Tutorial 5: Maps I

Leah Brooks

February 23, 2019

This is our first of two tutorials on maps, concentrating on the R package `sf`. This tutorial introduces digital maps. It then gives examples of how to present them, followed by some examples of the type of spatial analysis you can do with this package. In our second mapping class, we concentrate on choropleth maps.

As before, this tutorial begins with a very simple map and then works up to maps and techniques that are more complicated.

While making this tutorial, I consulted three online tutorials that I would recommend if you want more in-depth coverage of specific map issues. See

- The University of Chicago's Computing for the Social Sciences
- The authors of the the package, in-depth, but sometimes difficult to follow
- A random guy

A. Load packages

This week are are adding a new package to our repetoire: `sf`. This package is a quantum leap forward for mapping in R. It is fast and, relative to what was previously available, easy to use (you may not believe this after today's tutorial, but it is true). This package is also designed to work with `ggplot`, so today's lesson shows you how these packages integrate.

As a word of warning, `sf` is very new – not even officially released yet. As such, the online help is not as extensive as for `ggplot`, and every so often you run into odd errors, or things that you think the package should do but does not. Despite all this, we are learning it because I think it's the best for mapping and spatial analysis in R: comprehensive and fast.

All `sf` commands begin with `st_`, as you'll see in this tutorial. We introduce only a fraction of what `sf` can do in this tutorial, so look at the online help or discuss with me if you have more questions.

We also load `ggplot2` and `dplyr` – packages you've already installed. Here I use the `require()` command. This command checks if you have the package. If you do, it loads the package. If it doesn't it downloads the package and then loads the package.

```r
install.packages("sf", dependencies = TRUE)
```

```r
require(sf)
```

```
## Loading required package: sf

## Linking to GEOS 3.6.1, GDAL 2.2.0, proj.4 4.9.3
```

```r
require(ggplot2)
```

```
## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.4.4
```

```r
require(dplyr)
```

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

B. Load, explore and plot two shapefiles

We begin by loading a shapefile of wards in DC. There are 8 wards in DC (electoral districts for councilmembers – there are also 5 at-large councilmembers), so we expect this file to have 8 polygons. You can download this file from DC's Open Data site here. From the download menu, choose "shapefile" and then save in a location you'll remember.

B.1. Load

The `st_read()` command is specifically designed to work with shapefiles, which are files that contain spatial data. For more details on such files, see the lecture notes.

To load a shapefile into R with `sf`, you use the `st_read()` command. You tell R where the file is, and what type of file it is by the extension.

```
dc.wards <- st_read("H:/pppa_data_viz/2019/tutorial_data/lecture05/Ward_from_2012/Ward_from_2012.shp")
```

```
## Reading layer `Ward_from_2012' from data source `H:\pppa_data_viz\2019\tutorial_data\lecture05\Ward_
## Simple feature collection with 8 features and 82 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: -77.1198 ymin: 38.79164 xmax: -76.90915 ymax: 38.99597
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

Note that `sf` tells you a little about this file immediately upon loading. We can see that it is made up of polygons. R also tells us maximum and minimum latitude and longitude of the map and the projection number (more on this later).

B.2. Explore

We can also use all the standard data frame commands to learn about this file. Try

```
head(dc.wards)
```

```
## Simple feature collection with 6 features and 82 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: -77.08172 ymin: 38.79164 xmax: -76.90915 ymax: 38.9573
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
##   OBJECTID WARD   NAME          REP_NAME
## 1        1    8 Ward 8 Trayon White, Sr.
## 2        2    6 Ward 6     Charles Allen
## 3        3    7 Ward 7      Vincent Gray
## 4        4    2 Ward 2        Jack Evans
## 5        5    1 Ward 1    Brianne Nadeau
## 6        6    5 Ward 5   Kenyan McDuffie
##                                     WEB_URL      REP_PHONE
## 1 http://dccouncil.us/council/trayon-white-sr (202) 724-8045
## 2   http://dccouncil.us/council/charles-allen (202) 724-8072
## 3    http://dccouncil.us/council/vincent-gray (202) 724-8068
## 4      http://dccouncil.us/council/jack-evans (202) 724-8058
## 5  http://dccouncil.us/council/brianne-nadeau (202) 724-8181
## 6 http://dccouncil.us/council/kenyan-mcduffie (202) 724-8028
##               REP_EMAIL                                  REP_OFFICE
## 1    twhite@dccouncil.us 1350 Pennsylvania Ave, Suite 400, NW 20004
## 2    callen@dccouncil.us 1350 Pennsylvania Ave, Suite 406, NW 20004
## 3     vgray@dccouncil.us 1350 Pennsylvania Ave, Suite 406, NW 20004
## 4    jevans@dccouncil.us 1350 Pennsylvania Ave, Suite 106, NW 20004
## 5   bnadeau@dccouncil.us 1350 Pennsylvania Ave, Suite 102, NW 20004
## 6 kmcduffie@dccouncil.us 1350 Pennsylvania Ave, Suite 506, NW 20004
##   WARD_ID  LABEL  AREASQMI Shape_Leng Shape_Area POP_2000 POP_2010
## 1       8 Ward 8 11.937871   28714.07   30965852    74049    73662
## 2       6 Ward 6  6.221045   24157.98   16064917    70867    76238
## 3       7 Ward 7  8.809914   22345.23   22818183    69987    71748
## 4       2 Ward 2  8.684517   29545.80   22492798    63455    76645
## 5       1 Ward 1  2.535896   12925.38    6567941    71747    74462
## 6       5 Ward 5 10.390304   22893.40   26910761    71440    74308
##   POP_2011_2 POP_BLACK POP_NATIVE POP_ASIAN POP_HAWAII POP_OTHER_
## 1      81133     75259        110       310         12        711
## 2      84290     29909        295      3573         40       1233
## 3      73290     69005        219       225         17       1211
## 4      77645      6817        213      7640         30       2496
## 5      82859     25110        300      3509        111       6259
## 6      82049     57733        335      1622          9       3758
##   TWO_OR_MOR NOT_HISPAN HISPANIC_O POP_MALE POP_FEMALE AGE_0_5 AGE_5_9
## 1        872      79843       1290    35573      45560    7879    7061
## 2       2529      79000       5290    40411      43879    4779    2747
## 3        908      70987       2303    33916      39374    5230    4485
## 4       2875      69529       8116    39214      38431    2173    1110
## 5       2596      65755      17104    41368      41491    4733    2644
## 6       1915      75058       6991    38692      43357    5778    3972
##   AGE_10_14 AGE_15_17 AGE_18_19 AGE_20 AGE_21 AGE_22_24 AGE_25_29
## 1      5963      3596      2945   1800   1748      4070      6306
## 2      2235      1088      1370    794    724      4596     13427
## 3      4333      2944      2162   1347   1107      3170      5036
## 4       571       486      6200   3363   3485      6463     12614
## 5      1934      1223      2399   1613   1642      5487     15600
## 6      2654      2249      2817   1533   1531      3882      8514
##   AGE_30_34 AGE_35_39 AGE_40_44 AGE_45_49 AGE_50_54 AGE_55_59 AGE_60_61
## 1      5951      4617      4873      4429      4978      5001      1578
## 2     12512      8052      5474      4524      4350      4724      1830
## 3      5083      4154      5166      4794      5715      5272      1716
## 4     10475      6491      4175      3445      3413      3248      1447
## 5     12286      7378      5099      4784      3963      2618      1420
## 6      7439      5996      5228      4828      4905      5274      1736
##   AGE_65_66 AGE_67_69 AGE_70_74 AGE_75_79 AGE_80_84 AGE_85_PLU MEDIAN_AGE
## 1      1011      1211      1904      1026       634        432       29.3
## 2      1518      1596      2167      1522       816        958       33.9
## 3      1254      1680      2370      1719      1290       1142         37
## 4      1214      1325      1763       764       724        863       30.9
## 5       802      1208      1778       954       505        738       31.3
## 6      1108      1640      2361      2079      1947       2135       35.4
##   UNEMPLOYME TOTAL_HH FAMILY_HH       PCT_FAMILY NONFAMILY_
## 1       22.9    29470     17747 60.2205632846963      11723
## 2        6.3    40100     15110 37.6807980049875      24990
## 3       19.1    29266     15574 53.2153352012574      13692
## 4        3.7    38870      9071 23.3367635708773      29799
## 5        6.6    34907     12253 35.1018420374137      22654
## 6       14.1    31656     14893 47.0463735152894      16763
##         PCT_NONFAM PCT_BELOW_ PCT_BELO_1 PCT_BELO_2 PCT_BELO_3 PCT_BELO_4
## 1 39.7794367153037       37.7       35.3       10.8       38.8       54.1
## 2 62.3192019950125       12.5        9.6        4.5       25.7       13.9
## 3 46.7846647987426       27.2       23.6        8.9         28       12.8
## 4 76.6632364291227       13.4        4.7       10.1       33.2         26
## 5 64.8981579625863       13.5         11        7.7       23.3        8.7
## 6 52.9536264847106         19       13.5       12.5       20.4       19.1
##   PCT_BELO_5 PCT_BELO_6 PCT_BELO_7 PCT_BELO_8 POP_25_PLU POP_25_P_1
## 1       21.6          0       51.2       36.4       <NA>       1858
## 2       11.6          0       10.7        6.2       <NA>       1785
## 3        5.3          0       19.7         20       <NA>       2259
## 4       23.5          0       12.6       10.3       <NA>       1743
## 5        6.9          0       26.8       11.5       <NA>       4324
## 6       17.7          0       27.9       10.4       <NA>       2796
##   POP_25_P_2 MARRIED_CO MALE_HH_NO FEMALE_HH_ MEDIAN_HH_ PER_CAPITA
## 1       2614       <NA>       1886      11653      30910      17596
## 2      25882       <NA>        944       4157      94343      58354
## 3       3309       <NA>       1682       9499      39165      22917
## 4      28509       <NA>        458        754     100388      72388
## 5      22488       <NA>       1634       3133      82159      47982
## 6      11168       <NA>       1567       6726      57554      32449
##   PCT_BELO_9 PCT_BELO10 NO_DIPLOMA DIPLOMA_25 NO_DEGREE_ ASSOC_DEGR
## 1       36.3       10.8       5958      18736      10975       2149
## 2        6.3        4.5       3128       7079       6643       1852
## 3       22.9        8.9       6002      18683      10800       2569
## 4       13.5       10.1        988       2381       2980        940
## 5       17.9        7.7       3224       6984       5293       1258
## 6       21.6       12.5       5094      13788      11034       2196
##   BACH_DEGRE MED_VAL_OO Shape_Le_1 Shape_Ar_1
## 1       3781     229900   28714.07   30965852
## 2      19588     573200   24157.98   16064917
## 3       4890     238900   22345.23   22818183
## 4      16253     623500   29545.80   22492798
## 5      17613     542100   12925.38    6567941
## 6      11557     379800   22893.40   26910761
##                         geometry
## 1 POLYGON ((-76.97229 38.8728...
## 2 POLYGON ((-77.0179 38.9141,...
## 3 POLYGON ((-76.94186 38.9185...
## 4 POLYGON ((-77.04946 38.9199...
## 5 POLYGON ((-77.03523 38.9374...
## 6 POLYGON ((-76.99144 38.9573...
```

```
str(dc.wards)
```

```
## Classes 'sf' and 'data.frame':   8 obs. of  83 variables:
##  $ OBJECTID  : num  1 2 3 4 5 6 7 8
##  $ WARD      : num  8 6 7 2 1 5 3 4
##  $ NAME      : Factor w/ 8 levels "Ward 1","Ward 2",..: 8 6 7 2 1 5 3 4
##  $ REP_NAME  : Factor w/ 8 levels "Brandon T. Todd",..: 7 3 8 4 2 5 6 1
##  $ WEB_URL   : Factor w/ 8 levels "http://dccouncil.us/council/brandon-todd",..: 7 3 8 4 2 5 6 1
##  $ REP_PHONE : Factor w/ 8 levels "(202) 724-8028",..: 2 7 6 4 8 1 5 3
##  $ REP_EMAIL : Factor w/ 8 levels "bnadeau@dccouncil.us",..: 7 3 8 4 1 5 6 2
##  $ REP_OFFICE: Factor w/ 7 levels "1350 Pennsylvania Ave, Suite 102, NW 20004",..: 5 6 6 3 1 7 4 2
##  $ WARD_ID   : Factor w/ 8 levels "1","2","3","4",..: 8 6 7 2 1 5 3 4
##  $ LABEL     : Factor w/ 8 levels "Ward 1","Ward 2",..: 8 6 7 2 1 5 3 4
##  $ AREASQMI  : num  11.94 6.22 8.81 8.68 2.54 ...
##  $ Shape_Leng: num  28714 24158 22345 29546 12925 ...
##  $ Shape_Area: num  30965852 16064917 22818183 22492798 6567941 ...
##  $ POP_2000  : num  74049 70867 69987 63455 71747 ...
##  $ POP_2010  : num  73662 76238 71748 76645 74462 ...
##  $ POP_2011_2: num  81133 84290 73290 77645 82859 ...
##  $ POP_BLACK : Factor w/ 8 levels "25110","29909",..: 8 2 7 6 1 5 4 3
##  $ POP_NATIVE: Factor w/ 8 levels "110","163","213",..: 1 5 4 3 6 7 2 8
##  $ POP_ASIAN : Factor w/ 8 levels "1622","1755",..: 4 6 3 8 5 1 7 2
##  $ POP_HAWAII: Factor w/ 8 levels "0","111","12",..: 3 7 4 5 2 8 1 6
##  $ POP_OTHER_: Factor w/ 8 levels "1211","1233",..: 7 2 1 4 6 5 3 8
##  $ TWO_OR_MOR: Factor w/ 8 levels "1915","2287",..: 7 3 8 5 4 1 6 2
##  $ NOT_HISPAN: Factor w/ 8 levels "65755","66565",..: 8 7 4 3 1 6 5 2
##  $ HISPANIC_O: Factor w/ 8 levels "1290","16501",..: 1 5 4 7 3 6 8 2
##  $ POP_MALE  : Factor w/ 8 levels "33916","35573",..: 2 7 1 5 8 4 3 6
##  $ POP_FEMALE: Factor w/ 8 levels "38431","39374",..: 7 6 2 1 3 5 8 4
##  $ AGE_0_5   : Factor w/ 8 levels "2173","4259",..: 8 4 5 1 3 7 2 6
##  $ AGE_5_9   : Factor w/ 8 levels "1110","2644",..: 8 3 7 1 2 4 5 6
##  $ AGE_10_14 : Factor w/ 8 levels "1934","2235",..: 8 2 6 7 1 3 4 5
##  $ AGE_15_17 : Factor w/ 8 levels "1088","1223",..: 7 1 6 8 2 4 3 5
##  $ AGE_18_19 : Factor w/ 8 levels "1370","1379",..: 6 1 3 8 4 5 7 2
##  $ AGE_20    : Factor w/ 8 levels "1347","1497",..: 5 8 1 6 4 3 2 7
##  $ AGE_21    : Factor w/ 8 levels "1107","1499",..: 5 8 1 6 4 3 2 7
##  $ AGE_22_24 : Factor w/ 8 levels "2893","3170",..: 5 6 2 8 7 4 3 1
##  $ AGE_25_29 : Factor w/ 8 levels "12614","13427",..: 5 2 4 1 3 7 8 6
##  $ AGE_30_34 : Factor w/ 8 levels "10475","12286",..: 5 3 4 1 2 8 7 6
##  $ AGE_35_39 : Factor w/ 8 levels "4154","4617",..: 2 8 1 6 7 4 5 3
##  $ AGE_40_44 : Factor w/ 8 levels "4175","4873",..: 2 6 4 1 3 5 7 8
##  $ AGE_45_49 : Factor w/ 8 levels "3445","4429",..: 2 3 5 1 4 6 7 8
##  $ AGE_50_54 : Factor w/ 8 levels "3413","3963",..: 5 3 7 1 2 4 6 8
##  $ AGE_55_59 : Factor w/ 8 levels "2618","3248",..: 5 4 6 2 1 7 3 8
##  $ AGE_60_61 : Factor w/ 8 levels "1420","1447",..: 3 6 4 2 1 5 7 8
##  $ AGE_65_66 : Factor w/ 8 levels "1011","1108",..: 1 6 4 3 8 2 7 5
##  $ AGE_67_69 : Factor w/ 8 levels "1208","1211",..: 2 4 6 3 1 5 8 7
##  $ AGE_70_74 : Factor w/ 8 levels "1763","1778",..: 3 4 7 1 2 6 8 5
##  $ AGE_75_79 : Factor w/ 8 levels "1026","1522",..: 1 2 3 7 8 5 6 4
##  $ AGE_80_84 : Factor w/ 8 levels "1290","1349",..: 6 8 1 7 5 4 2 3
##  $ AGE_85_PLU: Factor w/ 8 levels "1142","1837",..: 5 8 1 7 6 3 2 4
##  $ MEDIAN_AGE: Factor w/ 7 levels "29.3","30.9",..: 1 4 6 2 3 5 6 7
##  $ UNEMPLOYME: Factor w/ 7 levels "14.1","19.1",..: 3 5 2 4 6 1 4 7
##  $ TOTAL_HH  : Factor w/ 8 levels "29266","29470",..: 2 8 1 7 5 4 6 3
##  $ FAMILY_HH : Factor w/ 8 levels "12253","14893",..: 7 3 4 8 1 2 5 6
##  $ PCT_FAMILY: Factor w/ 8 levels "23.3367635708773",..: 8 3 6 1 2 5 4 7
##  $ NONFAMILY_: Factor w/ 8 levels "11723","12840",..: 1 7 3 8 6 4 5 2
##  $ PCT_NONFAM: Factor w/ 8 levels "39.7794367153037",..: 1 6 3 8 7 4 5 2
##  $ PCT_BELOW_: Factor w/ 8 levels "11.9","12.5",..: 7 2 6 3 4 5 8 1
##  $ PCT_BELO_1: Factor w/ 8 levels "1.9","11","13.5",..: 5 8 4 6 2 3 1 7
##  $ PCT_BELO_2: Factor w/ 8 levels "10.1","10.8",..: 2 4 8 1 7 3 6 5
##  $ PCT_BELO_3: Factor w/ 8 levels "14.6","20.4",..: 8 5 6 7 4 2 3 1
##  $ PCT_BELO_4: Factor w/ 8 levels "0","12.8","13.9",..: 6 3 2 5 8 4 1 7
##  $ PCT_BELO_5: Factor w/ 8 levels "10.3","11.6",..: 5 2 7 6 8 4 3 1
##  $ PCT_BELO_6: Factor w/ 2 levels "0","29": 1 1 1 1 1 1 1 2
##  $ PCT_BELO_7: Factor w/ 8 levels "10.7","12.6",..: 8 1 4 2 5 7 6 3
##  $ PCT_BELO_8: Factor w/ 8 levels "10.3","10.4",..: 6 7 5 1 4 2 8 3
##  $ POP_25_PLU: Factor w/ 0 levels: NA NA NA NA NA NA NA NA
##  $ POP_25_P_1: Factor w/ 8 levels "1743","1785",..: 3 2 4 1 7 5 8 6
##  $ POP_25_P_2: Factor w/ 8 levels "11168","15399",..: 5 4 8 6 3 1 7 2
##  $ MARRIED_CO: Factor w/ 0 levels: NA NA NA NA NA NA NA NA
##  $ MALE_HH_NO: Factor w/ 8 levels "1567","1591",..: 5 8 4 6 3 1 7 2
##  $ FEMALE_HH_: Factor w/ 8 levels "11653","1511",..: 1 4 8 7 3 6 2 5
##  $ MEDIAN_HH_: Factor w/ 8 levels "100388","112873",..: 3 8 4 1 7 5 2 6
##  $ PER_CAPITA: Factor w/ 8 levels "17596","22917",..: 1 6 2 7 5 3 8 4
##  $ PCT_BELO_9: Factor w/ 8 levels "13.5","13.7",..: 7 8 6 1 4 5 2 3
##  $ PCT_BELO10: Factor w/ 8 levels "10.1","10.8",..: 2 4 8 1 7 3 6 5
##  $ NO_DIPLOMA: Factor w/ 8 levels "3128","3224",..: 5 1 6 8 2 4 7 3
##  $ DIPLOMA_25: Factor w/ 8 levels "11907","13788",..: 4 8 3 6 7 2 5 1
##  $ NO_DEGREE_: Factor w/ 8 levels "10500","10800",..: 3 8 2 5 7 4 6 1
##  $ ASSOC_DEGR: Factor w/ 8 levels "1003","1258",..: 5 3 7 8 2 6 1 4
##  $ BACH_DEGRE: Factor w/ 8 levels "11557","13032",..: 7 6 8 3 4 1 5 2
##  $ MED_VAL_00: Factor w/ 8 levels "229900","238900",..: 1 6 2 7 5 3 8 4
##  $ Shape_Le_1: num  28714 24158 22345 29546 12925 ...
##  $ Shape_Ar_1: num  30965852 16064917 22818183 22492798 6567941 ...
##  $ geometry  :sfc_POLYGON of length 8; first list element: List of 1
##   ..$ : num [1:3792, 1:2] -77 -77 -77 -77 -77 ...
##   ..- attr(*, "class")= chr  "XY" "POLYGON" "sfg"
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA NA NA NA NA ...
##   ..- attr(*, "names")= chr  "OBJECTID" "WARD" "NAME" "REP_NAME" ...
```

A key attribute of a shapefile is its projection (for more on projections, see lecture notes). To find the projection of a `sf` file, use the `st_crs()` command. This reports the projection of the file.

The EPSG code is a numeric reference for the projection of the file. “EPSG” stands for the European Petroleum Survey Group, which first made this numeric system of reference. They have since been absorbed by the International Association of Oil and Gas Producers (IOGP). The IOGP maintains a reigstry of projections and associated codes that are very useful to anyone using spatial data.

```
# make sure it has a projection
st_crs(dc.wards)
```

```
## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"
```

It’s useful to know the projection of a file if

- you'd like to do some analysis with that file and another shapefile – they should be in the same projection
- to know whether you've got the desired projection. For example, some US projections give a flat top to the US, and others give a curved top.

B.3. Plot

There are two major ways to make a map from a `sf` file. You can use the `plot` command or `ggplot`. I'll give a brief example of plot; the rest of the tutorial and class will focus on methods associated with `ggplot`.

The `plot` command is R's basic plotting command, and it takes all kinds of inputs, not just simple features files. If you use `plot` with a simple features file, it will make a choropleth map of all the variables in the dataframe. This is almost always a bad idea. To show just the outlines of the polygons in this file, you need to tell R to just use the polygons. You do this by writing `st_geometry(SF FILE)`.

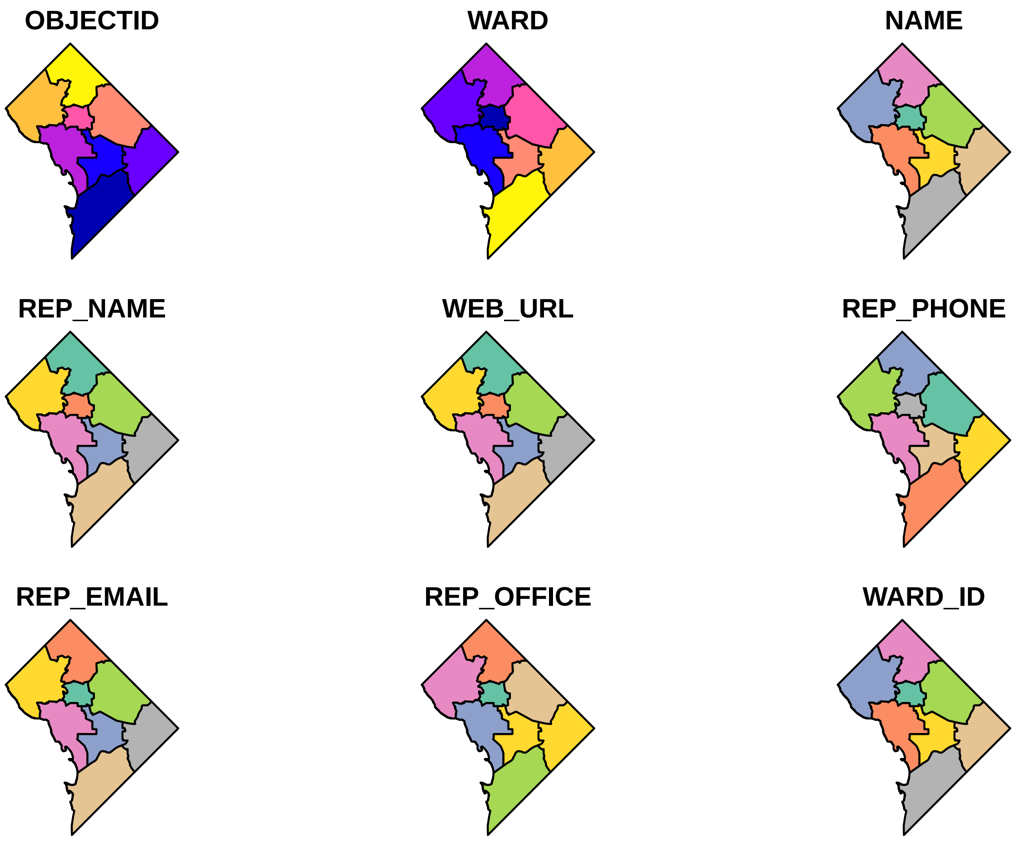
Try both of these:

```
# plot all variables
plot(dc.wards)
```

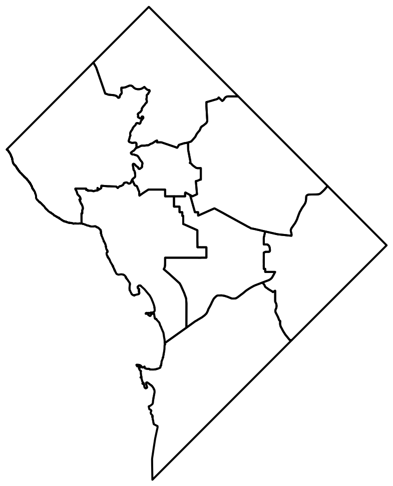
```
## Warning: plotting the first 9 out of 82 attributes; use max.plot = 82 to
## plot all

## Warning in classInt::classIntervals(na.omit(values), min(nbreaks, n.unq), :
## n same as number of different finite values\neach different finite value is
## a separate class

## Warning in classInt::classIntervals(na.omit(values), min(nbreaks, n.unq), :
## n same as number of different finite values\neach different finite value is
## a separate class
```



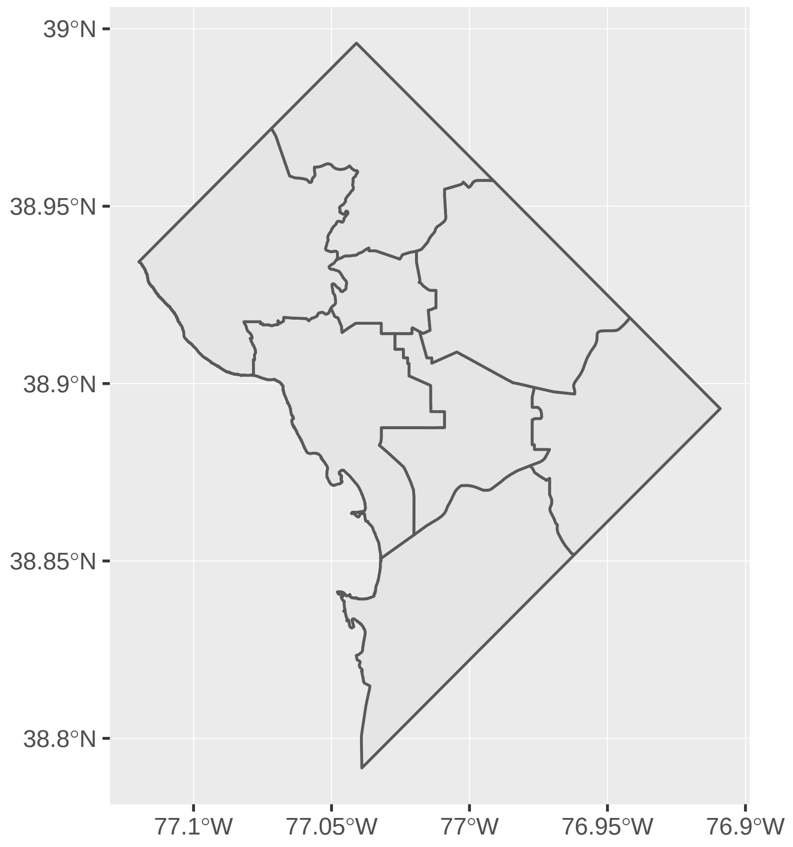
```r
# plot just the polygons
plot(st_geometry(dc.wards))
```



Rather than `plot`, we will stick in this class to plotting via `ggplot`. What we do here is very similar to what we've already done. Instead of `geom_bar()` or `geom_hist()`, we now rely on `geom_sf()`. This command takes a simple feature as the data input and plots the polygons.

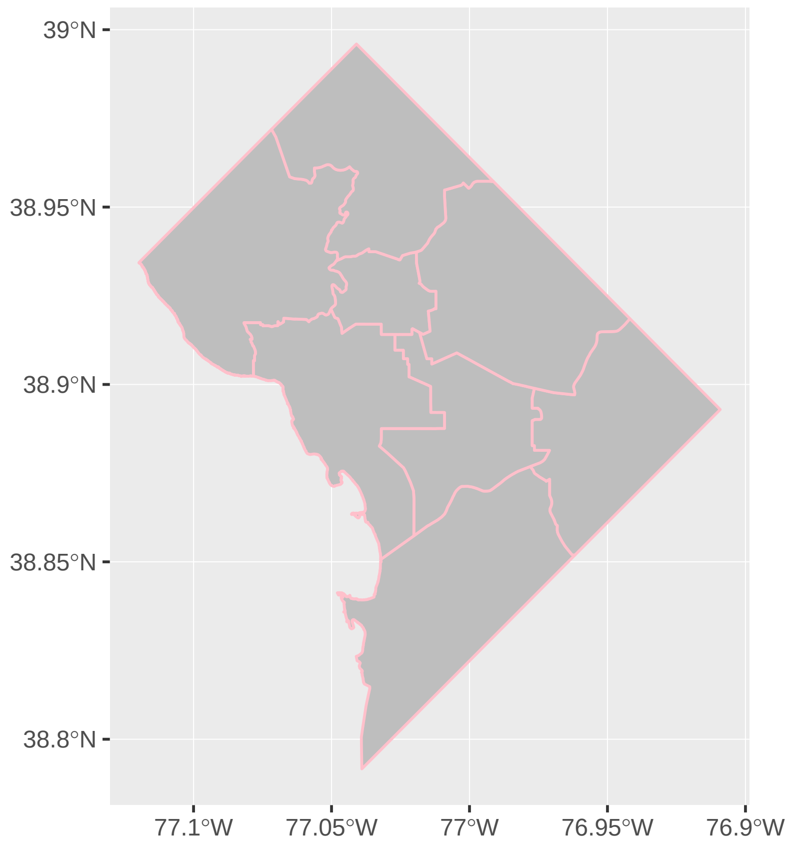
The below command is the simplest way to make a map from a "simple feature." As I did in previous tutorials with graphs, I create an object and then call it to see it. You could simply use `ggplot` without creating an object (though you wouldn't be able to save it later if you wanted).

```
# two ways to plot -- second
ward.map <- ggplot() +
               geom_sf(data = dc.wards)
ward.map
```



So the map above is decent, but you might want to change colors or shading. Some basic modifications include the color of the polygon lines and the polygon fill. Below I set the color of the lines to pink and the fill to grey; these commands are both inside the `geom_sf()` command, like graph options are inside `geom_bar()` or `geom_histogram()`.

```r
# with color and fill
ward.map <- ggplot() +
  geom_sf(data = dc.wards, color = "pink", fill = "grey")
ward.map
```



Usually we have maps to identify things. It would be nice to not just see the wards, but have some label for which ward is which. To do this in `ggplot`, we need to be able to tell R where to put the labels. Remember that a map is sort of like a graph, but where the coordinates are latitude and longitude.

Of course, the current map is polygons – it has the location of the points that are the boundary of the polygon, but it doesn't have the latitude and longitude of the center of the polygon, which is where we'd like to put the label.

So the first step is to find the latitude and longitude of the center of the polygon. To do this, we use two commands. The inner command is `st_centroid()`, which you could use in a plot command. For example, `plot(st_centroid(dc.wards))` would allow you to make a map of the centroids. However, we want to know the coordiantes of the centroids, not just map them, so we have to use `st_coordinates()` to get the coordinates our of the file.

The command `st_coordinates(st_centroid(dc.wards))` yields a matrix with two columns: `X` and `Y`, as you can see below.

```r
# get coordinates
dc.coords <- st_coordinates(st_centroid(dc.wards))
```

```
## Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
## of_largest_polygon): st_centroid does not give correct centroids for
## longitude/latitude data
```

```r
dc.coords
```

```
##           X        Y
## 1 -77.00659 38.84021
## 2 -77.00277 38.88681
## 3 -76.94784 38.88699
## 4 -77.04330 38.89323
## 5 -77.03142 38.92553
## 6 -76.98548 38.92544
## 7 -77.07899 38.93637
## 8 -77.03415 38.96384
```

Of course, to plot these, we need to put them into the shapefile. I am next going to do something that I would encourage you never ever to do in any other circumstances: put files together without merging. This is generally bad practice because it relies on the files being in the same order – which could change without you knowing. The best would be to add the ward number to the `dc.coords` file, but I couldn't figure out how to do this!

To put the `dc.coords` matrix together with the `dc.wards` shapefile, I just `cbind()`, which basically glues two dataframes together. The `c` in `cbind` is for column, and the command "binds" two sets of columns together. It requires (at least) two dataframes. Because `dc.coords` is a matrix and not a dataframe (a matrix allows fewer ooptions than a dataframe), you need to convert `dc.coords` into a dataframe, which you can do with `as.data.frame()`.

I check that my new dataframe has the additional `X` and `Y` columns by looking at the names. Note that this is still a polygon map – it just has the centroid of the polygon attached.

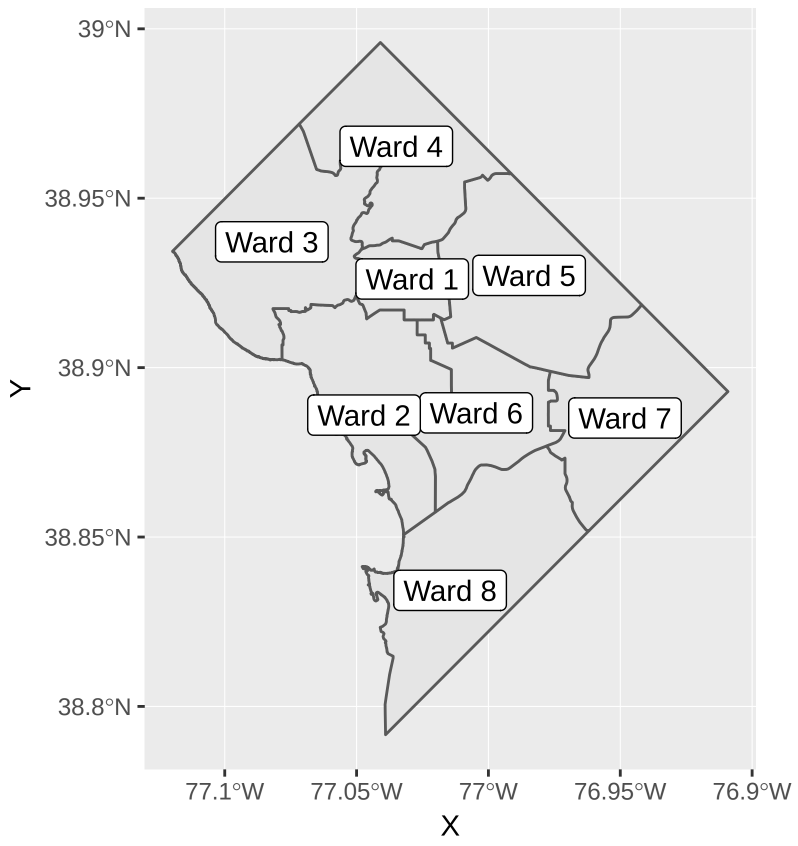
```r
# put coordinates together with map
dc.wards2 <- cbind(dc.wards,dc.coords)
names(dc.wards2)
```

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

Now we can modify the plot to add these labels using `geom_sf_label()`. This command uses a dataframe (we'll use the new one `dc.wards2` that we just created), and three key parts: where to put the label (X and Y), and what to put as the label (we use `NAME`).

```r
# plot
ward.map <- ggplot() +
  geom_sf(data = dc.wards) +
  geom_sf_label(data = dc.wards2, aes(x=X, y=Y, label=NAME))
ward.map
```

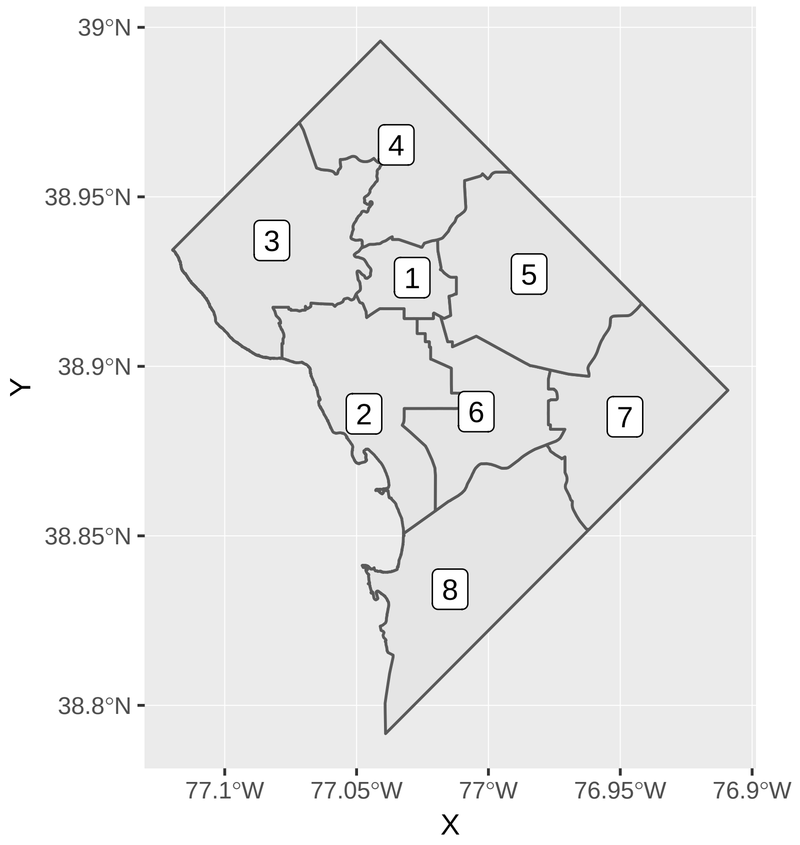
```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may
## not give correct results for longitude/latitude data
```



That looks ok, but you might prefer just the ward number, especially if your title makes it clear that these are all wards. You can change what the label says by changing the variable you associate with the label. Below we label by `WARD`, which is a number for the ward.

```r
# or you can do just the number
ward.map <- ggplot() +
  geom_sf(data = dc.wards) +
  geom_sf_label(data = dc.wards2, aes(x=X, y=Y, label=WARD))
ward.map
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may
## not give correct results for longitude/latitude data
```



As an aside, Jill points out that this code works just as well. It doesn't require the `cbind()` command, but it does rely on the rows in the coordinates file being in the order as the rows in the `dc.wards` file.

```r
dc.coords.df <- as.data.frame(dc.coords)
dc.coords.df

ward.map <- ggplot() +
  geom_sf(data = dc.wards) +
  geom_text(data = dc.coords.df, aes(x = X, y = Y,
                                      label = dc.wards$WARD_ID))
ward.map
```

B.4. Plot two things

One of the most powerful things about `sf` together with `ggplot` is your ability to plot mutliple features together. To illustrate this, download the location of DC public school administration here. Again, save as a shapefile somewhere you'll remember. This is a *points* datafile, different from the polygons file we were working with.

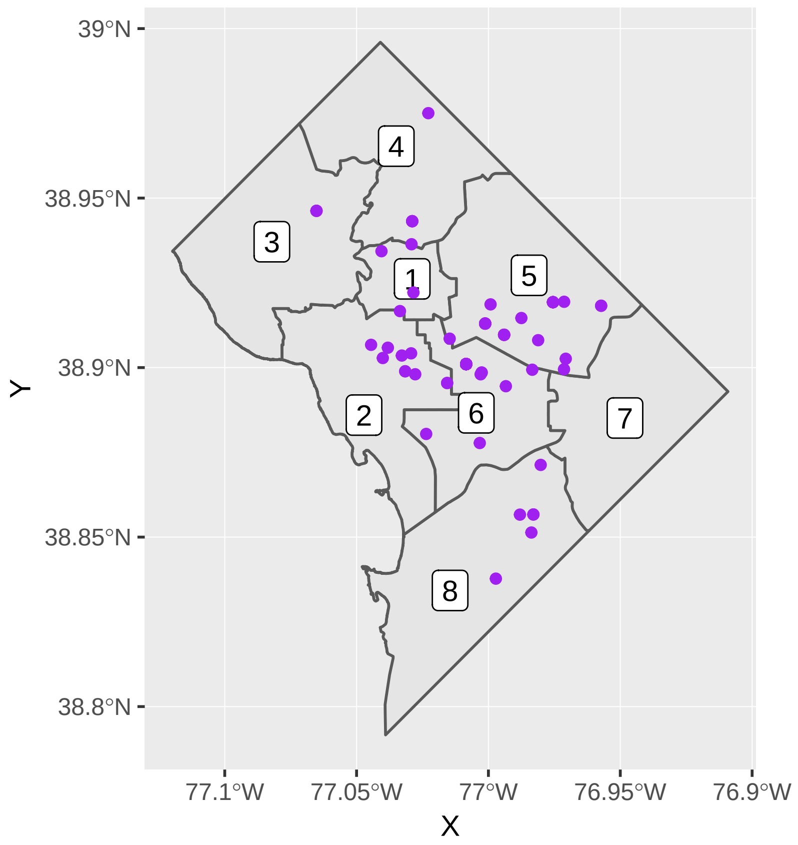
Below you add the data, using `st_read()` and then add them to the previous map with an additional `geom_sf()` command, making purple points. Note that you don't need separate plotting commands for diffent map types. R can figure out which is which.

```
# load some points
dcps.pts <- st_read("H:/pppa_data_viz/2019/tutorial_data/lecture05/Public_School_Administration_Points/P
```

```
## Reading layer `Public_School_Administration_Points' from data source `H:\pppa_data_viz\2019\tutorial_
## Simple feature collection with 163 features and 8 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -77.0652 ymin: 38.83775 xmax: -76.95726 ymax: 38.97505
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

```
# plot them
ward.map <- ggplot() +
  geom_sf(data = dc.wards) +
  geom_sf_label(data = dc.wards2, aes(x=X, y=Y, label=WARD)) +
  geom_sf(data = dcps.pts, color = "purple")
ward.map
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may
## not give correct results for longitude/latitude data
```



C. Merging in data

The final task with this small simple feature is to merge in an additional dataset. We do this to show the value of simple features: they work like regular old dataframes, but also have additional spatial information.

We'll merge in additional data about wireline adoption by ward (wireline is the old home phone). Download these data from here. Make sure to download the "spreadsheet," not the shapefile.

We can use `read.csv()` to load these data.

```
# read the new data
landline <- read.csv("H:/pppa_data_viz/2019/tutorial_data/lecture05/Residential_Wireline_Adoption_Rate_
landline
```

```
##   ï..OBJECTID WARD_ID  LABEL JUN2012  DEC2012  JUN2013  DEC2013
## 1           1       8 Ward 8    55.9 58.85759 60.98185 58.42421
## 2           2       6 Ward 6    73.7 77.10824 80.14122 82.52427
## 3           3       7 Ward 7    53.7 55.28880 57.39871 55.13108
## 4           4       2 Ward 2    82.9 87.24695 87.90285 85.95241
## 5           5       1 Ward 1    74.6 76.95666 79.61574 79.07470
## 6           6       5 Ward 5    62.0 63.08352 65.07573 65.55988
## 7           7       3 Ward 3    88.3 93.56102 93.39461 88.00533
## 8           8       4 Ward 4    78.1 81.90963 83.48707 82.35078
```

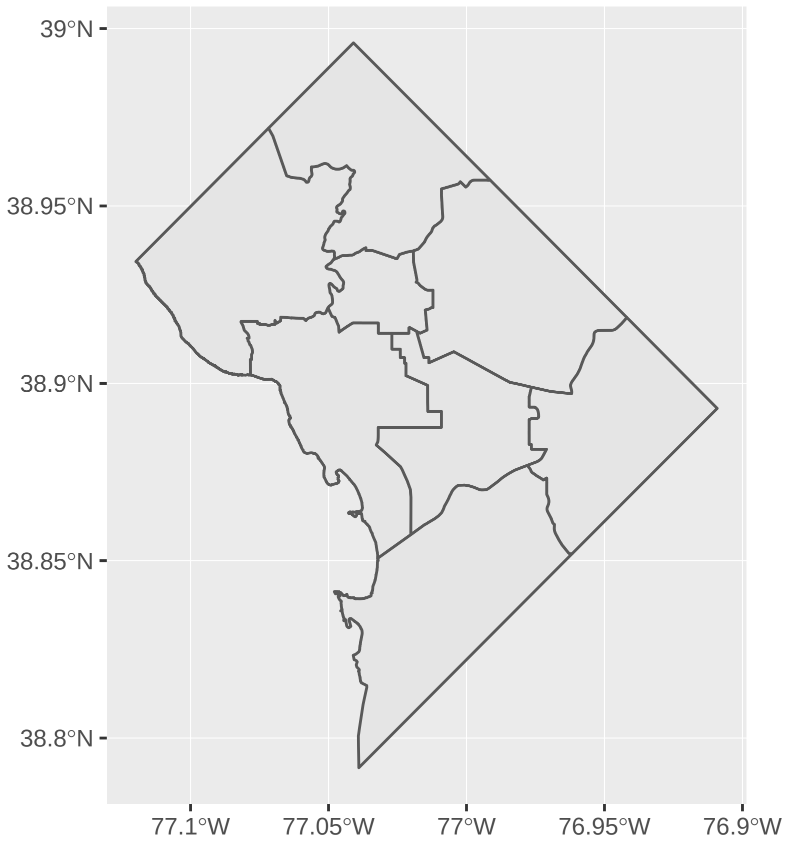
We then use `merge()` to put the simple feature (`dc.wards2`) together with the landline data. You've seen this command before; note here how the two inputs have different names for the same variable, which is why we use `by.x` and `by.y`. I keep all observations from both dataframes and then check the total number of observations (how many should it be?).

```r
# merge it in
dc.wards3 <- merge(x = dc.wards2, y = landline, by.x = "WARD", by.y = "WARD_ID", all = TRUE)
dim(dc.wards3)
```

```
## [1]  8 91
```

Now we can map these data. First we make the basic map:

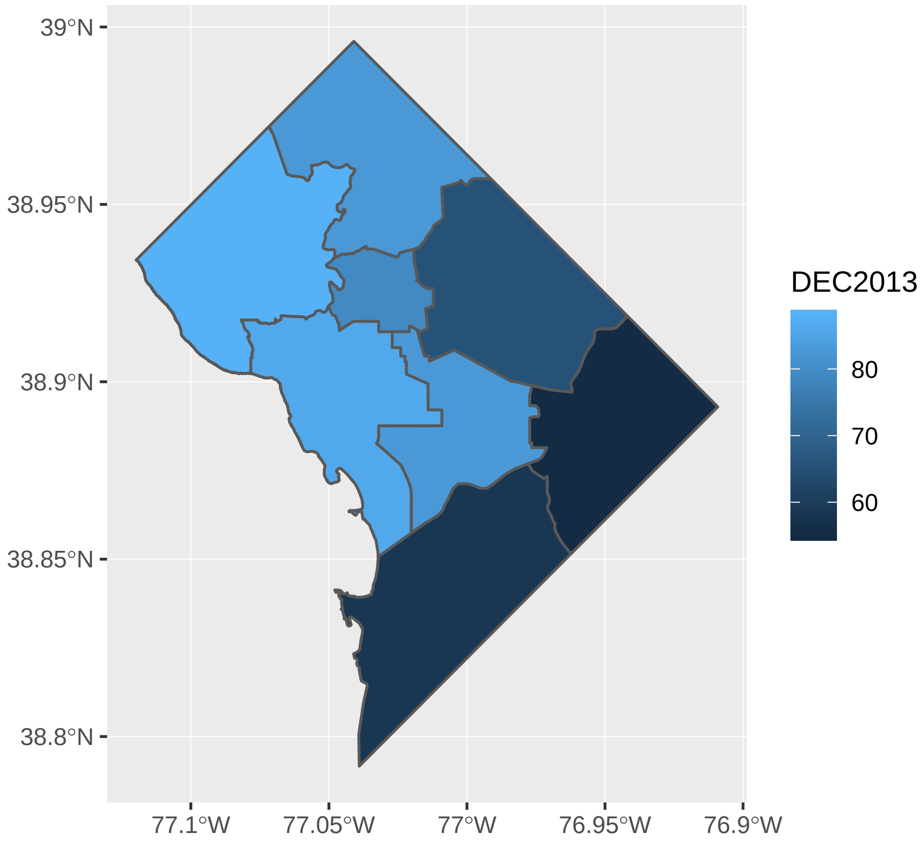
```
# make a basic map
landline.map <- ggplot() +
  geom_sf(data = dc.wards3)
landline.map
```



And now we show the values of landline penetration (why we went to all this trouble). This is a choropleth map, and we'll spend the second mapping tutorial working on these types of maps. In brief, we "fill" the polygons by the value of landline penetration; I use the latest values, as of December 2013.

There are many things not to like about the presentation of this map. Hold your list of concerns until the next mapping class.

```
# show values of landline penetration
landline.map <- ggplot() +
  geom_sf(data = dc.wards3, aes(fill = DEC2013))
landline.map
```



D. Use bigger data

Now we are going to use a larger dataset and try to make a presentation-quality map.

D.1. Load crime data

For this portion of the tutorial, we'll use DC crime data. Download 2018 DC crime data as a shapefile from here; then use `st_read()` to read in the file.

```
# load data
c2018 <- st_read("H:/pppa_data_viz/2019/tutorial_data/lecture05/Crime_Incidents_in_2018/Crime_Incidents_
```

```
## Reading layer `Crime_Incidents_in_2018' from data source `H:\pppa_data_viz\2019\tutorial_data\lectur
## Simple feature collection with 33645 features and 23 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -77.11232 ymin: 38.81467 xmax: -76.91002 ymax: 38.9937
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

It is always a good idea to do some quick checks on data quality. The note on DC Open data's page about missing values made me nervous, but using `summary()` to look at the geographic information suggests it is ok.

```r
# look at variables
str(c2018)
```

```
## Classes 'sf' and 'data.frame':   33645 obs. of  24 variables:
##  $ CCN       : Factor w/ 33638 levels "07006630","10954295",..: 1 2 3 9 8 12553 12554 12555 12556 125
##  $ REPORT_DAT: Factor w/ 33597 levels "2018-01-01T00:04:32.000Z",..: 17438 7411 18001 492 11323 1288
##  $ SHIFT     : Factor w/ 3 levels "DAY","EVENING",..: 3 2 3 3 1 3 3 3 3 3 ...
##  $ METHOD    : Factor w/ 3 levels "GUN","KNIFE",..: 1 3 3 1 3 3 3 1 3 3 ...
##  $ OFFENSE   : Factor w/ 9 levels "ARSON","ASSAULT W/DANGEROUS WEAPON",..: 4 7 4 4 8 9 5 2 9 8 ...
##  $ BLOCK     : Factor w/ 7119 levels "0 - 0 BLOCK OF COLUMBUS CIRCLE NE",..: 3779 4432 1880 3114 317
##  $ XBLOCK    : num  402412 393499 401872 400393 400863 ...
##  $ YBLOCK    : num  131645 138307 136822 132326 132294 ...
##  $ WARD      : Factor w/ 8 levels "1","2","3","4",..: 8 2 6 8 8 8 5 2 7 6 ...
##  $ ANC       : Factor w/ 40 levels "1A","1B","1C",..: 37 9 26 38 36 38 23 6 33 30 ...
##  $ DISTRICT  : Factor w/ 7 levels "1","2","3","4",..: 7 2 5 7 7 7 5 3 5 1 ...
##  $ PSA       : Factor w/ 57 levels "101","102","103",..: 51 14 41 52 52 56 37 18 41 1 ...
##  $ NEIGHBORHO: Factor w/ 39 levels "Cluster 1","Cluster 10",..: 30 34 16 31 31 33 17 36 18 38 ...
##  $ BLOCK_GROU: Factor w/ 449 levels "000100 1","000100 2",..: 261 5 311 252 257 430 447 210 315 190
##  $ CENSUS_TRA: Factor w/ 179 levels "000100","000201",..: 99 2 118 94 97 172 179 75 119 66 ...
##  $ VOTING_PRE: Factor w/ 143 levels "Precinct 1","Precinct 10",..: 19 100 124 23 22 28 113 46 123 1
##  $ LATITUDE  : num  38.9 38.9 38.9 38.9 38.9 ...
##  $ LONGITUDE : num  -77 -77.1 -77 -77 -77 ...
##  $ BID       : Factor w/ 10 levels "ADAMS MORGAN",..: NA NA NA 2 NA NA NA NA NA 8 ...
##  $ START_DATE: Factor w/ 33517 levels "1985-01-11T03:56:21.000Z",..: 5 7166 8 29 11459 13047 13054 1
##  $ END_DATE  : Factor w/ 27256 levels "1985-01-11T04:00:39.000Z",..: 4 6688 7 NA NA 10770 NA 10773 N
##  $ OBJECTID  : num  2.59e+08 2.59e+08 2.59e+08 2.59e+08 2.59e+08 ...
##  $ OCTO_RECOR: Factor w/ 33645 levels "07006630-01",..: 1 2 3 9 8 12559 12560 12561 12562 12563 ...
##  $ geometry  :sfc_POINT of length 33645; first list element: Classes 'XY', 'POINT', 'sfg'  num [1:2]
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA NA NA NA NA ...
##   ..- attr(*, "names")= chr  "CCN" "REPORT_DAT" "SHIFT" "METHOD" ...
```

```r
# size and missings
dim(c2018)
```

```
## [1] 33645    24
```

```r
summary(c2018$LATITUDE)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   38.81   38.89   38.91   38.91   38.92   38.99
```

```r
summary(c2018$LONGITUDE)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -77.11  -77.03  -77.01  -77.01  -76.99  -76.91
```

```
table(c2018$OFFENSE)
```

```
##
##                    ARSON ASSAULT W/DANGEROUS WEAPON
##                        5                       1668
##                 BURGLARY                   HOMICIDE
##                     1415                        160
##      MOTOR VEHICLE THEFT                    ROBBERY
##                     2388                       2019
##                SEX ABUSE               THEFT F/AUTO
##                      275                      11574
##              THEFT/OTHER
##                    14141
```

```
# nothing seems to have NAs
```

D.2. Making legible maps from these data

There are so many crimes in these data that a picture that maps them all is not a good idea. So we make a marker for the violent crimes (arson, assault, homicide, robbery and sex abuse) only. We use a `ifelse()` command to discriminate between the two types. After creating the new variable `ctype`, I use the `table()` command to check whether I've put the right crimes in each group. In this first command, I have not!

```
# wrong!
c2018$ctype <- ifelse(c2018$OFFENSE %in% c("ARSON","ASSAULT  W/DANGEROUS WEAPON",
                                                 "HOMICIDE", "ROBBERY", "SEX ABUSE"), 1, 0)
table(c2018$ctype, c2018$OFFENSE)
```

```
##
##     ARSON ASSAULT W/DANGEROUS WEAPON BURGLARY HOMICIDE MOTOR VEHICLE THEFT
##   0     0                       1668     1415        0                2388
##   1     5                          0        0      160                   0
##
##     ROBBERY SEX ABUSE THEFT F/AUTO THEFT/OTHER
##   0       0         0        11574       14141
##   1    2019       275            0           0
```

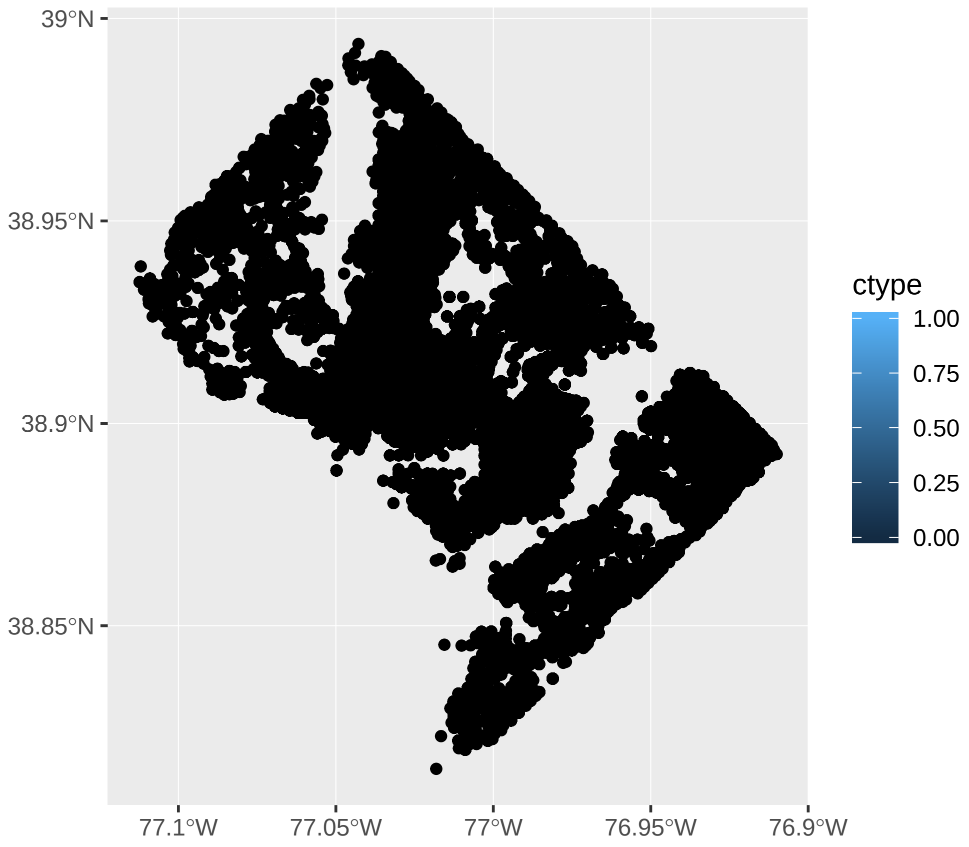
Now we fix:

```
# better!
c2018$ctype <- ifelse(c2018$OFFENSE %in% c("ARSON","ASSAULT W/DANGEROUS WEAPON",
                                           "HOMICIDE", "ROBBERY", "SEX ABUSE"), 1, 0)
table(c2018$ctype, c2018$OFFENSE)
```

```
##
##     ARSON ASSAULT W/DANGEROUS WEAPON BURGLARY HOMICIDE MOTOR VEHICLE THEFT
##   0     0                          0     1415        0                2388
##   1     5                       1668        0      160                   0
##
##     ROBBERY SEX ABUSE THEFT F/AUTO THEFT/OTHER
##   0       0         0        11574       14141
##   1    2019       275            0           0
```

Plot all of these (it may take a while; if your computer can't handle it, just ignore this step):

```
# plot
crimeo <- ggplot() +
  geom_sf(data = c2018, aes(fill = ctype))
crimeo
```

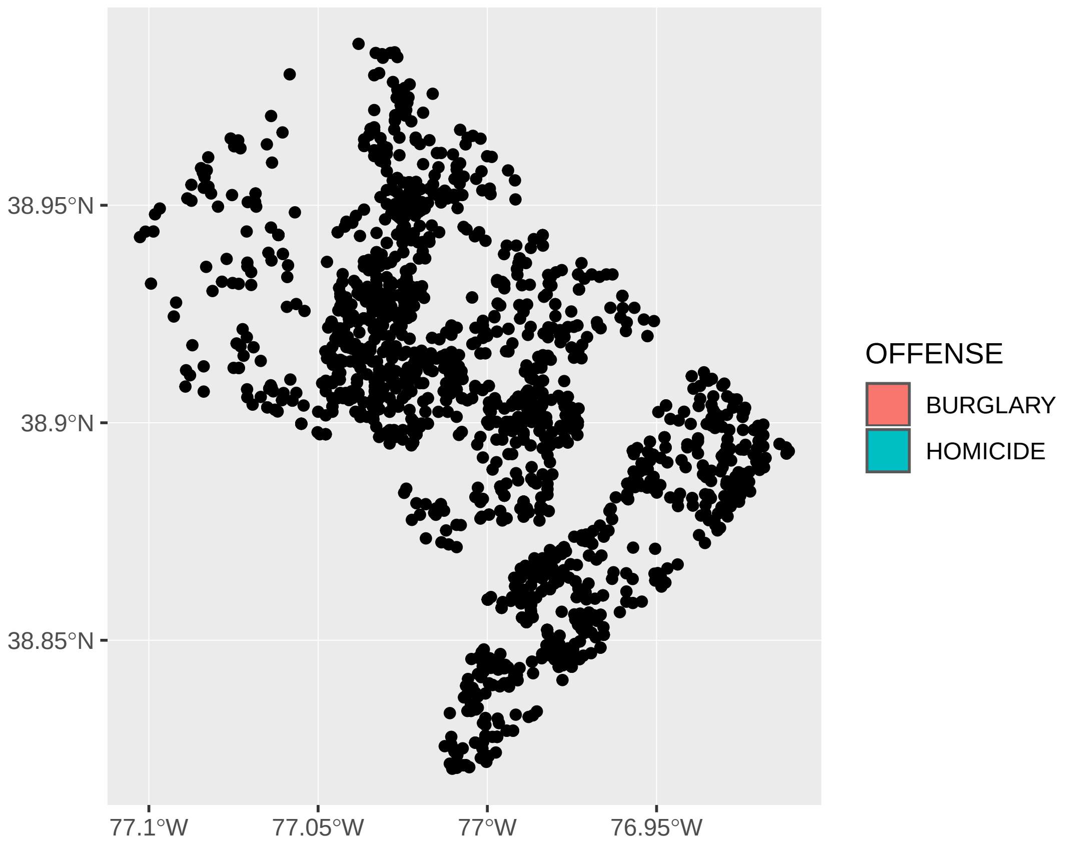


This still looks like an awful lot. So make a smaller dataframe with just homicides and burglaries by using the type of subsetting we've already learned. Recall that the statement `(c2018$OFFENSE == "HOMICIDE" | c2018$OFFENSE == "BURGLARY")` is true if the offense is *either* a homicide or a burglary. The `|` operator means "or" (in R and many other languages).

```
# make smaller dataframe
vc2018 <- c2018[which(c2018$OFFENSE == "HOMICIDE" | c2018$OFFENSE == "BURGLARY"),]
```

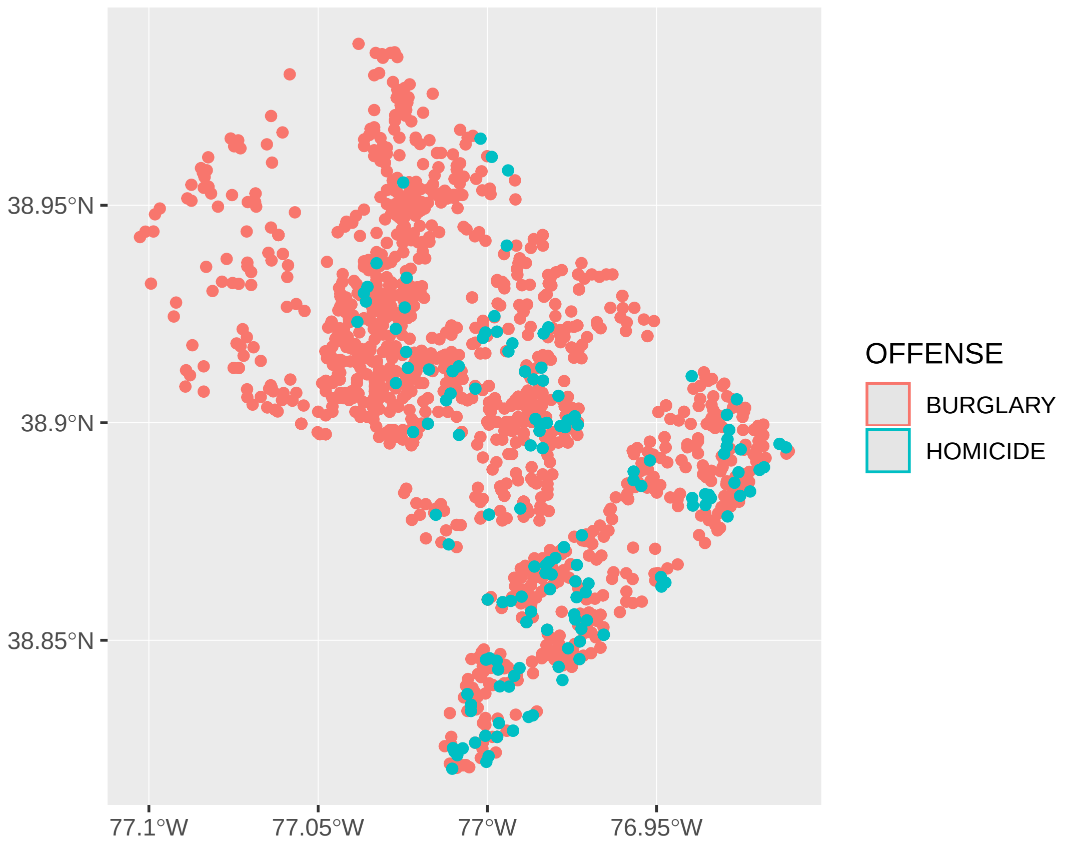
Now let's plot these two – notice the change in the `data =` input. Analogous to what we do with bar graphs, I use `fill` for the offense.

```r
# lets just do homicides and burglary -- bad
vc <- ggplot() +
  geom_sf(data = vc2018, aes(fill = OFFENSE))
vc
```



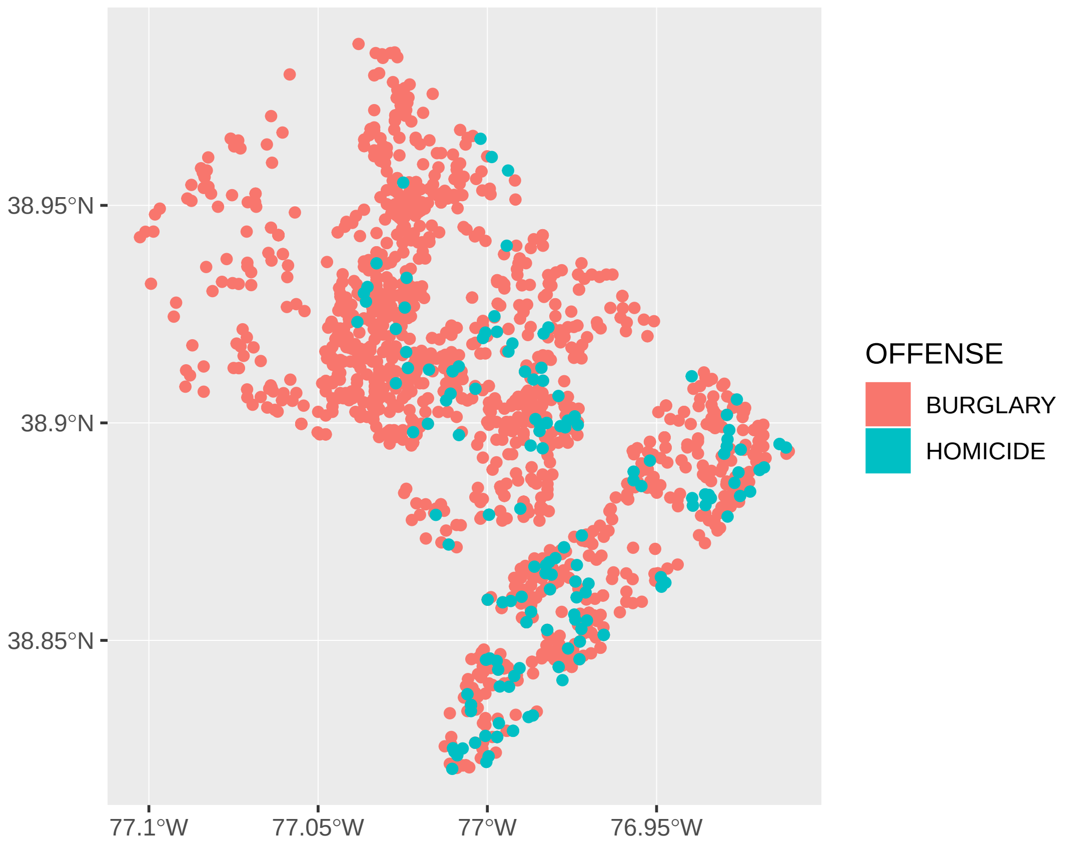
This was not a good idea! From the legend, R knows that we're trying to use these two types, but there's nothing to see in the plot. Rather than `fill=`, use `color=`.

```r
# lets just do homicides and burglary -- better
vc <- ggplot() +
  geom_sf(data = vc2018, aes(color = OFFENSE))
vc
```



But the legend now looks a little wacky. Use both `color` and `fill` to get something reasonable looking.

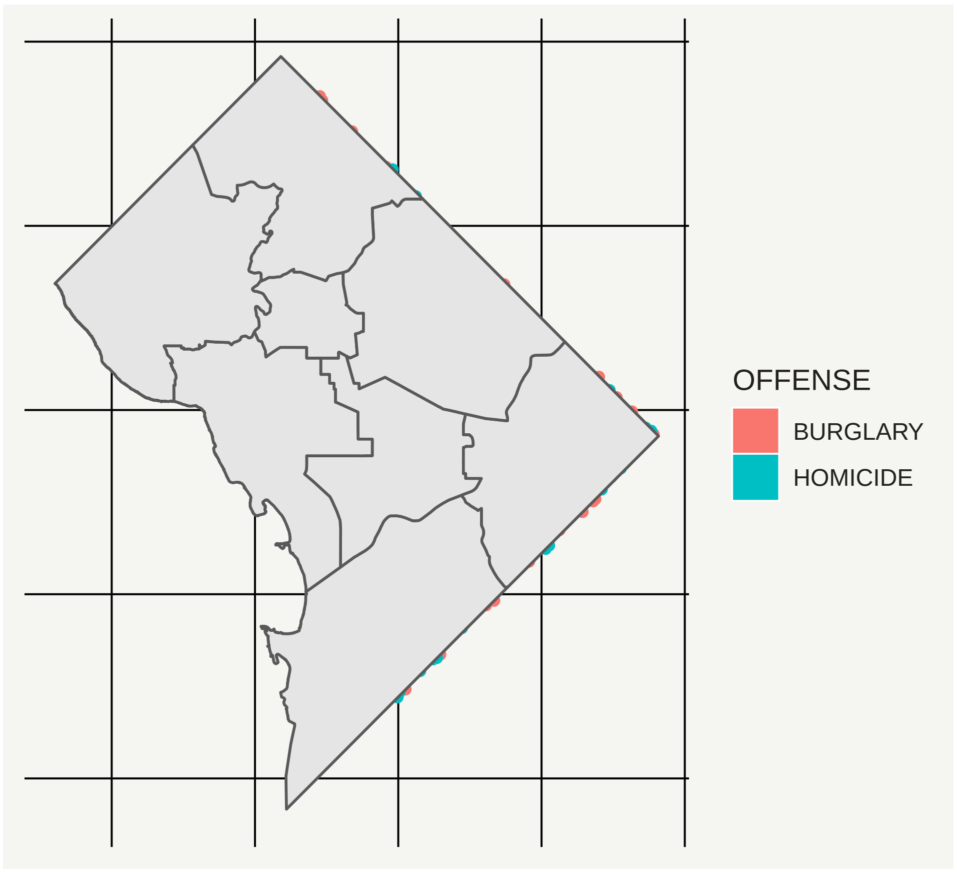
```r
# lets just do homicides and burglary -- best
vc <- ggplot() +
  geom_sf(data = vc2018, aes(color = OFFENSE, fill = OFFENSE))
vc
```



Of course, this map still looks ugly. I very much like the map here, made entirely in `ggplot` and `sf`. I've mostly copied Timo's work, particularly the theme elements of the map.

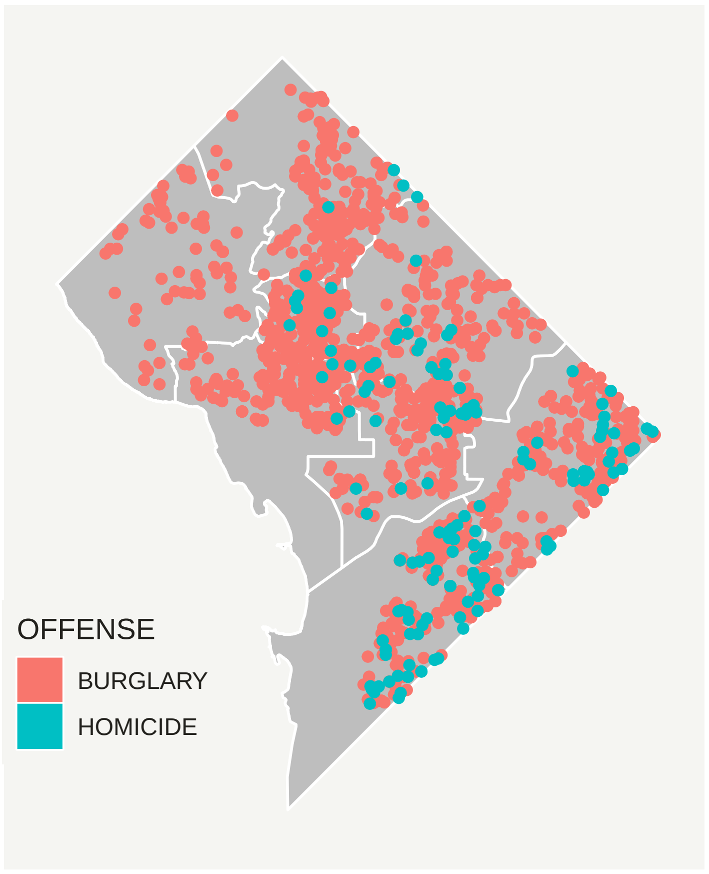
I'd like to also add ward boundaries to the map, with the points on top. The way I've tried to do this below is not a good idea! The wards are on top of the points.

```r
# make it look decent -- but wrong layer on top!
vc <- ggplot() +
  geom_sf(data = vc2018, aes(color = OFFENSE, fill = OFFENSE)) +
  geom_sf(data = dc.wards) +
  theme(
    text = element_text(color = "#22211d"),
    axis.line = element_blank(),
    axis.text.x = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks = element_blank(),
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    plot.background = element_rect(fill = "#f5f5f2", color = NA),
    panel.background = element_rect(fill = "#f5f5f2", color = NA),
    legend.background = element_rect(fill = "#f5f5f2", color = NA)
    )
vc
```



I can fix this ordering issue by switching the order of the layering. By putting the wards first in the command, they go first on the map. Then points (vc2018) on top.

```r
# make it look decent -- fixed layer issue
# subsantially copied from
# https://timogrossenbacher.ch/2016/12/beautiful-thematic-maps-with-ggplot2-only/
vc <- ggplot() +
  geom_sf(data = dc.wards, color = "white", fill = "grey") +
  geom_sf(data = vc2018, aes(color = OFFENSE, fill = OFFENSE)) +
  theme(
    text = element_text(color = "#22211d"),
    axis.line = element_blank(),
    axis.text.x = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks = element_blank(),
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    plot.background = element_rect(fill = "#f5f5f2", color = NA),
    panel.background = element_rect(fill = "#f5f5f2", color = NA),
    panel.grid = element_line(color = "#f5f5f2"),
    legend.background = element_rect(fill = "#f5f5f2", color = NA),
    legend.position = c(0.13,0.2)
  )
vc
```



Now save this file using `ggsave()`. This is code for saving an already created plot to a file. Before doing that, I make a variable (or a 1x1 matrix, if you prefer) that has today's date in it, in numeric form (e.g., 20190225). I like to always put the date in the name of anything I save so that when I work on it again I will not save over the older version.

R has a built in function to deliver today's date (`Sys.Date()`). I use the substring command (see lecture 4 notes) to extract the parts we need. I then use `paste0` to put the parts together. This command puts together all the text bits you put into it, with no spaces in between (the `0` part). You can look at all the parts separately if you'd like to better understand what's going on.

```
# save using ggsave
# todays date
dateo <- paste0(substr(Sys.Date(),1,4),substr(Sys.Date(),6,7),substr(Sys.Date(),9
                                                                          ,10))
dateo
```

```
## [1] "20190303"
```

Now with this date in hand, I make the filename with the date in it. As I've written, this file will save in the current directory. The currnet working directory is *not* the directory your program is in. (You can see what directory you're in with the `getwd()` command).

To save the file in a place I'd like, I also make a path variable, which now includes the filename. The other relevant parts of this command are the `plot=`, which asks which plot to save (this is one of the reasons to name your plots) and the "device" (jpg, tif, etc) to which you'd like to pass your plot. Finally (though there are plenty more options on the official page), we choose 300 dots per inch (good for printing) with `dpi = 300`.

```
fn <- paste0(dateo,"_violent_crime_2018.jpg")
patho <- paste0("H:/pppa_data_viz/2019/tutorials/tutorial_05/",fn)
ggsave(filename = patho,
       plot = vc,
       device = "jpg",
       dpi = 300)
```

```
## Saving 6.5 x 4.5 in image
```

E. Putting map data together with other data

Part of the value data in a spatial format is your ability to combine data spatially. This means you can combine data not just by merging variables, but by saying "what polygon does this point fall in?" or "which polygons does this polygon touch?"

We now perform such a spatial analysis. For our example, we'll use the crime data and find the block group in which each crime takes place. This will allow us to compute crime rates: crimes divided by population. Obviously we already have crimes, but we have to associate each crime with an area, and we need to know the population of each of these areas.

We begin by loading the block group map I downloaded from the DC open data website; be sure to download the shapefile.

Load the block group data and see what variables it has.

```
# load block group map
bg2010 <- st_read("H:/pppa_data_viz/2019/tutorial_data/lecture05/Census_Block_Groups__2010/Census_Block_
```

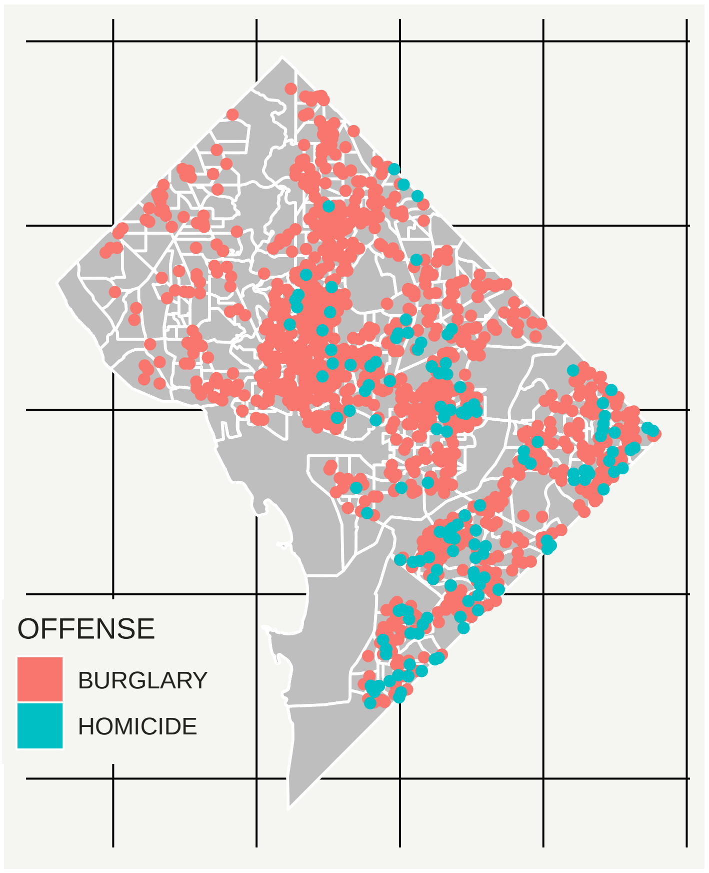
```
## Reading layer `Census_Block_Groups__2010' from data source `H:\pppa_data_viz\2019\tutorial_data\lectu
## Simple feature collection with 450 features and 54 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: -77.11976 ymin: 38.79165 xmax: -76.9094 ymax: 38.99581
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

```
names(bg2010)
```

```
##  [1] "OBJECTID"   "TRACT"      "BLKGRP"     "GEOID"      "P0010001"
##  [6] "P0010002"   "P0010003"   "P0010004"   "P0010005"   "P0010006"
## [11] "P0010007"   "P0010008"   "OP000001"   "OP000002"   "OP000003"
## [16] "OP000004"   "P0020002"   "P0020005"   "P0020006"   "P0020007"
## [21] "P0020008"   "P0020009"   "P0020010"   "OP00005"    "OP00006"
## [26] "OP00007"    "OP00008"    "P0030001"   "P0030003"   "P0030004"
## [31] "P0030005"   "P0030006"   "P0030007"   "P0030008"   "OP00009"
## [36] "OP00010"    "OP00011"    "OP00012"    "P0040002"   "P0040005"
## [41] "P0040006"   "P0040007"   "P0040008"   "P0040009"   "P0040010"
## [46] "OP000013"   "OP000014"   "OP000015"   "OP000016"   "H0010001"
## [51] "H0010002"   "H0010003"   "SHAPE_Leng" "SHAPE_Area" "geometry"
```

Let's plot it with the crimes to make sure that the maps are what we think they are.

```
# look at it with homicides and burglaries
cbg <- ggplot() +
  geom_sf(data = bg2010, color = "white", fill = "grey") +
  geom_sf(data = vc2018, aes(color = OFFENSE, fill = OFFENSE)) +
  theme(
    text = element_text(color = "#22211d"),
    axis.line = element_blank(),
    axis.text.x = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks = element_blank(),
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    plot.background = element_rect(fill = "#f5f5f2", color = NA),
    panel.background = element_rect(fill = "#f5f5f2", color = NA),
    panel.grid = element_blank(),
    legend.background = element_rect(fill = "#f5f5f2", color = NA),
    legend.position = c(0.13,0.2)
  )
cbg
```



Looks like all crimes fall inside a block group. Now we need to find which block group. To do this, we use a command called `st_intersection()`. When the shapefiles are [points, polygons] like the ones in the call below, R returns a points dataframe with information on the block group into which each point falls. Before this intersection, I make a smaller version of the block group data (`bg2010.small`) that has just the block group information – that's all I want to add to the points. Finally, I use `head()` to check on my new points data.

```
# find which crimes are in which block groups
bg2010.small <- bg2010[,c("TRACT","BLKGRP")]
cbg <- st_intersection(vc2018,bg2010.small)
```

```
## although coordinates are longitude/latitude, st_intersection assumes that they are planar

## Warning: attribute variables are assumed to be spatially constant
## throughout all geometries
```

```
head(cbg)
```

```
## Simple feature collection with 6 features and 26 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -77.0734 ymin: 38.90259 xmax: -77.05759 ymax: 38.91258
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
##            CCN               REPORT_DAT    SHIFT METHOD  OFFENSE
## 1826  18064713 2018-04-23T13:04:28.000Z      DAY OTHERS BURGLARY
## 11452 18134729 2018-08-14T03:10:42.000Z MIDNIGHT OTHERS BURGLARY
## 26674 18042061 2018-03-15T16:09:21.000Z  EVENING OTHERS BURGLARY
## 30745 18026424 2018-02-16T14:31:32.000Z      DAY OTHERS BURGLARY
## 30750 18026435 2018-02-16T15:42:03.000Z  EVENING OTHERS BURGLARY
## 23472 18136654 2018-08-17T04:38:28.000Z MIDNIGHT OTHERS BURGLARY
##                                              BLOCK XBLOCK YBLOCK WARD ANC
## 1826             3100 - 3199 BLOCK OF K STREET NW 394626 137194    2  2E
## 11452            3036 - 3099 BLOCK OF M STREET NW 394737 137483    2  2E
## 26674            3000 - 3099 BLOCK OF N STREET NW 394778 137665    2  2E
## 30745 2800 - 2899 BLOCK OF PENNSYLVANIA AVENUE NW 395005 137471    2  2E
## 30750 2800 - 2899 BLOCK OF PENNSYLVANIA AVENUE NW 395005 137471    2  2E
## 23472      3700 - 3799 BLOCK OF RESERVOIR ROAD NW 393634 138304    2  2E
##       DISTRICT PSA NEIGHBORHO BLOCK_GROU CENSUS_TRA VOTING_PRE LATITUDE
## 1826         2 206  Cluster 4   000100 4     000100 Precinct 5 38.90258
## 11452        2 206  Cluster 4   000100 4     000100 Precinct 5 38.90519
## 26674        2 206  Cluster 4   000100 4     000100 Precinct 5 38.90683
## 30745        2 206  Cluster 4   000100 4     000100 Precinct 5 38.90508
## 30750        2 206  Cluster 4   000100 4     000100 Precinct 5 38.90508
## 23472        2 206  Cluster 4   000201 1     000201 Precinct 6 38.91258
##       LONGITUDE        BID               START_DATE
## 1826  -77.06195 GEORGETOWN 2018-04-23T10:56:35.000Z
## 11452 -77.06068 GEORGETOWN 2018-08-13T22:52:11.000Z
## 26674 -77.06021       <NA> 2018-03-14T03:48:45.000Z
## 30745 -77.05759 GEORGETOWN 2018-02-16T01:25:28.000Z
## 30750 -77.05759 GEORGETOWN 2018-02-15T17:30:53.000Z
## 23472 -77.07340       <NA> 2018-08-17T03:36:29.000Z
##                       END_DATE  OBJECTID  OCTO_RECOR ctype  TRACT BLKGRP
## 1826                      <NA> 259183436 18064713-01     0 000100      4
## 11452 2018-08-14T00:19:42.000Z 259193062 18134729-01     0 000100      4
## 26674 2018-03-14T03:49:12.000Z 259240847 18042061-01     0 000100      4
## 30745 2018-02-16T02:30:15.000Z 259245997 18026424-01     0 000100      4
## 30750 2018-02-16T08:15:53.000Z 259246002 18026435-01     0 000100      4
## 23472 2018-08-17T03:46:04.000Z 259224604 18136654-01     0 000201      1
##                         geometry
## 1826  POINT (-77.06196 38.90259)
## 11452  POINT (-77.06068 38.9052)
## 26674 POINT (-77.06021 38.90684)
## 30745 POINT (-77.05759 38.90509)
## 30750 POINT (-77.05759 38.90509)
## 23472  POINT (-77.0734 38.91258)
```

Now I can start to find the average number of crimes per person. This is a multi-step process:

- find the block group for each crime (done)
- find the total number of crimes in each block group → block group level data
- add the block group population
- calculate a rate

We'll start by counting the number of crimes by block group. Remember that the current dataset is at the crime level – one observation per crime. We want a dataset at the block group/offense level – so we know how many offenses of each type are committed in each block group in 2018. We want R to just count the number of observations by block group and offense. We do this by grouping the data by tract/block group/offense, and then summarizing (counting) the number of observations in each group (`n()`).

We do an initial check on the final product (`cbgs`) by looking at how many rows it has relative to the original dataframe (`cbg`); it should have fewer.

```
# first count number of crimes by type by block group
require(dplyr)
dim(cbg)
```

```
## [1] 1567   27
```

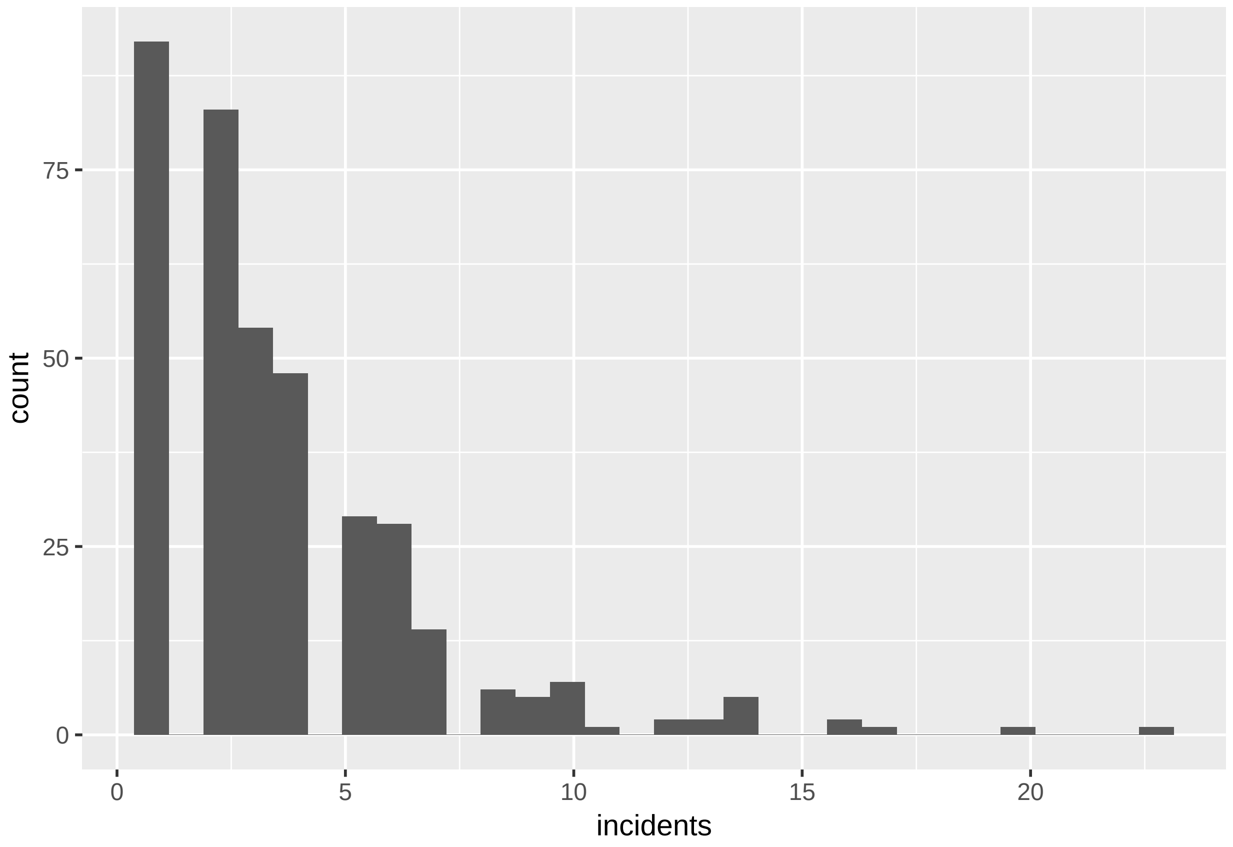
```
cbg <- group_by(.data = cbg, TRACT, BLKGRP, OFFENSE)
cbgs <- summarize(.data = cbg, incidents = n())
dim(cbgs)
```

```
## [1] 478   5
```

Let's check this work a different way by making a quick histogram of burglaries (homicides, thankfully, are more rare, so they are less amenable to this kind of check). Note that this shows only block groups with at least one burglary. (We could fix this with what we do later in this step.)

```
# make a histogram of the distribution of burlargies
burg.hist <- ggplot() +
  geom_histogram(data = cbgs[which(cbgs$OFFENSE == "BURGLARY"),], aes(x = incidents))
burg.hist
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Of course, it's also useful to know not just the level of crime, but the crime rate. We find a crime rate by dividing the number of crimes by the resident population (what is the right denominator is a big issue for criminologists; for now we'll suffice with resident population).

To make a rate, we need the resident population of each block group. I know that the block group shapefile has a population variable. It's poorly labeled online, but I know from other work that `P0010001` is actually total population.

We cannot merge a spatial dataframe (`cbgs`) and another spatial dataframe (`bgs.2010`). So let's make the block group file a regular dataframe, rather than a simple feature. We do this by setting the geometry part of the file to "NULL." Before this, we shrink the block group file for ease of use.

Next we merge the dataframe `bg2010.pop` with `cbg` by two variables: tract and block group. (If you were using more than just one state's worth of data, you would need to include the state ID as well; tract numbers repeat across states.) I keep all block groups so I can know which block groups have crime rates of zero.

After this merge (as with any merge), I check the size of the output dataframe relative to the size of the input dataframes.

```r
# need to merge in block group population
# i know that total population is P0010001
bg2010.pop <- bg2010[,c("TRACT","BLKGRP","P0010001")]
st_geometry(bg2010.pop) <- NULL
cbgs2 <- merge(x = cbgs, y = bg2010.pop, by = c("TRACT","BLKGRP"), all.x = TRUE)
dim(cbgs)
```

```
## [1] 478   5
```

```r
dim(bg2010.pop)
```

```
## [1] 450   3
```

```r
dim(cbgs2)
```

```
## [1] 478   6
```

```r
# check result of merge
summary(cbgs2$P0010001)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      33     930    1292    1382    1741    3916
```

How many block groups have no reported crime in 2018?

Now we are finally ready to make a crime rate. I set all missing values of `incidents` equal to zero, and then calculate a rate.

```r
# set incidents equal to zero if missing
summary(cbgs2$incidents)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   2.000   3.278   4.000  23.000
```

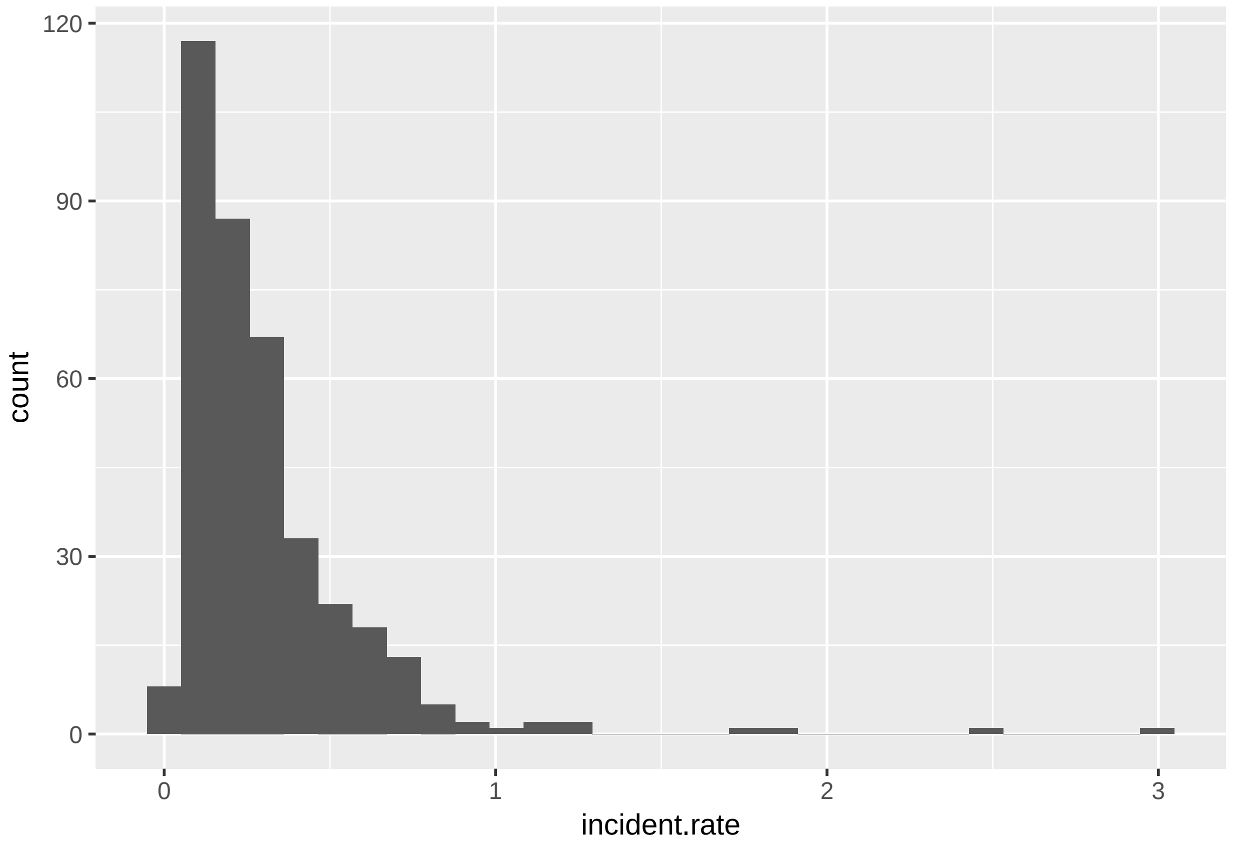
```r
cbgs2$incidents <- ifelse(is.na(cbgs2$incidents) == TRUE, 0, cbgs2$incidents)
# create crime rate
cbgs2$incident.rate = (cbgs2$incidents / (cbgs2$P0010001/100))
summary(cbgs2$incident.rate)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.03336 0.10564 0.18859 0.27505 0.33264 3.03030
```

Now make a histogram with these rates.

```r
# make the histogram
rburg.hist <- ggplot() +
  geom_histogram(data = cbgs2[which(cbgs2$OFFENSE == "BURGLARY"),], aes(x = incident.rate))
rburg.hist
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



F. Homework

1. Answer any questions in red througout the tutorial.
2. Get two other maps not used in this tutorial. Layer them and make them look decent.
3. Make a bar chart that shows the population density (people / area) by ward. You may wish to use `st_area()` to find the area of each ward. Population is already in the ward dataframe.