip code (an excel file).  
Read the MedianZIP.xlsx, use the proper function, and save it as mydata. Save the .xlsx file in this project’s data folder.

# Step 1 - Load the Data

## Step 1.1 - Read the data

Install the readxl and tidyverse package and load them. Read the MedianZIP.xlsx, use the proper function, and save it as mydata. If you need other packages for this lab, you may need to install them too. Try what works. There are so many ways to do this.

library(tidyverse)

## Warning: package 'tidyverse' was built under R version 4.2.2

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──  
## ✔ ggplot2 3.4.0 ✔ purrr 0.3.4   
## ✔ tibble 3.1.8 ✔ dplyr 1.0.10  
## ✔ tidyr 1.2.1 ✔ stringr 1.4.1   
## ✔ readr 2.1.3 ✔ forcats 0.5.2

## Warning: package 'ggplot2' was built under R version 4.2.2

## Warning: package 'tidyr' was built under R version 4.2.2

## Warning: package 'forcats' was built under R version 4.2.2

## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()

library(readxl)

## Warning: package 'readxl' was built under R version 4.2.2

mydata <- read\_excel("MedianZIP.xlsx")

## Warning: Expecting numeric in C5070 / R5070C3: got '.'

## Warning: Expecting numeric in C25733 / R25733C3: got '.'

## Warning: Expecting numeric in C25734 / R25734C3: got '.'

## Warning: Expecting numeric in C25735 / R25735C3: got '.'

## Warning: Expecting numeric in C25811 / R25811C3: got '.'

## Warning: Expecting numeric in C29646 / R29646C3: got '.'

## Warning: Expecting numeric in C29981 / R29981C3: got '.'

## Step 1.2 - Clean the data

Clean up the dataframe if needed and make sure the column names are zip, median, mean, population. Make sure the values in each column are “numeric”. If they are factors or characters, you must change them to quantitative data.

# Write your code below.  
str(mydata)

## tibble [32,634 × 4] (S3: tbl\_df/tbl/data.frame)  
## $ Zip : num [1:32634] 10001 10002 10003 10004 10005 ...  
## $ Median: num [1:32634] 71245 30844 89999 110184 115133 ...  
## $ Mean : num [1:32634] 123113 46259 139331 156683 163763 ...  
## $ Pop : num [1:32634] 17678 70878 53609 1271 1517 ...

## Step 1.3 - Load library and data

1. Load the zipcode package (install the zipcode package first, which is an archived package). The zipcode package can be installed by doing the following. Alternatively, you can use more recent ‘zipcodeR’ package instead.

install.packages("remotes")  
library(remotes)  
install\_version("zipcode", "1.0")  
library(zipcode)

1. Use data(zipcode) to load a dataframe that contains city, state, latitude, and longitude for US zip codes.
2. Double-check your environment to find the zipcode dataframe with five variables and 44336 rows.

# Write your code below.  
library(zipcode)  
data(zipcode)

## Step 1.4 - Merge the data

Merge the zipcode information from the two dataframes (merge into one dataframe).

1. First, clean up and standardize the zipcodes in mydata using the clean.zipcodes() function, and save the values to the zip column of mydata.
2. Merge mydata and zipcode by the common column zip and store the new dataframe as dfNew.
3. use the merge() function for this.

# Write your code below.  
mydata$Zip <- clean.zipcodes(mydata$Zip)  
dfNew <- merge(mydata, zipcode, by.x = "Zip", by.y = "zip")

## Step 1.5 - Clean the data again

Remove Hawaii and Alaska (just focus on the “lower 48” states). **HINT**: You can use the which() function we learned from Intro to Data Science or you can use dplyr to filter the proper rows (use of course the filter() function in the dplyr package).

* After removing the two states, you should have 32321 rows in your new dataframe. (mydata has 32634 rows in it.)

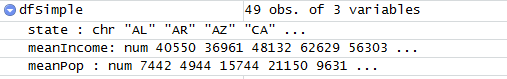
# Write your code below.  
dfNew <- dfNew[dfNew$state != "HI" & dfNew$state != "AK", ]

# Step 2 - Show the income and population per state

## Step 2.1 - Create a simpler dataframe

Create a simpler dataframe (call it dfSimple), with just the average median income and the population for each state.

* There are many ways to do this. But the simplest way is by using dplyr. Use group\_by() and summarize() from “dplyr” to do this.
* The new dataframe should look like this:



Step 2.1 Environment

# Write your code below.  
dfSimple <- dfNew  
dfSimple <- dfSimple %>%  
 group\_by(state) %>%  
 summarize(meanIncome = mean(Median), meanPop = mean(Pop))

## Step 2.2 - Update columns

Add the state abbreviations and the state names as new columns (make sure the state names are all lower case).

1. Get the state name (not just the abbreviations). Use the built-in state.name and state.abb datasets. This is the code: dfSimple$stateName <- state.name[match(dfSimple$state, state.abb)]
2. Convert stateName to lowercase and save the values in the stateName column.

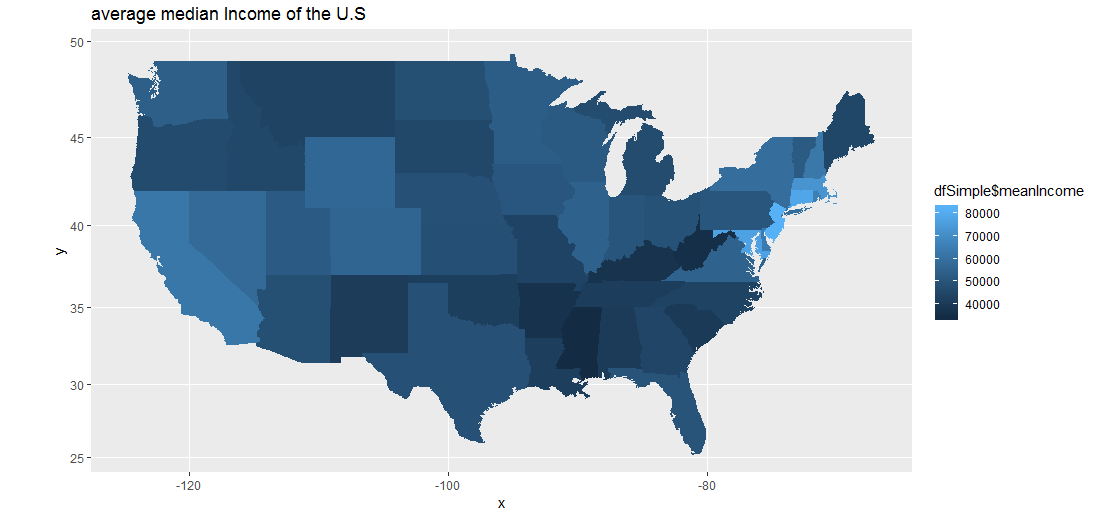
# Write your code below.  
dfSimple$stateName <- tolower(state.name[match(dfSimple$state, state.abb)])

## Step 2.3 - Visualize the US (pt1)

Show the U.S. map, using color to represent the average median income of each state.

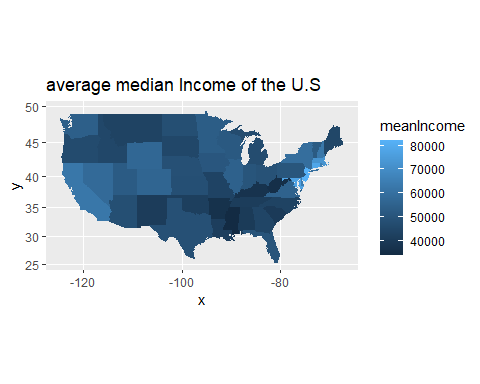
1. Get the data on the state to be mapped. Use map\_data() function to read "state" object and save the result as us.
2. Use dfSimple to create a map and set stateName as map\_id. (follow the course content practice, written in the textbook and in the video).

It should look like this (please do not forget to add the title of the map):



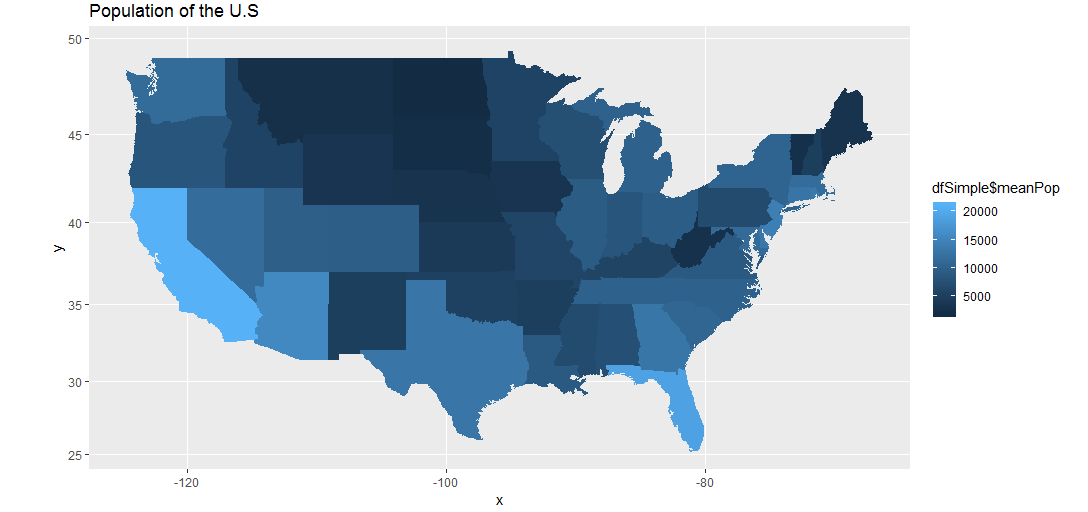
Step 2.3 Map

# Write your code below.  
us <- map\_data("state")  
map1 <- ggplot(dfSimple, aes(map\_id = stateName))  
map1 <- map1 + geom\_map(map = us, aes(fill=meanIncome))  
map1 <- map1 + expand\_limits(x = us$long, y = us$lat)  
map1 <- map1 + coord\_map() + ggtitle("average median Income of the U.S")  
map1



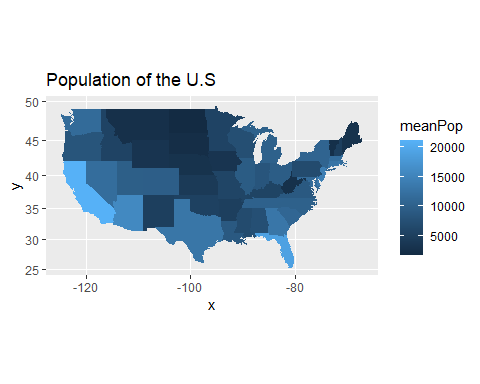
## Step 2.4 - Visualize the US (pt2)

Create a second map with color representing the population of the state. It should look like this:



Step 2.4 Map

# Write your code below.  
map2 <- ggplot(dfSimple, aes(map\_id = stateName))  
map2 <- map2 + geom\_map(map = us, aes(fill=meanPop))  
map2 <- map2 + expand\_limits(x = us$long, y = us$lat)  
map2 <- map2 + coord\_map() + ggtitle("Population of the U.S")  
map2

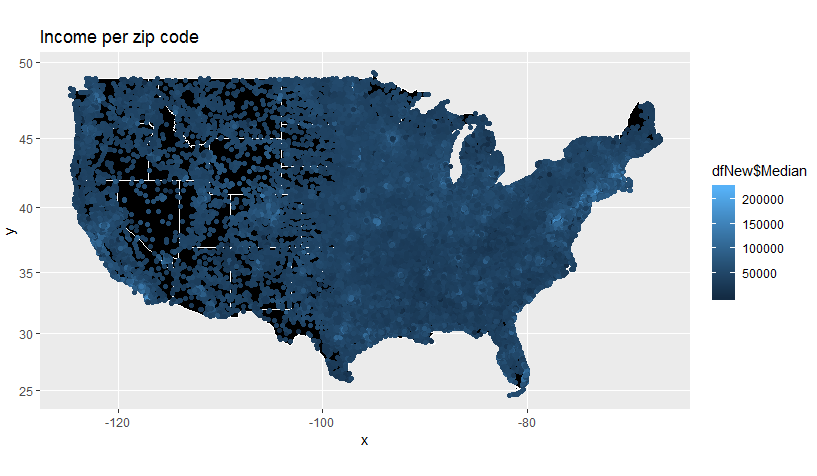


# Step 3 - Show the income per zip code

Draw each zipcode on the map, where the color of the “dot” is based on the median income. To make the map look appealing, set the background of the map to black.

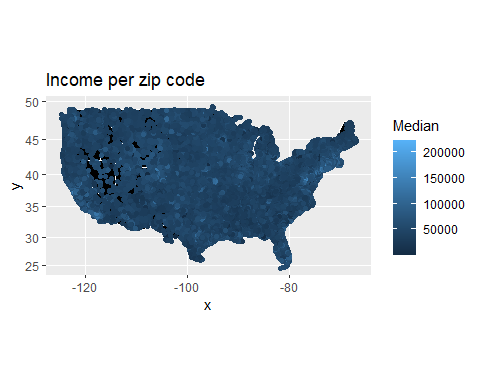
* Use dfNew to create a map and set stateName as the map ID
* Set the background color to black and line color to white
* Change the limits of the x and y axes to print the whole map (expand\_limits)
* Plot points on map, where each “dot” represents a zipcode and the color of the “dots” is based on median income (geom\_point, use color attribute using Median value).

The graph should look like this:



Step 3 Map

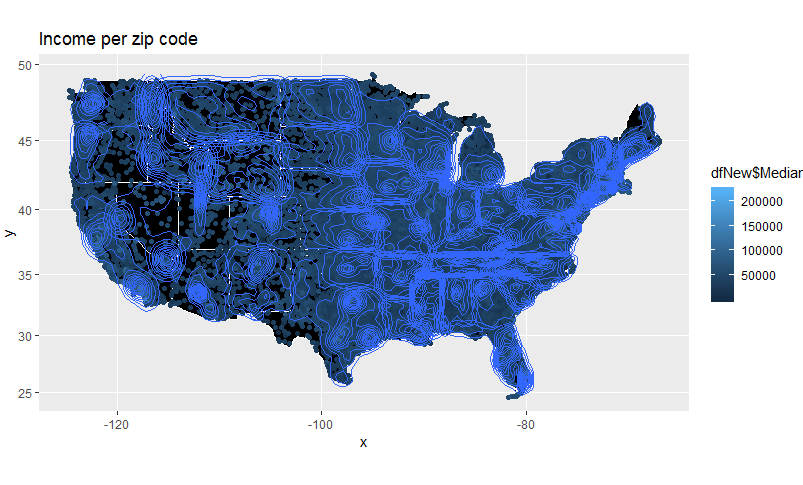
# Write your code below.  
dfNew$stateName <- tolower(state.name[match(dfNew$state, state.abb)])  
map3 <- ggplot(dfNew, aes(map\_id = stateName))  
map3 <- map3 + geom\_map(map = us,fill="black",color="white")  
map3 <- map3 + expand\_limits(x = us$long, y = us$lat)  
map3 <- map3 + geom\_point(aes(x = longitude, y = latitude, color = Median), size=1.25)  
map3 <- map3 + coord\_map() + ggtitle("Income per zip code")  
map3



# Step 4 - Show zip code density

Now generate a different map, one where we can easily see where there are lots of zipcodes and where there are few (using the stat\_density2d() function). We will name this as mapD.

It should look like this:



Step 4 Map

# Write your code below.  
map4 <- ggplot(dfNew, aes(map\_id=stateName))  
map4 <- map4 + geom\_map(map=us, fill="black", color="white")  
map4 <- map4 + expand\_limits(x=us$long,y=us$lat)  
map4 <- map4 + coord\_map() + ggtitle("Income per zip code")  
map4 <- map4 + stat\_density\_2d(data=dfNew,aes(x=longitude, y=latitude, fill=Median))

## Warning in stat\_density\_2d(data = dfNew, aes(x = longitude, y = latitude, :  
## Ignoring unknown aesthetics: fill

map4

## Warning: The following aesthetics were dropped during statistical transformation: fill  
## ℹ This can happen when ggplot fails to infer the correct grouping structure in  
## the data.  
## ℹ Did you forget to specify a `group` aesthetic or to convert a numerical  
## variable into a factor?

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

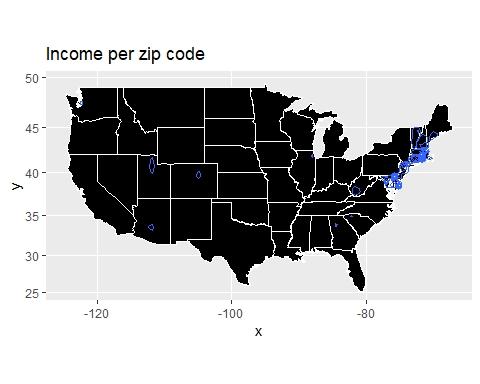
## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning: `stat\_contour()`: Zero contours were generated

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf



# Step 5 - Zoom in to the region around Tampa

Repeat steps 3 and 4, but have the image/map of the Tampa Bay area.

Below I am giving out the code for this:

# Before using geocode function, you must create Google API key. Follow directions in this url:  
https://setcompass.com/How-to-Get-Google-Maps-API-Key-Guide.htm  
# Google map requires your credit card information to avoid excessive use of Google resources.It will not charge you money as long as you use this to do this homework. It does not use a lot of calls.   
  
#register\_google(key = "your key here", write = TRUE) #### please delete your key information before submitting the compiled file. You can either compile as a docx file and delete this line, or use other software to hide the key information.   
  
# use geocode function to get latitude and longtitude of Tampa  
latlon <- geocode("Tampa, fl")  
  
# create the first zoomed map based on "mapZip", and plot a point representing Tampa  
mapZipZoomed <- mapZip + geom\_point(aes(x = latlon$lon, y = latlon$lat), color="darkred", size = 3)  
  
# zoom into the region arount Tampa with 10 degrees latitude and longtitude fluctuation (+/- 10)  
mapZipZoomed <- mapZipZoomed + xlim(latlon$lon-10, latlon$lon+10) + ylim(latlon$lat-10,latlon$lat+10) + coord\_map()  
  
# plot the map  
mapZipZoomed

# Write your code below.  
library(ggmap)

## Warning: package 'ggmap' was built under R version 4.2.2

## ℹ Google's Terms of Service: <]8;;https://mapsplatform.google.comhttps://mapsplatform.google.com]8;;>

## ℹ Please cite ggmap if you use it! Use `citation("ggmap")` for details.

library(ggplot2)  
mapZip <- ggplot(dfNew, aes(map\_id=stateName))  
mapZip <- mapZip + geom\_map(map=us, fill="black", color="white")  
mapZip <- mapZip + expand\_limits(x=us$long,y=us$lat)  
register\_google(key = "xxx", write = TRUE)

## ℹ Replacing old key (AIzaSyAn4T\_cRWEmBYnLchJ14hBMBvnt4XxwMx4) with new key in C:/Users/rocke/Documents/.Renviron

latlon <- geocode("Tampa, fl")

## ℹ <]8;;https://maps.googleapis.com/maps/api/geocode/json?address=Tampa,+fl&key=xxxhttps://maps.googleapis.com/maps/api/geocode/json?address=Tampa,+fl&key=xxx]8;;>

mapZipZoomed <- mapZip + geom\_point(aes(x= latlon$lon, y = latlon$lat), color="darkred", size = 3)  
mapZipZoomed <- mapZipZoomed + xlim(latlon$lon-10, latlon$lon+10) + ylim(latlon$lat-10,latlon$lat+10) + coord\_map()  
mapZipZoomed

