

# Maps, 2 of 3

*Leah Brooks*

*March 25, 2018*

Today's class investigates more of what R can do with maps. We undertake 2 major tasks

1. Pull in a map from google maps as a background
2. Calculate how many crimes occur in each ward in DC and make a map based on that

Each of these teaches you some additional R skills. The first task uses `ggmap()`, which is a command that works with `ggplot` to pull in maps from outside sources.

We also review skills from previous classes

- merging
- summarizing with `plyr`
- renaming variables
- conditional formatting: `ifelse()`
- looking at distributions with `table()`

## A. Pulling in google maps

### A.1. Install the relevant packages

To begin, you'll need to install yet another package: `ggmap`. You can find a helpful `ggmap` cheatsheet [here](#), and the full discussion of this package [here](#).

In this class, we'll work with two types of maps: vector maps and raster maps. Last class we worked with sets of points, polygons or lines. Those maps are all vector maps.

In contrast, the `ggmap` and `get_map` functions that we are learning today generate raster maps. The `get_map` function pulls in maps from your choice of source: google, open streetmap, and others. Raster maps are pixels that are colored in based on an attribute. In google maps, the pixel is usually a color based on whether the area is a road (yellow), or a park (green) or a border (black).

The `get_map()` command is structured like this (here I'm just highlighting the most important commands):

```
new.object <- get_map(location = address / c(lon = num, lat = num) / name of location,  
                      source = c("google", "osm", "stamen",  
                                zoom = some number)
```

You tell R the location you want to map, and where you want to get the map from (google, open street map, or others), and what zoom you'd like. R then grabs this raster map and puts it into `new.object`, which you can then plot with `ggmap()`.

### A.2. Make a map of DC from google maps

Now we'll use `ggmap`'s `get_map` to make a map of DC, and we'll rely on google's geocoding to find DC for us.

I will warn you that sometimes when I do this I'll get a "OVER\_QUERY\_LIMIT" error, which means that too many people at GW are querying google at the same time. In my (limited) experience, I re-run a few times and this error goes away. Google limits queries in two ways: a number of queries from the same IP

at the same time, and the total number of queries in a time period. My guess is that GW is hitting the same-time limit.

The commands below call the relevant libraries, pull in a google map, and then use the `ggmap` command to plot this new object.

```
# pull in relevant libraries  
library(ggmap)
```

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

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

```
library(rgdal)
```

```
## Loading required package: sp
```

```
## 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
```

```
# pull in map
```

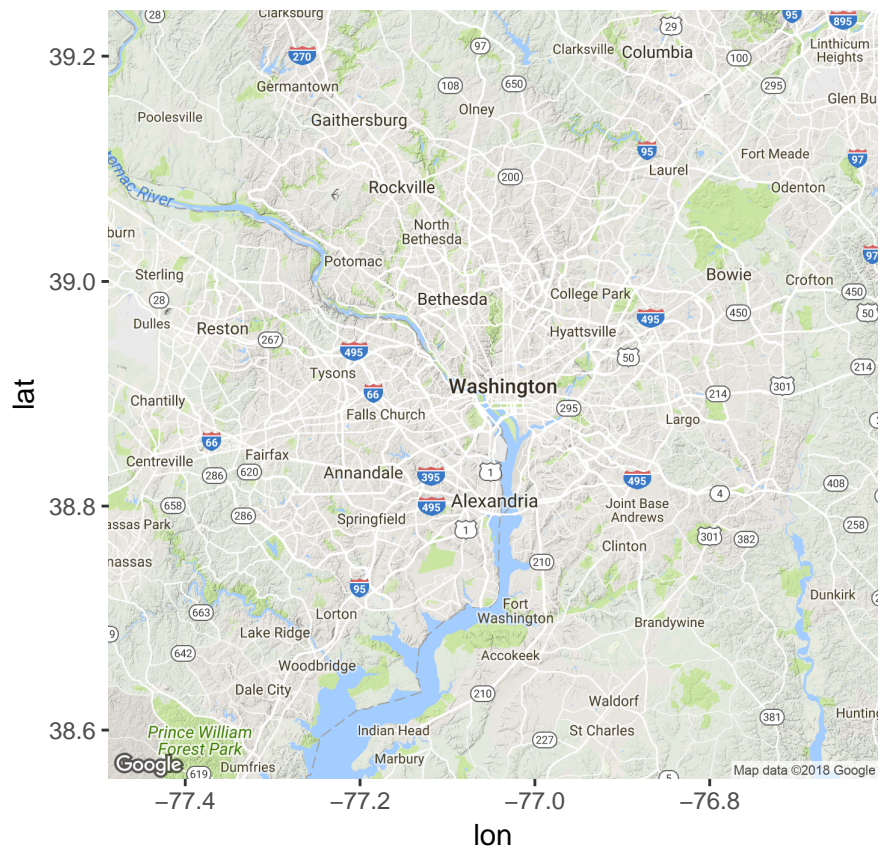
```
dmv.map <- get_map(location = "george washington university", source = "google")
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=george+washington+university&zoom=15
```

```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=george%20washington%20university&key=AIzaSyB...
```

```
# plot the map we pulled in
```

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



### A.3. Zooming in

What if we want to zoom in on GW? We can modify the zoom of what we pull in. We tell `get_map` a center point (`location = c(lon = -77.05, lat = 38.9)`) and a zoom level, here `zoom = 16`.

```
near.us.map <- get_map(location = c(lon = -77.05, lat = 38.9), zoom = 16, source = "google")
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=38.9,-77.05&zoom=16&size=640x640
```

```
ggmap(near.us.map)
```



Does this look right to you?

#### A.4. Changing google map options

This map we just pulled in has an awful lot of junk on it. Now we re-write our command to just get the part of the map (the layers) we'd like.

To do this, we'll use the `get_googlemap` command, which allows for us to be very specific about what we pull from the map.

You can make your own choices about what to get by going to the “styling wizard” [here](#). When you arrive at the wizard, choose “create a style.” I used the “standard” theme, and then clicked on “more options” to change what I saw on the map. After all my choices, when I clicked “finish,” google gives two options, and I took the “grab the url” for getting the following weblink:

https://maps.googleapis.com/maps/api/staticmap?key=YOUR\_API\_KEY&center=38.899050194500695,-77.0476046&zoom=16&format=png&maptype=roadmap&style=feature:landscape.man\_made%



[7Cvisibility:off&style=feature:poi%7Cvisibility:off&size=480x360](http://maps.googleapis.com/maps/api/staticmap?center=38.899050194500695,-77.0476046&zoom=16&format=png&maptype=roadmap&style=feature:poi%7Cvisibility:off&size=480x360)

Look carefully at the hyperlink and you can see that it has all the parts of the command you'll need to call, separated by &s. First, there is a `center=38.899050194500695,-77.0476046`. Then there is the zoom level with `zoom=16`. The default format seems to be `png`, and I requested a roadmap (`maptype=roadmap`), though I could equally well have requested a satellite map or some other type.

The rest of the URL is the style I've requested. I've turned off the "man-made landscape", and well as the points of interest (poi). The trickiest bit in making this into a command is how to enter all of the style features from the URL. See the command below for details.

```
# define the styles from the URL
style1 <- c(feature = "landscape.man_made", visibility = "off")
style2 <- c("&style=", feature = "poi", visibility = "off")

# put them into one variable
style <- c(style1, style2)

# pull the google map with these styles
near.us.map2 <- get_googlemap(center = c(lon = -77.0476046, lat = 38.899050194500695),
                             zoom = 16,
                             format = "png8",
                             maptype = "roadmap",
                             style = style,
                             size = c(480,360))
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=38.89905,-77.047605&zoom=16&size=
```

```
# plot it
ggmap(near.us.map2)
```



## A.5. Adding other stuff to your google map

One very powerful part of this mapping is that you can add additional layers on top of this google map. Today we'll use a dataset from last class: the 2017 DC crimes.

To do this, we need to walk through a few steps \* pull in the 2017 crime data as a shapefile \* make the shapefile into a data frame (so it's plottable with `ggplot`) \* plot it and the DC map together

Instead of the crime data we downloaded for the previous class, today we'll grab the crime shapefile. This has all the data from the previous class, plus the spatial location of each crime. Download the shapefile by going to [this page](#), and then choosing "download," and "shapefile."

Along the way, I'll show you how you could just plot the points from the crimes dataframe (not shapefile) by themselves, and how it might look if you do it wrong.

To pull in the shapefile, use the same `readOGR()` command as last class.

To put what's in this shapefile as a simple scatter plot, you need to take the data out of the shapefile. For points data like these, you can do this with the `data.frame()` command. So `crime.2017.data` is a regular old dataframe, while `crime.2017` is a spatial dataframe.

We then use the familiar `ggplot` to try plotting the data, using just longitude and latitude. You'll see that it doesn't look as nice as what we'll do later.

Finally, when Rosa did the tutorial, she loaded only 33,069 crimes, rather than the 33,073 crimes I did. We could not find the source of the discrepancy. Online, [opendatadc.gov](#) seems to indicate there should be 33,069 crimes, so either outcome is ok.

```

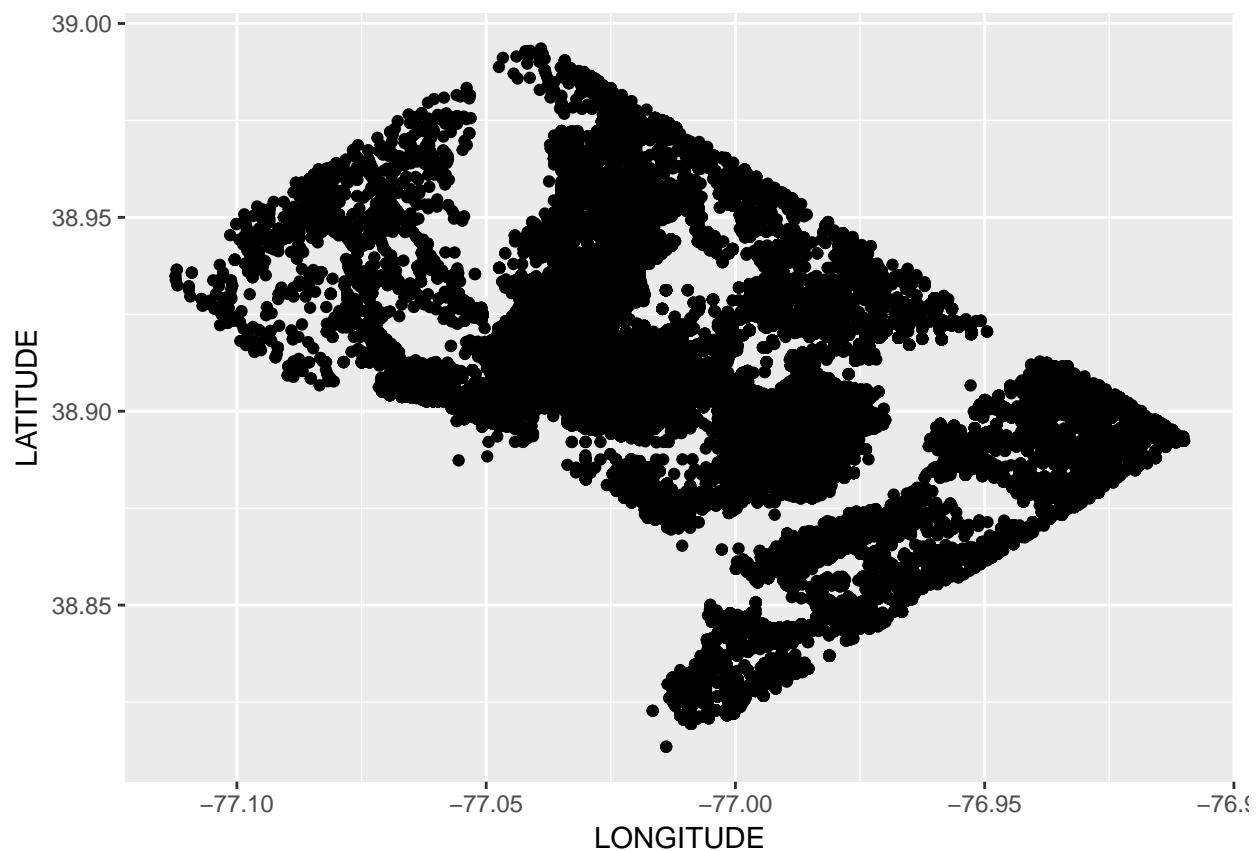
# load a shapefile of the 2017 dc crime data
library(rgdal)
crime.2017 <- readOGR(dsn = "H:/pppa_data_viz/2018/tutorials/lecture10/data/2018-03-20_Crime_Incidents_2017.shp",
                      layer = "Crime_Incidents_in_2017")

## OGR data source with driver: ESRI Shapefile
## Source: "H:/pppa_data_viz/2018/tutorials/lecture10/data/2018-03-20_Crime_Incidents_in_2017", layer: Crime_Incidents_in_2017
## with 33073 features
## It has 22 fields
## Integer64 fields read as strings:  OBJECTID

# need to make this a dataframe
# this is ok with points, for polygons you need fortify
crime.2017.df <- data.frame(crime.2017)

# look at map of
ggplot(crime.2017.df) +
  geom_point(aes(x=LONGITUDE, y = LATITUDE))

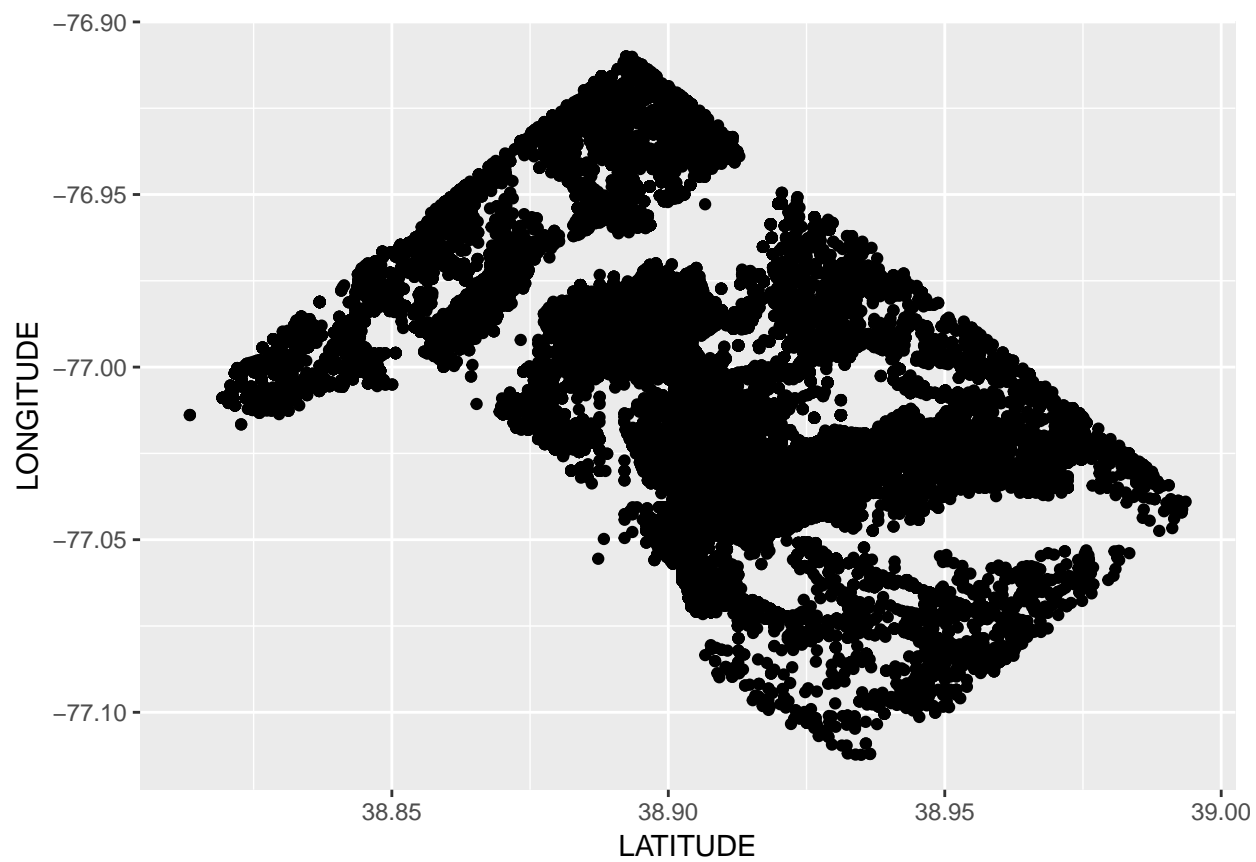
```



```

# what if you made a mistake and did it backward?
ggplot(crime.2017.df) +
  geom_point(aes(x=LATITUDE, y = LONGITUDE))

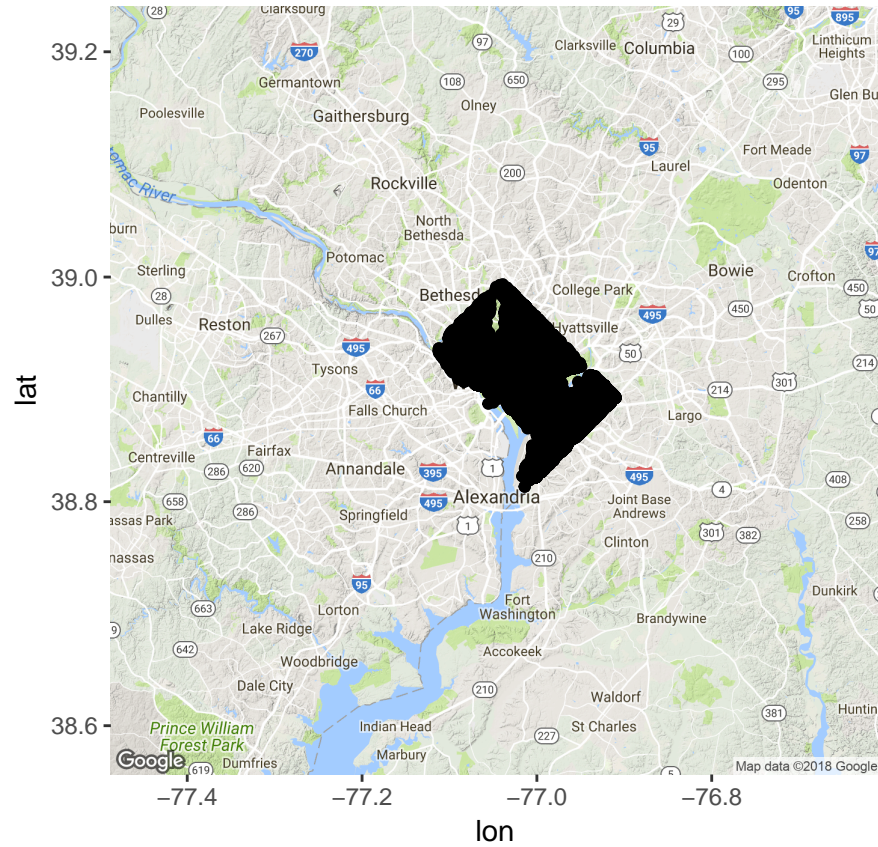
```



And now we do the combination of interest: crime points on top of the google map.

*# put the crime map on top of DC map we loaded before*

```
ggmap(dmv.map) +  
  geom_point(data = crime.2017.df, aes(x=LONGITUDE, y=LATITUDE))
```



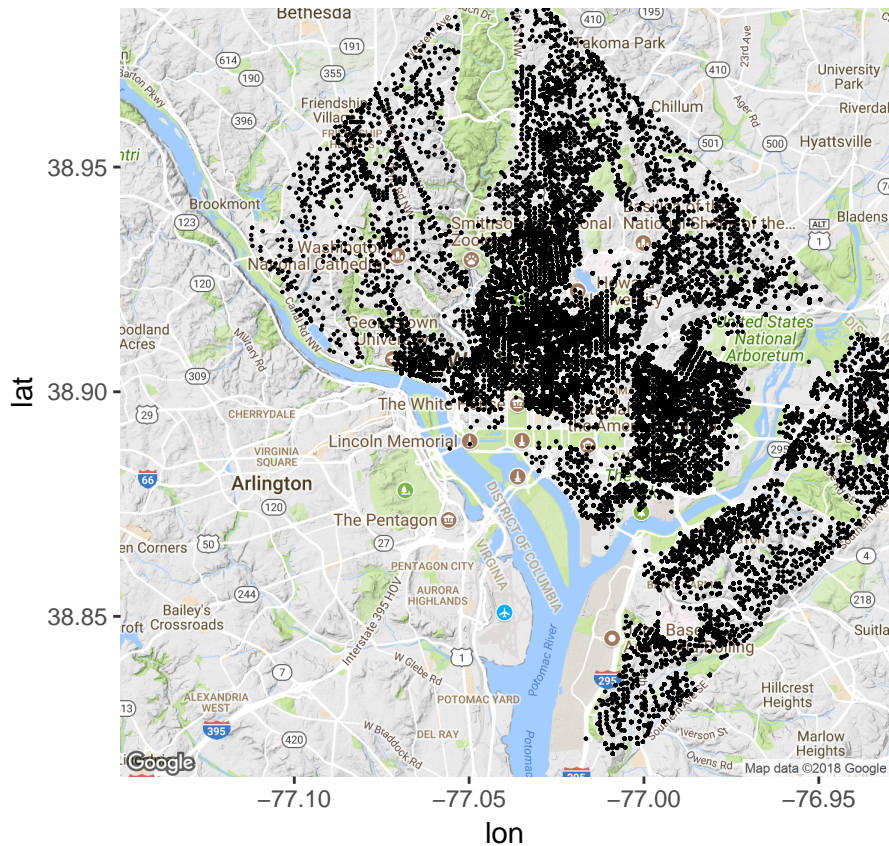
Rosa points out that this could look a lot better if we zoomed in and make the points smaller. Following her words of wisdom, we see

```
# get a new dc map
dmv.map2 <- 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

# plot the crime points on top of it
ggmap(dmv.map2) +
  geom_point(data = crime.2017.df, aes(x=LONGITUDE, y=LATITUDE), size = 0.1)

## Warning: Removed 1068 rows containing missing values (geom_point).
```



## B. Spatial analysis

Apart from simply drawing maps, R also allows you to do calculations based on location when you have properly entered spatial data. In this section, we'll learn how to find the ward for each 2017 crime. We'll then see how our calculation matches what the city did, and we'll plot where our work and the city's don't agree. Finally, we'll make a choropleth map showing crime intensity by ward.

### B.1. Clean up ward name in crime data

The 2017 crime data already come with a ward number in the data. Because we are going to calculate our own ward number, we are going to change the name of this original data, going from "WARD" to "org\_ward".



To do so, use the following code

```
names(crime.2017)[names(crime.2017) == "WARD"] <- "org_ward"
```

## B.2. Pull in ward map

To find the ward for each crime, we need a shapefile with wards. I went to [opendatac.gov](https://opendatac.gov) and downloaded the 2012 ward map, which is [here](#) and may not be perfect, but which I thought was sufficient for our purposes. To download, go to “download” at the upper right and choose “shapefile.”

Now load it into R:

```
ward.map <- readOGR(dsn = "H:/pppa_data_viz/2018/tutorials/lecture10/data/2018-03-25_Ward_from_2012",
                    layer = "Ward_from_2012")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "H:/pppa_data_viz/2018/tutorials/lecture10/data/2018-03-25_Ward_from_2012", layer: "Ward_from_2012"
## with 8 features
## It has 82 fields
## Integer64 fields read as strings: OBJECTID WARD POP_2000 POP_2010 POP_2011_2
```

## B.3. Now intersect the crime and ward map

And here’s where the spatial magic happens. We use the `intersect` function from the `raster` package to intersect these two shapefiles. We write `raster::intersect` so that R knows that we want the intersect command from the `raster` package and not some other package that might have an intersect command (others do).

What does the intersect command do? It basically asks, for each row in geometry A, does it have a match in geometry B? In our case, geometry A is a set of points (crimes), and geometry B is a set of polygons (wards). So for each point, we ask, is it in one of the polygons from set B? If yes, add the attributes of the polygon in which it falls to the shapefile.

The intersect command has the following (simplified here) syntax.

```
new.object <- raster::intersect(mapA, mapB)
```

The output is in the format of `mapA` – so if `mapA` is points and `mapB` is polygons, the intersect is points. If `mapA` is lines and `mapB` is points, `mapA` is lines, with one observation per line/point intersection.

To be clear about what’s happening in our intersection, I have R show us the names of variables/attributes in both files before we do the intersect, and the number of rows and attributes. Then, after the intersect, we look at the names in the intersected file. It has variable names from both files.

```
names(crime.2017)
```

```
## [1] "CCN"      "REPORT_DAT" "SHIFT"      "METHOD"      "OFFENSE"
## [6] "BLOCK"    "XBLOCK"     "YBLOCK"     "org_ward"    "ANC"
## [11] "DISTRICT" "PSA"        "NEIGHBORHO" "BLOCK_GROU"  "CENSUS_TRA"
## [16] "VOTING_PRE" "LATITUDE"   "LONGITUDE"  "BID"         "START_DATE"
## [21] "END_DATE"  "OBJECTID"
```

```
dim(crime.2017)
```

```
## [1] 33073 22
```

```
names(ward.map)
```

```
## [1] "OBJECTID" "WARD" "NAME" "REP_NAME" "WEB_URL"
## [6] "REP_PHONE" "REP_EMAIL" "REP_OFFICE" "WARD_ID" "LABEL"
## [11] "AREASQMI" "POP_2000" "POP_2010" "POP_2011_2" "POP_BLACK"
## [16] "POP_NATIVE" "POP_ASIAN" "POP_HAWAII" "POP_OTHER_" "TWO_OR_MOR"
## [21] "NOT_HISPAN" "HISPANIC_0" "POP_MALE" "POP_FEMALE" "AGE_0_5"
## [26] "AGE_5_9" "AGE_10_14" "AGE_15_17" "AGE_18_19" "AGE_20"
## [31] "AGE_21" "AGE_22_24" "AGE_25_29" "AGE_30_34" "AGE_35_39"
## [36] "AGE_40_44" "AGE_45_49" "AGE_50_54" "AGE_55_59" "AGE_60_61"
## [41] "AGE_65_66" "AGE_67_69" "AGE_70_74" "AGE_75_79" "AGE_80_84"
## [46] "AGE_85_PLU" "MEDIAN_AGE" "UNEMPLOYME" "TOTAL_HH" "FAMILY_HH"
## [51] "PCT_FAMILY" "NONFAMILY_" "PCT_NONFAM" "PCT_BELOW_" "PCT_BELO_1"
## [56] "PCT_BELO_2" "PCT_BELO_3" "PCT_BELO_4" "PCT_BELO_5" "PCT_BELO_6"
## [61] "PCT_BELO_7" "PCT_BELO_8" "POP_25_PLU" "POP_25_P_1" "POP_25_P_2"
## [66] "MARRIED_CO" "MALE_HH_NO" "FEMALE_HH_" "MEDIAN_HH_" "PER_CAPITA"
## [71] "PCT_BELO_9" "PCT_BELO10" "NO_DIPLOMA" "DIPLOMA_25" "NO_DEGREE_"
## [76] "ASSOC_DEGR" "BACH_DEGRE" "MED_VAL_00" "Shape_Leng" "Shape_Area"
## [81] "Shape_Le_1" "Shape_Ar_1"
```

```
dim(ward.map)
```

```
## [1] 8 82
```

```
crimes.wards <- raster::intersect(crime.2017,ward.map)
```

```
## Loading required namespace: rgeos
```

```
names(crimes.wards)
```

```
## [1] "CCN" "REPORT_DAT" "SHIFT" "METHOD" "OFFENSE"
## [6] "BLOCK" "XBLOCK" "YBLOCK" "org_ward" "ANC"
## [11] "DISTRICT" "PSA" "NEIGHBORHO" "BLOCK_GROU" "CENSUS_TRA"
## [16] "VOTING_PRE" "LATITUDE" "LONGITUDE" "BID" "START_DATE"
## [21] "END_DATE" "OBJECTID" "OBJECTID" "WARD" "NAME"
## [26] "REP_NAME" "WEB_URL" "REP_PHONE" "REP_EMAIL" "REP_OFFICE"
## [31] "WARD_ID" "LABEL" "AREASQMI" "POP_2000" "POP_2010"
## [36] "POP_2011_2" "POP_BLACK" "POP_NATIVE" "POP_ASIAN" "POP_HAWAII"
## [41] "POP_OTHER_" "TWO_OR_MOR" "NOT_HISPAN" "HISPANIC_0" "POP_MALE"
## [46] "POP_FEMALE" "AGE_0_5" "AGE_5_9" "AGE_10_14" "AGE_15_17"
## [51] "AGE_18_19" "AGE_20" "AGE_21" "AGE_22_24" "AGE_25_29"
## [56] "AGE_30_34" "AGE_35_39" "AGE_40_44" "AGE_45_49" "AGE_50_54"
## [61] "AGE_55_59" "AGE_60_61" "AGE_65_66" "AGE_67_69" "AGE_70_74"
## [66] "AGE_75_79" "AGE_80_84" "AGE_85_PLU" "MEDIAN_AGE" "UNEMPLOYME"
## [71] "TOTAL_HH" "FAMILY_HH" "PCT_FAMILY" "NONFAMILY_" "PCT_NONFAM"
## [76] "PCT_BELOW_" "PCT_BELO_1" "PCT_BELO_2" "PCT_BELO_3" "PCT_BELO_4"
## [81] "PCT_BELO_5" "PCT_BELO_6" "PCT_BELO_7" "PCT_BELO_8" "POP_25_PLU"
## [86] "POP_25_P_1" "POP_25_P_2" "MARRIED_CO" "MALE_HH_NO" "FEMALE_HH_"
## [91] "MEDIAN_HH_" "PER_CAPITA" "PCT_BELO_9" "PCT_BELO10" "NO_DIPLOMA"
## [96] "DIPLOMA_25" "NO_DEGREE_" "ASSOC_DEGR" "BACH_DEGRE" "MED_VAL_00"
## [101] "Shape_Leng" "Shape_Area" "Shape_Le_1" "Shape_Ar_1"
```

```
dim(crimes.wards)
```

```
## [1] 33073 104
```

Note that this new file has the attributes of both the original file and the ward file.

## B.4. Assess the result

Whenever you do a calculation, it is always wise to see if the results are as expected. We have already looked at the size and attributes of the file. In this case, we'll also look at whether the ward variable we calculated is the same as the one the city did when they shared the crime data file.

To do this, we'll create a new variable that takes the value 1 when these two measures of ward differ and zero otherwise.

```
# first: does our new ward variable have the same number of observations in each ward  
# as the original?  
table(crimes.wards$WARD)
```

```
##  
##      1      2      3      4      5      6      7      8  
## 4615 5953 1577 2889 4785 5526 4309 3419
```

```
table(crimes.wards$org_ward)
```

```
##  
##      1      2      3      4      5      6      7      8  
## 4624 5949 1577 2889 4794 5512 4310 3418
```

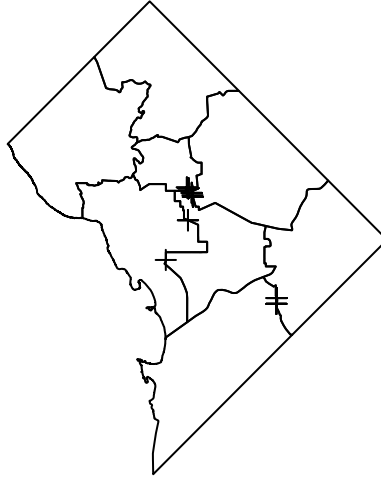
```
# create a new variable for instances of disagreement  
crimes.wards$do.not.agree <- ifelse(crimes.wards$WARD == crimes.wards$org_ward, 0, 1)
```

```
# how many instances of disagreement are there?  
table(crimes.wards$do.not.agree)
```

```
##  
##      0      1  
## 33040     33
```

So we see that for 33 of about 33,000 crimes we don't find the same ward as the DC data people do. Let's take a closer look to see where these differences are by making a plot, as we do below.

```
# look at the instances of disagreement  
plot(ward.map)  
plot(crimes.wards[crimes.wards$do.not.agree == 1,], add = TRUE)
```



It seems that the only disagreements are on the very borders of wards. This is heartening.

## B.5. Calculate total crimes by ward

Now that we have this new variable, what could we do with it? First, we'll calculate the total number of crimes by ward, and crimes per capita per ward. To do this, we need to extract the dataframe from the crimes shapefile, and use the `ddply` summarize command as we have before.

Note that the population variable `POP_2010` is an integer variable, which is a type of factor. First we need to make it a character (`as.character()`) and then into numeric (`as.numeric()`). If we try to make it a numeric variable directly, R will do it, but give us something that is garbage.

```
# get a dataframe from our new shapefile
crimes.wards.df <- data.frame(crimes.wards)

library("plyr")
# make the 2010 population a numeric variable (or everything else is garbage)
# need as.character, b/c integers are a type of factor, i think
crimes.wards.df$pop_2010_num <- as.numeric(as.character(crimes.wards.df$POP_2010))
is.numeric(crimes.wards.df$POP_2010)
```

```
## [1] FALSE
```

```
is.numeric(crimes.wards.df$pop_2010_num)
```

```
## [1] TRUE
```

```
summary(crimes.wards.df$POP_2010)

## 71748 73662 74308 74462 75773 76238 76645 78887
## 4309 3419 4785 4615 2889 5526 5953 1577

summary(crimes.wards.df$pop_2010_num)

##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
## 71748 74308 74462 75019 76238 78887

# make a ward-level dataset with crime totals
# we don't add up population (b/c that would be population*number of crimes)
# instead, we take the average of the ward population by ward -- which is the ward population
totals.by.wards <- ddply(crimes.wards.df, "WARD", "summarize",
                        crime.count = length(WARD),
                        pop = mean(pop_2010_num, na.rm = TRUE))

# again, make the integer value of crime.count into a numeric variable
totals.by.wards$crime.num <- as.numeric(as.character(totals.by.wards$crime.count))
typeof(totals.by.wards$crime.num)

## [1] "double"

summary(totals.by.wards$crime.num)

##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
## 1577 3286 4462 4134 4970 5953

# create a total crimes per capita measure
totals.by.wards$crime.pc <- totals.by.wards$crime.num / (totals.by.wards$pop / 1000)
totals.by.wards

##   WARD crime.count   pop crime.num crime.pc
## 1     1      4615 74462      4615 61.97792
## 2     2      5953 76645      5953 77.66978
## 3     3      1577 78887      1577 19.99062
## 4     4      2889 75773      2889 38.12704
## 5     5      4785 74308      4785 64.39414
## 6     6      5526 76238      5526 72.48354
## 7     7      4309 71748      4309 60.05742
## 8     8      3419 73662      3419 46.41471
```

## B.6. Make a map of crime intensity by ward

Finally, we'll make a map of crime intensity by ward. Remember that we couldn't do this with the original `crimes.2017` shapefile, which had point-level crime data.

To make a ward-level plot, we'll need to add our just-created ward-level measures to the ward spatial data. We can do this with the `merge` command. We then use the `spplot` command to plot the wards and color them in by crime intensity per capita and use the `plot` command to put a ward map on top.

You may need to install the `RColorBrewer` package for this section, unless you have already done so.

```
# add ward-level crime stats to the ward map
totals.by.wards

##   WARD crime.count   pop crime.num crime.pc
## 1     1      4615 74462      4615 61.97792
```

```
## 2      2      5953 76645      5953 77.66978
## 3      3      1577 78887      1577 19.99062
## 4      4      2889 75773      2889 38.12704
## 5      5      4785 74308      4785 64.39414
## 6      6      5526 76238      5526 72.48354
## 7      7      4309 71748      4309 60.05742
## 8      8      3419 73662      3419 46.41471
```

```
ward.map.w.crime <- merge(ward.map, totals.by.wards, by.x = "WARD", by.y = "WARD", all = TRUE)
```

```
# make colors for the wards
```

```
# see notes here
```

```
# http://www.nickeubank.com/wp-content/uploads/2015/10/RGIS3\_MakingMaps\_part1\_mappingVectorData.html
```

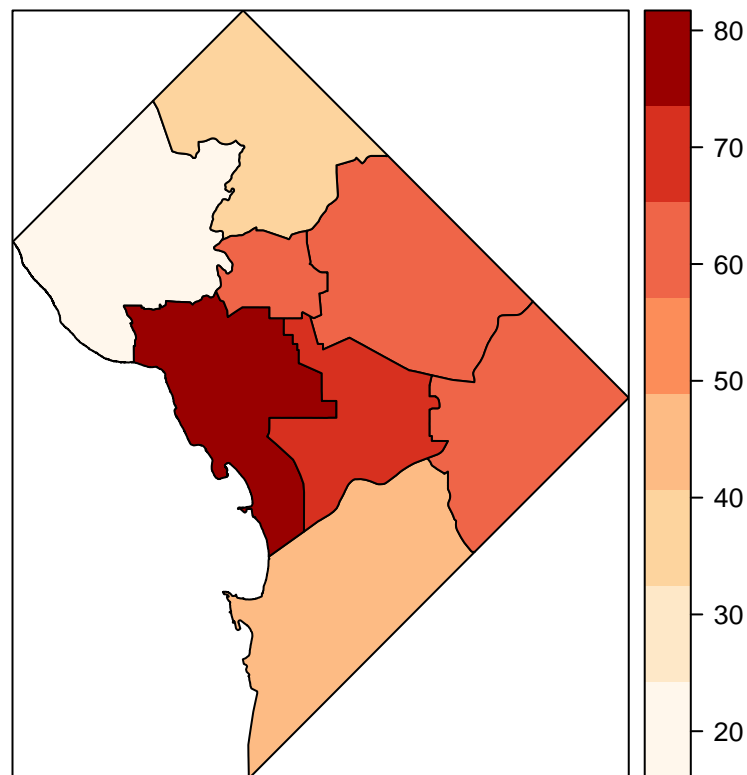
```
library(RColorBrewer)
```

```
# this chooses 8 colors from this palette
```

```
my.palette <- brewer.pal(n=8, name = "OrRd")
```

```
# make a choropleth map by crime intensity per capita showing ward borders
```

```
spplot(ward.map.w.crime, "crime.pc", col.regions = my.palette, cuts = 7)
```



## C. Homework

1. Make a map as in section B.6. for one specific type of crime. You'll need to subset the crime data to do this; we've done a similar subset command in previous tutorials.



2. Plot two other spatial files of your choosing, then make an intersection between these two spatial files of your choosing, and plot the intersection as well. As we did in class, do a little bit of checking on the results.

The intersect will not work if the two spatial files are not in the same projection. If you need to change projection of one of your files, you can try something like the below, where `mapA` and `mapB` are your two spatial datasets.

```
# find projection of each file
proj.A <- CRS(mapA)
proj.A
proj.B <- CRS(mapB)
proj.B

# change projection of mapB to be that of mapA
mapBA <- spTransform(mapB, proj.A)
```