

Maps, 3 of 3

Leah Brooks

April 13, 2018

Today is our last formal mapping tutorial. We make choropleth maps, graduated symbol maps, and dot density maps. But first we get data into shape to map.

A. Get data into shape

We use the same block group data as in previous tutorials. See the tutorial from lecture 2 on where to find these data.

For this class, we'll also use the block group map, which I've downloaded from NHGIS.org, an organization that makes available consistent historical geographic information. You can download the DC block group file [here](#); this is the one we'll use in today's tutorial. If you would like the shapefiles for [Maryland](#) and [Virginia](#), follow the links. Any others, you can download directly from nhgis.org

Also, today we avoid anything from `sf`. It is superior for almost all of the tasks below, but not everyone got it to load properly – it seems to be quite finicky.

A.1 Some libraries

We'll start off loading shapefiles, which means we need to load a few libraries.

```
library(sp)
library(rgdal)

## rgdal: version: 1.2-16, (SVN revision 701)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.2.0, released 2017/04/28
##  Path to GDAL shared files: C:/Users/lbrooks/Documents/R/win-library/3.4/rgdal/gdal
##  GDAL binary built with GEOS: TRUE
##  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##  Path to PROJ.4 shared files: C:/Users/lbrooks/Documents/R/win-library/3.4/rgdal/proj
##  Linking to sp version: 1.2-7
```

A.2. Load block group maps and data, subset to DC

We begin by loading the DC block group map using `readOGR` as we did last class. Remember that this is a shapefile, so it includes information about the location of these places in space.

After loading the shapefile, take a look at its column names, the first few rows of the associated data, and use `dim()` to see how many block groups the file has.

```
# load a block group map
dcbgs.map <- readOGR(dsn = "H:/pppa_data_viz/2018/tutorials/lecture11/data/nhgis0018_shape/nhgis0018_shp",
                    layer = "DC_blkgrp_2010")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "H:/pppa_data_viz/2018/tutorials/lecture11/data/nhgis0018_shape/nhgis0018_shape/nhgis0018_sh
## with 450 features
## It has 15 fields
## Integer64 fields read as strings:  ALAND10 AWATER10
```

```
# take a look at this file
```

```
names(dcbgs.map)
```

```
## [1] "STATEFP10" "COUNTYFP10" "TRACTCE10" "BLKGRPCE10" "GEOID10"
## [6] "NAMELSAD10" "MTFCC10" "FUNCSTAT10" "ALAND10" "AWATER10"
## [11] "INTPTLAT10" "INTPTLON10" "GISJOIN" "Shape_area" "Shape_len"
```

```
dcbgs.map@data[1:5,]
```

```
## STATEFP10 COUNTYFP10 TRACTCE10 BLKGRPCE10 GEOID10 NAMELSAD10
## 0 11 001 006801 2 110010068012 Block Group 2
## 1 11 001 006700 3 110010067003 Block Group 3
## 2 11 001 007803 2 110010078032 Block Group 2
## 3 11 001 007803 3 110010078033 Block Group 3
## 4 11 001 007502 2 110010075022 Block Group 2
## MTFCC10 FUNCSTAT10 ALAND10 AWATER10 INTPTLAT10 INTPTLON10
## 0 G5030 S 90744 0 +38.8881502 -076.9821208
## 1 G5030 S 167777 0 +38.8872631 -076.9932764
## 2 G5030 S 137978 0 +38.8962653 -076.9371881
## 3 G5030 S 181559 0 +38.8923288 -076.9369370
## 4 G5030 S 346034 0 +38.8546445 -076.9674792
## GISJOIN Shape_area Shape_len
## 0 G11000100068012 90742.78 1260.214
## 1 G11000100067003 167777.14 1711.157
## 2 G11000100078032 137978.41 1525.427
## 3 G11000100078033 181560.19 1728.923
## 4 G11000100075022 346034.20 3475.022
```

```
dim(dcbgs.map)
```

```
## [1] 450 15
```

Make sure that we have what we think we do by plotting this map.

```
# plot it
```

```
plot(dcbgs.map)
```



Now we load block group data. This is the file we worked with before that has information about the attributes of individual block groups. I use the `table()` function to see how many states this file has. Since we'll just be working with DC, I subset the file to DC.

```
# load block group data
block.groups <- read.csv(
  "h:/pppa_data_viz/2018/tutorials/lecture02/acs_bgs20082012_dmv_20180123.csv")
#names(block.groups)
dim(block.groups)
```

```
## [1] 9708 3063
```

```
table(block.groups$STATE)
```

```
##
```

```
## 11 24 51
```

```
## 450 3926 5332
```

```
# subset to dc
```

```
block.groups.dc <- block.groups[which(block.groups$STATE == 11),]
```

```
# lets verify a variable we'll use in a minute
```

```
summary(block.groups.dc$B07201e1)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##         0      855    1222    1330    1644    4835
```

A.3. Merge map file and data, version 1

Recall that a block group is identified by a combination of state id (2 digit) + county id (3 digits) + tract id (6 digits) + block group id (1 digit). Thus, when we combine the block group map with the data, we need to merge on all four fields.

Another way of saying this is that tract 001001, block group 2 might exist in both Washington, DC and Prince George's County. Thus you need both the state and county ids as well as the tract and block group ones.

As we did last class, we use `sp`'s merge feature to add data to the shapefile.

```
dcbgs.map.test <- sp::merge(dcbgs.map,
                             block.groups.dc,
                             by.x = c("STATEFP10", "COUNTYFP10", "TRACTCE10", "BLKGRPC10"),
                             by.y = c("STATE", "COUNTY", "TRACT", "BLKGRP"),
                             all.x = TRUE)
summary(dcbgs.map.test@data$B07201e1)
```

The variable B07201 is the total "population 1 year and over living in a Metropolitan Statistical Area in the United States" (Census). Note that while it had values before, it doesn't now. This tells us that there is something screwy in our merge.

Always check merges!

A.4. Merge map file and data, version 2

To figure out why these files didn't merge, I looked at the merging variables in both datasets to see how they differed. I realized that the tract IDs in the data file (`block.groups.dc`) were not always six digits with leading and lagging zeros.

I fix this below (this took a fair amount of trial and error to get right). There are no new commands here, but putting a bunch of commands that we have used before together. The fundamental problem was that the tract field in the data file was missing leading zeros (zeros that go before the number to make it the full six digits).

```
### add leading and lagging zeros to tract ids
# first, variable is an integer, so we need a string variable to work with
block.groups.dc$tract.str <- as.character(block.groups.dc$TRACT)
# now add leading/lagging zeros as needed
block.groups.dc$county.fp <- ifelse(block.groups.dc$COUNTY == 1, "001", NA)
block.groups.dc$tract.fp <- block.groups.dc$tract.str
block.groups.dc$tract.fp <- ifelse(nchar(block.groups.dc$tract.str) == 5,
                                   paste0("0", block.groups.dc$tract.str),
                                   block.groups.dc$tract.fp)
block.groups.dc$tract.fp <- ifelse(nchar(block.groups.dc$tract.str) == 4,
                                   paste0("00", block.groups.dc$tract.str),
                                   block.groups.dc$tract.fp)
block.groups.dc$tract.fp <- ifelse(nchar(block.groups.dc$tract.str) == 3,
                                   paste0("000", block.groups.dc$tract.str),
                                   block.groups.dc$tract.fp)
# check results
block.groups.dc[which((block.groups.dc$tract.fp == "003000" & block.groups.dc$BLKGRP == 1) |
                      (block.groups.dc$tract.fp == "003000" & block.groups.dc$BLKGRP == 2) |
                      (block.groups.dc$tract.fp == NA)),
                 c("STATE", "COUNTY", "TRACT", "tract.fp", "county.fp", "BLKGRP", "tract.str")]
```

```
##      STATE COUNTY TRACT tract.fp county.fp BLKGRP tract.str
## 142     11      1  3000   003000      001      1      3000
## 143     11      1  3000   003000      001      2      3000
```

After I do all this fixing, I do some checking. In datasets like these, the file should have only one observation for each block group (state+county+tract+block group). We say that the file is “unique” by block group. Below I check whether this is the case.

Non-unique dataframes cause big troubles in merging.

To check for uniqueness, I use a `plyr` command called `count`, which counts unique values of the columns you request. The command `count` creates a variable in the output dataframe called `freq`, which is the number of times – the frequency – of each ID group that you put into the command.

In general, when you are merging, at least one of your dataframes should be unique by the merging variables. If both are not unique by the merging variables, you’re going to get unexpected (and perhaps unwanted) results.

Thus, to tell whether your dataframe is unique by the variables you’ve specified, you can look (as we do below) at the distribution of the count of by ID frequencies. If the variable `freq` always takes on the value 1, the dataset is unique by the ID variables.

```
# merge with spatial polygon
dim(block.groups.dc)
```

```
## [1] 450 3066
```

```
library(plyr)
# check if data obs are unique
d.unique <- count(block.groups.dc[,c("STATE","county.fp","tract.fp","BLKGRP")])
table(d.unique$freq)
```

```
##
## 1
## 450
```

```
d.unique.m <- d.unique[which(d.unique$freq > 1),]
d.unique.m
```

```
## [1] STATE      county.fp tract.fp BLKGRP      freq
## <0 rows> (or 0-length row.names)
```

```
# check if map obs are unique
dim(dcbgs.map)
```

```
## [1] 450 15
```

```
m.unique <- count(dcbgs.map@data[,c("STATEFP10","COUNTYFP10","TRACTCE10","BLKGRPCE10")])
table(m.unique$freq)
```

```
##
## 1
## 450
```

Now that the data are fixed, let’s try another merge, and check the variable that we looked at initially. As a reminder, the first two parts of the merge statement are the input dataframes. Next I specify the merging variables from the x and y datasets, and then finally, we tell R to keep all observations from dataframe x (`all.x = TRUE`) and dataframe y (`all.y = TRUE`).

```
dcbgs.map.data <- sp::merge(dcbgs.map,
                           block.groups.dc,
```

```

by.x = c("STATEFP10", "COUNTYFP10", "TRACTCE10", "BLKGRPCE10"),
by.y = c("STATE", "county.fp", "tract.fp", "BLKGRP"),
all.x = TRUE, all.y = TRUE)
dim(dcbgs.map.data@data)

```

```
## [1] 450 3077
```

```
summary(dcbgs.map.data@data$B07201e1)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##         0      855    1222    1330    1644    4835
```

As an aside, (no need to do this yourself) here is how I finally figured out what was going on in the problematic merge. I took a dataframe from the map, merged it with the block group data, and looked at the non-matches.

```

## how i finally solved the problem
# get two dataframes and check for non-matches
# get dataframe from map file
dcbgs.map.dataf <- as.data.frame(dcbgs.map)
dcbgs.map.dataf$md <- 1
block.groups.dc$dd <- 1
bgdc.dat <- block.groups.dc[,c("STATE", "county.fp", "tract.fp", "BLKGRP", "B00001e1", "dd", "TRACT")]
checker <- merge(dcbgs.map.dataf,
                 bgdc.dat,
                 by.x = c("STATEFP10", "COUNTYFP10", "TRACTCE10", "BLKGRPCE10"),
                 by.y = c("STATE", "county.fp", "tract.fp", "BLKGRP"),
                 all.x = TRUE, all.y = TRUE)
dim(checker)
checker[which(is.na(checker$md) == TRUE | is.na(checker$dd) == TRUE ),
        c("STATEFP10", "COUNTYFP10", "TRACTCE10", "TRACT", "BLKGRPCE10", "md", "dd")]
names(checker)

```

B. Create tract-level data

Remember that each tract is made up of individual block groups. Sometimes block groups are so small that they are difficult to see on a map. In this case, a researcher may wish to turn to tracts. Below we create a tract-level spatial dataframe.

I do this for both the data file – here we aggregate to tracts – and the map file, where we take a spatial union of block groups by tract.

B.1. Prepare the map

We begin with the map. I use the `unionSpatialPolygons()` function to do this. The inputs to this function are the original shapefile and the second argument is you telling R how to combine. In our case we combine by tract ID. Unfortunately, this output is not a spatial polygons dataframe – it's just a spatial polygon.

The next three lines of code take the spatial polygons and make it a spatial polygons dataframe; the final lines of code check this.

```

# make tract-level map
library(maptools)

```

```
## Checking rgeos availability: TRUE
```

```

library(raster)
class(dcbgs.map)

## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"

dcts.map.sp <- unionSpatialPolygons(dcbgs.map, dcbgs.map$TRACTCE10)
class(dcts.map.sp)

## [1] "SpatialPolygons"
## attr(,"package")
## [1] "sp"

# this output is a spatialpolygon file, not a spatial polygons dataframe
# so we need to add some basic data
# this seems to put tract ids as row names
# thank you example here
# https://gis.stackexchange.com/questions/61633/convert-a-spatial-polygon-object-to-data-frame-using-r
dcts.map.df<- data.frame(tract.fp = getSpPPolygonsIDSlots(dcts.map.sp))

## Warning: use *apply and slot directly
row.names(dcts.map.df) <- getSpPPolygonsIDSlots(dcts.map.sp)

## Warning: use *apply and slot directly
dcts.map <- SpatialPolygonsDataFrame(dcts.map.sp, data=dcts.map.df)
names(dcts.map@data)

## [1] "tract.fp"
dcts.map@data[1:5,]

## [1] 000100 000201 000202 000300 000400
## 179 Levels: 000100 000201 000202 000300 000400 000501 000502 ... 011100

```

B.2. Prepare the data

Now we need to make tract-level data from the block group data. We do this by summarizing the data using dplyr's ddply function. I then check the variables I just created, and I can therefore tell you that there is something wrong with the population variable. This is a problem from my initial creation of these data, but I haven't been able to find the error. You should not use variables B00001e1 and B00002e2 – apologies.

```

# make tract-level data
tracts.dc <- ddply(block.groups.dc, "tract.fp", summarize,
  mob.same.house = sum(B07201e2, na.rm = TRUE),
  mob.tot.pop = sum(B07201e1, na.rm = TRUE))
summary(tracts.dc$mob.same.house)

```

```

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##         0    1959     2487    2686    3356    6533

```

```
summary(tracts.dc$mob.tot.pop)
```

```

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##         0    2440     3077    3345    4164    7907

```

```
dim(tracts.dc)

## [1] 179 3

tracts.dc[1:5,]

##   tract.fp mob.same.house mob.tot.pop
## 1  000100          3862          4895
## 2  000201          2787          4074
## 3  000202          3385          4923
## 4  000300          4550          5310
## 5  000400          1325          1501
```

B.3. Merge the data

Now we are finally ready to have a tract-level spatial dataset by merging the map and data files we just created. After the merge I check the size of the file, and whether there are any tracts with missing values for the variable we just created `mob.tot.pop`.

```
# merge the two
dim(dcts.map)

## [1] 179 1

dim(tracts.dc)

## [1] 179 3

dcts.map.data <- sp::merge(dcts.map,
                           tracts.dc,
                           by.x = c("tract.fp"),
                           by.y = c("tract.fp"),
                           all.x = TRUE, all.y = TRUE)

dim(dcts.map.data@data)

## [1] 179 3

dcts.map.data@data[which(is.na(dcts.map.data@data$mob.tot.pop) == TRUE),]

## [1] tract.fp      mob.same.house mob.tot.pop
## <0 rows> (or 0-length row.names)
```

B.4. Create new variables

Finally, we create a new variable that tells us the share of people over the age of one who lived in the same house one year ago. Make summaries of the variable just created and its components to check if things seem reasonable.

```
dcts.map.data@data$mob.same.hs.shr <-
  dcts.map.data@data$mob.same.house / dcts.map.data@data$mob.tot.pop
summary(dcts.map.data@data$mob.same.house)

##   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      0    1959    2487    2686    3356    6533

summary(dcts.map.data@data$mob.tot.pop)
```



```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##           0    2440    3077    3345    4164    7907
```

```
summary(dcts.map.data@data$mob.same.hs.shr)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's
## 0.2304  0.7566  0.8201  0.8068  0.8786  0.9889      1
```

C. Choropleth maps

Now, with all these data prepared, we are ready to make some choropleth maps. This tutorial owes a big debt to Claudia Engel's work [here](#), which you might like to look at.

Using the `sp` package, we can make a very simple choropleth plot. We need to tell R just the spatial polygon dataframe and the variable you would like to plot. R chooses cutoffs and colors.

```
# lets try with defaults
```

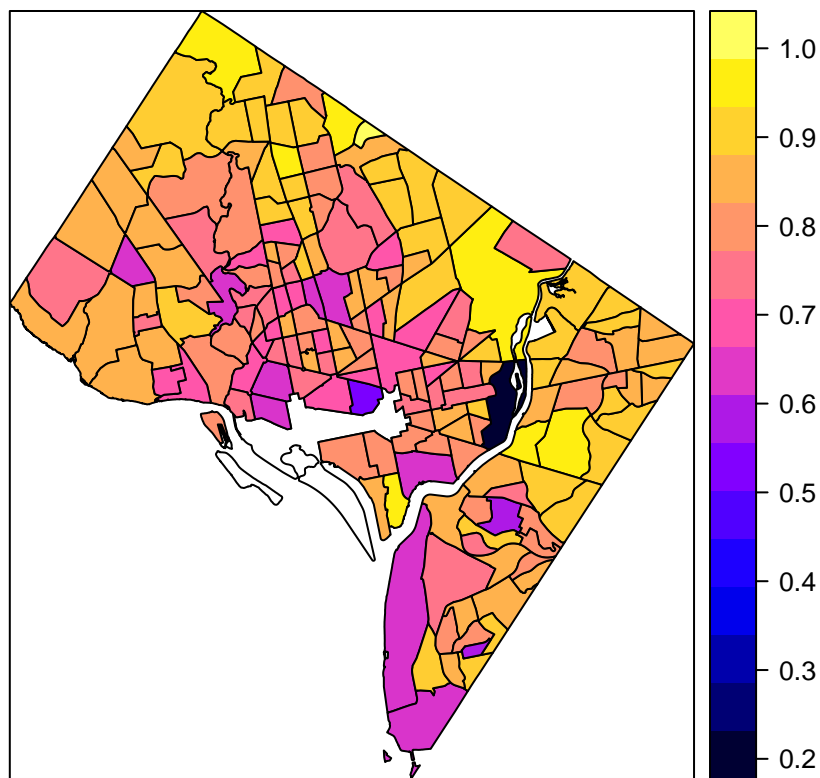
```
names(dcts.map.data@data)
```

```
## [1] "tract.fp"      "mob.same.house" "mob.tot.pop"    "mob.same.hs.shr"
```

```
dcts.map.data@data[1:5,]
```

```
##      tract.fp mob.same.house mob.tot.pop mob.same.hs.shr
## 1      000100          3862        4895      0.7889683
## 2      000201          2787        4074      0.6840943
## 3      000202          3385        4923      0.6875889
## 4      000300          4550        5310      0.8568738
## 5      000400          1325        1501      0.8827448
```

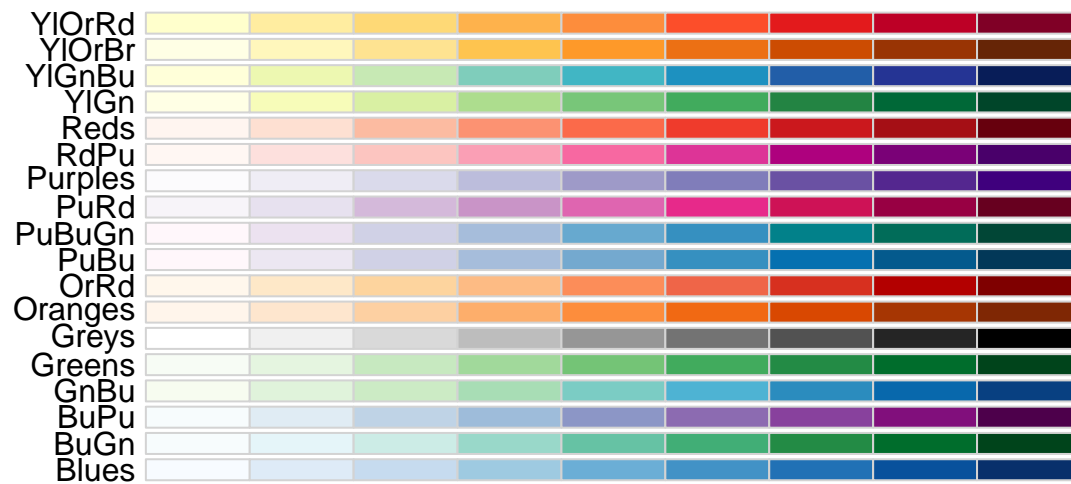
```
spplot(dcts.map.data, "mob.same.hs.shr")
```



Hopefully we can do a little better with color. We are using a new library which pulls up good-looking palettes. You may need to install the package `RColorBrewer`. I use the `display.brewer.all()` command to see what types of sequential palettes the program has. I create a palette of greys with 5 colors called `grey.pal` using the `brewer.pal` function, which takes the number of colors and the name of the colors (from the earlier plot) as arguments.

With this palette, I make the choropleth plot with 4 cut (thus requiring five colors). I have no idea how R cuts the data for this.

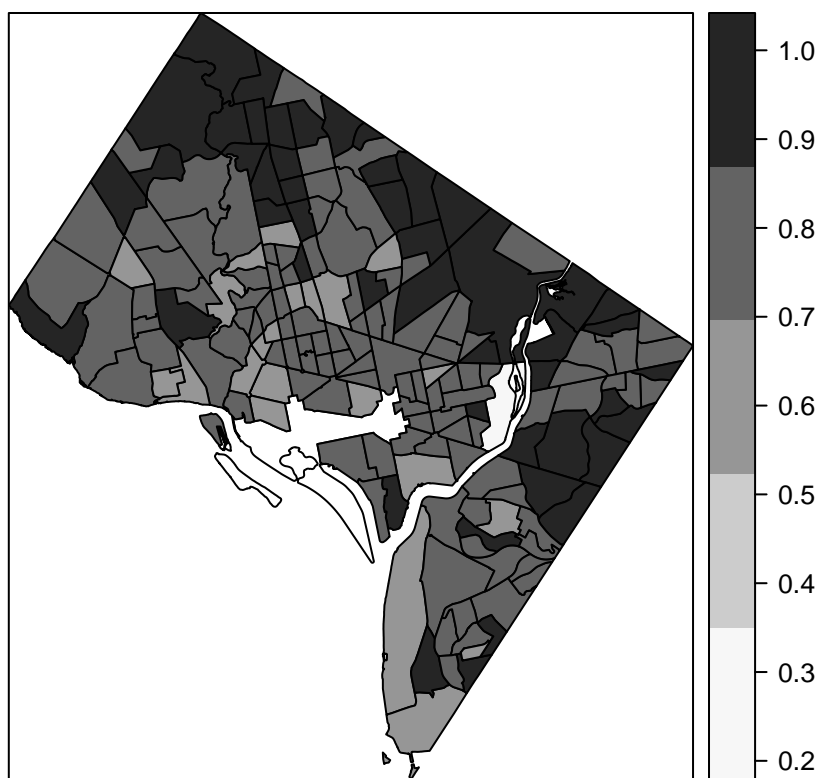
```
# lets try to do better with color  
library(RColorBrewer)  
display.brewer.all(type="seq")
```



```
grey.pal <- brewer.pal(5,"Greys")
class(grey.pal)
```

```
## [1] "character"
```

```
# notice that cuts is one less than the number of colors
spplot(dcts.map.data, "mob.same.hs.shr", col.regions = grey.pal, cuts = 4)
```



Though how you should just the data depends on what question you are asking, it is frequently useful to make bins with equal numbers of observations. Here I divide the data into quintiles, as we did in a previous lecture. But I do not need to create a quintiles variable (which I do just for kicks). I also add a title and use the `at=` option to tell R that we are dividing by the quint variable.

```
# now lets use quintiles instead
quints <- quantile(dcts.map.data$mob.same.hs.shr, probs = seq(0,1,0.2), na.rm = TRUE)
quints
```

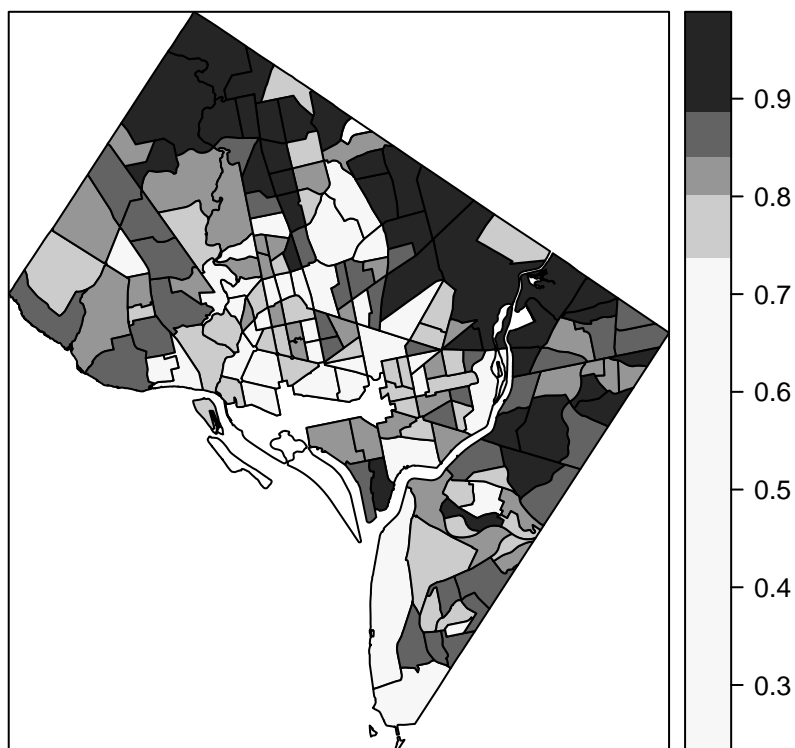
```
##           0%          20%          40%          60%          80%          100%
## 0.2304081 0.7374368 0.8013798 0.8408305 0.8871339 0.9889120
```

```
dcts.map.data@data <- within(dcts.map.data@data,
                             quintile <- as.integer(cut(mob.same.hs.shr,quints,
                                                         include.lowest = TRUE)))
table(dcts.map.data@data$quintile)
```

```
##
##  1  2  3  4  5
## 36 35 36 35 36
```

```
spplot(dcts.map.data, "mob.same.hs.shr",
       at = quint,
       col.regions = grey.pal,
       main = "DC: Share of Population in Same House One Year Ago")
```

DC: Share of Population in Same House One Year Ago



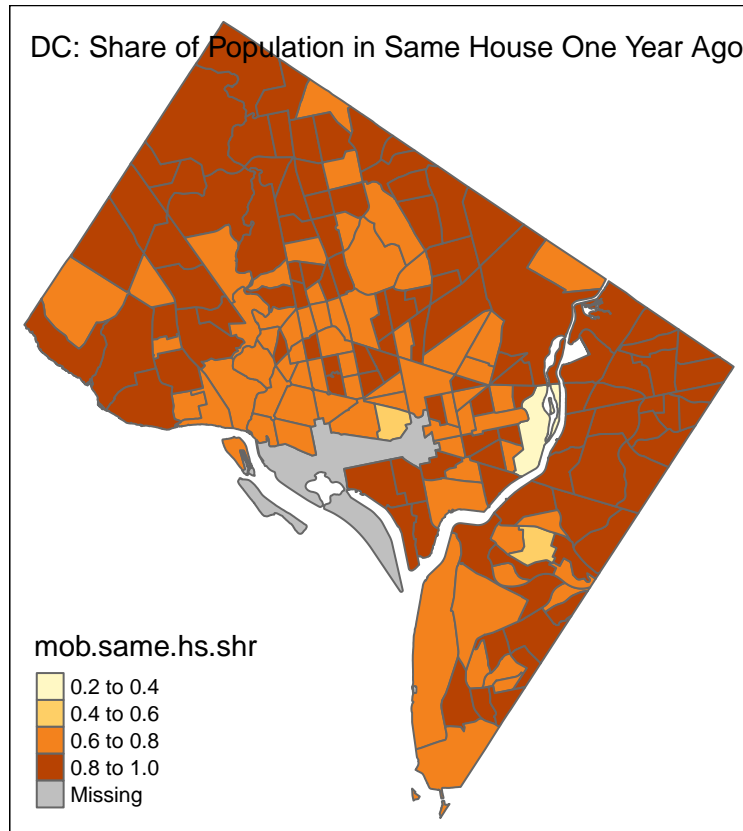
Rosa provides a better legend solution: Let's use other map package. First, check whether you have the `tmap` and `tmtools` packages. If you do not have, please install these packages first.

With the `tmap` package, you can generate choropleth maps with great flexibility and the syntax is similar to that of `ggplot2`. Here is the basic command.

Note that this basic command automatically makes an equal division of categories between 0 and 1; this is likely a poor choice for these particular data.

```
library(tmap)
library(tmtools)

qtm(dcts.map.data, fill = "mob.same.hs.shr",
    title = "DC: Share of Population in Same House One Year Ago")
```



D. Graduated Symbol Maps

Graduated symbol maps, unlike choropleth maps, rely on points. Our first job, therefore, is to make a new point version of the polygon census tract map. Then, with the map in hand, we can add data and plot the points. I relied heavily on [this](#) tutorial when crafting this section.

D.1. Make point data

We begin by using the `gCentroid` function to extract the point at the center of each tract (R calculates this for you). The `byid = TRUE` part of this command tells R to make a centroid for each ID in your dataframe. We then make these points into a spatial points dataframe with the `SpatialPointsDataFrame()` command.

I check the output a bit with `class()`, `head()`, `plot()` and `summary()`.

```
# first, lets find the x/y centroid of every census tract, so we'll have points to map
# see explanation
# https://stackoverflow.com/questions/32571956/get-data-frame-with-polygons-id-and-centroid-lat-long-in
library(rgeos)

## rgeos version: 0.3-26, (SVN revision 560)
## GEOS runtime version: 3.6.1-CAPI-1.10.1 r0
## Linking to sp version: 1.2-7
## Polygon checking: TRUE
```

```
tpoints <- SpatialPointsDataFrame(gCentroid(dcts.map.data, byid = TRUE),
                                  dcts.map.data@data)
```

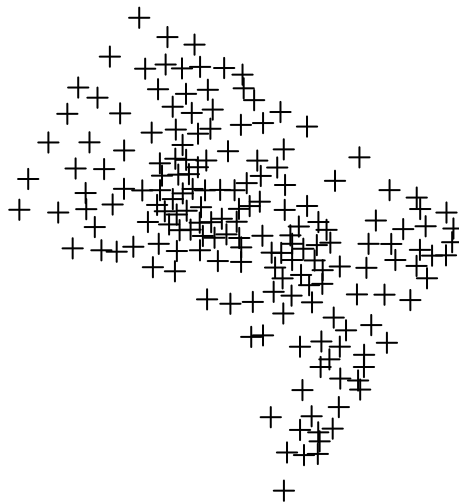
```
class(tpoints)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
head(tpoints)
```

```
##   tract.fp mob.same.house mob.tot.pop mob.same.hs.shr quintile
## 1   000100         3862         4895         0.7889683         2
## 2   000201         2787         4074         0.6840943         1
## 3   000202         3385         4923         0.6875889         1
## 4   000300         4550         5310         0.8568738         4
## 5   000400         1325         1501         0.8827448         4
## 6   000501         2070         3189         0.6491063         1
```

```
plot(tpoints)
```



```
summary(tpoints)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##           min      max
## x 1611978.6 1628568.3
## y  310127.5  328201.1
## Is projected: TRUE
```

```
## proj4string :
## [+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=37.5 +lon_0=-96 +x_0=0
## +y_0=0 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0]
## Number of points: 179
## Data attributes:
##      tract.fp  mob.same.house  mob.tot.pop  mob.same.hs.shr  quintile
## 000100 : 1  Min.      : 0  Min.      : 0  Min.      :0.2304  Min.      :1
## 000201 : 1  1st Qu.:1959  1st Qu.:2440  1st Qu.:0.7566  1st Qu.:2
## 000202 : 1  Median :2487  Median :3077  Median :0.8201  Median :3
## 000300 : 1  Mean   :2686  Mean   :3345  Mean   :0.8068  Mean   :3
## 000400 : 1  3rd Qu.:3356  3rd Qu.:4164  3rd Qu.:0.8786  3rd Qu.:4
## 000501 : 1  Max.    :6533  Max.    :7907  Max.    :0.9889  Max.    :5
## (Other):173                      NA's     :1      NA's     :1
```

Unfortunately, I know from painful experience that the projection we just created will not match with the google map we'd like to match to. So we need to change the projection of this map to something that has coordinates in latitude and longitude, rather than meters (from some point).

I use the `spTransform()` function to tell R that we want to go from the current coordinate reference system to EPSG number 4326. See this [handy website](#) with information on virtually all projections.

```
# but we need to put points into lat/long to get them to agree with the google map
tpoints.geog <- spTransform(tpoints, CRS("+init=epsg:4326"))
crs(tpoints.geog)
```

```
## CRS arguments:
## +init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
## +towgs84=0,0,0
```

Of course, the mapping we'll do below requires a dataframe, not a spatial points dataframe. This means that we need a dataframe with the x/y coordinates of each tract. We do this below. First we find the coordinates of `tpoints.geog` using the `coordinates()` function, and then make a dataframe with the `data.frame()` function.

Unfortunately, R puts the tract ID as a row name, rather than as a variable. I fix this with the `plyr` function `rownames_to_column` from the `tibble` package.

Finally, I check the output. Does it look ok?

```
# make a dataframe with coordinates
tract.coords <- data.frame(coordinates(tpoints.geog))
library(dplyr)
```

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

## The following objects are masked from 'package:rgeos':
##
##      intersect, setdiff, union

## The following objects are masked from 'package:raster':
##
##      intersect, select, union

## The following objects are masked from 'package:plyr':
##
##      arrange, count, desc, failwith, id, mutate, rename, summarise,
##      summarize

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



```
##
## filter, lag
## The following objects are masked from 'package:base':
##
## intersect, setdiff, setequal, union
class(tract.coords)

## [1] "data.frame"
tract.coords <- tibble::rownames_to_column(tract.coords, "tract.fp")
names(tract.coords)

## [1] "tract.fp" "x" "y"
tract.coords[1:5,]

## tract.fp x y
## 1 000100 -77.06059 38.90836
## 2 000201 -77.07483 38.90934
## 3 000202 -77.06828 38.90784
## 4 000300 -77.07566 38.91755
## 5 000400 -77.06590 38.92389
```

D.2. Add in data to points

Of course, these points are not very useful without any data. Here we merge in the tract-level data, and check on the results. Since I already merged in tract level data to `dcts.map.data`, I just tell R to treat that as a dataframe (using the `as.data.frame()` command) and merge it in. Finally, I check the results.

```
# merge in the data
names(dcts.map.data)

## [1] "tract.fp" "mob.same.house" "mob.tot.pop" "mob.same.hs.shr"
## [5] "quintile"
tract.coords <- merge(tract.coords, as.data.frame(dcts.map.data),
                      by = "tract.fp")
dim(tract.coords)

## [1] 179 7
tract.coords[1:5,]

## tract.fp x y mob.same.house mob.tot.pop mob.same.hs.shr
## 1 000100 -77.06059 38.90836 3862 4895 0.7889683
## 2 000201 -77.07483 38.90934 2787 4074 0.6840943
## 3 000202 -77.06828 38.90784 3385 4923 0.6875889
## 4 000300 -77.07566 38.91755 4550 5310 0.8568738
## 5 000400 -77.06590 38.92389 1325 1501 0.8827448
## quintile
## 1 2
## 2 1
## 3 1
## 4 4
## 5 4
```

D.3. Make a map

Now we are ready to make a graduated density map. First we'll pull a map from google as the background. For more details, see last week's tutorial.

```
# lets get a dc map from google
library(ggmap)
```

```
## Warning: package 'ggmap' was built under R version 3.4.4
```

```
## Loading required package: ggplot2
```

```
##
```

```
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:raster':
```

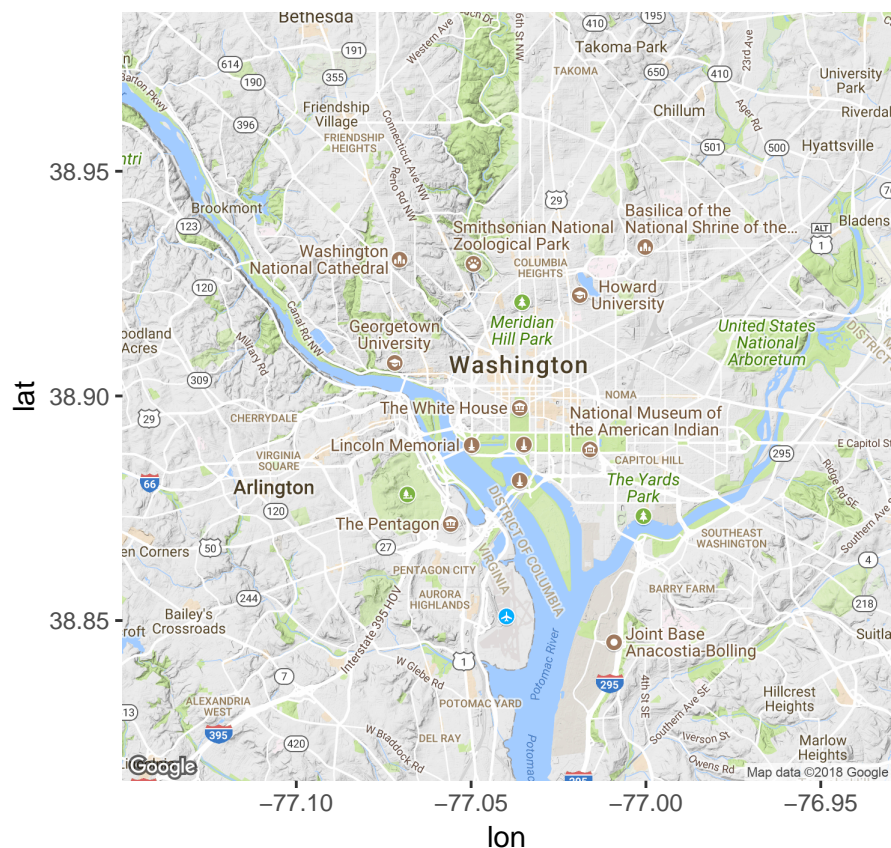
```
##
```

```
##      calc
```

```
dmv.map.gl <- get_map(location = c(lon = -77.04, lat = 38.9), zoom = 12, source = "google")
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=38.9,-77.04&zoom=12&size=640x640
```

```
ggmap(dmv.map.gl)
```



Now that I have this map, I double-check the data, and use ggplot to make a combination of graduated size circles (from `geom_point()`) on top of the google map.

```
# plot number of people in the same house a year ago
summary(tract.coords$mob.same.house)
```

```
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##         0    1959    2487     2686    3356    6533
```

```
summary(tract.coords$x)
```

```
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
## -77.11 -77.03 -77.01 -77.01 -76.99 -76.92
```

```
names(tract.coords)
```

```
## [1] "tract.fp"      "x"              "y"              "mob.same.house"
## [5] "mob.tot.pop"   "mob.same.hs.shr" "quintile"
```

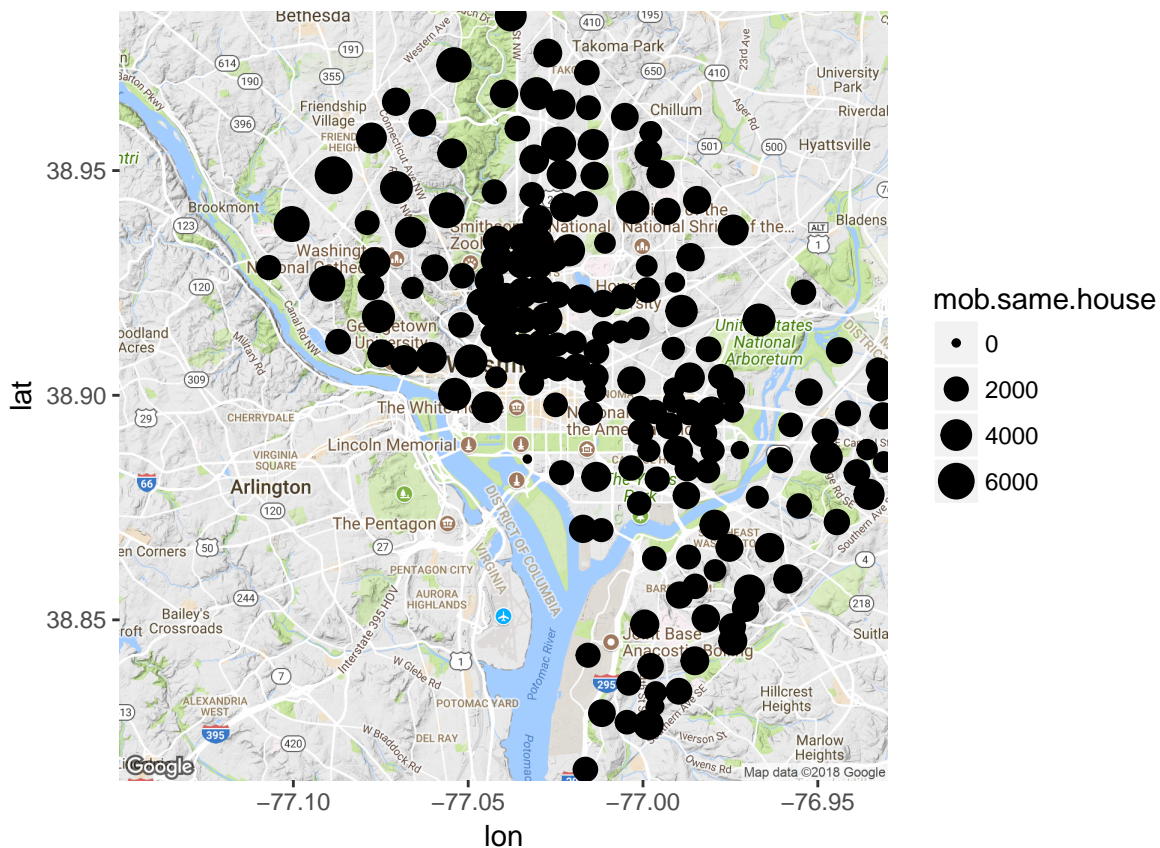
```
# plot people in same city one year ago with circles of varying size
```

```
library(ggplot2)
```

```
ggmap(dmv.map.gl) +
```

```
  geom_point(data=tract.coords, aes(x=x, y = y, size = mob.same.house))
```

```
## Warning: Removed 4 rows containing missing values (geom_point).
```



I don't love this final product. I think the google map background is too noisy – recall that this is fixable with input options – and that the circles are probably too dark and too equally-sized to convey information well.

E. Dot Density Maps

We wrap up with dot density plots. This portion of the tutorial is very heavily draw from [this work](#), and I am appreciative.

For this map, we'll use the block group data, since it will generate more finely spaced dots. The idea here is that you use dots to represent a certain number of people – here each dot represents 10.

We begin by making a subset of the block group dataframe, keeping only five columns that describe your housing one year ago: Lived in the same house, lived in the same metro area, lived in a different metro area, lived in a non-metro area, and lived outside the US. We divide these figures by 10 so that each dot will represent ten people.

```
# we'll start with the block group data
num.dots <- select(block.groups.dc,
                   c("B07201e2", "B07201e4", "B07201e7", "B07201e13", "B07201e14")) / 10
dim(num.dots)

## [1] 450 5
```

The trick to the dot density is putting dots in polygons. We rely on a R function called `dotsInPolys()` that does this for us. Here, for each column in `num.dots` we are creating a list of polygons called `sp.dfs` that have dots randomly assigned within the relevant polygon (`f="random"`).

I will go over this command in greater detail during the lecture, but the idea is that we are using R's `lapply()` function to do things over a list of `x`, where `x` takes on the values of the names of the columns in `num.dots` (`names(num.dots)`). The output is a list of dataframes, each of which is created by the `dotsInPolys()` command.

Note that because dots are randomly assigned within polygons of the tracts, we could theoretically get a different looking map each time we do this. However, tracts are quite small, so the map is unlikely to look different in practice (as Rosa can attest, since she tried it.)

```
# then put dots randomly in the polygons
sp.dfs <- lapply(names(num.dots), function(x) {
  dotsInPolys(dcbgs.map, as.integer(num.dots[, x]), f="random")
})
```

Finally, we plot this dot density map. First we look at the coordinates of the `tpoints` file, so I can properly locate my legend (though I want to admit this took a lot of trial and error).

(As an aside, we can use the original projection here, since we are not matching this map with any other. We could alternatively use the second one; it doesn't really matter.)

We choose another Brewer palette and add a legend to the plot.

```
# needed for figuring out legend
summary(coordinates(tpoints))

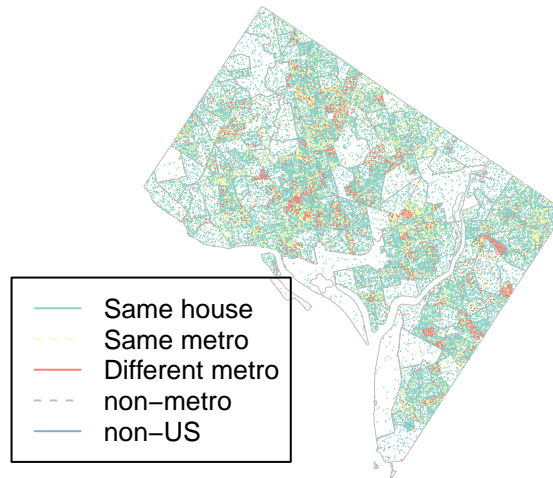
##           x           y
## Min.   :1611979 Min.   :310128
## 1st Qu.:1618026 1st Qu.:318600
## Median :1620668 Median :320302
## Mean   :1620742 Mean   :320177
## 3rd Qu.:1623404 3rd Qu.:322430
## Max.   :1628568 Max.   :328201

# plot them
plot(dcbgs.map, lwd = 0.01, border = "grey")
pal <- c("#8dd3c7", "#ffffb3", "#fb8072", "#bebada", "#80b1d3")
```

```

for (i in 1:length(sp.dfs)) {
  plot(sp.dfs[[i]], add = T, pch = 16, cex = 0.1, col = pal[i])
}
legend(1605000, 317400, legend=c("Same house", "Same metro", "Different metro", "non-metro", "non-US"),
      col=pal, lty=1:2, cex=0.8)

```



```
summary(dcbgs.map)
```

```

## Object of class SpatialPolygonsDataFrame
## Coordinates:
##      min      max
## x 1610830.2 1629412
## y  308504.6 329361
## Is projected: TRUE
## proj4string :
## [+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=37.5 +lon_0=-96 +x_0=0
## +y_0=0 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0]
## Data attributes:
## STATEFP10 COUNTYFP10  TRACTCE10  BLKGRPC10      GEOID10
## 11:450      001:450    001001 : 6    1:179      110010001001: 1
##                               002102 : 6    2:156      110010001002: 1
##                               001500 : 5    3: 74      110010001003: 1
##                               002101 : 5    4: 33      110010001004: 1
##                               005500 : 5    5:  6      110010002011: 1
##                               007601 : 5    6:  2      110010002021: 1
##                               (Other):418      (Other)      :444

```

```

##          NAMELSAD10    MTFCC10    FUNCSTAT10    ALAND10    AWATER10
## Block Group 1:179    G5030:450    S:450          100233 : 1    0          :378
## Block Group 2:156                                1011677: 1    83          : 2
## Block Group 3: 74                                102295 : 1    1059         : 1
## Block Group 4: 33                                1033566: 1    106          : 1
## Block Group 5:  6                                1040527: 1    11701        : 1
## Block Group 6:  2                                104202 : 1    124342       : 1
##                                                    (Other):444    (Other): 66
##          INTPTLAT10    INTPTLON10    GISJOIN
## +38.8132451: 1    -076.9143351: 1    G11000100001001: 1
## +38.8221670: 1    -076.9178170: 1    G11000100001002: 1
## +38.8259640: 1    -076.9185396: 1    G11000100001003: 1
## +38.8260370: 1    -076.9200165: 1    G11000100001004: 1
## +38.8263382: 1    -076.9222541: 1    G11000100002011: 1
## +38.8269677: 1    -076.9232083: 1    G11000100002021: 1
## (Other)      :444    (Other)      :444    (Other)      :444
##          Shape_area    Shape_len
## Min.      : 31158    Min.      : 745.7
## 1st Qu.: 126137    1st Qu.: 1547.6
## Median : 214271    Median : 2184.4
## Mean      : 353585    Mean      : 2632.4
## 3rd Qu.: 366470    3rd Qu.: 2975.5
## Max.      :6636981    Max.      :33406.2
##

```

F. Homework

1. How many block groups are there in Washington, DC? How many tracts?
2. Make a different choropleth map using a non-grey ColorBrewer palette.
3. Make either a graduated density plot or a dot density plot using at least one dataset not from this tutorial.