CSSS508, Lecture 9

Mapping

Michael Pearce
(based on slides from Chuck Lanfear)

May 24, 2023

Topics

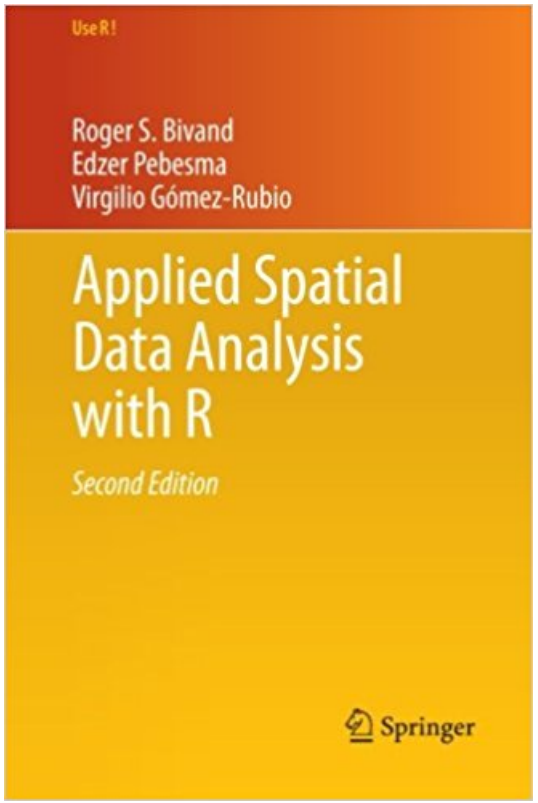
Last time, we learned about,

1. Basics of Strings
2. Strings in Base R
3. Strings in `stringr` (Tidyverse)

Today, we will cover,

1. Basic mapping:
    - Simple `ggplot`s with coordinates
    - Density plots in `ggmap`
    - Labeling points with `ggrepel`
2. Advanced mapping:
    - GIS with `sf`
    - `tidycensus`

Mapping in R: A quick plug



Mapping can be **complex!** If you are interested in digging deeper, here are some resources:

- If you are interested in mapping, GIS, and geospatial analysis in R, *acquire this book*.
- RSpatial.org is a great resource for working with spatial data.
- For more information on *spatial statistics*, consider taking Jon Wakefield's **CSSS 554: Statistical Methods for Spatial Data** in the winter quarter.

1. Basic mapping with `ggmap`

One Day of SPD Incidents

Today, we'll study data from the Seattle Police Department regarding incidents on just one day: March 25, 2016.

The data can be downloaded from my predecessor's Github using the code below (code in the R companion file).

```
library(ggplot2)
library(readr)
library(dplyr)
library(tidyr)
library(stringr)
```

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

Taking a `glimpse()`

```
glimpse(spd_raw)
```

```
## Rows: 706
## Columns: 19
## $ `CAD CDW ID`                 <dbl> 1701856, 1701857, 1701853, 170…
## $ `CAD Event Number`           <dbl> 16000104006, 16000103970, 1600…
## $ `General Offense Number`     <dbl> 2016104006, 2016103970, 201610…
## $ `Event Clearance Code`       <chr> "063", "064", "161", "245", "2…
## $ `Event Clearance Description` <chr> "THEFT - CAR PROWL", "SHOPLIFT…
## $ `Event Clearance SubGroup`   <chr> "CAR PROWL", "THEFT", "TRESPAS…
## $ `Event Clearance Group`      <chr> "CAR PROWL", "SHOPLIFTING", "T…
## $ `Event Clearance Date`       <chr> "03/25/2016 11:58:30 PM", "03/…
## $ `Hundred Block Location`     <chr> "S KING ST / 8 AV S", "92XX BL…
## $ `District/Sector`            <chr> "K", "S", "D", "M", "M", "B", …
## $ `Zone/Beat`                  <chr> "K3", "S3", "D2", "M1", "M3", …
## $ `Census Tract`               <dbl> 9100.102, 11800.602, 7200.106,…
## $ Longitude                    <dbl> -122.3225, -122.2680, -122.342…
## $ Latitude                     <dbl> 47.59835, 47.51985, 47.61422, …
## $ `Incident Location`          <chr> "(47.598347, -122.32245)", "(4…
## $ `Initial Type Description`   <chr> "THEFT (DOES NOT INCLUDE SHOPL…
## $ `Initial Type Subgroup`      <chr> "OTHER PROPERTY", "SHOPLIFTING…
## $ `Initial Type Group`         <chr> "THEFT", "THEFT", "TRESPASS", …
## $ `At Scene Time`              <chr> "03/25/2016 10:25:51 PM", "03/…
```

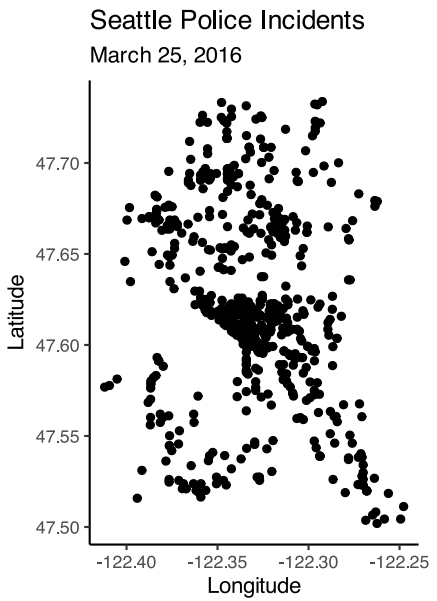
What does each **row represent?** What are the **variables and values?**

Simple Plotting: Regular `ggplots`

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() + # evenly spaces x and y
  ggtitle("Seattle Police Incidents",
          subtitle="March 25, 2016") +
  theme_classic()
```



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

Better plots with `ggmap`

`ggmap` is a package that works with `ggplot2` to plot spatial data directly on map images.

What this package does for you:

1. Queries servers for a map at the location and scale you want
2. Plots the 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 (beyond this course)

Installation

We can install `ggmap` like other packages:

```
install.packages("ggmap")
```

Note: If prompted to "install from sources the package which needs compilation", I find that typing "no" works best!

Because the map APIs it uses change frequently, sometimes you may need to get a newer development version of `ggmap` from the author's GitHub. This can be done using the `remotes` package.

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

Note, this may require compilation on your computer.

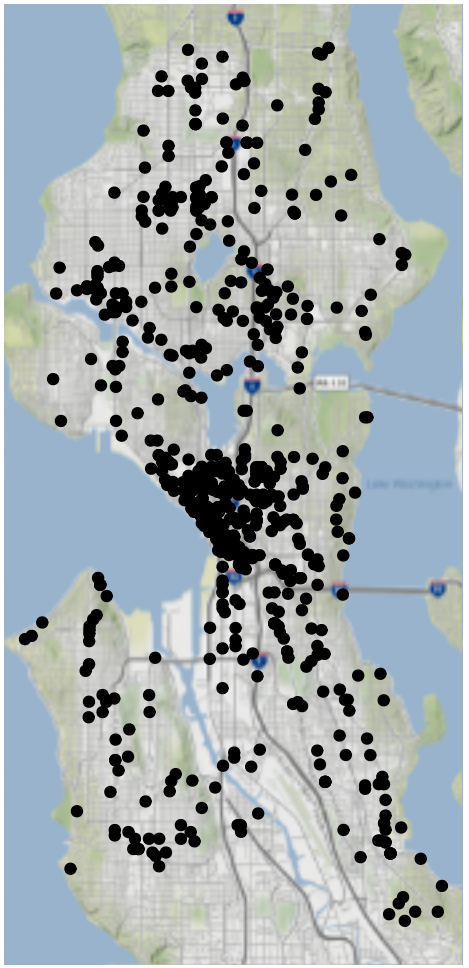
```
library(ggmap)
```

Quick Maps with `qmplot()`

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

```
qmplot(data = spd_raw,
       x = Longitude,
       y = Latitude)
```

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



`get_map()`

`qmplot()` internally uses the function `get_map()`, which 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)
- `maptype=`: `"watercolor"`, `"toner"`, `"toner-background"`, `"toner-lite"`
- `color=`: `"color"` or `"bw"`

There are fancier options available, but many require a Google Maps API.

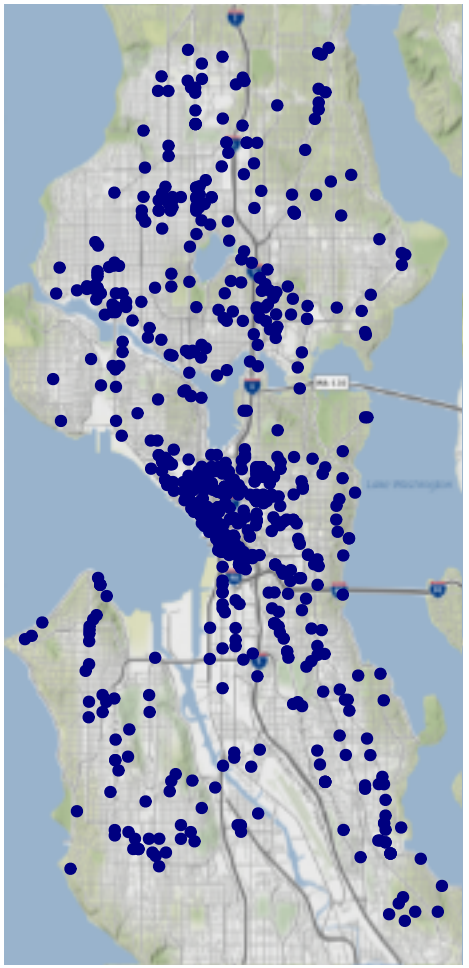
See the **help page** for `gmplot()` or `get_map` for more information!

Nicer Example

Let's add *color* to our plot from before

```
qmplot(data = spd_raw,
       x = Longitude,
       y = Latitude,
       color=I("navy")
)
```

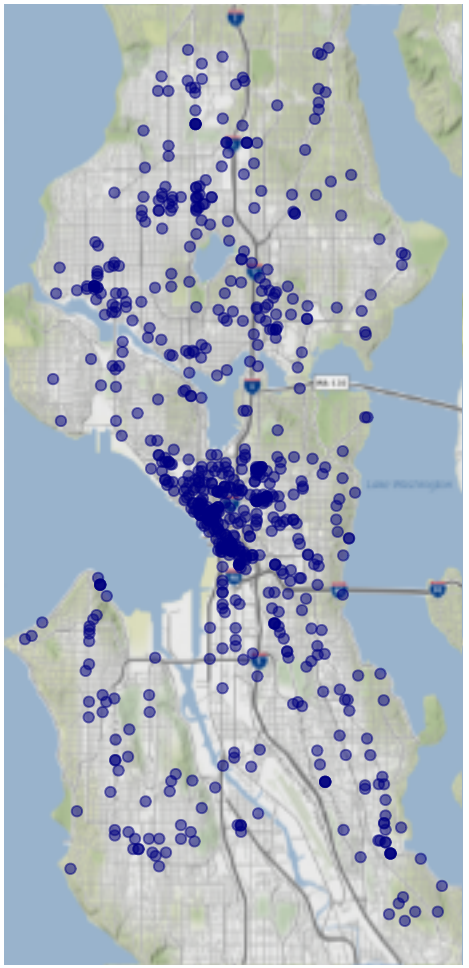
`I()` is used here to specify *set* (constant) rather than *mapped* values.



Nicer Example

Add *transparency*

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

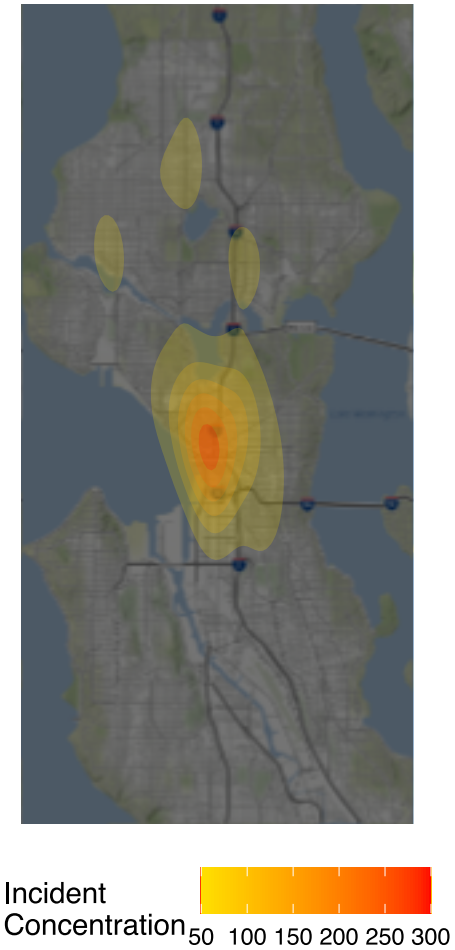


Density plots

Density Layers

We can create a **spatial density plot** using the layer function `stat_density_2d()`:

- What to use when creating a "density" plot (where the observations occur)
- Aesthetics of the density plot
- Additional layer functions for customization



Basic Map

Start with basic plot, remove points:

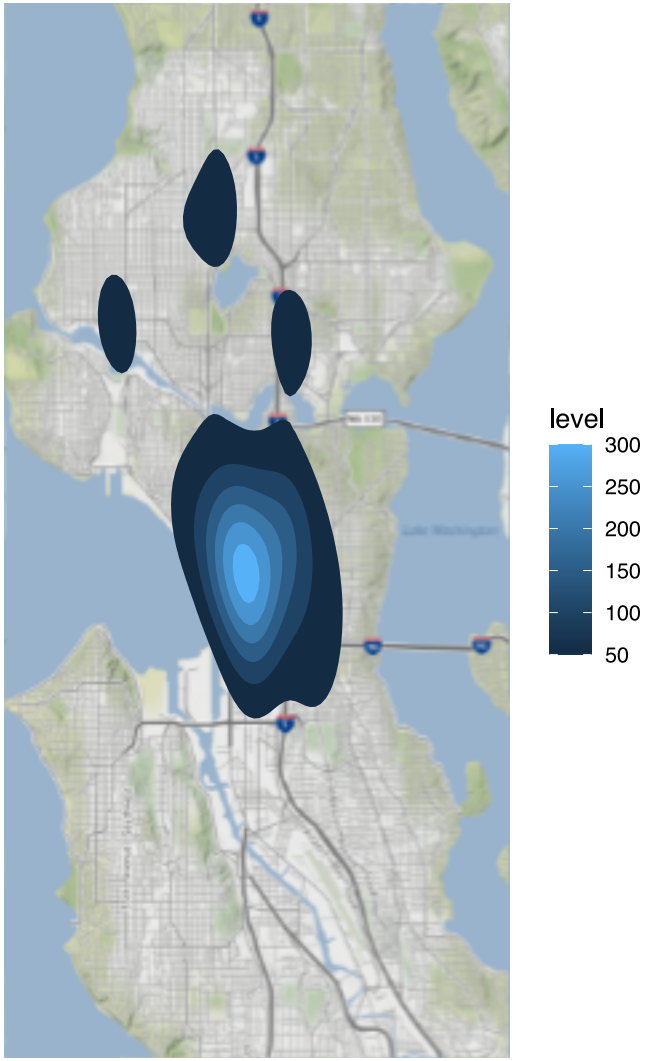
```
qmplot(data = spd_raw,
       geom = "blank",
       x = Longitude,
       y = Latitude)
```

Add Density Layer

Add `stat_density_2d` layer:

```
qmplot(data = spd_raw,
       geom = "blank",
       x = Longitude,
       y = Latitude)+
  stat_density_2d(
    aes(fill = stat(level)),
    geom = "polygon")
```

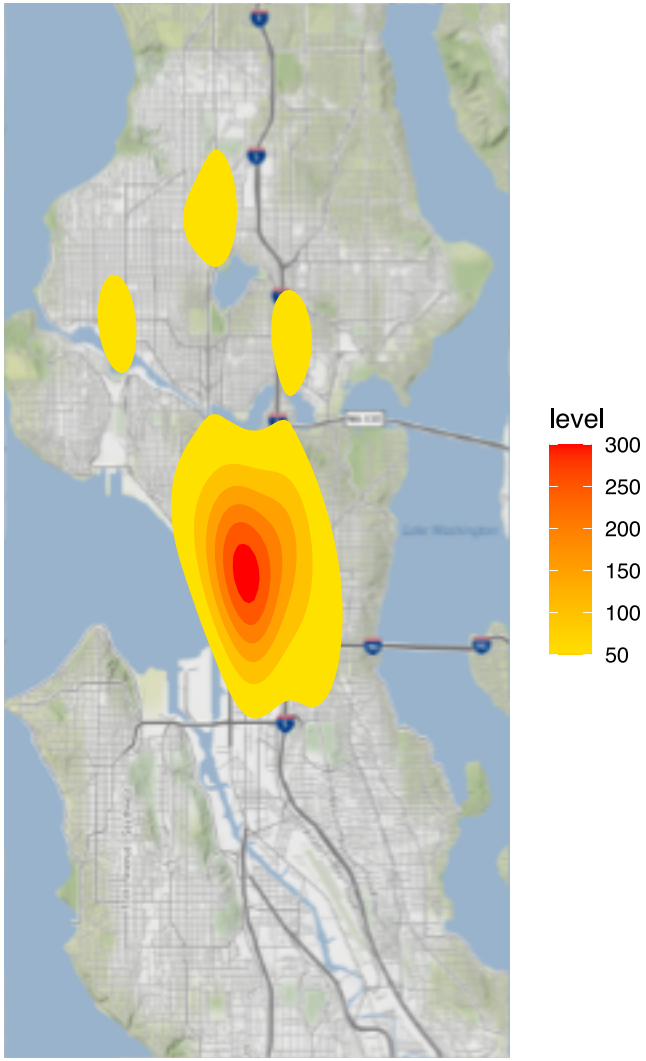
`stat(level)` indicates we want `fill=` to be based on `level` values calculated by the layer (i.e., number of observations).



Color Scale

Specify color gradient

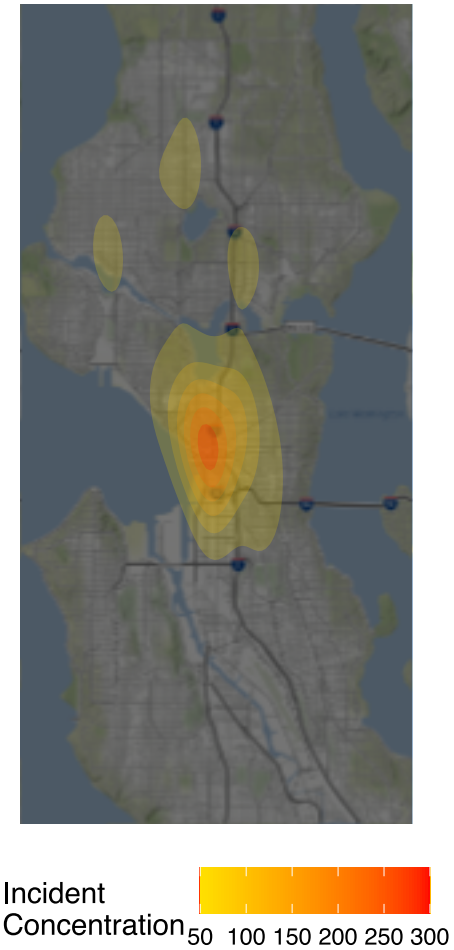
```
qmplot(data = spd_raw,
       geom = "blank",
       x = Longitude,
       y = Latitude)+
  stat_density_2d(
    aes(fill = stat(level)),
    geom = "polygon",
  )+
  scale_fill_gradient2(
    low = "white",
    mid = "yellow",
    high = "red")
```



Customize transparency and legend

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

Done!



Labeling points with `ggrepel`

Focus in on Downtown Seattle

First, let's filter our data 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)
```

We'll plot just these values from now on!

Let's also make a dataframe that includes just assaults and robberies downtown:

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

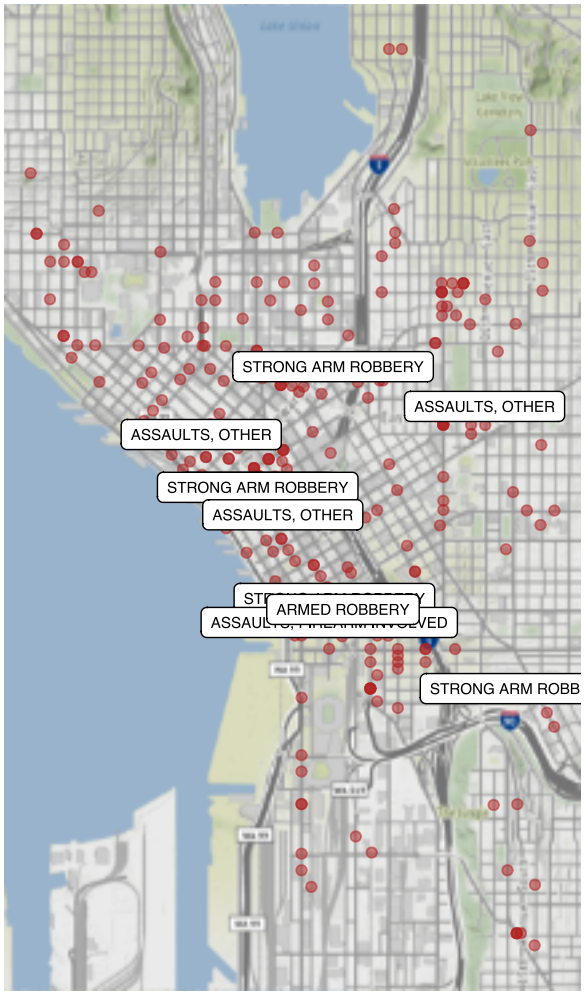
We'll **label** these observations!

Labels

Now let's plot the events and label them with `geom_label()`:

```
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),
       size=2)
```

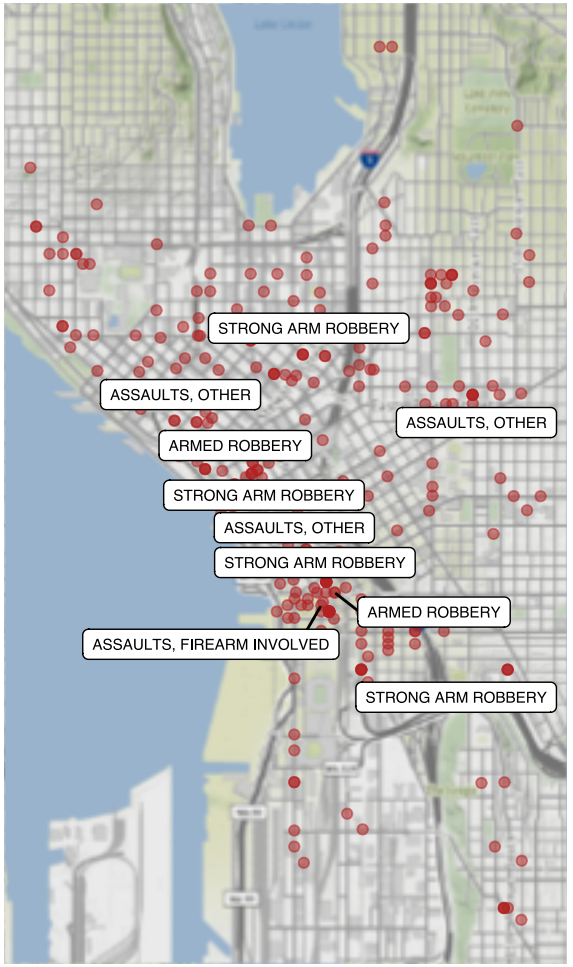
Note that one dataframe is used for points (`downtown`) and another for labels (`assaults`)!!



Fixing Overlapping Labels

The `ggrepel` package lets use fix or reduce overlapping labels using the function `geom_label_repel()`.

```
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),
    size=2)
```



2. Advanced mapping

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* (e.g. 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 in `sf`:

- **EPSG codes** (`epsg` in R)
  - Numeric codes which *refer to a predefined CRS*
  - Example: WGS84 is `4326`
- **PROJ.4 strings** (`proj4string` in R)
  - Text strings of parameters that *define a CRS*
  - Example: NAD83(NSRS2007) / Washington North

```
+proj=lcc +lat_1=48.73333333333333 +lat_2=47.5 +lat_0=47
+lon_0=-120.8333333333333 +x_0=500000 +y_0=0 +ellps=GRS80
+towgs84=0,0,0,0,0,0,0 +units=m +no_defs
```

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.

[2] 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_as_sf()`: Convert a data frame into an `sf` object.
- `st_join()`: Join data by spatial relationship.
- `st_transform()`: Convert between CRS.
- `st_drop_geometry()`: Remove geometry from a `sf` data frame.
- `st_relate()`: Compute relationships between geometries (like neighbor matrices).
- `st_interpolate_aw()`: Areal-weighted interpolation of polygons.[1]

[1] I recommend the dedicated `areal` package for this though!

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
##   `/Users/pearce790/CSSS508/Lectures/Lecture9/data/district/votdst.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 2592 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 1220179 ymin: 31555.16 xmax: 1583562 ymax: 287678
## Projected CRS: NAD83(HARN) / Washington North (ftUS)
```

If following along, click here to download a zip of the shapefile.

Voting Data: Processing

```
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) %>%
  st_as_sf() # Makes sure resulting object is an sf dataframe
```

Taking a `glimpse()`

```
glimpse(precincts_votes_sf)
```

```
## Rows: 960
## Columns: 5
## Groups: Precinct [960]
## $ Precinct <chr> "SEA 11-1256", "SEA 11-1550", "SEA 11-1552", "SEA 1…
## $ P_Dem    <dbl> 0.7707510, 0.8168421, 0.7507987, 0.8376328, 0.83259…
## $ P_Rep    <dbl> 0.15612648, 0.07789474, 0.13418530, 0.08649469, 0.0…
## $ Turnout  <dbl> 0.6931507, 0.7274119, 0.7347418, 0.7522831, 0.75792…
## $ geometry <MULTIPOLYGON [US_survey_foot]> MULTIPOLYGON (((1273698 1…
```

Notice the `geometry` column and its unusual class: `MULTIPOLYGON`

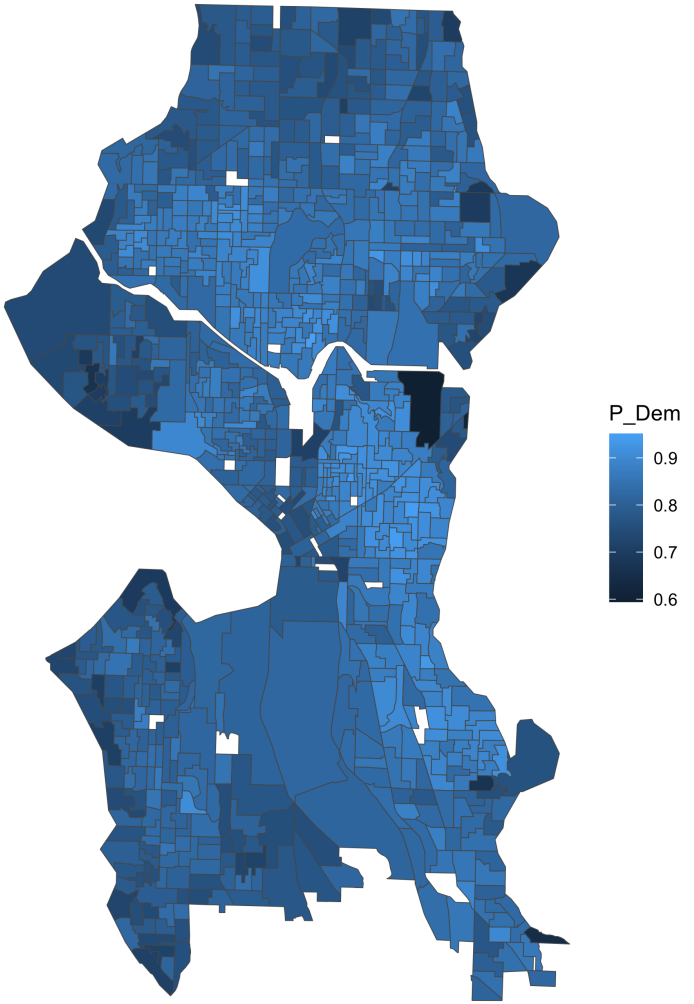
A single observation (row) has a geometry which may consist of multiple polygons.

Voting Map

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

```
ggplot(precincts_votes_sf,
       aes(fill=P_Dem)) +
  geom_sf(size=NA) +
  theme_void() +
  theme(legend.position =
          "right")
```

- `fill=P_Dem` maps color inside precincts to `P_Dem`.
- `size=NA` removes precinct outlines.
- `theme_void()` removes axes and background.



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.

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 × 4
##   name        label                                        concept geography
##   <chr>       <chr>                                        <chr>   <chr>
## 1 B01001A_008 Estimate!!Total!!Male!!20 to 24 years        SEX BY… tract
## 2 B01001A_009 Estimate!!Total!!Male!!25 to 29 years        SEX BY… tract
## 3 B01001A_010 Estimate!!Total!!Male!!30 to 34 years        SEX BY… tract
## 4 B01001A_011 Estimate!!Total!!Male!!35 to 44 years        SEX BY… tract
## 5 B01001A_012 Estimate!!Total!!Male!!45 to 54 years        SEX BY… tract
## 6 B01001A_013 Estimate!!Total!!Male!!55 to 64 years        SEX BY… tract
## 7 B01001A_014 Estimate!!Total!!Male!!65 to 74 years        SEX BY… tract
## 8 B01001A_015 Estimate!!Total!!Male!!75 to 84 years        SEX BY… tract
## 9 B01001A_016 Estimate!!Total!!Male!!85 years and o…       SEX BY… tract
```

Getting Data

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

What do these look like?

```
glimpse(king_county)
```

```
## Rows: 495
## Columns: 7
## $ GEOID       <chr> "53033010102", "53033005803", "53033004901", "53…
## $ NAME        <chr> "Census Tract 101.02, King County, Washington", …
## $ B02001_001E <dbl> 4539, 3268, 4126, 2574, 4032, 3274, 3421, 3652, …
## $ B02001_001M <dbl> 697, 394, 532, 473, 531, 534, 367, 306, 488, 455…
## $ B02009_001E <dbl> 876, 138, 143, 225, 1187, 229, 215, 276, 16, 101…
## $ B02009_001M <dbl> 445, 133, 95, 207, 382, 239, 144, 178, 16, 90, 4…
## $ geometry    <MULTIPOLYGON [°]> MULTIPOLYGON (((-122.291 47..., MUL…
```

With `output="wide"`, **estimates** end in `E` and *error margins* in `M`.

Processing Data

We can drop the margins of error, rename the estimates then, `mutate()` into a proportion `Any Black` measure.

```
king_county <-  king_county %>%
  select(-ends_with("M")) %>%
  rename(`Total Population`=B02001_001E,
         `Any Black`=B02009_001E) %>%
  mutate(`Any Black` = `Any Black` / `Total Population`)
glimpse(king_county)
```

```
## Rows: 495
## Columns: 5
## $ GEOID              <chr> "53033010102", "53033005803", "5303300490…
## $ NAME               <chr> "Census Tract 101.02, King County, Washin…
## $ `Total Population` <dbl> 4539, 3268, 4126, 2574, 4032, 3274, 3421,…
## $ `Any Black`        <dbl> 0.192994052, 0.042227662, 0.034658265, 0.…
## $ geometry           <MULTIPOLYGON [°]> MULTIPOLYGON (((-122.291 47.…
```

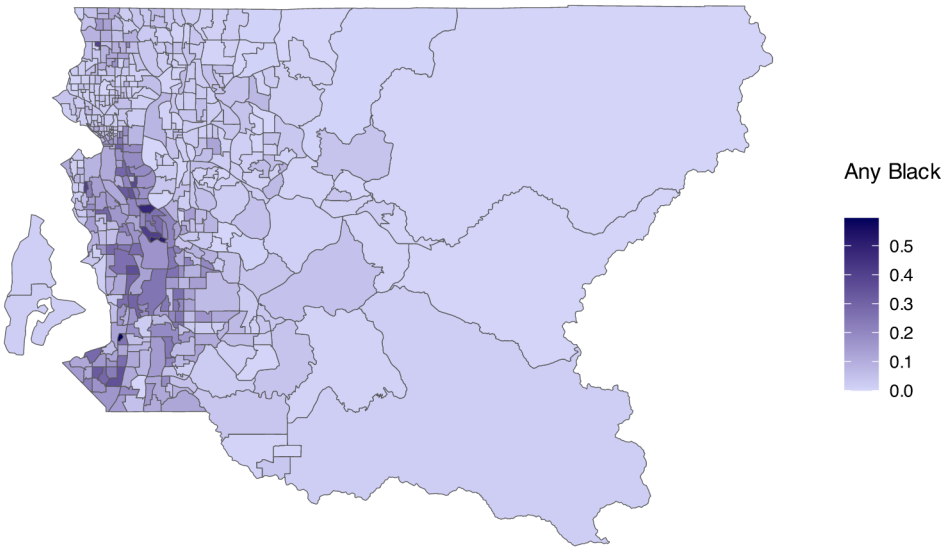
Mapping Code

```
king_county %>%
  ggplot(aes(fill = `Any Black`)) +
  geom_sf(size = NA) +
  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.
  - `size = NA` removes outlines
- `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

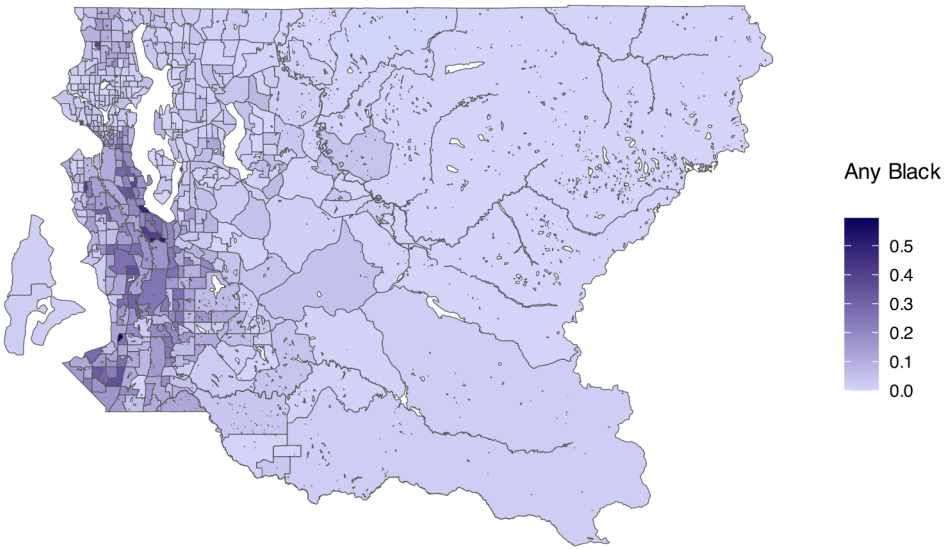
With a simple function and boundaries of water bodies in King County, we can replace water with empty space.

```
st_erase <- function(x, y) {
  st_difference(x, st_make_valid(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`
- `st_make_valid()` fixes geometry errors from subtraction
- `area_water()` obtains `sf` geometry of water bodies

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

Proportion Any Black



Lab/Homework

For the remainder of class, we'll work on Homework 8, which is due next week!

- The first part is based on Lecture 8 (strings)
  - This includes lab questions from last lecture!
- The second part is based on Lecture 9 (mapping)
  - This includes an analysis of the restaurant data from last week!

There is no lab next week due to Memorial Day!