Spatial Data Basics

Prepared for PAP New Jersey Trainings

Kelly Pierce, Texas Advanced Computing Center

February 10, 2022

Contents

Coordinate reference systems (CRS) and projected reference systems **1**

Example: New Jersey Bus Stops **2**

Load geographic data from CSV and convert to simple features. 2

Load a shapefile from file 4

Identify the coordinate reference system and geometry type 4

Change the coordinate reference system 6

Map the bus stops 7

Basic point map 7

Add a basemap 8

Get a bounding box for the basemap 8

Load street data basemap 8

Map bus stops on streetmap 9

Population choropleth map 9

Aggregate bus stops into a grid 11

Create a five-mile “fishnet” grid 11

Point-in-polygon join 13

Map gridded bus stop counts by type 14

Exercise: New Jersey Hospital Locations **16**

Coordinate reference systems (CRS) and projected reference systems

We use two reference systems, depending on the task:

- WGS84 coordinate reference system is *unprojected*; it references points on an ellipsoid surface. Paradoxically, we use this reference system for drawing maps because the functions we use for map-making understand this reference system and know how to project it to a planar surface. Coordinates in this reference system are measured in decimal degrees or degrees:minutes:seconds.
- The New Jersey State Planar *projected* reference system is locally tailored for the state of New Jersey for accurate spatial representations, measured in meters, and can be used for euclidean distance calculations that would otherwise be distorted if attempted on an ellipsoid reference system such as WGS84. However, mapping functions generally do not know how to interpret this reference system.

The ArcGIS blog has a nice explanation of geographic coordinate systems and projected coordinate systems: https://www.esri.com/arcgis-blog/products/arcgis-pro/mapping/gcs_vs_pcs/ if you'd like further reading.

These coordinate reference systems can be referred to by standardized codes. We'll define two constants referring to those codes here, and we'll refer to the coordinate references by these variable names later on.

```
WGS84 = 4326
NJ_PLANAR = 'ESRI:102311'
```

Example: New Jersey Bus Stops

We'll use New Jersey bus stop locations to explore some of the analyses you can perform with spatial data. Navigate to New Jersey Geographic Information Network (https://njogis-newjersey.opendata.arcgis.com/datasets/NJTRANSIT::bus-stops-of-nj-transit/explore) and download the shapefile dataset by clikcing on the download icon in the right-hand panel and selecting the "Shapefile" option. This will download the data as a `zip` archive. Unpack the `zip` archive and make note of the directory path. As you work through these exercises on your own, you can either move the shapefile directory to your working directory and load with a relative file path, or load the data by specifying its full file path.

Load geographic data from CSV and convert to simple features.

We will use the built-in `read.csv()` function to load our data. This function needs only the path to a valid CSV file (a file where columns are separated by "," and rows are separated by newlines) to load our data. Then we will use the `st_as_sf()` function from the library `sf` to convert the data to a simple features object which can represent geographic data. Other functions from the library `sf` will help us explore this dataset. Read more about the `sf` library here.

```
bus_csv <- read.csv('data/Bus_Stops_of_NJ_Transit.csv')
head(bus_csv)
```

```
##     FID COUNTY DLAT_GIS DLONG_GIS LINE STOP_NUM STOP_TYPE STREET_DIR
## 1 30448  Essex 40.72093 -74.22520   37    17819        NS          N
## 2 30449  Essex 40.72093 -74.22520  107    17819        NS          N
## 3 30450  Essex 40.72115 -74.22518   90    17841        NS          S
## 4 30451  Essex 40.79417 -74.23053   97    19442        NS          N
## 5 30452  Essex 40.79419 -74.23067   97    19445        FS          S
## 6 30453  Essex 40.77684 -74.17063   99    18659        NS          E
##          DESCRIPTION_BSL DIRECTION_OP    MUNICIPALITY
## 1 GROVE ST AT HERPERS ST           Ou   IRVINGTON TWP
## 2 GROVE ST AT HERPERS ST           Ou   IRVINGTON TWP
## 3 GROVE ST AT HERPERS ST           Ou   IRVINGTON TWP
```

```
## 4 HARRISON AVE AT ELM ST           In WEST ORANGE TWP
## 5 HARRISON AVE AT ELM ST           Ou WEST ORANGE TWP
## 6 HELLER PKWY AT LAKE ST           In          NEWARK
##                                GlobalID
## 1 9dd47d93-3ac8-4167-85bd-20fefa063e33
## 2 f3cde248-5286-4a0d-8710-94c38b3bafd8
## 3 8fc8919f-8a25-4eec-aa65-f5c9d6d793dd
## 4 65a3bde8-d8f0-475d-9f49-ff86a81d387f
## 5 4a3fa65f-a13a-4f07-bfcd-977f7562c5c4
## 6 70298ad9-9b33-419d-b883-1d2434264126
```

After inspecting the data, it appears the latitude and longitude data are in columns `DLAT_GIS` and `DLONG_GIS`. We have to tell `st_as_sf()` which CRS to use; unfortunately, CSVs do not have CRS metadata so we'll assume the common WGS84 CRS was used.

```
library(sf)
```

```
## Linking to GEOS 3.8.1, GDAL 3.2.1, PROJ 7.2.1; sf_use_s2() is TRUE
```

```
bus_sf <- bus_csv %>% st_as_sf(.,coords=c("DLONG_GIS","DLAT_GIS"), crs=WGS84)
head(bus_sf)
```

```
## Simple feature collection with 6 features and 10 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -74.23067 ymin: 40.72093 xmax: -74.17063 ymax: 40.79419
## Geodetic CRS:  WGS 84
##     FID COUNTY LINE STOP_NUM STOP_TYPE STREET_DIR        DESCRIPTION_BSL
## 1 30448  Essex   37    17819        NS          N GROVE ST AT HERPERS ST
## 2 30449  Essex  107    17819        NS          N GROVE ST AT HERPERS ST
## 3 30450  Essex   90    17841        NS          S GROVE ST AT HERPERS ST
## 4 30451  Essex   97    19442        NS          N HARRISON AVE AT ELM ST
## 5 30452  Essex   97    19445        FS          S HARRISON AVE AT ELM ST
## 6 30453  Essex   99    18659        NS          E HELLER PKWY AT LAKE ST
##   DIRECTION_OP    MUNICIPALITY                              GlobalID
## 1           Ou   IRVINGTON TWP 9dd47d93-3ac8-4167-85bd-20fefa063e33
## 2           Ou   IRVINGTON TWP f3cde248-5286-4a0d-8710-94c38b3bafd8
## 3           Ou   IRVINGTON TWP 8fc8919f-8a25-4eec-aa65-f5c9d6d793dd
## 4           In WEST ORANGE TWP 65a3bde8-d8f0-475d-9f49-ff86a81d387f
## 5           Ou WEST ORANGE TWP 4a3fa65f-a13a-4f07-bfcd-977f7562c5c4
## 6           In          NEWARK 70298ad9-9b33-419d-b883-1d2434264126
##                     geometry
## 1  POINT (-74.2252 40.72093)
## 2  POINT (-74.2252 40.72093)
## 3 POINT (-74.22518 40.72115)
## 4 POINT (-74.23053 40.79417)
## 5 POINT (-74.23067 40.79419)
## 6 POINT (-74.17063 40.77684)
```

We see now that a `geometry` column has been added to our dataframe, with coordinate `POINT` values. We could save this file using the `st_write()` function (though we will skip that for this exercise). We will use the actual shapefile for the remainder of our work, so we remove the `bus_sf` variable to avoid confusion:

```
rm(bus_sf)
```

Load a shapefile from file

We will use the `st_read()` function from the library `sf`, which we loaded previously. We're also going to load the `tidyverse` library, which has additional useful methods for data exploration.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
bus <- st_read('data/Bus_Stops_of_NJ_Transit.shp')
```

```
## Reading layer 'Bus_Stops_of_NJ_Transit' from data source
##   '/Users/kpierce/data_trainings/PAP-TACC-2022/data/Bus_Stops_of_NJ_Transit.shp'
##   using driver 'ESRI Shapefile'
## Simple feature collection with 30499 features and 12 fields
## Geometry type: MULTIPOINT
## Dimension:     XY
## Bounding box:  xmin: -8406962 ymin: 4712207 xmax: -8230860 ymax: 5050000
## Projected CRS: WGS 84 / Pseudo-Mercator
```

Identify the coordinate reference system and geometry type

This dataset is loaded as a "simple feature collection", but you can treat it almost the same as a dataframe or `tbl_df`. This dataset does have two additional features that make it better suited for spatial data analysis than a regular dataframe:

Shapefiles contain metadata about the coordinate reference system (CRS) that are loaded with the raw data are loaded. You may have noticed some of this metadata printed when we loaded the data; we can als inspect the CRS with the `st_crs()` function:

```
st_crs(bus)
```

```
## Coordinate Reference System:
##   User input: WGS 84 / Pseudo-Mercator
##   wkt:
## PROJCRS["WGS 84 / Pseudo-Mercator",
##     BASEGEOGCRS["WGS 84",
##         DATUM["World Geodetic System 1984",
##             ELLIPSOID["WGS 84",6378137,298.257223563,
```

```
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4326]],
##     CONVERSION["Popular Visualisation Pseudo-Mercator",
##         METHOD["Popular Visualisation Pseudo Mercator",
##             ID["EPSG",1024]],
##         PARAMETER["Latitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["False easting",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]]],
##     CS[Cartesian,2],
##         AXIS["easting (X)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["northing (Y)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Web mapping and visualisation."],
##         AREA["World between 85.06°S and 85.06°N."],
##         BBOX[-85.06,-180,85.06,180]],
##     ID["EPSG",3857]]
```

Second, simple feature collections contain a special `geometry` column that contains spatial data. If we look at the first element of the `bus` dataset `geometry` column, we see that this column contains `MULTIPOINT` data. It's unclear why `st_read()` specified this as `MULTIPOINT` data, but this won't matter for the work we're doing. Still, if we wanted to convert the `geometry` column to `POINT` data we could do so with `st_cast(bus$geometry, "POINT")`.

```
bus$geometry[1]
```

```
## Geometry set for 1 feature
## Geometry type: MULTIPOINT
## Dimension:     XY
## Bounding box:  xmin: -8262706 ymin: 4971264 xmax: -8262706 ymax: 4971264
## Projected CRS: WGS 84 / Pseudo-Mercator
```

```
## MULTIPOINT ((-8262706 4971264))
```

All values in the `geometry` column must be of the same type, so even though this first entry is a single point there are likely other bus stops in the data set that have multiple points. We'll work to understand the contents of the `geometry` column later in this example.

Change the coordinate reference system

At this point you may have noticed that the values in the `geometry` column don't look like regular latitude and longitude values. Indeed, the CRS for the bus stop data is indicated as "WGS 84 / Pseudo-Mercator". This is a planar projection of the data, but not the one we want. We can use the `st_transform()` function to do the geometric calculations to translate between coordinate reference systems.

We'll make a new data object in the WGS84 CRS:

```
bus_wgs84 <- st_transform(bus, crs=WGS84)
```

We'll also make a new data object in the NJ Planar projection:

```
bus_nj <- st_transform(bus, crs=NJ_PLANAR)
```

We can double check that our transformations changed the data by looking at the `head()` of our new datasets and inspecting the CRS metadata:

```
head(bus_wgs84)
```

```
## Simple feature collection with 6 features and 12 fields
## Geometry type: MULTIPOINT
## Dimension:     XY
## Bounding box:  xmin: -74.23071 ymin: 40.72092 xmax: -74.17066 ymax: 40.79424
## Geodetic CRS:  WGS 84
##     FID COUNTY DLAT_GIS DLONG_GIS LINE STOP_NUM STOP_TYPE STREET_DIR
## 1 30448  Essex 40.72093 -74.22520   37    17819        NS          N
## 2 30449  Essex 40.72093 -74.22520  107    17819        NS          N
## 3 30450  Essex 40.72115 -74.22518   90    17841        NS          S
## 4 30451  Essex 40.79417 -74.23053   97    19442        NS          N
## 5 30452  Essex 40.79419 -74.23067   97    19445        FS          S
## 6 30453  Essex 40.77684 -74.17063   99    18659        NS          E
##               DESCRIPTIO DIRECTION_      MUNICIPALI
## 1 GROVE ST AT HERPERS ST         Ou   IRVINGTON TWP
## 2 GROVE ST AT HERPERS ST         Ou   IRVINGTON TWP
## 3 GROVE ST AT HERPERS ST         Ou   IRVINGTON TWP
## 4 HARRISON AVE AT ELM ST         In WEST ORANGE TWP
## 5 HARRISON AVE AT ELM ST         Ou WEST ORANGE TWP
## 6 HELLER PKWY AT LAKE ST         In          NEWARK
##                               GlobalID                       geometry
## 1 9dd47d93-3ac8-4167-85bd-20fefa063e33 MULTIPOINT ((-74.22516 40.7...
## 2 f3cde248-5286-4a0d-8710-94c38b3bafd8 MULTIPOINT ((-74.22516 40.7...
## 3 8fc8919f-8a25-4eec-aa65-f5c9d6d793dd MULTIPOINT ((-74.22523 40.7...
## 4 65a3bde8-d8f0-475d-9f49-ff86a81d387f MULTIPOINT ((-74.2305 40.79...
## 5 4a3fa65f-a13a-4f07-bfcd-977f7562c5c4 MULTIPOINT ((-74.23071 40.7...
## 6 70298ad9-9b33-419d-b883-1d2434264126 MULTIPOINT ((-74.17066 40.7...
```

```
head(bus_nj)
```

```
## Simple feature collection with 6 features and 12 fields
## Geometry type: MULTIPOINT
## Dimension:     XY
```

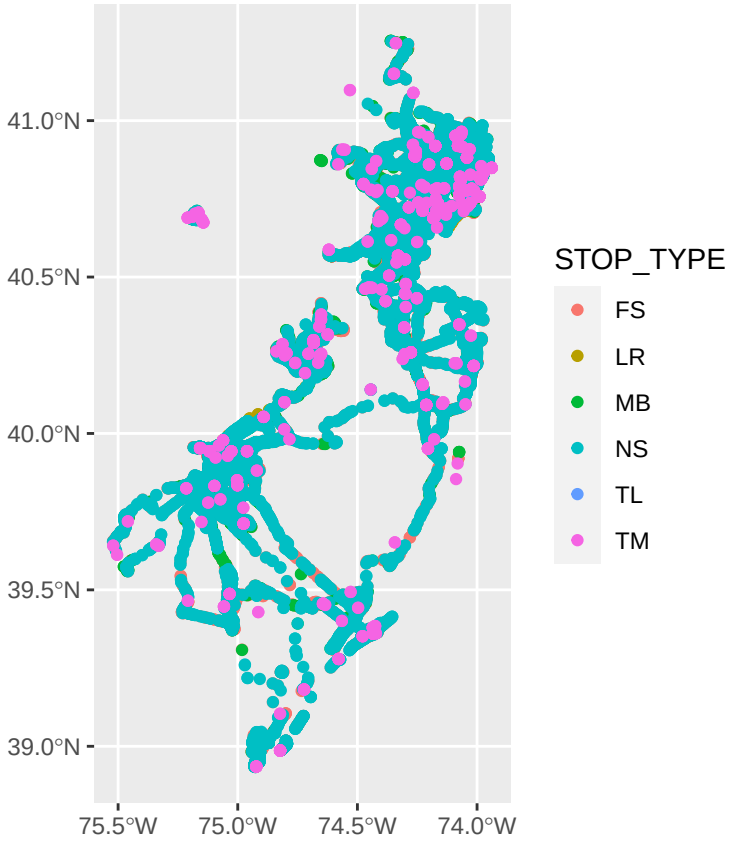
```
## Bounding box:  xmin: 172725.2 ymin: 209594.1 xmax: 177799.7 ymax: 217734.3
## Projected CRS: NAD_1983_HARN_StatePlane_New_Jersey_FIPS_2900
##     FID COUNTY DLAT_GIS DLONG_GIS LINE STOP_NUM STOP_TYPE STREET_DIR
## 1 30448  Essex 40.72093 -74.22520   37    17819        NS          N
## 2 30449  Essex 40.72093 -74.22520  107    17819        NS          N
## 3 30450  Essex 40.72115 -74.22518   90    17841        NS          S
## 4 30451  Essex 40.79417 -74.23053   97    19442        NS          N
## 5 30452  Essex 40.79419 -74.23067   97    19445        FS          S
## 6 30453  Essex 40.77684 -74.17063   99    18659        NS          E
##               DESCRIPTIO DIRECTION_      MUNICIPALI
## 1 GROVE ST AT HERPERS ST         Ou   IRVINGTON TWP
## 2 GROVE ST AT HERPERS ST         Ou   IRVINGTON TWP
## 3 GROVE ST AT HERPERS ST         Ou   IRVINGTON TWP
## 4 HARRISON AVE AT ELM ST         In WEST ORANGE TWP
## 5 HARRISON AVE AT ELM ST         Ou WEST ORANGE TWP
## 6 HELLER PKWY AT LAKE ST         In          NEWARK
##                               GlobalID                       geometry
## 1 9dd47d93-3ac8-4167-85bd-20fefa063e33 MULTIPOINT ((173219.2 20959...
## 2 f3cde248-5286-4a0d-8710-94c38b3bafd8 MULTIPOINT ((173219.2 20959...
## 3 8fc8919f-8a25-4eec-aa65-f5c9d6d793dd MULTIPOINT ((173212.4 20962...
## 4 65a3bde8-d8f0-475d-9f49-ff86a81d387f MULTIPOINT ((172742.3 21772...
## 5 4a3fa65f-a13a-4f07-bfcd-977f7562c5c4 MULTIPOINT ((172725.2 21773...
## 6 70298ad9-9b33-419d-b883-1d2434264126 MULTIPOINT ((177799.7 21581...
```

Map the bus stops

Basic point map

We'll use the library `ggplot2` to build our map. `ggplot2` is part of the `tidyverse` set of packages, and uses layers to construct images. Our first layer is a blank plotting space, which we create by calling the `ggplot()` function without any arguments. We then add the actual data we want to visualize in another layer with the `geom_sf` function, which understands how to interpret the `geometry` column in `bus_wgs84`.

```
nj_bus_map <- ggplot() +
  geom_sf(data=bus_wgs84, aes(color=STOP_TYPE), inherit.aes = FALSE)
nj_bus_map
```



Add a basemap

The bus stop points alone are hard to interpret, so we can add information to our map by adding a basemap – a background layer showing streets and landmarks to help contextualize the bus stop data. Basemap tiles can be downloaded from Stamen Maps.

Get a bounding box for the basemap We have to specify a bounding box, the coordinates for a box that contains the state of New Jersey, in order to download the basemap tiles. The easiest way to get a vaild bounding box is to use the `st_bbox()` function on a polygon to extract the bounding box coordinates.

We will load polygons for the state of New Jersey and its county boundaries directly from the ArcGIS Open Data by downloading a `geojson` file from a URL. `geojson` files are specially formatted text files that can be interpreted by the `read_sf()` function. We will transform these data to the WGS84 CRS.

```
nj <- read_sf(
  'https://opendata.arcgis.com/datasets/5f45e1ece6e14ef5866974a7b57d3b95_1.geojson'
  )
nj <- nj %>% st_transform(crs=WGS84)
```

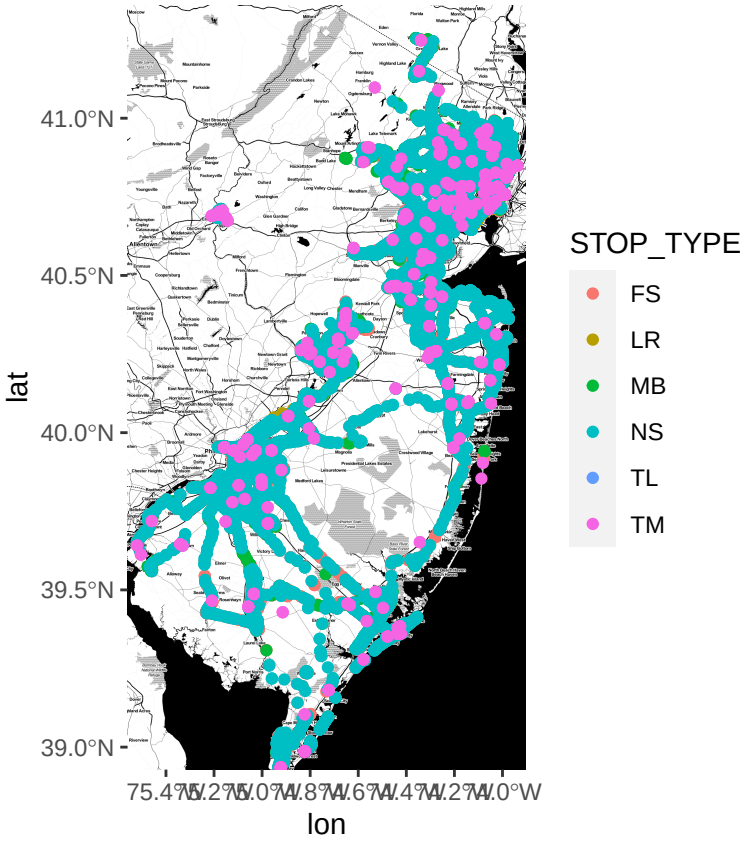
Load street data basemap Now that we have our New Jersey polygon, we can determine its bounding box and use the `get_stamenmap()` function to download a stylized street map background from Stamen Maps.

```
nj_bbox <- unname(st_bbox(nj))
nj_base_map <- get_stamenmap(bbox=nj_bbox, maptype="toner", force=TRUE)
```

Map bus stops on streetmap

Next we'll build a map of bus stops in with the same layer-based approach we used for the basic point map example above. This time we will make our first layer with the basemap by calling the `ggmap()` function from the `ggmap` library and passing our base map variable as an argument. Our next layer will be the bus stop data, and for this example we will color points by the bus stop type (one of the columns in the dataset). The argument `inherit.aes = FALSE` instructs to the function to ignore default plot aesthetics in favor of the aesthetic instructions we have provided.

```
library(ggmap)
nj_bus_map <- ggmap(nj_base_map) +
  geom_sf(data=bus_wgs84, aes(color=STOP_TYPE), inherit.aes = FALSE)
nj_bus_map
```



Population choropleth map

This dataset happens to include a number of measures for each county. Choropleth maps shade polygons by values. We'll take one measure, `POP2010` (the population of each county from the 2010 decennial census) and make a choropleth map.

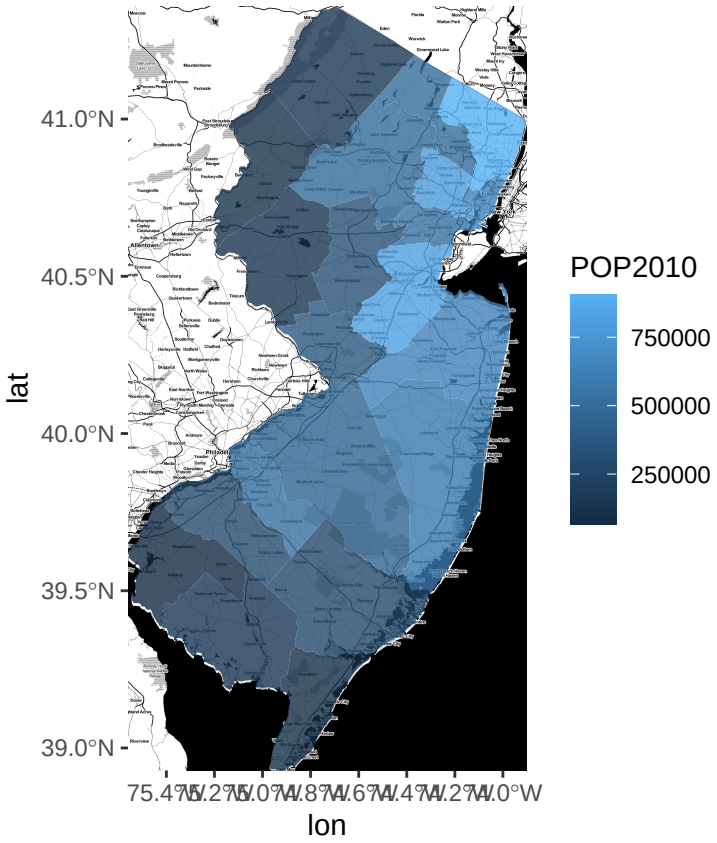
```r
head(nj)
```

```
## Simple feature collection with 6 features and 22 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -75.41973 ymin: 38.92852 xmax: -73.90245 ymax: 41.13372
## Geodetic CRS:  WGS 84
## # A tibble: 6 x 23
##   OBJECTID COUNTY     COUNTY_LABEL  CO    GNIS_NAME GNIS  FIPSSTCO FIPSCO  ACRES
##      <int> <chr>      <chr>         <chr> <chr>     <chr> <chr>    <chr>   <dbl>
## 1        1 ATLANTIC   Atlantic Cou~ ATL   County o~ 8822~ 34001    1      3.91e5
## 2        2 BERGEN     Bergen County BER   County o~ 8822~ 34003    3      1.53e5
## 3        3 BURLINGTON Burlington C~ BUR   County o~ 8822~ 34005    5      5.25e5
## 4        4 CAMDEN     Camden County CAM   County o~ 8822~ 34007    7      1.46e5
## 5        5 CAPE MAY   Cape May Cou~ CAP   County o~ 8822~ 34009    9      1.83e5
## 6        6 CUMBERLAND Cumberland C~ CUM   County o~ 8822~ 34011    11     3.21e5
## # ... with 14 more variables: SQ_MILES <dbl>, POP2010 <int>, POP2000 <int>,
## #   POP1990 <int>, POP1980 <int>, POPDEN2010 <int>, POPDEN2000 <int>,
## #   POPDEN1990 <int>, POPDEN1980 <int>, REGION <chr>, GLOBALID <chr>,
## #   Shape_Length <dbl>, Shape_Area <dbl>, geometry <MULTIPOLYGON [°]>
```

```r
nj_map <- ggmap(nj_base_map) +
  geom_sf(data=nj, aes(fill=POP2010), inherit.aes = FALSE, color = NA, alpha = 0.8)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```

```r
nj_map
```



Aggregate bus stops into a grid

Rather than viewing the individual points where bus stops are located, we can aggregate bus stops spatially. By counting the number of stops in different areas, we may see patterns about service level coverage that are hard to pick out when viewing stop locations individually.

We will first create a grid that covers the state of New Jersey, then we will identify the bus stops that fall inside each grid cell.

Create a five-mile "fishnet" grid

We will need the following elements to build our fishnet grid: 1. The coverage area, in our case the entire state of New Jersey 2. The width of each square in the grid, specified in meters

We will use the `NJ_PLANAR` projection because we want a flat grid. This projection measures spatial distances in meters, and so we specify our fishnet width in meters. We would like to overlay a five-mile grid on the state of New Jersey, so we convert five miles into meters using the constant of 1,609 meters per mile.

```
nj_flat <- st_transform(nj, crs=NJ_PLANAR)
fishnet_width <- 1609 * 5
```

Next we need to adjust our New Jersey shapefile so it contains only the state border, and not the internal county borders. The `st_union()` function accomplishes this. We also use the `st_make_valid()` function to correct any loops or places where the border appears to cross itself.

```
nj_border <- st_make_valid(st_union(nj_flat))
```

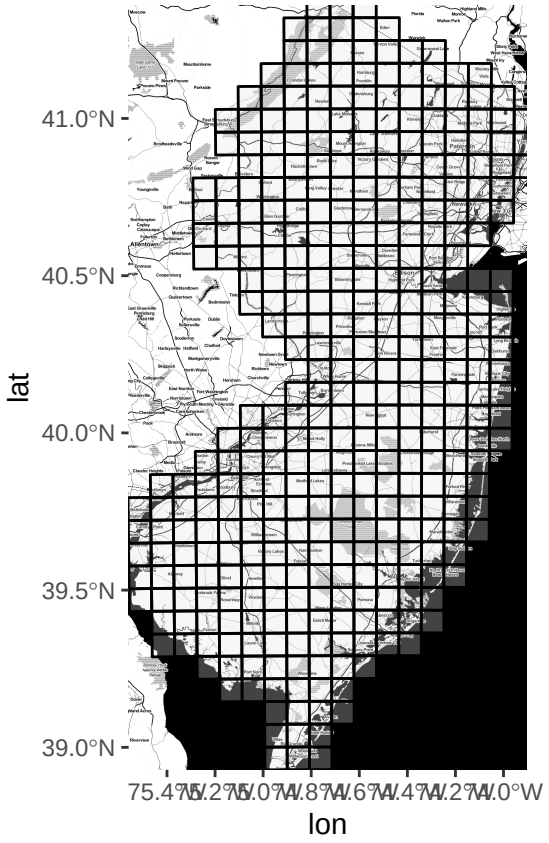
Now that we have our coverage area and our fishnet grid size determined, we are ready to use the `st_make_grid()` function to generate our grid. After we make the grid, we will assign each cell a unique identifier (a "net_id"). We will remove grid cells that fall outside the state border, and reshape the list of grid objects into a dataframe that is easier to work with.

```
net <- st_make_grid(
  x=nj_border, cellsize=fishnet_width, what='polygons', square=TRUE, crs=NJ_PLANAR
  )
net_agg <- st_as_sf(net) %>% tibble::rowid_to_column(., "net_id")
net_intersect <- st_intersects(nj_border, net_agg)
fishnet <- net_agg[unique(unlist(net_intersect)),]
```

We can make a quick visual inspection of our map with `ggmap`:

```
ggmap(nj_base_map) +
  geom_sf(
    data=st_transform(fishnet, WGS84),
    inherit.aes=FALSE, color='black', alpha=0.3
    )
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



Point-in-polygon join

With our fishnet created, we're ready to join our bus stop data. We use the `st_join()` function to identify the bus stops in each fishnet cell. This join preserves only the bus stop geometry, but we're really interested in the grid square shapes. So we drop the geometry column – we will add it back later with another join.

```
bus_net <- st_join(bus_nj, fishnet, join=st_within) %>%
    st_drop_geometry()
head(bus_net)
```

```
##     FID COUNTY DLAT_GIS DLONG_GIS LINE STOP_NUM STOP_TYPE STREET_DIR
## 1 30448  Essex 40.72093 -74.22520   37    17819        NS          N
## 2 30449  Essex 40.72093 -74.22520  107    17819        NS          N
## 3 30450  Essex 40.72115 -74.22518   90    17841        NS          S
## 4 30451  Essex 40.79417 -74.23053   97    19442        NS          N
## 5 30452  Essex 40.79419 -74.23067   97    19445        FS          S
## 6 30453  Essex 40.77684 -74.17063   99    18659        NS          E
##               DESCRIPTIO DIRECTION_      MUNICIPALI
## 1 GROVE ST AT HERPERS ST         Ou   IRVINGTON TWP
## 2 GROVE ST AT HERPERS ST         Ou   IRVINGTON TWP
## 3 GROVE ST AT HERPERS ST         Ou   IRVINGTON TWP
## 4 HARRISON AVE AT ELM ST         In WEST ORANGE TWP
## 5 HARRISON AVE AT ELM ST         Ou WEST ORANGE TWP
## 6 HELLER PKWY AT LAKE ST         In          NEWARK
##                                GlobalID net_id
## 1 9dd47d93-3ac8-4167-85bd-20fefa063e33    447
## 2 f3cde248-5286-4a0d-8710-94c38b3bafd8    447
## 3 8fc8919f-8a25-4eec-aa65-f5c9d6d793dd    447
## 4 65a3bde8-d8f0-475d-9f49-ff86a81d387f    465
## 5 4a3fa65f-a13a-4f07-bfcd-977f7562c5c4    465
## 6 70298ad9-9b33-419d-b883-1d2434264126    465
```

Now that we have bus stops associated with `net_id` values, we can count the number of bus stops in each cell by grouping rows by the `net_id` value. To make our analysis a bit more interesting, we will keep the `STOP_TYPE` data by using that as a grouping variable as well. This allows us to look at bus stop counts by stop type to see how service type varies spatially. The `summarise(count=n())` operation is what performs this count aggergation by counting the rows in each group.

```
bus_net_summary <- bus_net %>%
  group_by(net_id, STOP_TYPE) %>%
  summarise(count=n())
```

```
## `summarise()` has grouped output by 'net_id'. You can override using the
## `.groups` argument.
```

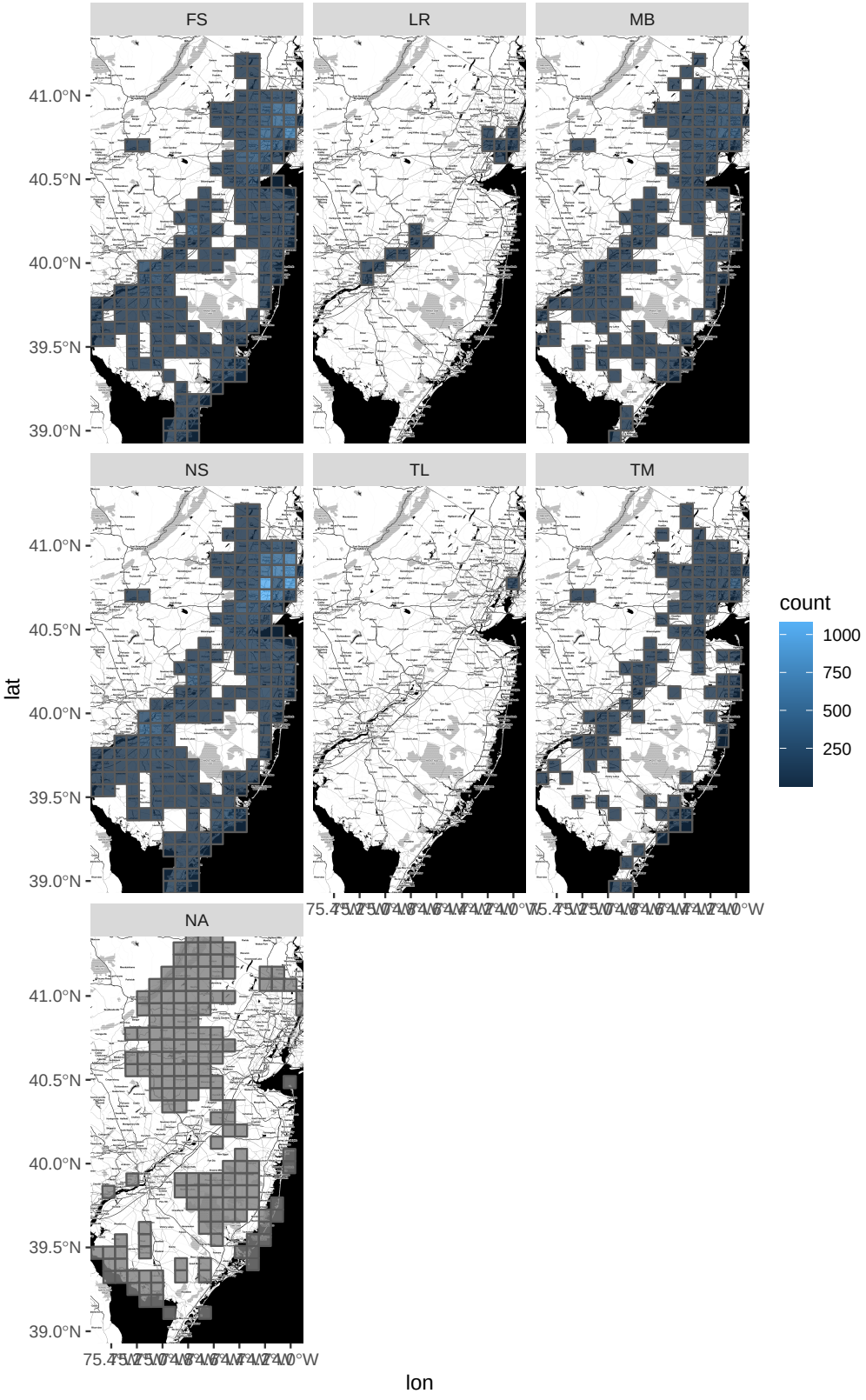
Earlier we dropped the `geometry` column, but now we're ready to add it back. We specifically want to add the fishnet `geometry` (not the bus stop point locations), so we'll join our `bus_net_summary` data to our original `fishnet` on the column `net_id`. We have to wrap that join in the `st_as_sf()` function to ensure the resulting dataframe is an `sf` object; we also specify that we would like to use the CRS already present in the `fishnet` data.

```
bus_net_final <- st_as_sf(
  left_join(
    fishnet,
    bus_net_summary,
    by='net_id'
    ),
  crs=st_crs(fishnet))
```

Map gridded bus stop counts by type

Now we're ready to map our data. We will use the `nj_base_map` as the first layer of our map, and we'll use the fishnet grids as the second layer. The `facet_wrap()` function allows us to easily make separate panels to display each bus stop type individually. We'll spread our facets over two rows for easier viewing:

```
bus_type_map <- ggmap(nj_base_map) +
  geom_sf(
    data=st_transform(bus_net_final, crs=WGS84),
    aes(fill=count),
    inherit.aes = FALSE, alpha=0.8
    ) +
  facet_wrap(facets='STOP_TYPE', nrow=3)
bus_type_map
```


FS
LR
MB
NS
TL
TM
NA
41.0°N
40.5°N
40.0°N
39.5°N
39.0°N
lat
lon
count
1000
750
500
250

Exercise: New Jersey Hospital Locations

1. Download "Hospitals in NJ" from the NJGIN Open Data portal: https://njogis-newjersey.opendata.arcgis.com/datasets/hospitals-in-nj/explore?location=40.145600%2C-74.726050%2C8.85&showTable=true

2. Load and inspect the data.

    a. What is the CRS used?
    b. How many columns are in the data set?
    c. How many rows?
    d. What are the data types as understood by R? Do those data types match your understanding of the data?

3. Make a faceted histogram of beds by region. Adjust the histogram bin size as necessary.

4. Assign each hospital to a grid cell in the 5-mile grid.

    a. Plot the corresponding choropleth map of hospitals per grid cell.
    b. Plot a histogram of the hospital per grid cell count.

5. Make a county-level fishnet.

    a. Use the `filter()` method to subset a single county's data from the hospital dataset.
    b. Make a new fishnet grid for that county only, this time with half-mile resolution.
    c. Aggregate that county's hospital data into the half-mile fishnet grid.
    d. Make the corresponding choropleth map and histogram for this county only.