CSSS508, Week 9

Mapping

Chuck Lanfear

May 29, 2019

Updated: May 29, 2019

Today

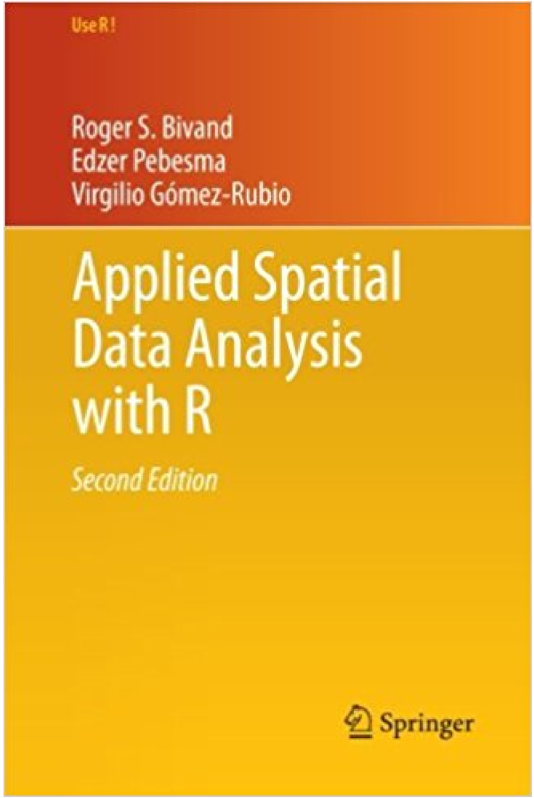
Basic Mapping in `ggplot2`

- Mapping with raw `ggplot2` using coordinates
- `ggmap` for mashing up maps with `ggplot2`
- Labeling points and using `ggrepel` to avoid overlaps

Advanced Mapping

- `sf`: Simple Features geometry for R
- `tidycensus` for obtaining Census Bureau data

Mapping in R: A quick plug



If you are interested in mapping, GIS, and geospatial analysis in R, *acquire this book*.

RSpatial.org is a great resource as well.

You may also consider taking Jon Wakefield's **CSSS 554: Statistical Methods for Spatial Data**, however it is challenging and focuses more heavily on statistics than mapping.

CSDE offers workshops using QGIS and/or ArcGIS. I recommend QGIS because it is free software with an extensive feature set and large user community.

Basic Mapping

`ggplot2` and `ggmap`

One Day of SPD Incidents

In Week 5, we looked at types of incidents the Seattle Police Department responded to in a single day. Now, we'll look at where those were.

```
library(tidyverse)
```

```
spd_raw <- read_csv("https://clanfear.github.io/CSSS508/Seattle_Polic
```

Taking a `glimpse()`

```
glimpse(spd_raw)
```

```
## Observations: 706
## Variables: 19
## $ `CAD CDW ID`                  <dbl> 1701856, 1701857, 1701853, 17018...
## $ `CAD Event Number`            <dbl> 16000104006, 16000103970, 160001...
## $ `General Offense Number`      <dbl> 2016104006, 2016103970, 20161041...
## $ `Event Clearance Code`        <chr> "063", "064", "161", "245", "202...
## $ `Event Clearance Description` <chr> "THEFT - CAR PROWL", "SHOPLIFT",...
## $ `Event Clearance SubGroup`    <chr> "CAR PROWL", "THEFT", "TRESPASS"...
## $ `Event Clearance Group`       <chr> "CAR PROWL", "SHOPLIFTING", "TRE...
## $ `Event Clearance Date`        <chr> "03/25/2016 11:58:30 PM", "03/25...
## $ `Hundred Block Location`      <chr> "S KING ST / 8 AV S", "92XX BLOC...
## $ `District/Sector`             <chr> "K", "S", "D", "M", "M", "B", "B...
## $ `Zone/Beat`                   <chr> "K3", "S3", "D2", "M1", "M3", "B...
## $ `Census Tract`                <dbl> 9100.102, 11800.602, 7200.106, 8...
## $ Longitude                     <dbl> -122.3225, -122.2680, -122.3420,...
## $ Latitude                      <dbl> 47.59835, 47.51985, 47.61422, 47...
## $ `Incident Location`           <chr> "(47.598347, -122.32245)", "(47....
## $ `Initial Type Description`    <chr> "THEFT (DOES NOT INCLUDE SHOPLIF...
## $ `Initial Type Subgroup`       <chr> "OTHER PROPERTY", "SHOPLIFTING",...
## $ `Initial Type Group`          <chr> "THEFT", "THEFT", "TRESPASS", "C...
## $ `At Scene Time`               <chr> "03/25/2016 10:25:51 PM", "03/25...
```

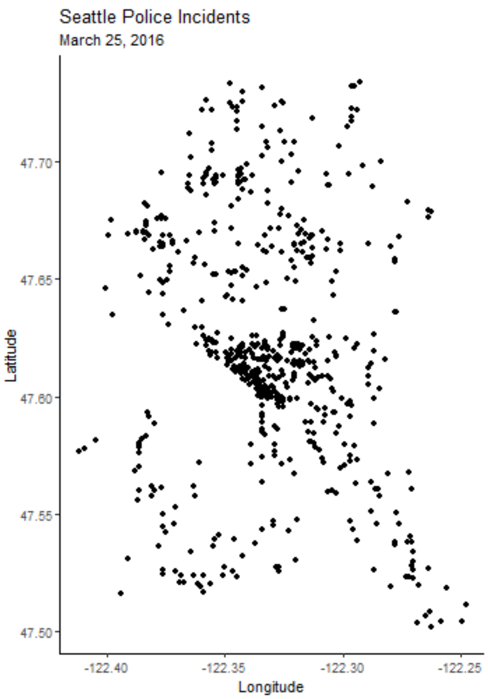
`x`,`y` as Coordinates

Coordinates, such as longitude and latitude, can be provided in `aes()` as `x` and `y` values.

This is ideal when you don't need to place points over some map for reference.

```
ggplot(spd_raw,
       aes(Longitude, Latitude)) +
  geom_point() +
  coord_fixed() +
  ggtitle("Seattle Police Incidents",
          subtitle="March 25, 2016") +
  theme_classic()
```

Sometimes, however, we want to plot these points over existing maps.



`ggmap`

`ggmap`

`ggmap` is a package that works with `ggplot2` to plot spatial data directly on map images downloaded from Google Maps[1], OpenStreetMap, and Stamen Maps (good artistic/minimal options).

What this package does for you:

1. Queries servers for a map (`get_map()`) at the location and scale you want
2. Plots the **raster** (bitmap) image as a `ggplot` object
3. Lets you add more `ggplot` layers like points, 2D density plots, text annotations
4. Additional functions for interacting with Google Maps (e.g. getting distances by bike)

[1] Requires an API Key now.

Installation

As of May 29, 2019, the current version of `ggmap` must be downloaded from GitHub.

This can be done using the `devtools` package.

```
if(!requireNamespace("devtools")) install.packages("devtools")
devtools::install_github("dkahle/ggmap", ref = "tidyup")
```

```
library(ggmap)
```

This may require compilation on your computer. If you get errors during install, check with me after class or in lab.

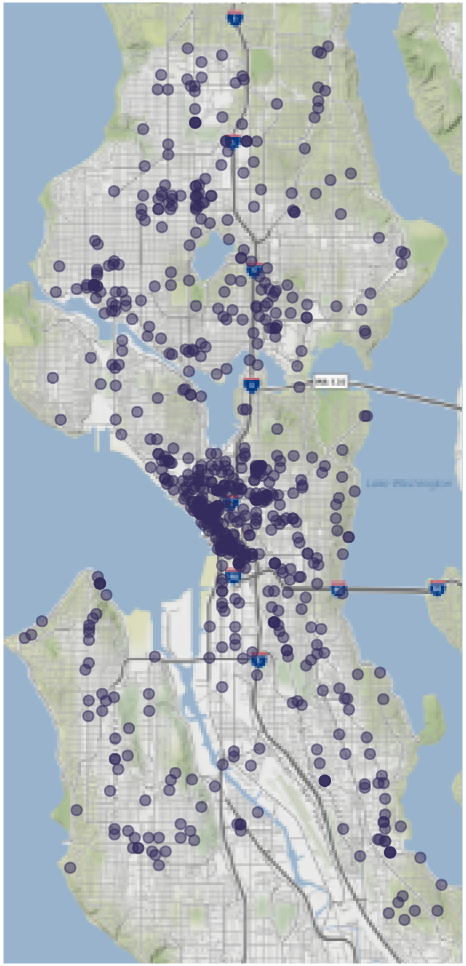
Quick Maps with `qmplot()`

`qmplot` will automatically set the map region based on your data:

```
qmplot(data = spd_raw,
       x = Longitude,
       y = Latitude,
       color = I("#342c5c"),
       alpha = I(0.5))
```

All I provided was numeric latitude and longitude, and it placed the data points correctly on a raster map of Seattle.

`I()` is used here to specify *set* (constant) rather than *mapped* values.



`get_map()`

Both `qmplot()` and `qmap()` are wrappers for a function called `get_map()` that retrieves a base map layer. Some options:

- `location=` search query or numeric vector of longitude and latitude
- `zoom=` a zoom level (3 = continent, 10 = city, 21 = building)
- `source=`
  - `"google"`: Google Maps for general purpose maps[1]
  - `"osm"`: OpenStreetMaps, general purpose but open access
  - `"stamen"`: Aesthetically pleasing alternatives based on OpenStreetMaps
- `maptype=`
  - Google types: `"terrain"`, `"terrain-background"`, `"satellite"`, `"roadmap"`, `"hybrid"`
  - Stamen types: `"watercolor"`, `"toner"`, `"toner-background"`, `"toner-lite"`
- `color=` `"color"` or `"bw"`

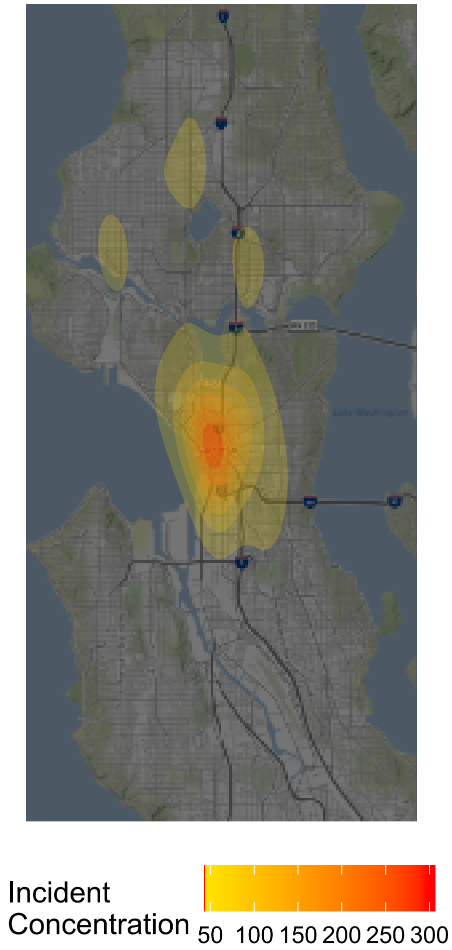
[1] Requires API key!

Adding Density Layers

Call `qmplot()` with no `geom()`, and then add density layers:

```
qmplot(data = spd_raw, geom = "blank",
  x = Longitude, y = Latitude,
  maptype = "toner-lite",
  darken = 0.5) +
  stat_density_2d(
    aes(fill = stat(level)),
      geom = "polygon",
      alpha = .2, color = NA) +
  scale_fill_gradient2(
      "Incident\nConcentration",
      low = "white",
      mid = "yellow",
      high = "red") +
  theme(legend.position = "bottom")
```

`stat(level)` indicates we want `fill=` to be based on `level` values calculated by the layer.



Labeling Points

Let's label the assaults and robberies specifically in downtown:

First filter to downtown based on values "eyeballed" from our earlier map:

```
downtown <- spd_raw %>%
  filter(Latitude > 47.58, Latitude < 47.64,
         Longitude > -122.36, Longitude < -122.31)
```

Then make a dataframe of just assaults and robberies:

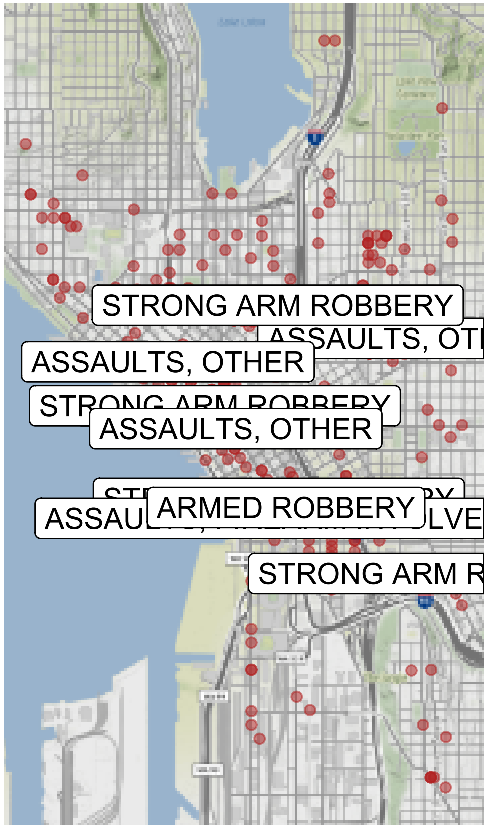
```
assaults <- downtown %>%
  mutate(assault_label =
           ifelse(`Event Clearance Group` %in%
                    c("ASSAULTS", "ROBBERY"),
                  `Event Clearance Description`, "")) %>%
  filter(assault_label != "")
```

Plotting with Labels

Now let's plot the events and label them with `geom_label()` (`geom_text()` without background or border):

```
qmplot(data = downtown,
       x = Longitude,
       y = Latitude,
       maptype = "toner-lite",
       color = I("firebrick"),
       alpha = I(0.5)) +
  geom_label(data = assaults,
       aes(label = assault_label))
```

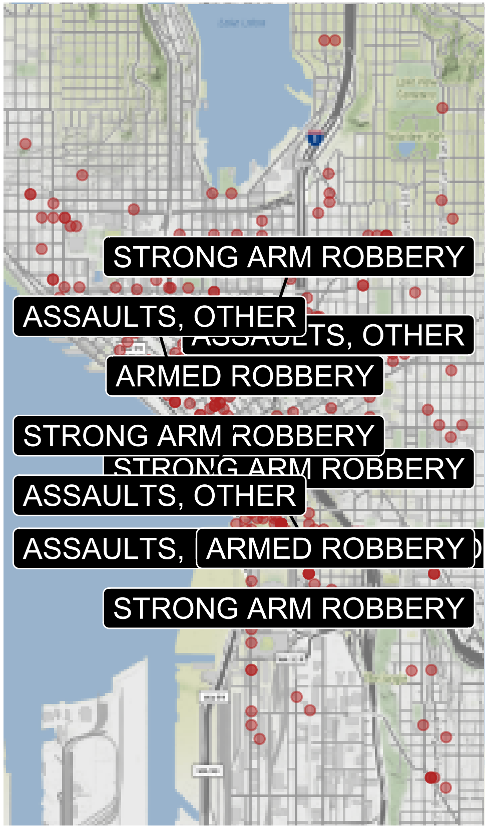
Placing the arguments for `color=` and `alpha=` inside `I()` prevents them from also applying to the labels. We would get transparent red labels otherwise!



`ggrepel`

You can also try `geom_label_repel()` or `geom_text_repel()` in the `ggrepel` package to fix or reduce overlaps (total space is limited here):

```
library(ggrepel)
qmplot(data =
    downtown,
    x = Longitude,
    y = Latitude,
    maptype = "toner-lite",
    color = I("firebrick"),
    alpha = I(0.5)) +
  geom_label_repel(
    data = assaults,
    aes(label = assault_label),
    fill = "black",
    color = "white",
    segment.color = "black")
```



Advanced Mapping

GIS and R with `sf`

Terminology

- Simple Features (`sf`)
- Coordinate Reference System (CRS)
- Shapefile

`sf`

Until recently, the main way to work with geospatial data in R was through the `sp` package. `sp` works well but does not store data the same way as most GIS packages and can be bulky and complicated.

The more recent `sf` package implements the GIS standard of **Simple Features** in R.

`sf` is also integrated into the `tidyverse`: e.g. `geom_sf()` in `ggplot2`.

The package is somewhat new but is expected to *replace* `sp` eventually. The principle authors and contributors to `sf` are the same authors as `sp` but with new developers from the `tidyverse` as well.

Because `sf` is the new standard, we will focus on it today.

```
library(sf)
```

Simple Features

A **Simple Feature** is a single observation with some defined geospatial location(s). Features are stored in special data frames (class `sf`) with two properties:

- **Geometry**: Properties describing a location (usually on Earth).
  - Usually 2 dimensions, but support for up to 4.
  - Stored in a single reserved *list-column* (`geom`, of class `sfc`).[1]
  - Contain a defined coordinate reference system.

- **Attributes**: Characteristics of the location (such as population).
  - These are non-spatial measures that describe a feature.
  - Standard data frame columns.

[1] A list-column is the same length as all other columns in the data, but each element contains *sub-elements* (class `sfg`) with all the geometrical components.

List-columns require special functions to manipulate, *including removing them*.

Coordinate Reference Systems

Coordinate reference systems (**CRS**) specify what location on Earth geometry coordinates are *relative to* (i.e. what location is (0,0) when plotting).

The most commonly used is **WGS84**, the standard for Google Earth, the Department of Defense, and GPS satellites.

There are two common ways to define a CRS:

- **EPSG codes** (`epsg` in R)
  - Numeric codes which *refer to a predefined projection*
  - Ex: WGS84 is `4326`
- **PROJ.4 strings** (`proj4string` in R)
  - Text strings which *define a projection*
  - WGS84 looks like this:

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

Shapefiles

Geospatial data is typically stored in **shapefiles** which store geometric data as **vectors** with associated attributes (variables)

Shapefiles actually consist of multiple individual files. There are usually at least three (but up to 10+):

- `.shp`: The feature geometries
- `.shx`: Shape positional index
- `.dbf`: Attributes describing features[1]

Often there will also be a `.prj` file defining the coordinate system.

[1] This is just a dBase IV file which is an ancient and common database storage file format.

Using `sf`

Selected `sf` Functions

`sf` is a huge, feature-rich package. Here is a sample of useful functions:

- `st_read()`, `st_write()`: Read and write shapefiles.
- `geom_sf()`: `ggplot()` layer for `sf` objects.
- `st_join()`: Join data by spatial relationship.
- `st_transform()`: Convert between CRS.
- `st_set_geometry(., NULL)`: Remove geometry from a `sf` data frame.
- `st_relate()`: Compute relationships between geometries (like neighbor matrices).
- `st_interpolate_aw()`: Areal-weighted interpolation of polygons.

Loading Data

We will work with the voting data from Homework 5. You can obtain a shape file of King County voting precincts from the county GIS data portal.

We can load the file using `st_read()`.

```
precinct_shape <- st_read("./data/district/votdst.shp",
                          stringsAsFactors = F) %>%
  select(Precinct=NAME, geometry)
```

```
## Reading layer `votdst' from data source `C:\Users\cclan\OneDrive\GitHub\CS
## Simple feature collection with 2592 features and 5 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 1220000 ymin: 31600 xmax: 1580000 ymax: 288000
## epsg (SRID):    NA
## proj4string:    +proj=lcc +lat_1=47.5 +lat_2=48.73333333333333 +lat_0=47 +
```

Voting Data: Processing

```r
precincts_votes_sf <-
  read_csv("./data/king_county_elections_2016.txt") %>%
  filter(Race=="US President & Vice President",
         str_detect(Precinct, "SEA ")) %>%
  select(Precinct, CounterType, SumOfCount) %>%
  group_by(Precinct) %>%
  filter(CounterType %in%
           c("Donald J. Trump & Michael R. Pence",
             "Hillary Clinton & Tim Kaine",
             "Registered Voters",
             "Times Counted")) %>%
  mutate(CounterType =
           recode(CounterType,
                  `Donald J. Trump & Michael R. Pence` = "Trump",
                  `Hillary Clinton & Tim Kaine` = "Clinton",
                  `Registered Voters`="RegisteredVoters",
                  `Times Counted` = "TotalVotes")) %>%
  spread(CounterType, SumOfCount) %>%
  mutate(P_Dem = Clinton / TotalVotes,
         P_Rep = Trump / TotalVotes,
         Turnout = TotalVotes / RegisteredVoters) %>%
  select(Precinct, P_Dem, P_Rep, Turnout) %>%
  filter(!is.na(P_Dem)) %>%
  left_join(precinct_shape)
```

Taking a `glimpse()`

```
glimpse(precincts_votes_sf)
```

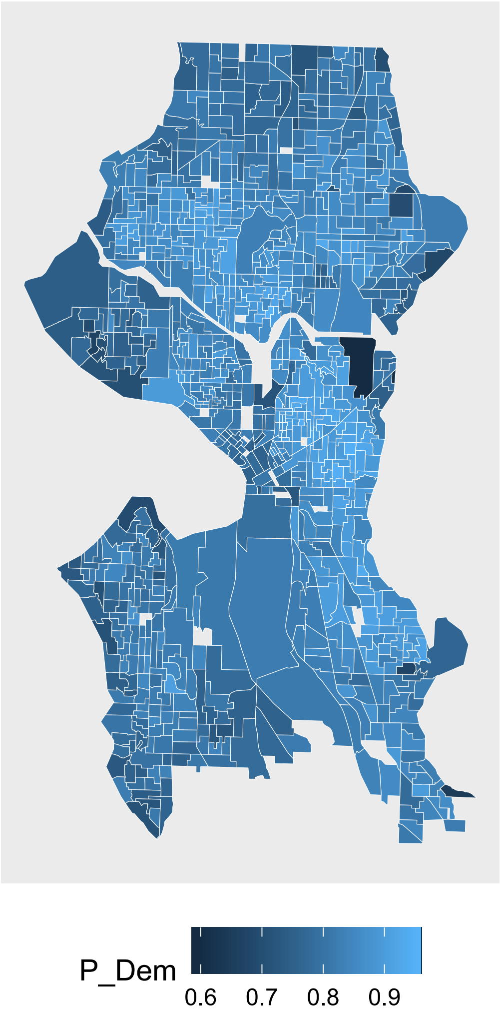
```
## Observations: 960
## Variables: 5
## Groups: Precinct [960]
## $ Precinct <chr> "SEA 11-1256", "SEA 11-1550", "SEA 11-1552", "SEA 11-...
## $ P_Dem    <dbl> 0.7707510, 0.8168421, 0.7507987, 0.8376328, 0.8325991...
## $ P_Rep    <dbl> 0.15612648, 0.07789474, 0.13418530, 0.08649469, 0.085...
## $ Turnout  <dbl> 0.6931507, 0.7274119, 0.7347418, 0.7522831, 0.7579299...
## $ geometry <MULTIPOLYGON [US_survey_foot]> MULTIPOLYGON (((1273698 193...
```

Voting Map

We can plot `sf` geometry using `geom_sf()`.

```
ggplot(precincts_votes_sf,
       aes(fill=P_Dem)) +
  geom_sf(color="white",
          size=0.1) +
  coord_sf(datum=NULL) +
  theme(legend.position =
          "bottom")
```

- `fill=P_Dem` maps color inside precincts to `P_Dem`.
- `color="white"` sets boundaries to white.
- `coord_sf(datum=NULL)` removes graticule lines.



`tidycensus`

`tidycensus`

`tidycensus` can be used to search the American Community Survey (ACS) and Dicennial Census for variables, then download them and automatically format them as tidy dataframes.

These dataframes include geographical boundaries such as tracts!

This package utilizes the Census API, so you will need to obtain a Census API key. I talk more about APIs in the Social Media Data and Text Mining lecture.

Application Program Interface (API): A type of computer interface that exists as the "native" method of communication between computers, often via http (usable via `httr` package).

- R packages that interface with websites and databases typically use APIs.
- APIs make accessing data easy while allowing websites to control access.

See the developer's GitHub page for detailed instructions.

Key `tidycensus` Functions

- `census_api_key()` - Install a census api key.
  - Note you will need to run this prior to using any `tidycensus` functions.
- `load_variables()` - Load searchable variable lists.
  - `year =`: Sets census year or endyear of 5-year ACS
  - `dataset =`: Sets dataset (see `?load_variables`)
- `get_decennial()` - Load Census variables and geographical boundaries.
  - `variables =`: Provide vector of variable IDs
  - `geography =`: Sets unit of analysis (e.g. `state`, `tract`, `block`)
  - `year =`: Census year (`1990`, `2000`, or `2010`)
  - `geometry = TRUE`: Returns `sf` geometry
- `get_acs()` - Load ACS variables and boundaries.

Searching for Variables

```
library(tidycensus)
# census_api_key("PUT YOUR KEY HERE", install=TRUE)
acs_2015_vars <- load_variables(2015, "acs5")
acs_2015_vars[10:18,] %>% print()
```

```
## # A tibble: 9 x 3
##   name       label                                 concept
##   <chr>      <chr>                                 <chr>
## 1 B01001_008 Estimate!!Total!!Male!!20 years       SEX BY AGE
## 2 B01001_009 Estimate!!Total!!Male!!21 years       SEX BY AGE
## 3 B01001_010 Estimate!!Total!!Male!!22 to 24 years SEX BY AGE
## 4 B01001_011 Estimate!!Total!!Male!!25 to 29 years SEX BY AGE
## 5 B01001_012 Estimate!!Total!!Male!!30 to 34 years SEX BY AGE
## 6 B01001_013 Estimate!!Total!!Male!!35 to 39 years SEX BY AGE
## 7 B01001_014 Estimate!!Total!!Male!!40 to 44 years SEX BY AGE
## 8 B01001_015 Estimate!!Total!!Male!!45 to 49 years SEX BY AGE
## 9 B01001_016 Estimate!!Total!!Male!!50 to 54 years SEX BY AGE
```

Getting Data

```r
king_county <- get_acs(geography="tract", state="WA",
                       county="King", geometry = TRUE,
                       variables=c("B02001_001E",
                                   "B02009_001E"))
```

What do these look like?

```r
glimpse(king_county)
```

```
## Observations: 796
## Variables: 6
## $ GEOID    <chr> "53033000100", "53033000100", "53033000200", "5303300...
## $ NAME     <chr> "Census Tract 1, King County, Washington", "Census Tr...
## $ variable <chr> "B02001_001", "B02009_001", "B02001_001", "B02009_001...
## $ estimate <dbl> 7963, 1550, 7973, 422, 2822, 170, 6721, 1168, 5300, 3...
## $ moe      <dbl> 529, 449, 377, 254, 201, 92, 454, 441, 571, 218, 165,...
## $ geometry <MULTIPOLYGON [°]> MULTIPOLYGON (((-122.2965 4..., MULTIPOL...
```

Variable names are in `variables` and values are in `estimate`.

Processing Data

To get one row per precinct, we want to `spread()` key (`variable`) and value (`estimate`) columns into separate columns for each variable. Then we `mutate()` `Any Black` into a proportion measure.

```
king_county <-  king_county %>%
  select(-moe) %>%
  group_by(GEOID) %>%
  spread(variable, estimate) %>%
  rename(`Total Population`=B02001_001,
         `Any Black`=B02009_001) %>%
  mutate(`Any Black` = `Any Black` / `Total Population`)
```

I remove the *margin of error* (`moe`) column before spreading.

The Data

```
head(king_county, 10)
```

```
## Simple feature collection with 10 features and 4 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -122.3799 ymin: 47.69462 xmax: -122.2551 ymax: 47.73416
## epsg (SRID):    4269
## proj4string:    +proj=longlat +datum=NAD83 +no_defs
## # A tibble: 10 x 5
## # Groups:   GEOID [10]
##    GEOID  NAME    `Total Populati~ `Any Black`                     geometry
##    <chr>  <chr>              <dbl>       <dbl>           <MULTIPOLYGON [°]>
##  1 53033~ Census~             7963      0.195  (((-122.2965 47.73378, -122~
##  2 53033~ Census~             7973      0.0529 (((-122.3236 47.72309, -122~
##  3 53033~ Census~             2822      0.0602 (((-122.3452 47.73413, -122~
##  4 53033~ Census~             6721      0.174  (((-122.3609 47.73414, -122~
##  5 53033~ Census~             5300      0.0570 (((-122.3609 47.72507, -122~
##  6 53033~ Census~             3052      0.0131 (((-122.3764 47.71652, -122~
##  7 53033~ Census~             8061      0.111  (((-122.345 47.72868, -122.~
##  8 53033~ Census~             4854      0.142  (((-122.3127 47.71944, -122~
##  9 53033~ Census~             2582      0.0352 (((-122.2915 47.7155, -122.~
## 10 53033~ Census~             1972      0.0152 (((-122.285 47.7192, -122.2~
```

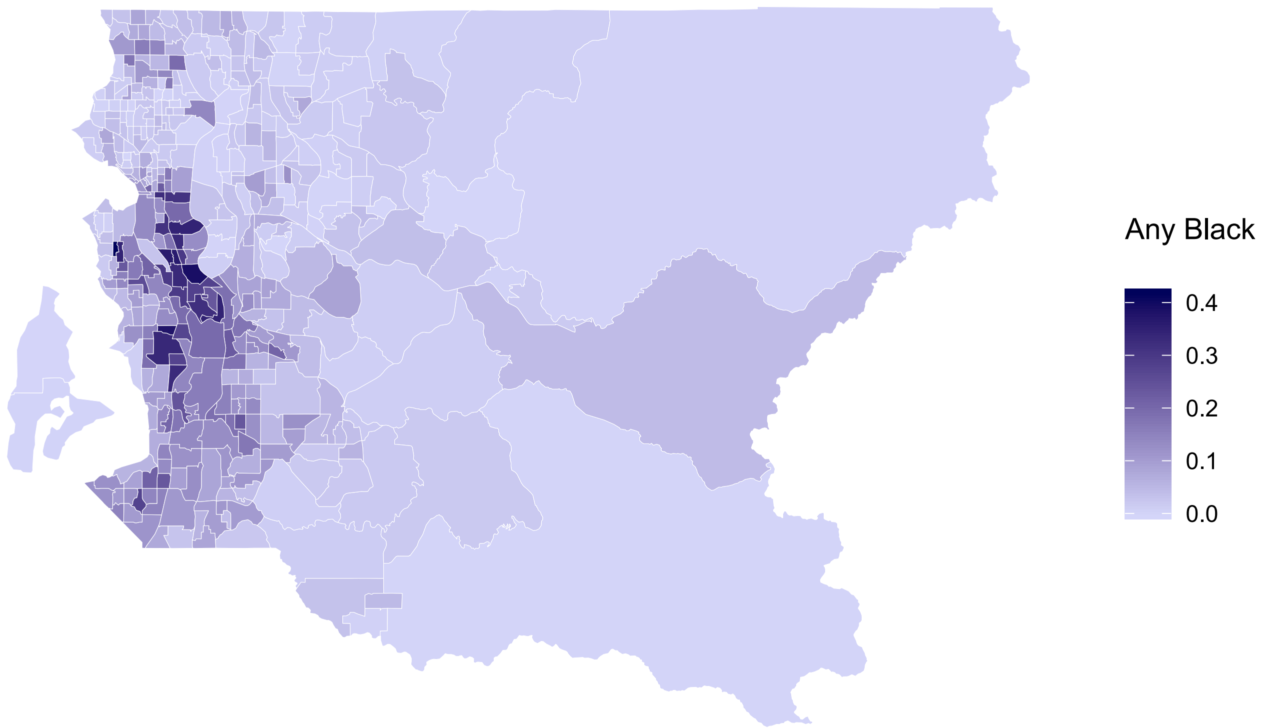
Mapping Code

```
king_county %>%
  ggplot(aes(fill=`Any Black`)) +
  geom_sf(size=0.1, color="white") +
  coord_sf(crs = "+proj=longlat +datum=WGS84", datum=NA) +
  scale_fill_continuous(name="Any Black\n",
                        low="#d4d5f9",
                        high="#00025b") +
  theme_minimal() + ggtitle("Proportion Any Black")
```

New functions:

- `geom_sf()` draws Simple Features coordinate data.
- `coord_sf()` is used here with these arguments:
    - `crs`: Modifies the coordinate reference system (CRS); WGS84 is possibly the most commonly used CRS.
    - `datum=NA`: Removes graticule lines, which are geographical lines such as meridians and parallels.

Proportion Any Black



Removing Water

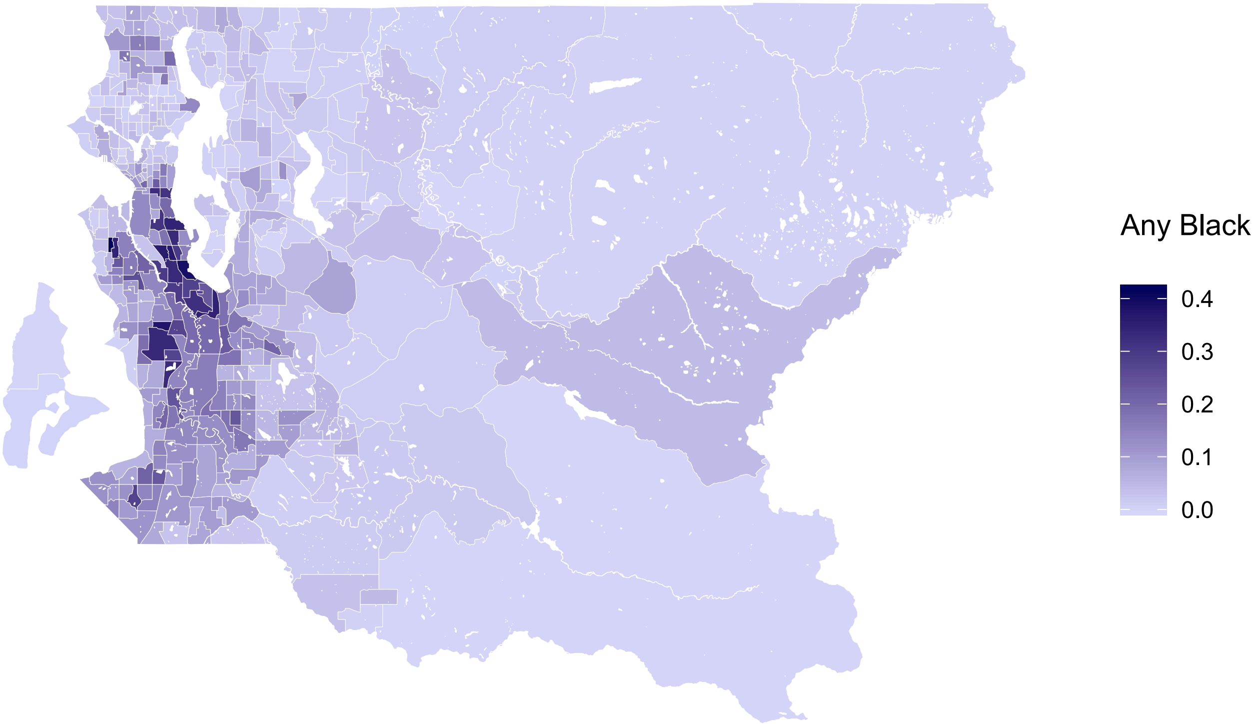
With a simple function and boundaries of water bodies in King County, we can replace water with empty space.

```
st_erase <- function(x, y) {
  st_difference(x, st_union(st_combine(y)))
}

kc_water <- tigris::area_water("WA", "King", class = "sf")

kc_nowater <- king_county %>%
  st_erase(kc_water)
```

- `st_combine()` merges all geometries into one
- `st_union()` resolves internal boundaries
- `st_difference()` subtracts `y` geometry from `x`
- `area_water()` obtains `sf` geometry of water bodies.

Then we can reproduce the same plot using `kc_nowater`...

Proportion Any Black

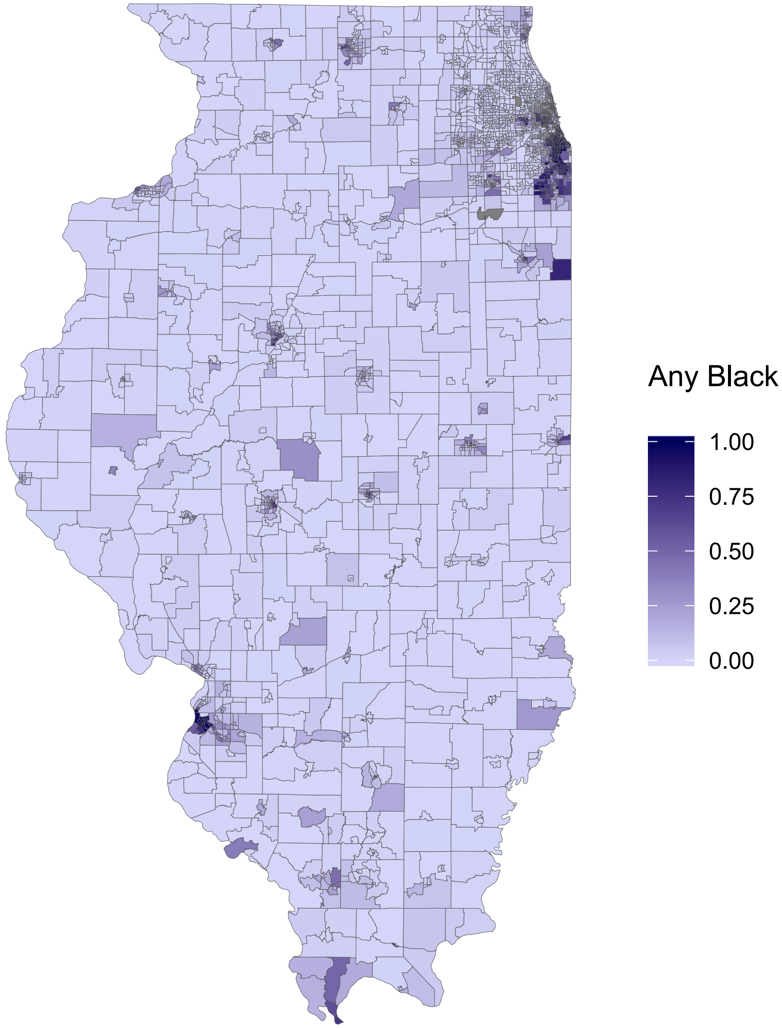


State Example Data

```
pb_state <-
  get_acs(geography = "tract", state = "IL",
          geometry  = TRUE,
          variables = c("B02001_001E",
                        "B02009_001E")) %>%
  select(-moe) %>%
  group_by(GEOID) %>%
  spread(variable, estimate) %>%
  rename(`Total Population` = B02001_001,
         `Any Black` = B02009_001) %>%
  mutate(`Any Black` = `Any Black` / `Total Population`)
```

State Example Plot

```
pb_state %>%
  ggplot(aes(fill=`Any Black`)) +
  geom_sf(lwd=0) +
  coord_sf(crs = "+proj=longlat +datum=WGS84", datum=NA) +
  scale_fill_continuous(name="Any Black\n",
                        low="#d4d5f9",
                        high="#00025b") +
  theme_minimal()
```


Any Black
1.00
0.75
0.50
0.25
0.00

Homework

Homework 6, part 2 is due next week.

Optional Exercise

Use the HW 7 template to practice making maps of the restaurant inspection data.

If you wish to submit it for bonus points, turn it in via Canvas by 11:59 PM on June 11th.