Geographic Information Systems (GIS) in R

Welcome!

Jennifer Lin

PS 490: R Workshop

2022-05-04

Goals

1. Become familiar with how to get geographic data and how to parse that data.
2. Become familiar with shape files, how to access them and how to plot them using `ggplot2`
3. Become familiar with distance calculations and learn to add additional layers of data to an existing map.

Assumptions

1. You have broad-based familiarity with `ggplot2`. There is no need to know how to write a chunk of `ggplot` code from memory, but you should have an understanding of what each line of code does.
2. You should know how to install and read in packages in R. We will cover many packages today.
3. You can load in data based in different file types wrangle data in `dplyr`. Again, there is no need to know how to wrangle data off memory, but you should be familiar with functions like `filter()`, `select()` and `mutate()`. You should also know how to use a pipe (`%>%`).

What is GIS?

> A geographic information system (GIS) is **a system that creates, manages, analyzes, and maps all types of data**. GIS **connects data to a map, integrating location data (where things are) with all types of descriptive information (what things are like there)**. This provides a foundation for mapping and analysis that is used in science and almost every industry. GIS helps users understand patterns, relationships, and geographic context. The benefits include improved communication and efficiency as well as better management and decision making.

Why Make Maps?

> Mapping is important for political scientists because pinpointing where a particular political phenomenon occurred often provides clues as to why it occurred.
>
> -- Cho and Gimpel, 2012, *Annual Reviews*

Applications of GIS Tools to Political Science

Politics is inherently spatial. See Cho and Gimpel, 2012

1. Electoral Redistricting
2. Political Representation
3. Political Participation
4. Political Attitudes
5. Social Interactions

... and so on.

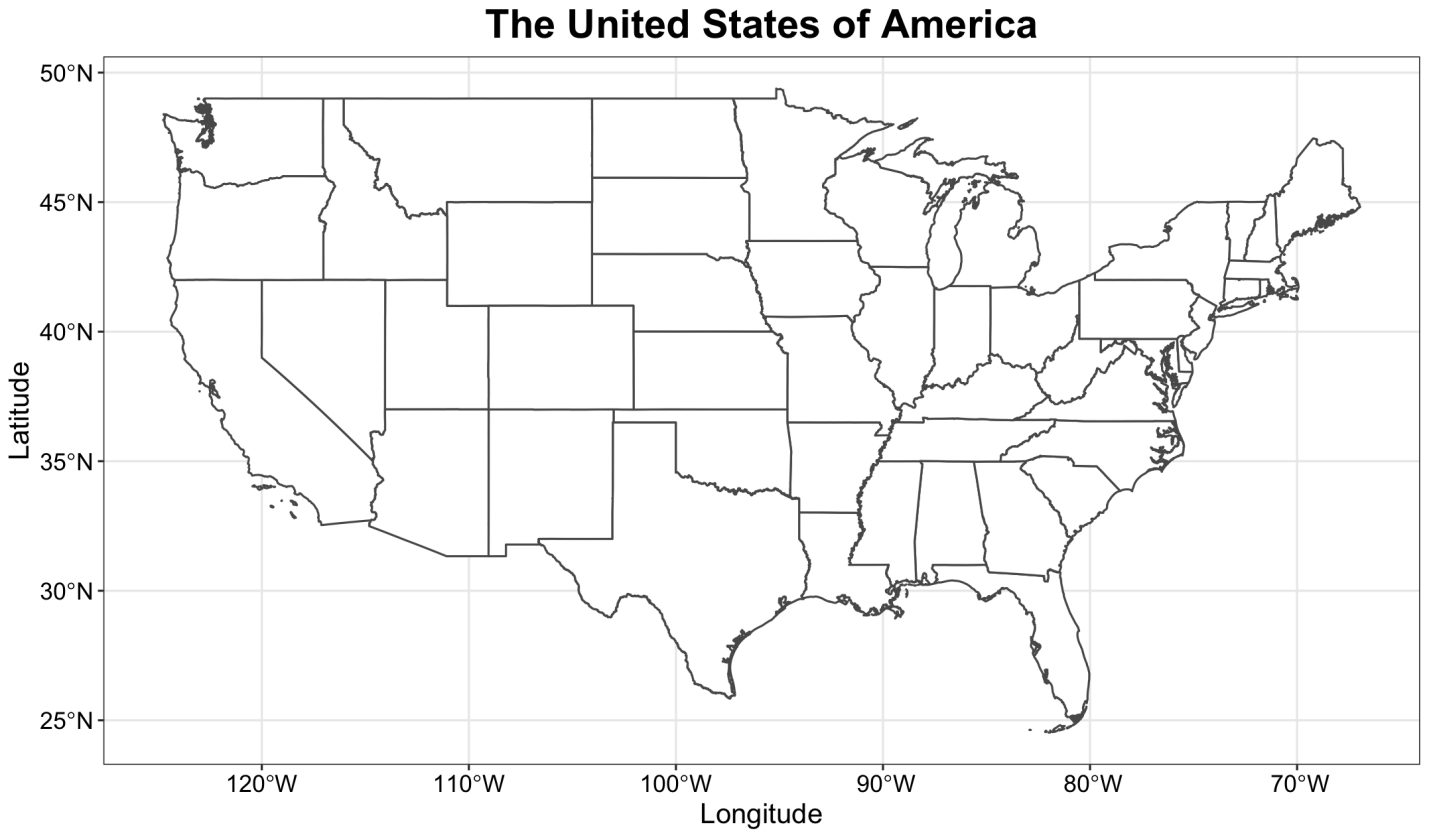
Review of Geography

Recall:

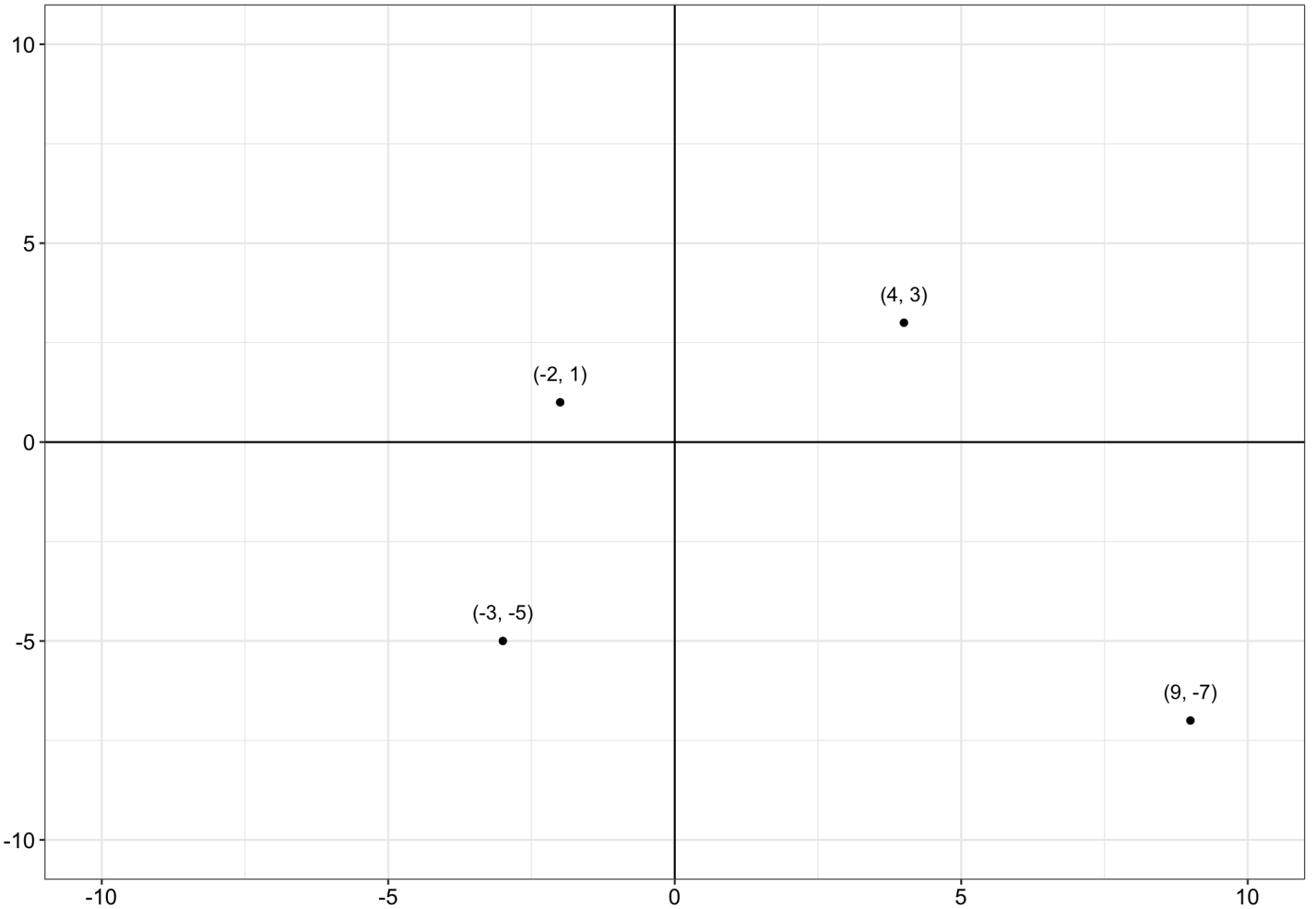
In elementary school, you might have learned how to interpret a map or a globe. There, you may have encountered the terms "latitude" and "longitude". Google defines these terms as follows:

latitude : The angular distance of a place north or south of the earth's equator, or of a celestial object north or south of the celestial equator, usually expressed in degrees and minutes.

longitude : The angular distance of a place east or west of the meridian at Greenwich, England, or west of the standard meridian of a celestial object, usually expressed in degrees and minutes.


The United States of America
50°N
45°N
40°N
35°N
30°N
25°N
Latitude
120°W
110°W
100°W
90°W
80°W
70°W
Longitude

The Coordinate Plane



The Distance Formula

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The distance between the points $(-2, 1)$ and $(4, 3)$ is

$$\begin{aligned} d &= \sqrt{(4 - (-2))^2 + (3 - 1)^2} \\ d &= \sqrt{36 + 4} \\ d &= \sqrt{40} \approx 6.324555 \end{aligned}$$

Why?

1. Geocoding -- Places location points in terms of longitude and latitude coordinates
2. Mapping with shape files -- The Earth is a large coordinate plane and shape files give us these location points too
3. Distance -- Application of the distance formula but factors in the curvature of the Earth

Data Sources

Geographic Data

Geographic data are data that give us information about where a place is in reference to the Earth's coordinate grid.

- Latitude and Longitude coordinates
- Government developed reference systems (i.e. Federal Information Processing Standards (FIPS) Codes)

```
data("fips_codes")
```

Examples

- Number of protests at a given location
- Number of political rallies at a given location
- Number of residents registered to vote in a household
- COVID Testing location hours of operation (Here are data from Wisconsin).

Geocoding

> Geocoding is the process of attaching geographic coordinates (usually as latitude and longitude or as x–y coordinates) to data that are in more traditional geographic formats, such as, for example, street addresses, zip codes, district numbers, county identifiers, or FIPS (Federal Information Processing Standard) codes. (Cho and Gimpel, 2012)

Geocoding -- Example

Suppose we want to geocode Scott Hall (601 University Place, Evanston, IL 60208)

There are three different ways we can do this

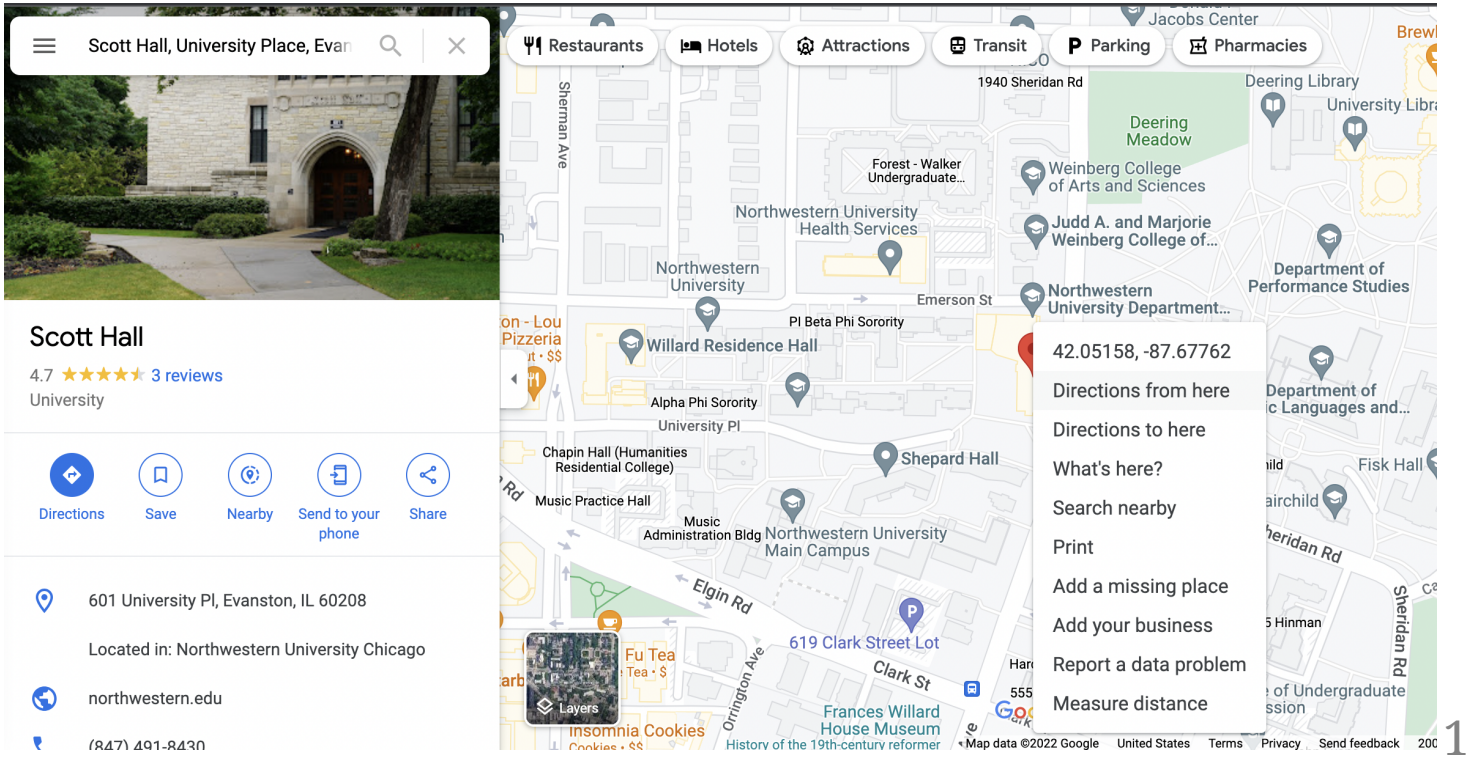
1. Brute Force: Google Maps
2. Brute Force: Census Geocoder API
3. By Script: `tidygeocoder`

```
geolocate <- data.frame(
  place = "Scott Hall",
  address = "601 University Place, Evanston, IL 60208"
)
```

Geocoding -- Brute Force Method

Google Maps

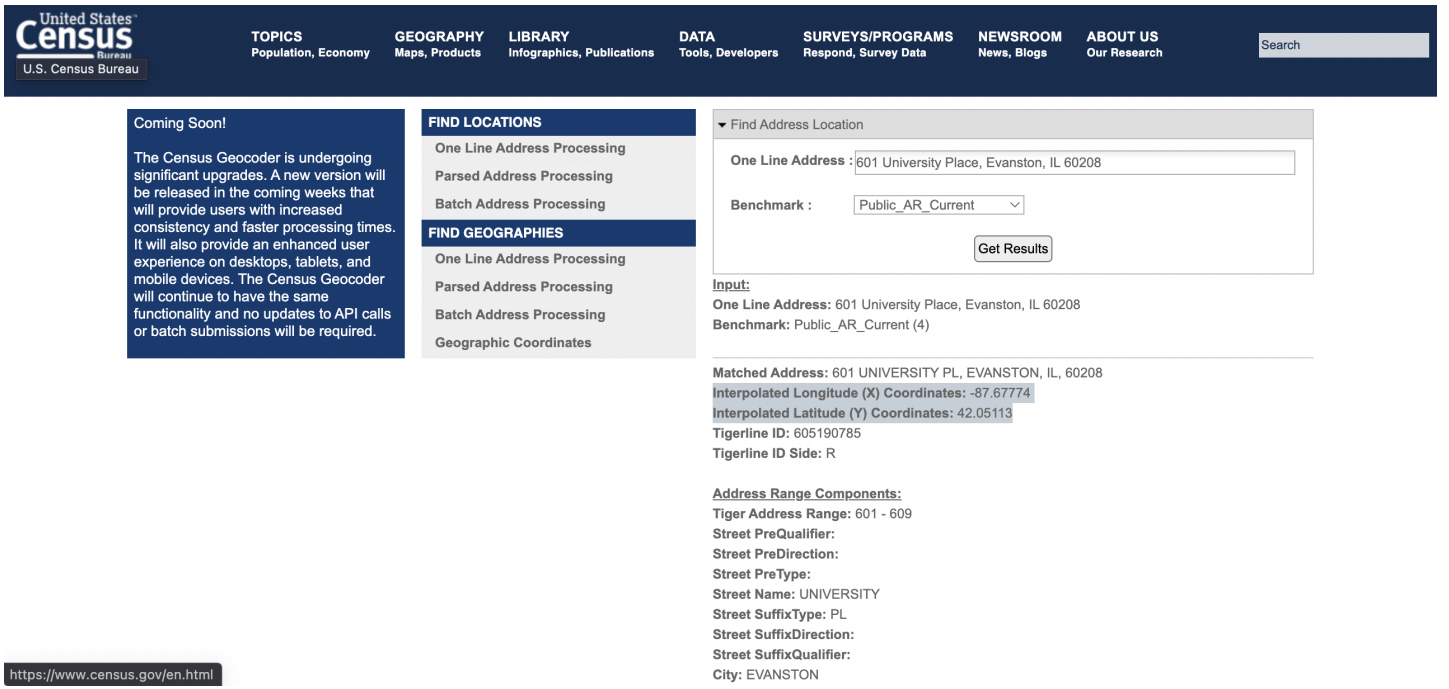
Scott Hall is at Lat = 42.05 and Long = -87.68



Geocoding -- Brute Force Method

US Census API

Scott Hall is at Lat = 42.05 and Long = -87.68



Geocoding -- Script

`tidygeocoder`

Provides two open source options: OpenStreetMaps (`osm`) and Census Geocoder (`census`).

Let's see OSM first

```
osm_geo <- geolocate %>%
  geocode(address, method = 'osm', lat = latitude , long = longitu
osm_geo
```

```
## # A tibble: 1 × 4
##   place      address                          latitude longitude
##   <chr>      <chr>                               <dbl>     <dbl>
## 1 Scott Hall 601 University Place, Evans…         42.1     -87.7
```

Geocoding -- Script

`tidygeocoder`

Now Census

```
census_geo <- geolocate %>%
  geocode(address, method = 'census', lat = latitude , long = long
census_geo
```

```
## # A tibble: 1 × 4
##   place      address                          latitude longitude
##   <chr>      <chr>                               <dbl>     <dbl>
## 1 Scott Hall 601 University Place, Evans…         42.1     -87.7
```

Reverse Geocoding

Takes a set of longitude and latitude codes and attempts to match it with an address

```
reverse_osm <- census_geo %>%
  reverse_geocode(lat = latitude, long = longitude, method = 'osm'
                  address = address_found, full_results = TRUE)
```

```
##                [,1]
## place          "Scott Hall"
## address        "601 University Place, Evanston, IL 60208"
## latitude       42.05113
## longitude      -87.67774
## address_found "Northwestern University, 633, Clark Street, Downtown, Eva
## place_id       282635476
## licence        "Data © OpenStreetMap contributors, ODbL 1.0. https://osm.
## osm_type       "relation"
## osm_id         2105485
## osm_lat        "42.0551164"
## osm_lon        "-87.67581113482166"
## amenity        "Northwestern University"
## house_number   "633"
## road           "Clark Street"
## neighbourhood "Downtown"
## town           "Evanston"
## municipality   "Evanston Township"
## county         "Cook County"
## state          "Illinois"
## postcode       "60208"
## country        "United States"
## country_code   "us"
## boundingbox    character,4
```

Census GEOID

In the United States, each geographic area is identified using the Census GEOID, which is a 15 digit numeric string that describes the state (S), county (C), census tract (T) and census block (B) such that it is [SS][CCC][TTTTTT][BBBB].

We can get this ID for any address using `tigris::call_geolocator_latlon()`.

```
call_geolocator_latlon(census_geo$latitude, census_geo$longitude)
```

For Scott Hall:

```
call_geolocator_latlon(census_geo$latitude, census_geo$longitude)
```

```
## [1] "170318087021003"
```

- State: 17
- County: 17031
- Census Tract: 17031808702
- Census Block: 170318087021003

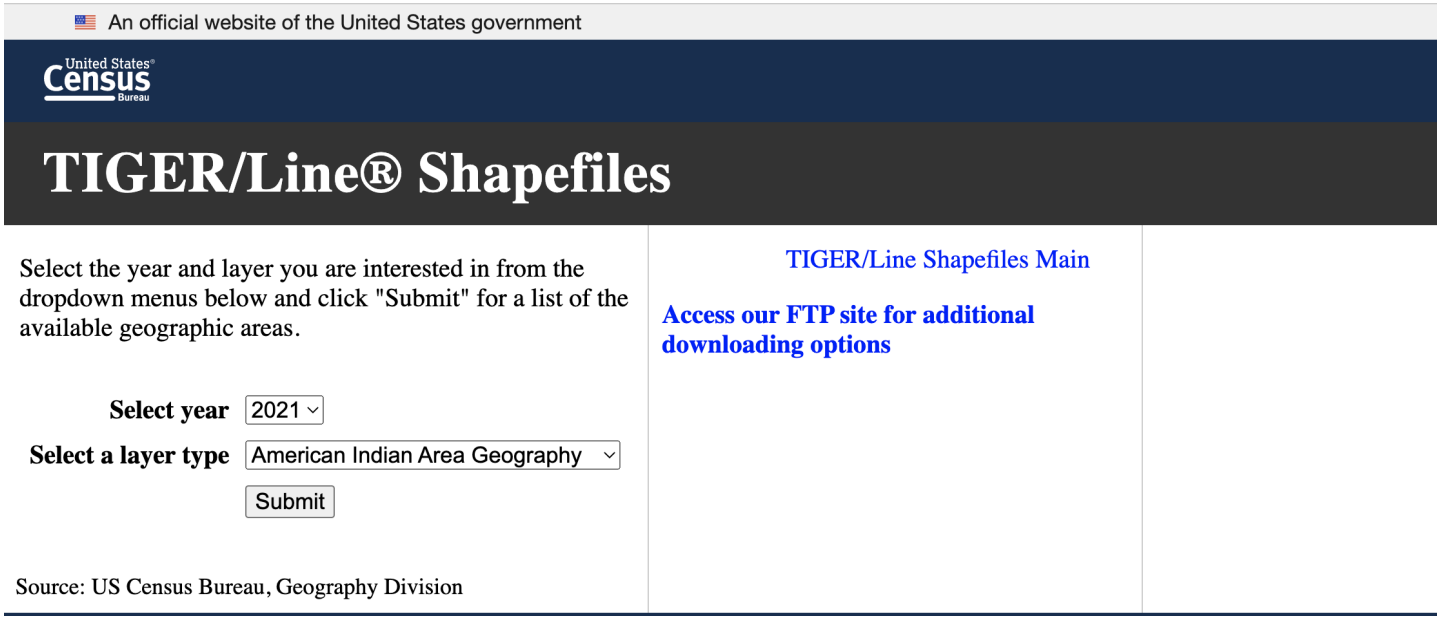
Shape Files

- Data frames that provide rough outlines for map features
- Includes geographic points to show where the boundaries are or where locations are.
- Includes information on projections, which is often more implicit in the dataset.

Shape Files

Getting Data -- Brute Force

https://www.census.gov/cgi-bin/geo/shapefiles/index.php



Shape Files

Getting Data -- Brute Force

Two ways:

1. Point and Click download from the Census TIGER archive
2. Modify the following code:

```
url <- "https://www2.census.gov/geo/tiger/"
extension <- "TIGER2021/TRACT/"
file <- "tl_2021_11_tract.zip"
shp_url <- paste0(url, extension, file)

if (! dir.exists(here::here("data/DC_Tract/"))) {
  download.file(
    shp_url,
    dest = here::here("data/DC_Tract.zip")
  )
  unzip(here::here("data/DC_Tract.zip"),
        exdir = here::here("data/DC_Tract"))
}
```

Shape Files

Getting Data -- Brute Force

The Brute Force method is good for us to see what is actually in the back end for shape files.

```
dir(here::here("data/DC_Tract/"))
```

```
## [1] "tl_2021_11_tract.cpg"
## [2] "tl_2021_11_tract.dbf"
## [3] "tl_2021_11_tract.prj"
## [4] "tl_2021_11_tract.shp"
## [5] "tl_2021_11_tract.shp.ea.iso.xml"
## [6] "tl_2021_11_tract.shp.iso.xml"
## [7] "tl_2021_11_tract.shx"
```

That is a lot of files! All you need to read in is the `.shp`.

```
DC_Tract_sf <- read_sf(
  here::here("data/DC_Tract/tl_2021_11_tract.shp")
```

Shape Files

Getting Data -- `tigris`

The Brute Force method is useful if you have data that does not come from an API. Thankfully, the Census has an easier way to get the shapes because we can leverage the API.

The `tigris` package provides commands to get us shape files

```
DC_County <- counties("11", year = 2020)
DC_Tract <- tracts("11", year = 2020)
```

This does the same thing

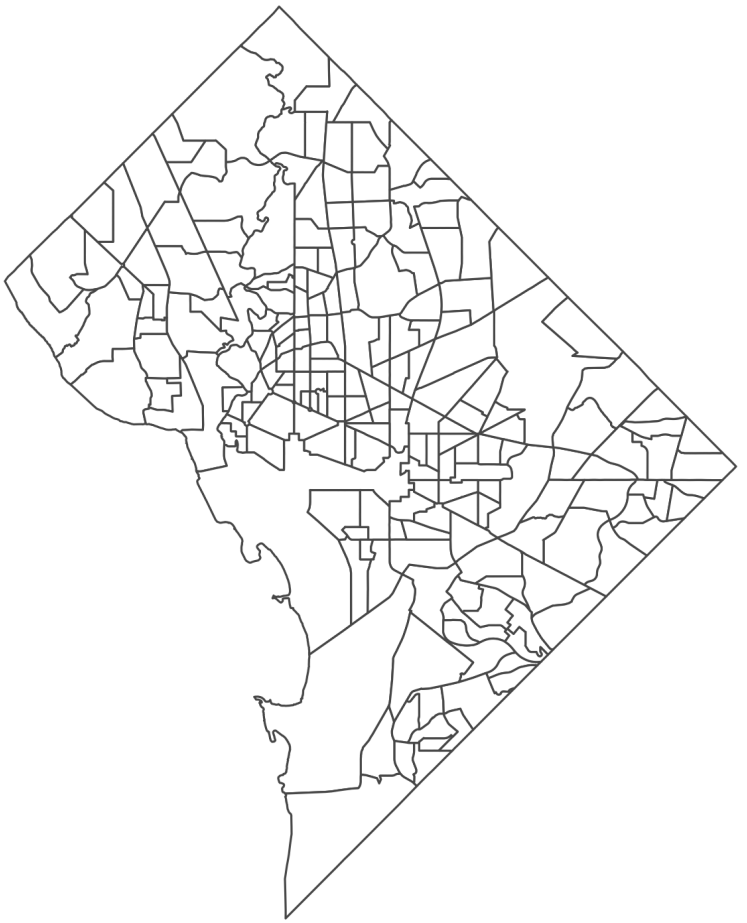
```
DC_County <- counties("DC", year = 2020)
DC_Tract <- tracts("DC", year = 2020)
```

With each shape file, we want to get a sense of its structure.

```
st_crs(DC_County)
```

```
## Coordinate Reference System:
##   User input: NAD83
##   wkt:
## GEOGCRS["NAD83",
##     DATUM["North American Datum 1983",
##         ELLIPSOID["GRS 1980",6378137,298.257222101,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4269]]
```

That was all the projection information to give us this map.



Exercise

1. Replicate the DC shape file read in.
2. Select a state of your choice. Get the county shape file from the Census database.

Map-Making

Data

We previously got part of the data to make a map.

Now, we should also get some other data, like physical locations or other area level data, like vote share

Here, we will use the Washington DC Places of Interest from the DC government.

```
url <- "https://opendata.arcgis.com/api/v3/datasets/"
extension <- "f323f677b3f34fe08956b8fcce3ace44_3/downloads/"
file <- "data?format=shp&spatialRefId=4326"
data_url <- paste0(url, extension, file)

if (! dir.exists(here::here("data/DC_POI/"))) {
  download.file(
    data_url,
    dest = here::here("data/DC_POI.zip")
  )
  unzip(here::here("data/DC_POI.zip"),
        exdir = here::here("data/DC_POI"))
}
```

This folder is a shape file too.

```
dir(here::here("data/DC_POI/")
```

x
Points_of_Interest.cpg
Points_of_Interest.dbf
Points_of_Interest.prj
Points_of_Interest.shp
Points_of_Interest.shx
Points_of_Interest.xml

Since there is no API to use with reading this shape file in, we need to brute force it.

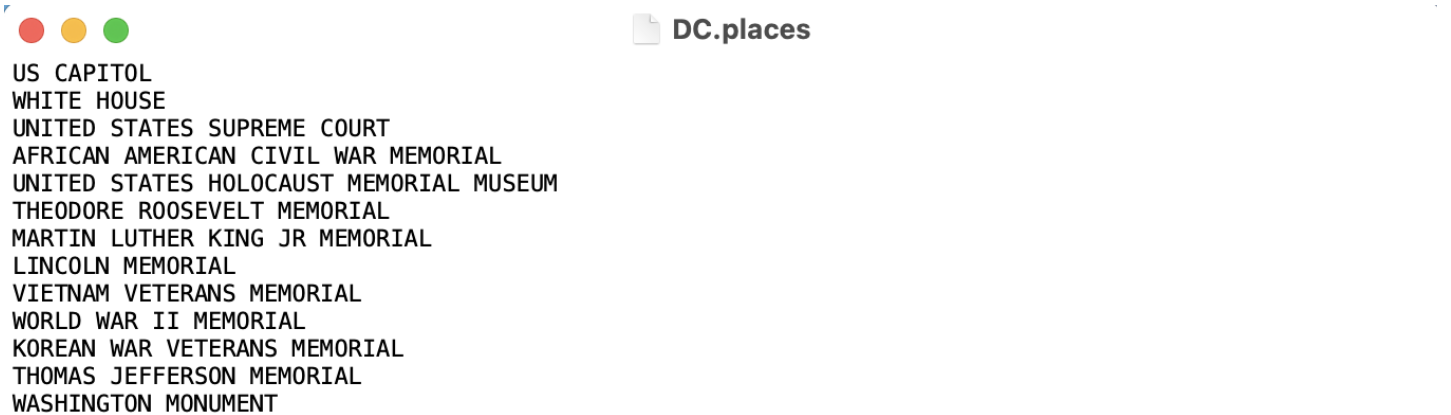
```
DC_POI <- read_sf(
  here::here("data/DC_POI/Points_of_Interest.shp")
) %>%
  mutate(
    long = unlist(map(geometry,1)),
    lat = unlist(map(geometry,2))
  )
```

I have selected a few places of interest that we can use

```
Select_Places <- readLines("DC.places")

head(Select_Places)
```

```
## [1] "US CAPITOL"
## [2] "WHITE HOUSE"
## [3] "UNITED STATES SUPREME COURT"
## [4] "AFRICAN AMERICAN CIVIL WAR MEMORIAL"
## [5] "UNITED STATES HOLOCAUST MEMORIAL MUSEUM"
## [6] "THEODORE ROOSEVELT MEMORIAL"
```



As always, it is a good idea to look at the projections and overall structure.

```
st_crs(DC_POI)
```

```
## Coordinate Reference System:
##   User input: WGS 84
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4326]]
```

Notice anything different from the previous one?

Coordinate Reference Systems

The DC Census Tract file the NAD83 projection (CRS 4269).

The DC Places uses WGS 84 (CRS 4326)

We need to so some conversions in order for a map to work. Use `st_transform()`

```
DC_Destinations <- DC_POI %>%
  filter(ALIASNAME %in% Select_Places) %>%
  st_transform(crs = 4269)
```

```
st_crs(DC_Destinations)
```

```
## Coordinate Reference System:
##   User input: EPSG:4269
##   wkt:
## GEOGCRS["NAD83",
##     DATUM["North American Datum 1983",
##         ELLIPSOID["GRS 1980",6378137,298.257222101,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["Geodesy."],
##         AREA["North America - onshore and offshore: Canada - Alberta; Br
##         BBOX[14.92,167.65,86.46,-47.74]],
##     ID["EPSG",4269]]
```

Mapping a Vacation

To put points on a map, let's pick a few locations of interest.

```
vacation <- c(
  "THOMAS JEFFERSON MEMORIAL",
  "UNITED STATES HOLOCAUST MEMORIAL MUSEUM",
  "US DEPARTMENT OF THE TREASURY",
  "US NAVY MUSEUM",
  "NATIONAL ZOO",
  "US CAPITOL",
  "DUPONT CIRCLE"
)
```

Take these locations out of the main data frame since this is all that we want.

```
chosen_places <- DC_Destinations %>%
  filter(ALIASNAME %in% vacation) %>%
  select(ALIASNAME, long, lat)
```

Distance Calculations

Haversine Distance

The straight line, "as the crow flies", distance between two points while keeping in mind the curvature of the Earth.

$$\begin{aligned}\operatorname{haversin}\left(\frac{d}{r}\right) &= \operatorname{haversin}(\phi_{\text{p1 lat}} - \phi_{\text{p2 lat}}) \\ &+ \cos(\phi_{\text{p2 lat}})\cos(\phi_{\text{p1 lat}})\operatorname{haversin}(\lambda_{\text{p1 long}} - \lambda_{\text{p2 long}})\end{aligned}$$

When we rearrange the formula, this is what we are calculating when we calculate distance

$$d = 2r$$

$$\arcsin\left(\sqrt{\sin^2\frac{\phi_{\text{p1 lat}} - \phi_{\text{p2 lat}}}{2} + \cos(\phi_{\text{p2 lat}})\cos(\phi_{\text{p1 lat}})\sin^2\frac{\lambda_{\text{p1 long}}}{}\right.$$

Preparing the Data

1. Generate a random (or purposeful) to and from list
2. Label start and end points as such

```
chosen_places <- cbind(
  slice(chosen_places, 1:n()),
  slice(chosen_places, 2:n(), 1)
) %>%
  rename(
    from       = ALIASNAME,
    start_long = long,
    start_lat  = lat,
    to         = ALIASNAME.1,
    end_long   = long.1,
    end_lat    = lat.1
  ) %>%
  select(-contains("geometry"))
```

Calculate the Distance

`geosphere::distHaversine()` allows us to easily apply the Haversine math to points.

Returns results in meters, which we convert to miles and kilometers at the end.

```
chosen_places <- chosen_places %>%
  mutate(
    distance = distHaversine(
      p1 = cbind(start_long, start_lat),
      p2 = cbind(end_long, end_lat))
    ) %>%
  mutate(
    miles = distance/1609.344,
    km    = distance/1000
  )
```

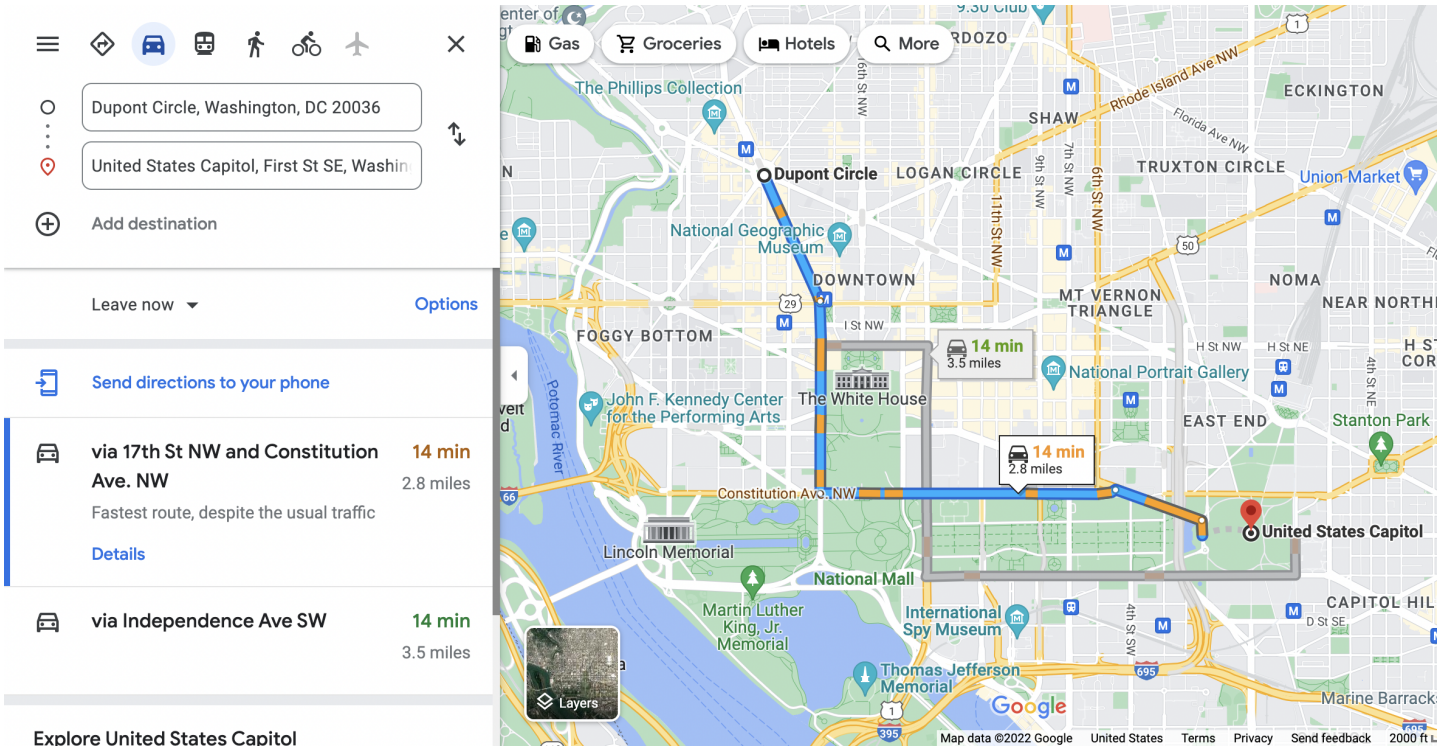
Let's look at the results.

```
chosen_places %>%
  select(from, to, miles) %>%
  st_set_geometry(NULL)
```

from	to	miles
US DEPARTMENT OF THE TREASURY	NATIONAL ZOO	2.491286
NATIONAL ZOO	UNITED STATES HOLOCAUST MEMORIAL MUSEUM	3.256168
UNITED STATES HOLOCAUST MEMORIAL MUSEUM	DUPONT CIRCLE	1.691316
DUPONT CIRCLE	US CAPITOL	2.305162
US CAPITOL	US NAVY MUSEUM	1.303710
US NAVY MUSEUM	THOMAS JEFFERSON MEMORIAL	2.306954
THOMAS JEFFERSON MEMORIAL	US DEPARTMENT OF THE TREASURY	1.165118

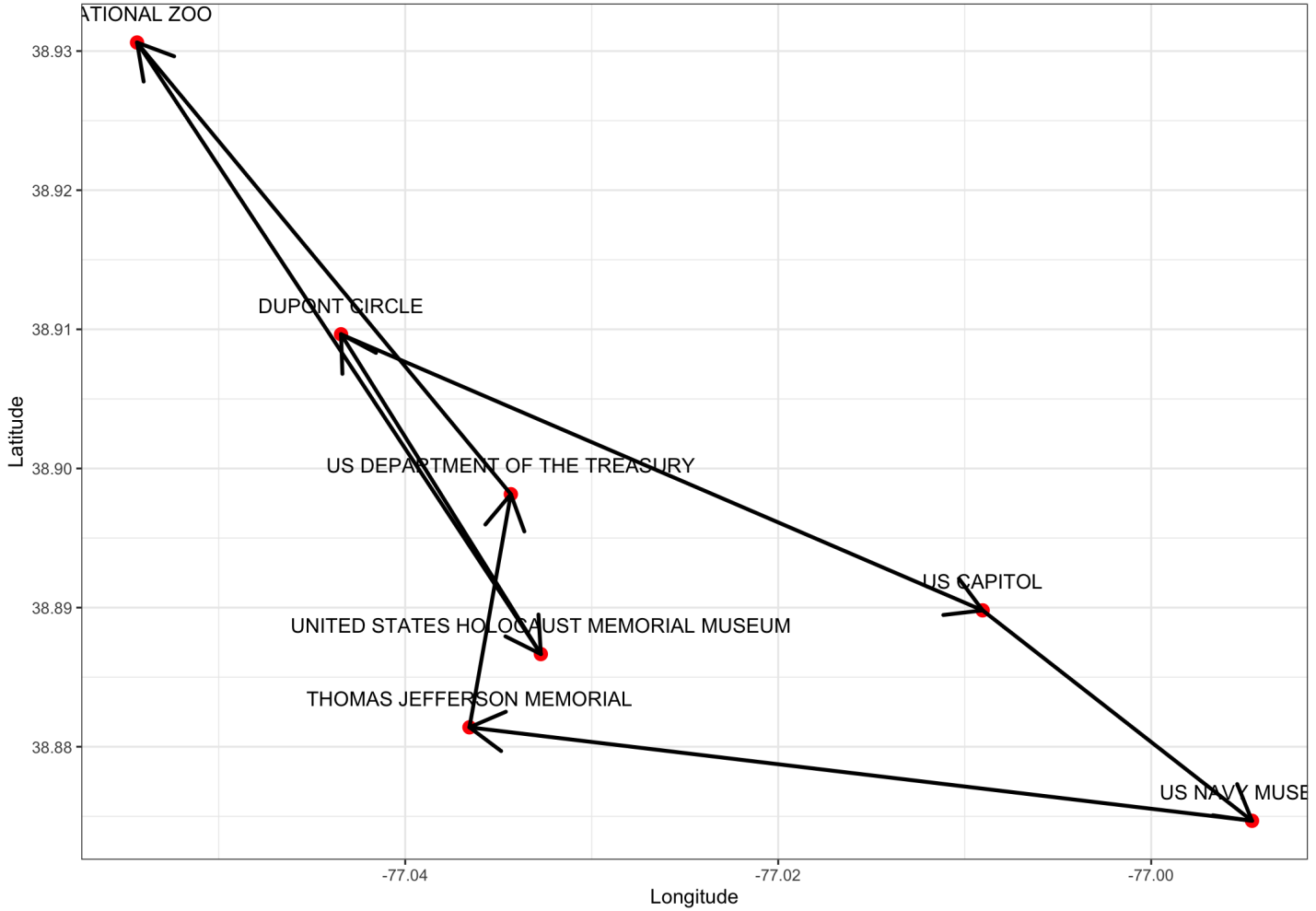
How did we do?

We got 2.3 mines from US Capitol to Dupont Circle.



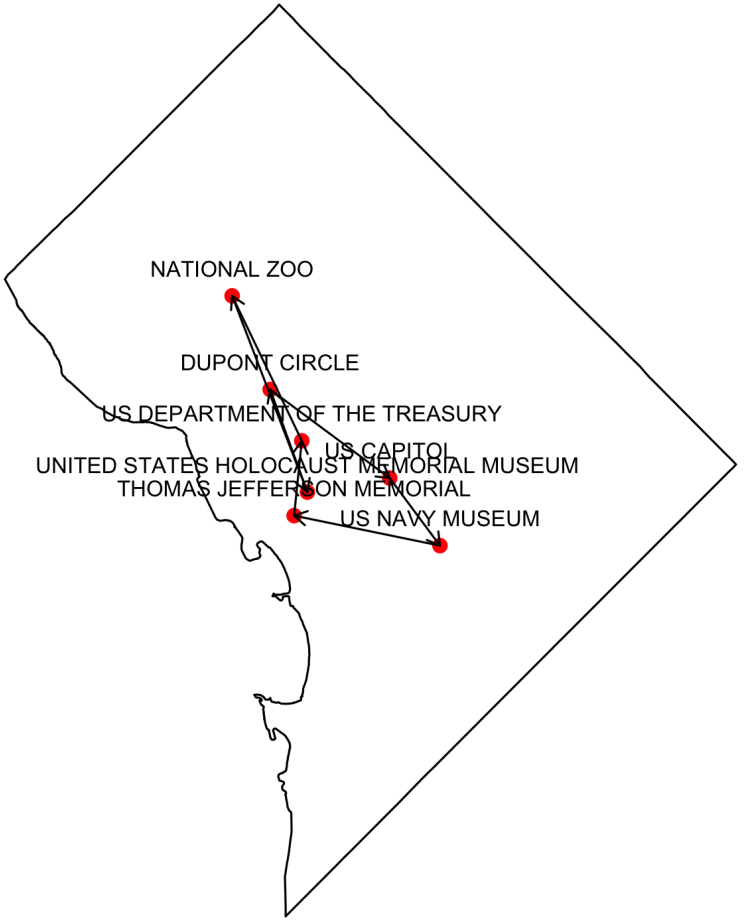
We can plot this movement using a standard `ggplot2`

```
ggplot(chosen_places, aes(x = start_long, y = start_lat))+
  geom_point(size = 3, color = "red")+
  geom_text(aes(label = from), vjust = -1.5)+
  geom_segment(
    aes(
    x = start_long,
    y = start_lat,
    xend = end_long,
    yend = end_lat
    ),
    arrow = arrow(length = unit(0.3, "inches")),
    size = 1,
    lineend = "round",
    linejoin = "round"
  )+
  xlab("Longitude")+
  ylab("Latitude")+
  theme_bw()
```


ATIONAL ZOO
DUPONT CIRCLE
US DEPARTMENT OF THE TREASURY
US CAPITOL
UNITED STATES HOLOCAUST MEMORIAL MUSEUM
THOMAS JEFFERSON MEMORIAL
US NAVY MUSE
38.93
38.92
38.91
38.90
38.89
38.88
Latitude
-77.04
-77.02
-77.00
Longitude

... Or put this in context of the DC county map. This is where having the same CRS is important!

```
ggplot(DC_County)+
  geom_sf(fill = "white", color = "black")+
  geom_point(
    chosen_places,
    mapping = aes(x = start_long, y = start_lat),
    size = 3, color = "red")+
  geom_text(
    chosen_places,
    mapping = aes(
      x = start_long, y = start_lat,
      label = from),
    vjust = -1.25)+
  geom_segment(
    chosen_places,
    mapping = aes(
      x = start_long,
      y = start_lat,
      xend = end_long,
      yend = end_lat
    ),
    arrow = arrow(length = unit(0.1, "inches")),
    size = 0.5,
    lineend = "round",
```


NATIONAL ZOO
DUPONT CIRCLE
US DEPARTMENT OF THE TREASURY
US CAPITOL
UNITED STATES HOLOCAUST MEMORIAL MUSEUM
THOMAS JEFFERSON MEMORIAL
US NAVY MUSEUM

Exercise

1. Pick a handful of locations from the DC places file.
2. Calculate the distance between the locations.
3. Place the locations on a map, tracing a plausible path for a tourist vacation.

CHALLENGE: Take the state for which you got the county shape file from earlier in this session. Pick one county. Make a map that shades that county as a different color.

Solution to CHALLENGE

```
FL <- counties("FL", cb = TRUE)

FL <- FL %>%
  mutate(
    pick = case_when(
      NAME == "Broward" ~ "selected",
      NAME == "Sarasota" ~ "selected",
      TRUE ~ "not selected"
    ))

ggplot(FL)+
  geom_sf(aes(fill = pick))+
  scale_fill_manual(
    breaks = c("selected", "not selected"),
    values = c(
      "selected" = "grey50",
      "not selected" = "white"
    ))+
  theme_rgis_void()+
  theme(
    legend.position = 'none'
  )
```