DS 116 Data Visualization

Spatial data visualization

Habet Madoyan

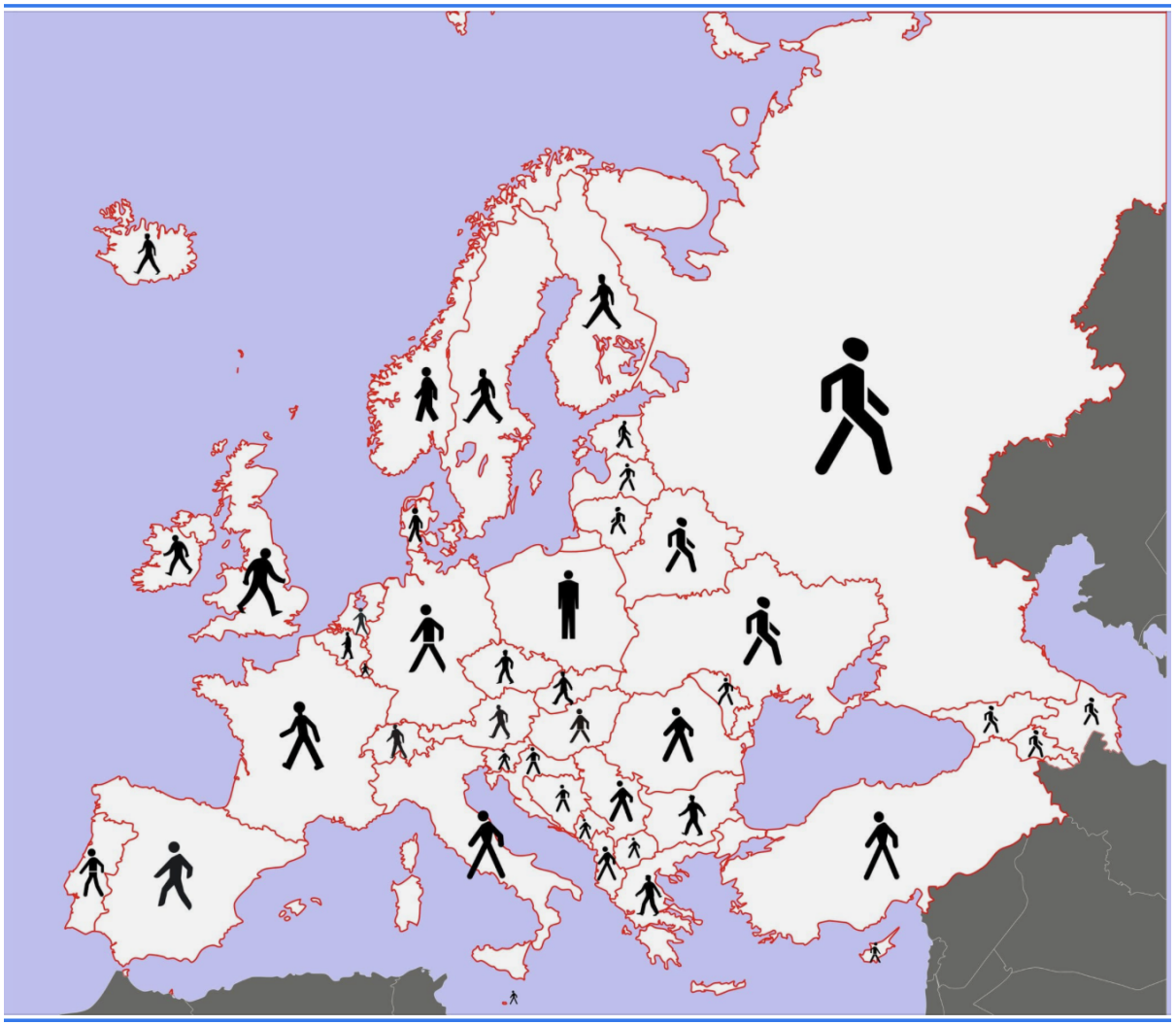
American University of Armenia

Spatial Data

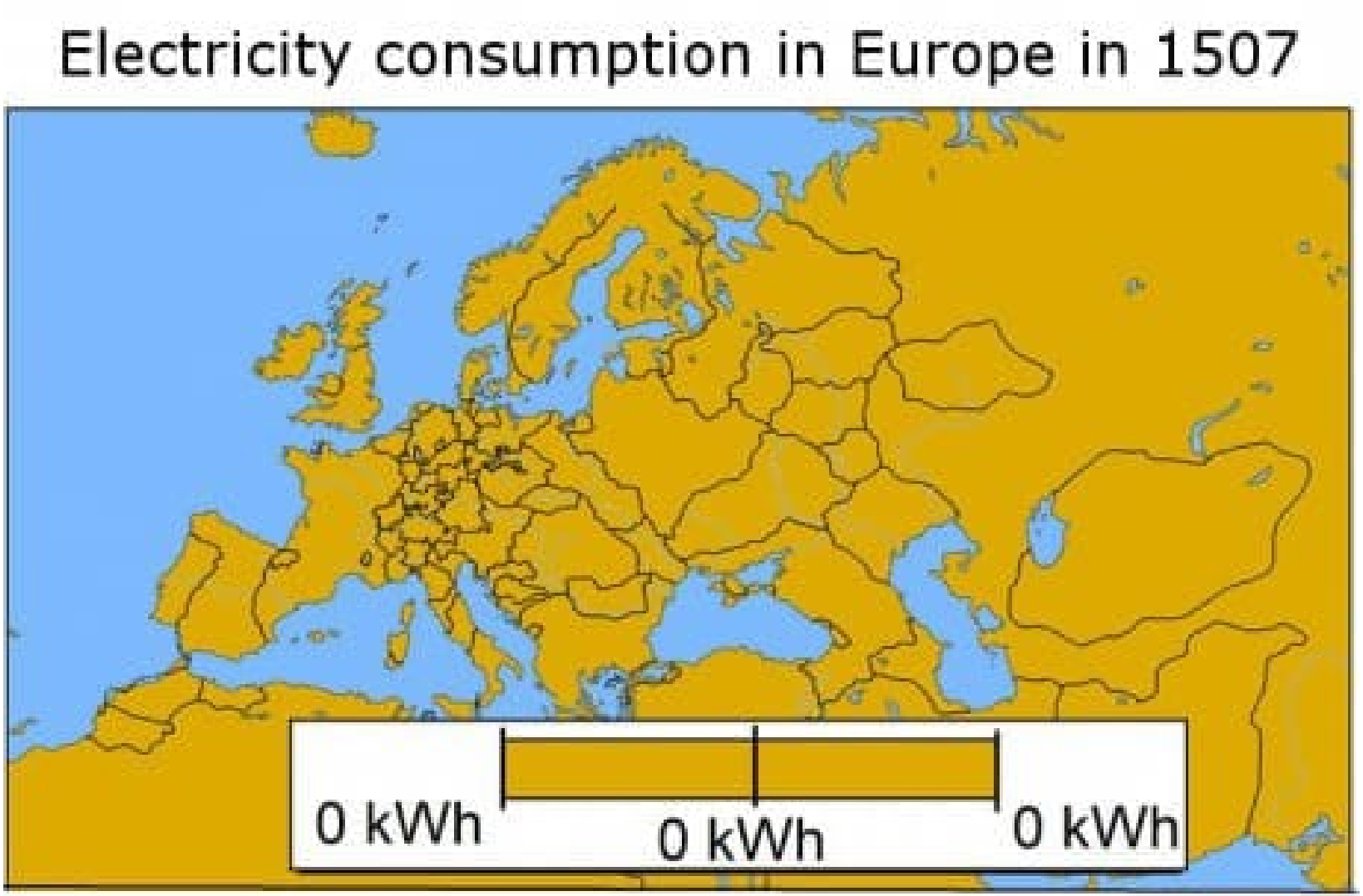
When data is associated with locations, we call it spatial data.
When those locations are on the earth, we call it **geospatial data**.
Location is described with the set of coordinates and the coordinate reference system (CRS)

Spatial Data

What pedestrians look like across Europe

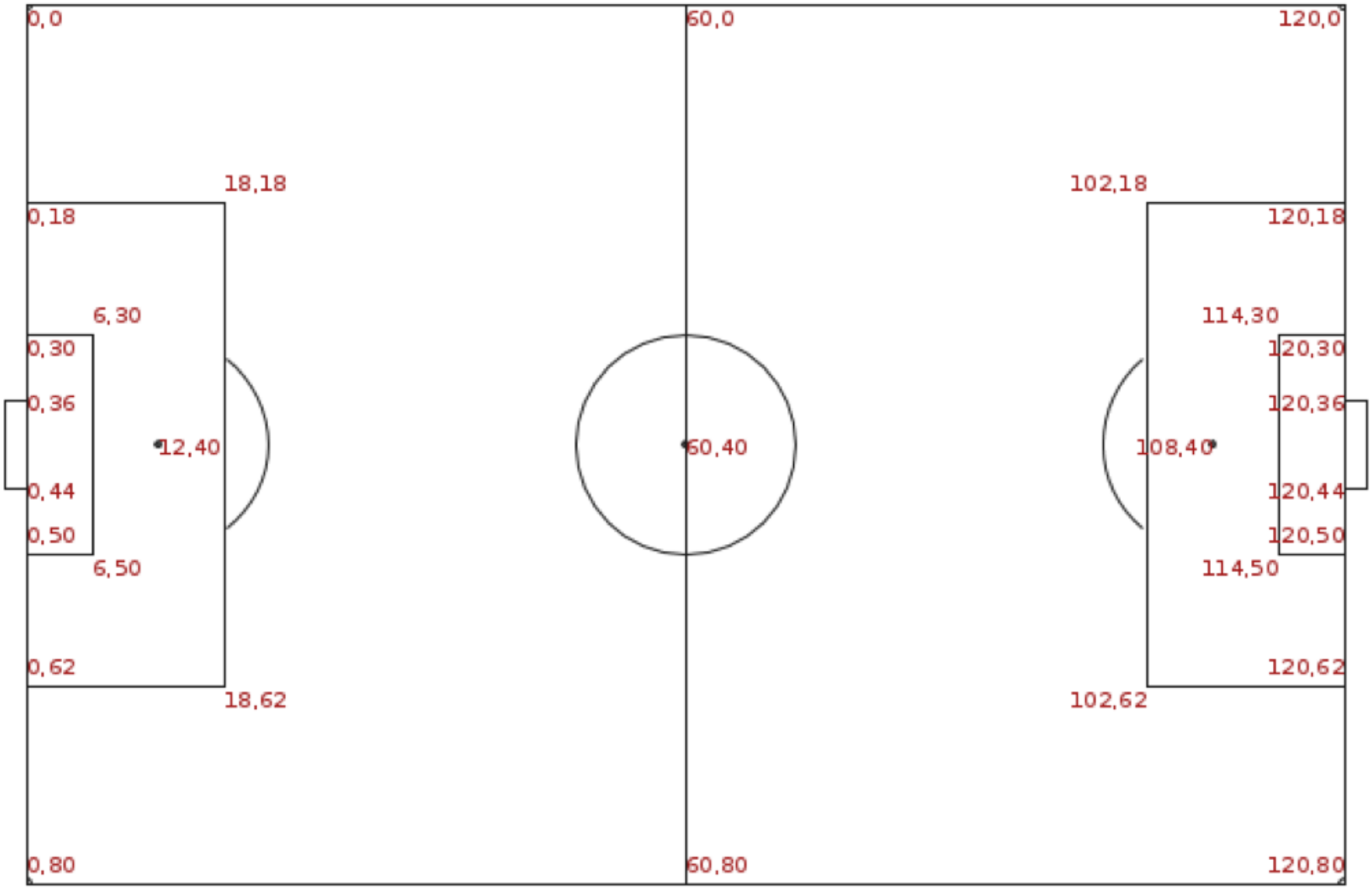


Spatial Data



Spatial Data

Spatial Data

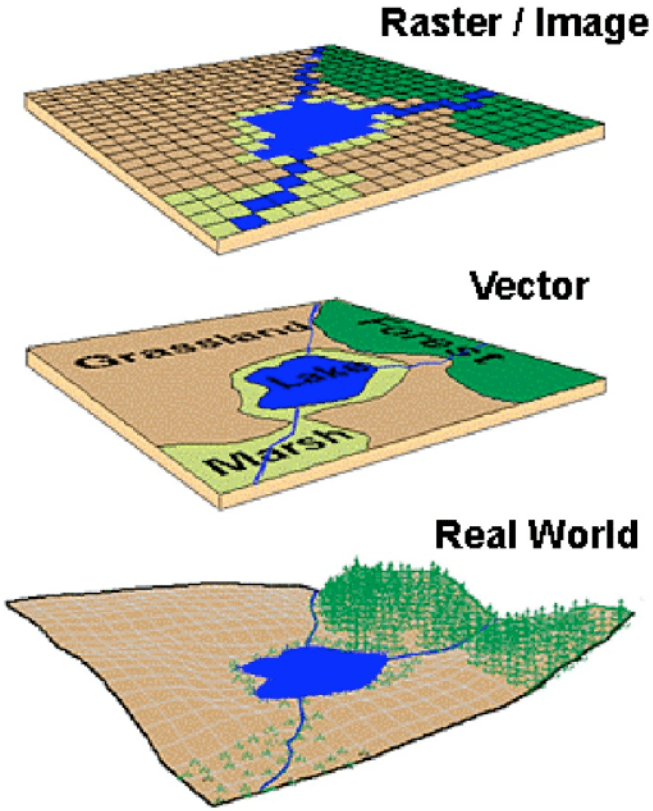


Geospatial Data Visualization

Data types in GIS

There are two main data types in GIS:

- Vector data
- Raster data



Geospatial Data Visualization

- Vector data is good for representing categorical and multivariate data.
- It also has attribute tables where every record or row corresponds to a feature or an object and every column corresponds to different attributes.

Vector data has three different types:

1. points,
2. lines,
3. polygons.

Geospatial Data Visualization

- Points data normally refers to data collected at some point in space.
- Examples:
  - City on the map
  - School
  - Drop off point for taxi
  - etc

Geospatial Data Visualization

Points connect to each other to become lines and represent features such as

- Highways,
- Rivers,
- Taxi routes,
- etc

Geospatial Data Visualization

If those lines add up to make a closed shape, we get a polygon. Examples of usage of polygons are country or administrative region boundaries.

Geospatial Data Visualization

Vector data file formats are called shapefiles.
A shapefile is a simple, nontopological format for storing the geometric location and attribute information of geographic features.
Shapefile is used to describe vector features such as points, lines, and polygons.

Geospatial Data Visualization

For doing so, shapefile uses three to seven files for a single map file to describe different components such as geometries or shape format, projection, attributes

File extension	Content
.dbf	Attribute information
.shp	Feature geometry
.shx	Feature geometry index
.aih	Attribute index
.ain	Attribute index
.prj	Coordinate system information
.sbn	Spatial index file
.sbx	Spatial index file

Geospatial Data Visualization

There are few ways to read vector data into R we will use sf library

Geospatial Data Visualization

Using rgdal
Specify the folder with all your spatial data

```
arm <- st_read('Data/Armenia_Marzes/Armenia_Marzes.shp')
```

```
## Reading layer 'Armenia_Marzes' from data source
##   '/Users/habetmadoyan/Documents/Education/DataViz/Lectures/Spatial Data Visualization/Data/Armenia_Marzes/Armenia_Marzes.shp'
##   using driver 'ESRI Shapefile'
## Simple feature collection with 11 features and 16 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 369676.6 ymin: 4299770 xmax: 641124.7 ymax: 4572280
## Projected CRS: WGS 84 / UTM zone 38N
```

```
class(arm)
```

```
## [1] "sf"         "data.frame"
```

Geospatial Data Visualization

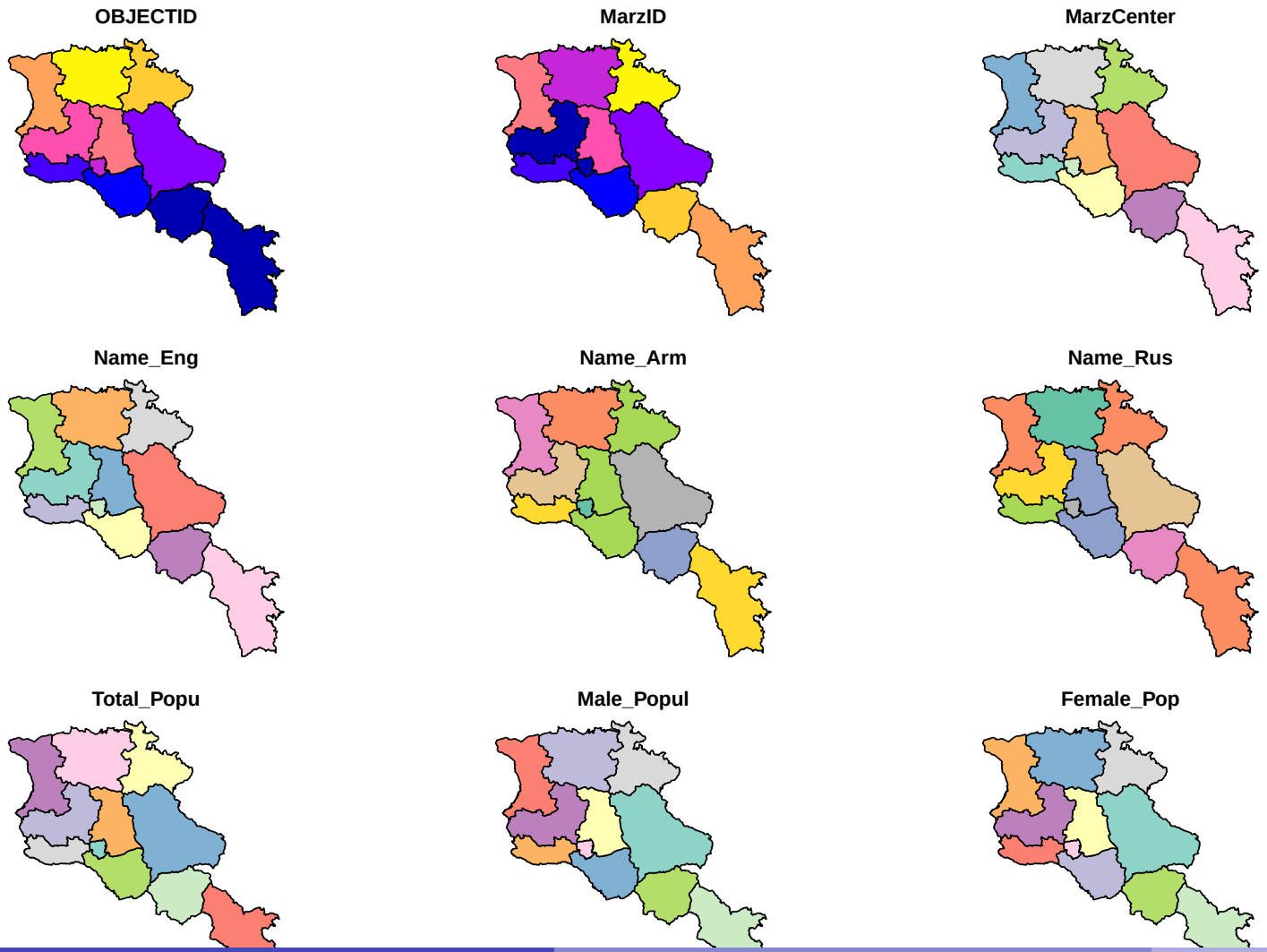
```
names(arm)
```

```
##  [1] "OBJECTID"   "MarzID"     "MarzCenter" "Name_Eng"   "Name_Arm"
##  [6] "Name_Rus"   "Total_Popu" "Male_Popul" "Female_Pop" "Total_Po_1"
## [11] "Male_Pop_1" "Female_P_1" "Link"       "Shape_Leng" "Shape_Le_1"
## [16] "Shape_Area" "geometry"
```

Geospatial Data Visualization

You can use generic plot method

```
plot(arm)
```



Geospatial Data Visualization

X is the Longitude Y is the Latitude

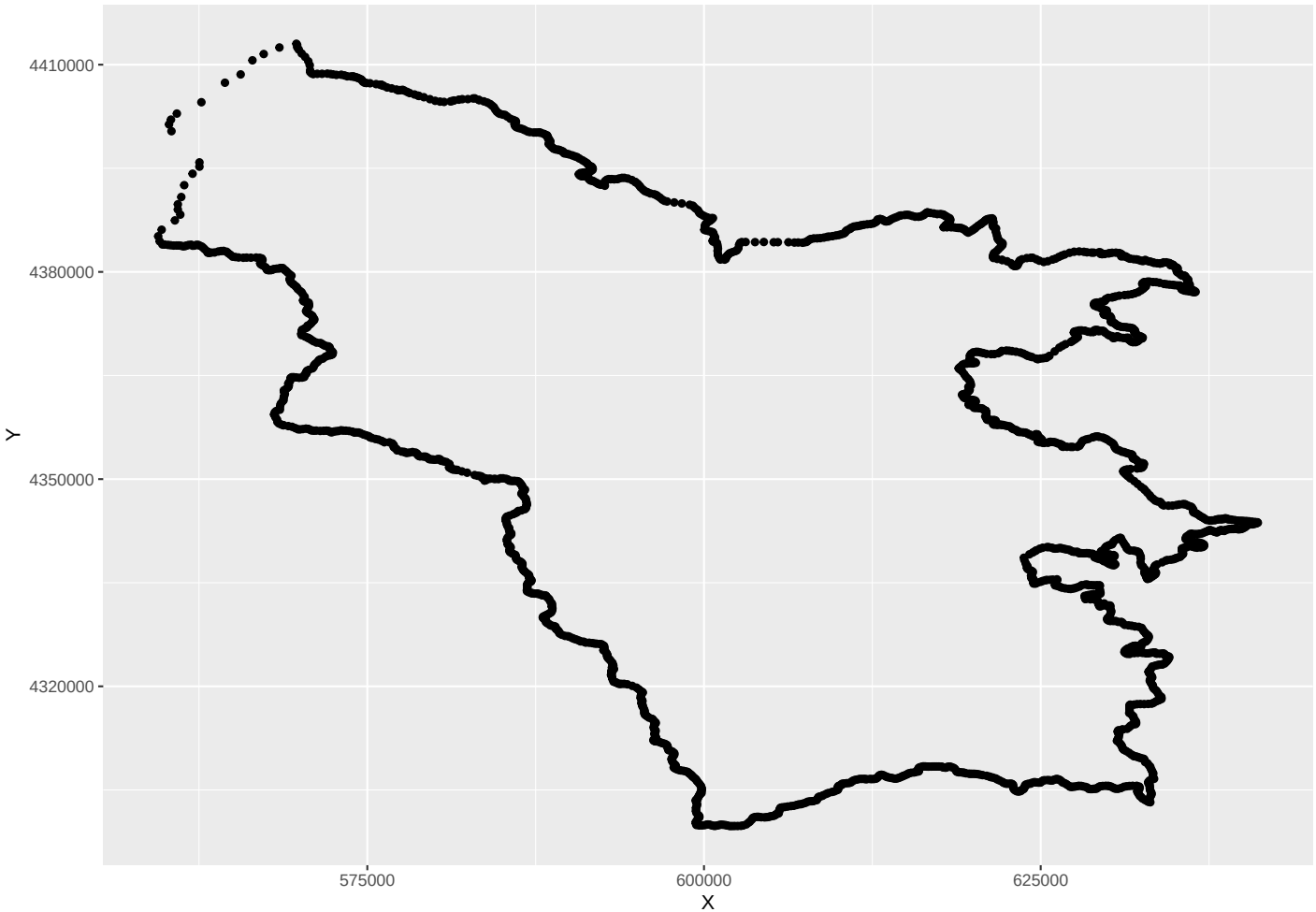
```
arm_sf <- st_read('Data/Armenia_Marzes/Armenia_Marzes.shp', quiet = TRUE)

arm_df <- arm_sf %>% group_by(MarzID) %>%
    st_cast("POINT")


arm_df = data.frame(Marz = arm_df$Name_Eng, st_coordinates(arm_df))

head(arm_df)
```

```
##     Marz        X       Y
## 1 SYUNIK 563130.7 4382890
## 2 SYUNIK 563109.7 4382983
## 3 SYUNIK 562920.9 4383285
## 4 SYUNIK 562807.7 4383511
## 5 SYUNIK 562694.4 4383690
## 6 SYUNIK 562533.5 4383823
```

Geospatial Data Visualization

Plot one region

```
arm_df %>% filter(Marz == "SYUNIK") %>%
  ggplot(aes(x = X, y = Y)) + geom_point()
```

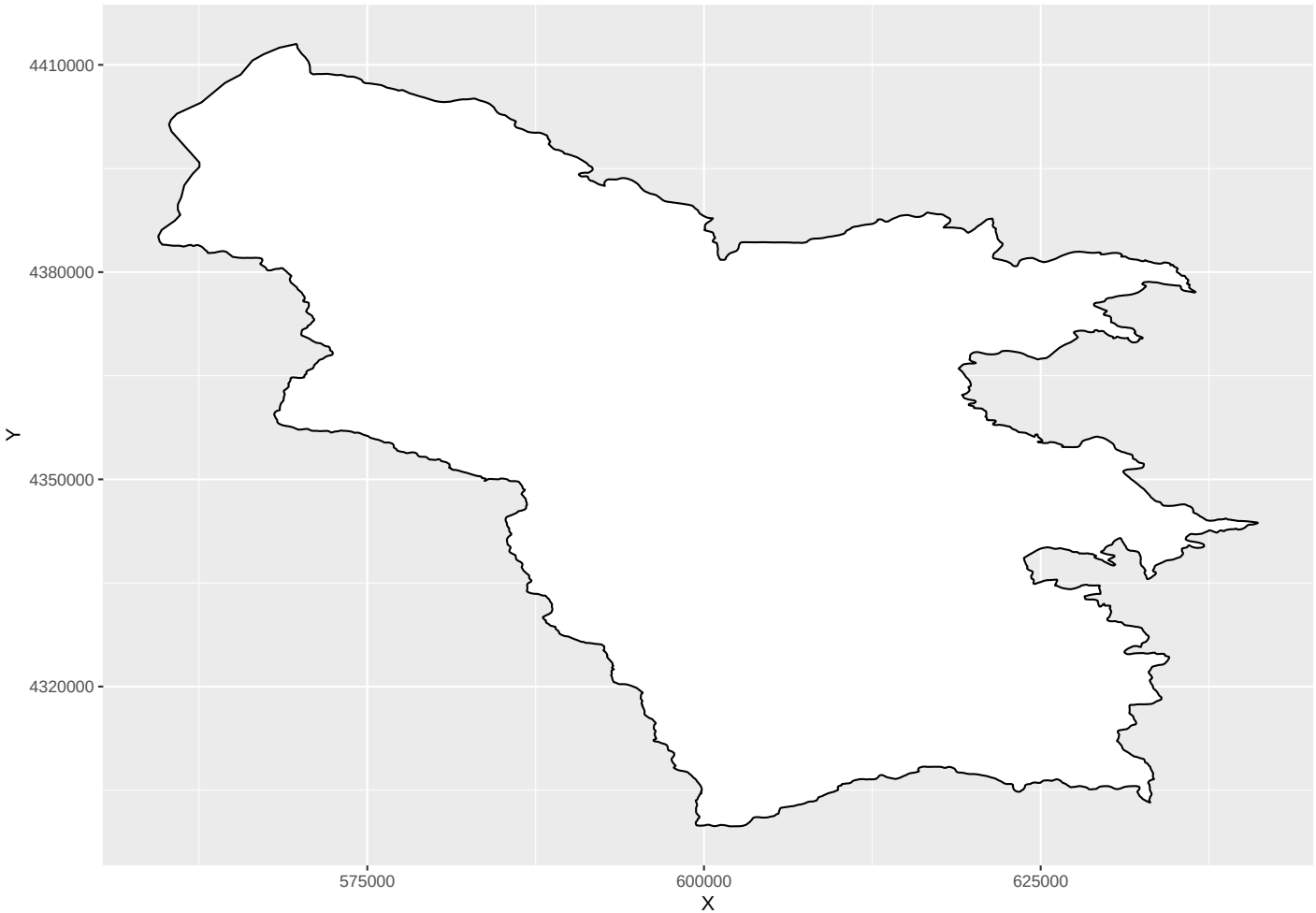


Geospatial Data Visualization

```
arm_df %>% filter(Marz == "SYUNIK") %>%
    ggplot(aes(x = X, y = Y)) + geom_polygon()
```

Geospatial Data Visualization

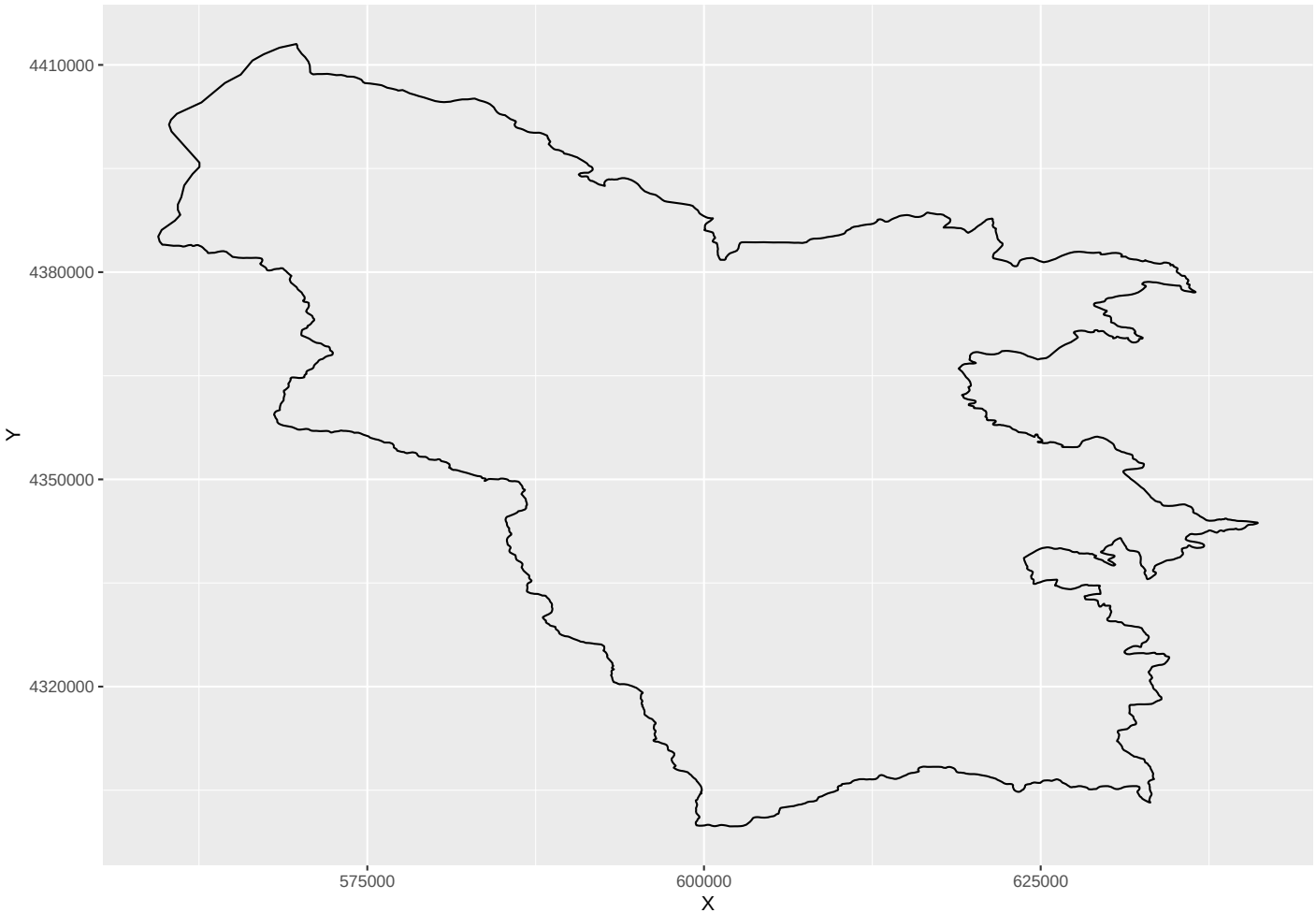
```
arm_df %>% filter(Marz == "SYUNIK") %>%
   ggplot(aes(x = X, y = Y)) +
  geom_polygon(fill = 'white', color = 'black')
```



Geospatial Data Visualization

Or geom_path()

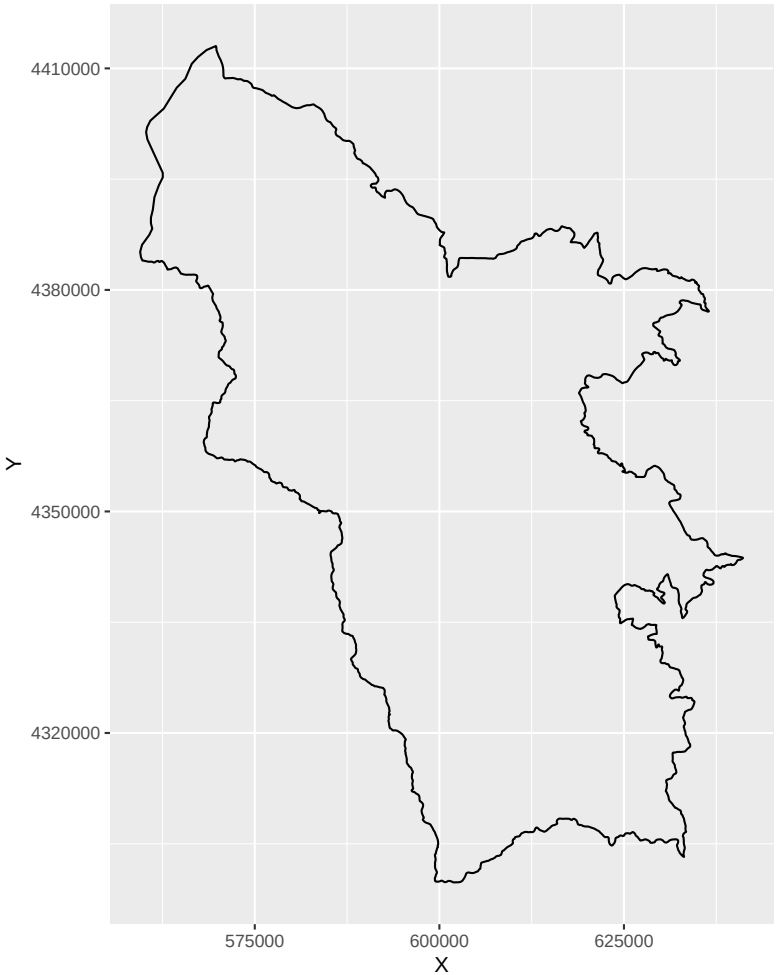
```
arm_df %>% filter(Marz == "SYUNIK") %>%
    ggplot(aes(x = X, y = Y)) + geom_path()
```



Geospatial Data Visualization

Add coord_fixed()

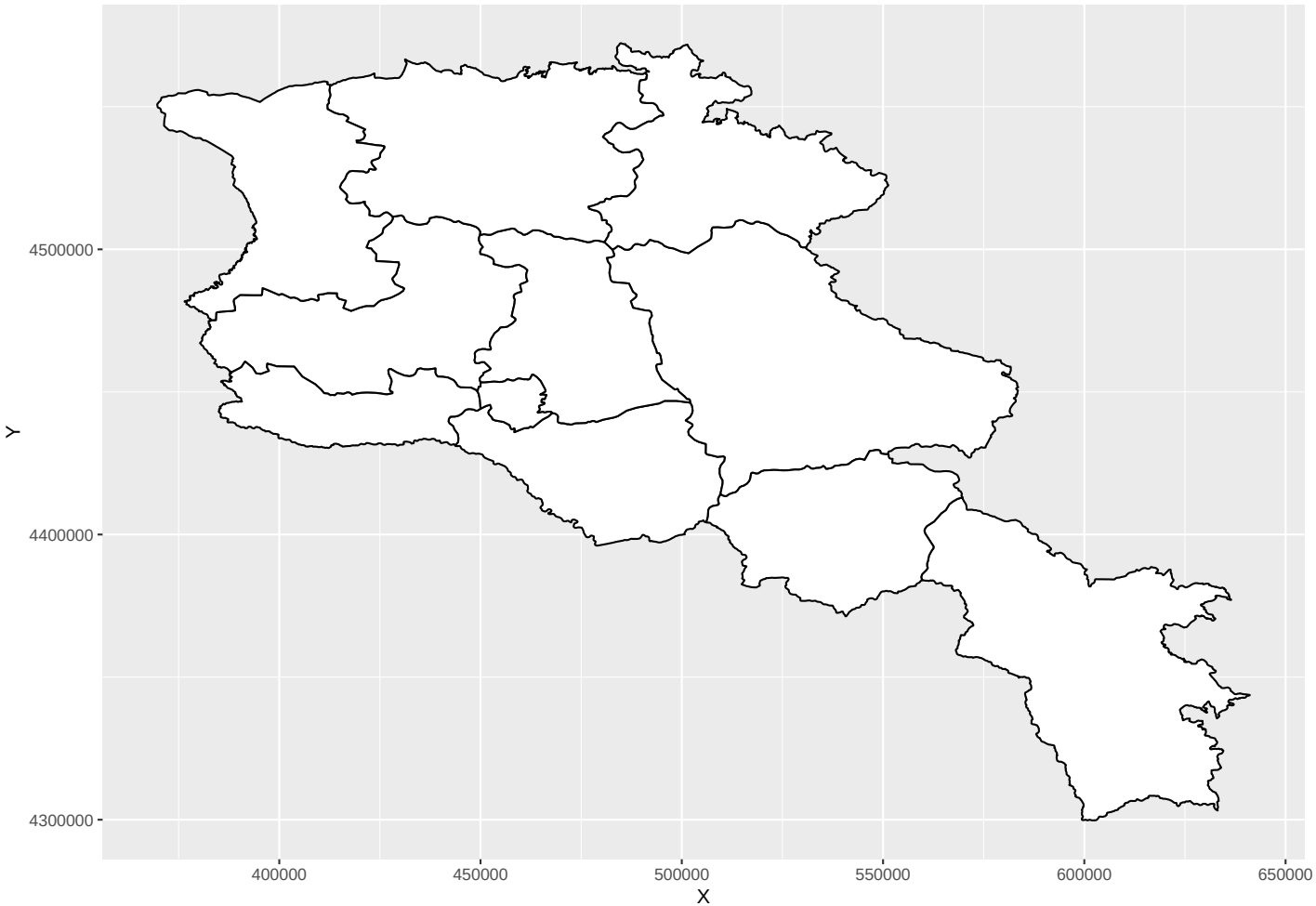
```
arm_df %>% filter(Marz == "SYUNIK") %>%
    ggplot(aes(x = X, y = Y)) + geom_path() + coord_fixed()
```



Geospatial Data Visualization

To get the whole map of Armenia, add group aesthetics

```
ggplot(arm_df, aes(x = X, y = Y, group = Marz)) +
  geom_polygon(fill = 'white', color = 'black')
```



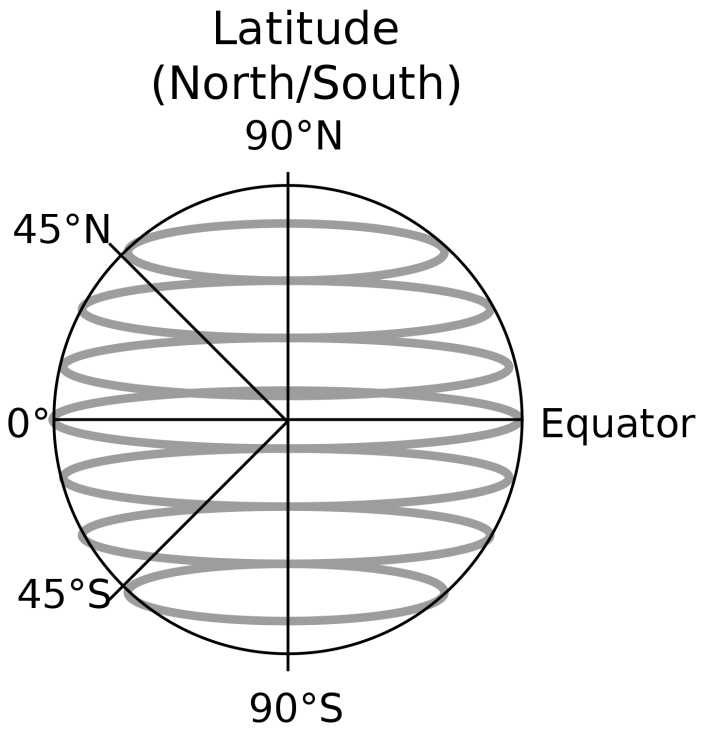
Geospatial Data Visualization

To get the whole map of Armenia, add group aesthetics

```
ggplot(arm_df, aes(x = X, y = Y, group = Marz)) +
  geom_polygon(fill = 'white', color = 'black') + coord_fixed() +theme_void
```

Geospatial Data Visualization

If you look at the x, y axis you can see that those are not longitude and latitude



Geospatial Data Visualization

- Look at the **coordinate reference system** for the shapefile
- If you want to overlay maps on each other, you need to be sure that they have the same CRS

```
st_crs(arm)
```

```
## Coordinate Reference System:
##   User input: WGS 84 / UTM zone 38N
##   wkt:
## PROJCRS["WGS 84 / UTM zone 38N",
##     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["UTM zone 38N",
##         METHOD["Transverse Mercator",
##             ID["EPSG",9807]],
##         PARAMETER["Latitude of natural origin",0,
##             ANGLEUNIT["Degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",45,
##             ANGLEUNIT["Degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["Scale factor at natural origin",0.9996,
##             SCALEUNIT["unity",1],
##             ID["EPSG",8805]],
##         PARAMETER["False easting",500000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",0,
##             LENGTHUNIT["metre",1],
```

Geospatial Data Visualization

Raster

- Raster data is set of spatial data in the form of a grid of pixels.
- Each pixel will hold a value which relates to some feature of interest at that point in space.
- in R we will use library(raster) to read and work with raster data

Geospatial Data Visualization

Raster file for Direct Normal Irradiation

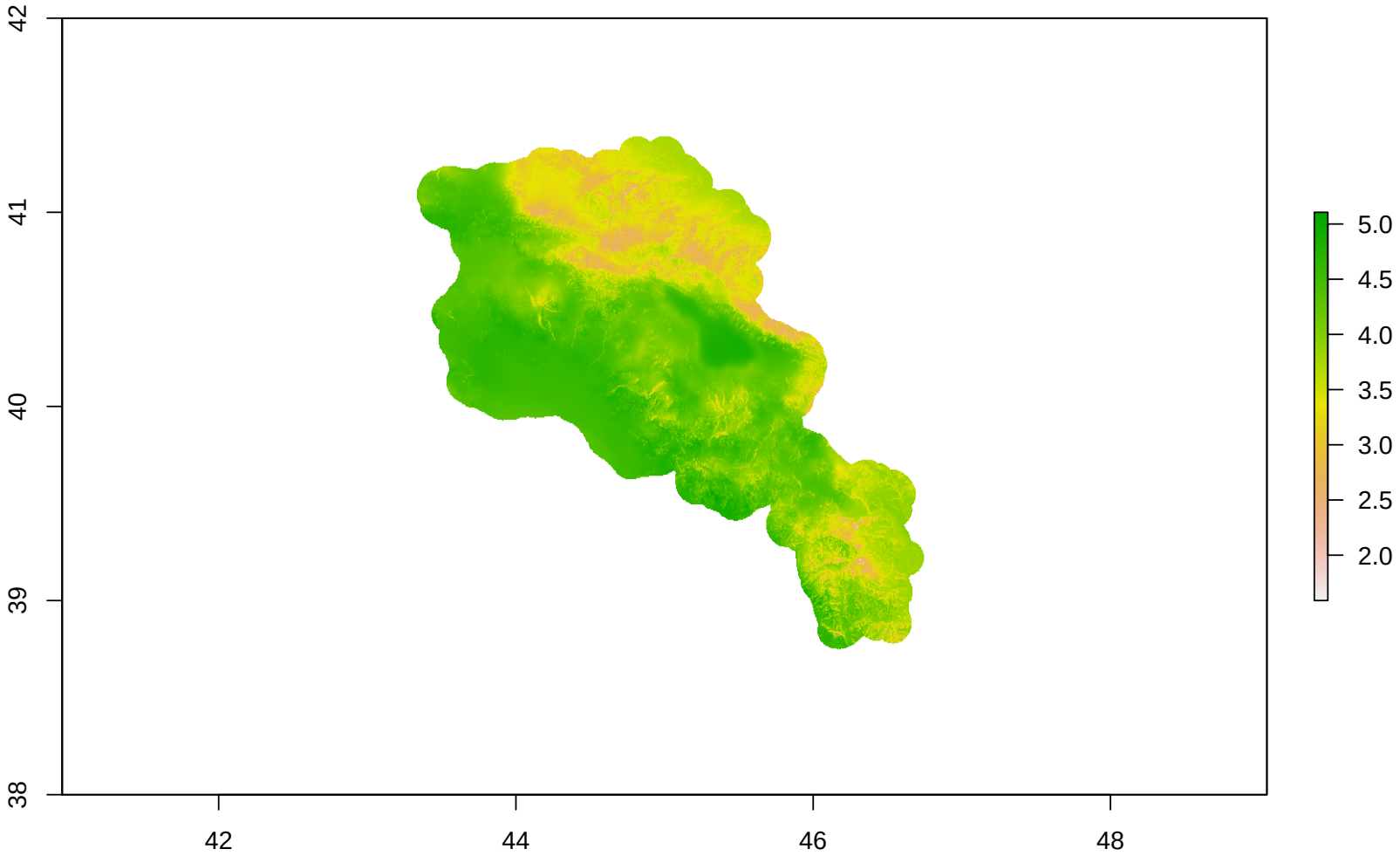
```
arm_solar <- raster('Data/Armenia_GlobalSolarAtlas-v2_GEOTIFF/DNI.tif')
arm_solar
```

```
## class      : RasterLayer
## dimensions : 1600, 1600, 2560000  (nrow, ncol, ncell)
## resolution : 0.0025, 0.0025  (x, y)
## extent     : 43, 47, 38, 42  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## source     : DNI.tif
## names      : DNI
## values     : 1.272, 5.129  (min, max)
```

Geospatial Data Visualization

You can use generic plot() to get the plot of the raster data

```
plot(arm_solar)
```



Geospatial Data Visualization

Transform into a dataframe

```
arm_solar_df <- as.data.frame(arm_solar, xy = T)
head(arm_solar_df)
```

```
##          x        y DNI
## 1 43.00125 41.99875  NA
## 2 43.00375 41.99875  NA
## 3 43.00625 41.99875  NA
## 4 43.00875 41.99875  NA
## 5 43.01125 41.99875  NA
## 6 43.01375 41.99875  NA
```

Geospatial Data Visualization

Lots of Missing values

```
dim(arm_solar_df)
```

```
## [1] 2560000       3
```

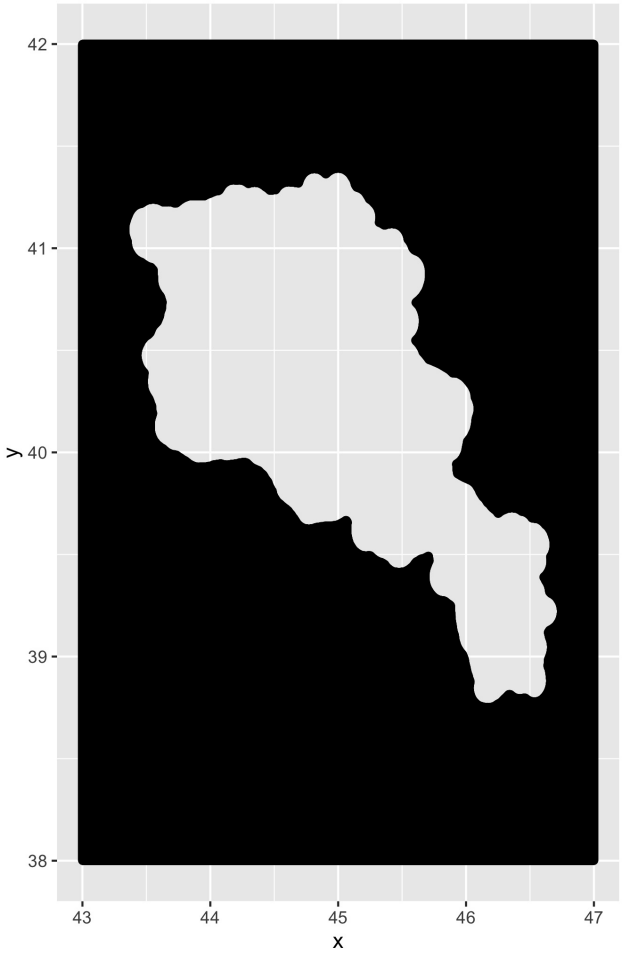
```
summary(arm_solar_df)
```

```
##        x             y             DNI
##  Min.   :43   Min.   :38   Min.   :1.3
##  1st Qu.:44   1st Qu.:39   1st Qu.:3.6
##  Median :45   Median :40   Median :4.1
##  Mean   :45   Mean   :40   Mean   :4.0
##  3rd Qu.:46   3rd Qu.:41   3rd Qu.:4.5
##  Max.   :47   Max.   :42   Max.   :5.1
##                            NA's   :1862183
```

Geospatial Data Visualization

Why do we have missing values ?

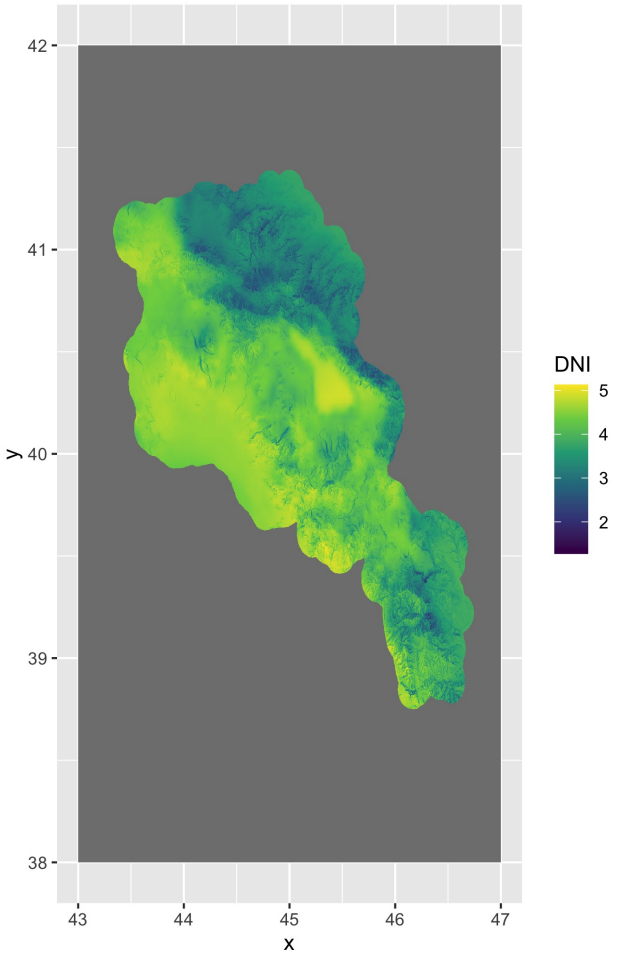
```
arm_solar_df %>% filter(is.na(DNI)) %>%
  ggplot() + geom_point(aes(x =x, y = y))
```



Geospatial Data Visualization

geom_raster() is used to visualize the raster data

```
ggplot(arm_solar_df ,aes(x =x, y = y, fill = DNI)) + geom_raster() +
  scale_fill_viridis_c()
```



Geospatial Data Visualization

Now we will try to plot DNI and administrative division of Armenia on the same plot. But we need to have them in the same CRS.

```
crs(arm_solar)
```

```
## Coordinate Reference System:
## Deprecated Proj.4 representation: +proj=longlat +datum=WGS84 +no_defs
## WKT2 2019 representation:
## GEOGCRS["unknown",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]],
##         ID["EPSG",6326]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433],
##         ID["EPSG",8901]],
##     CS[ellipsoidal,2],
##         AXIS["longitude",east,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433,
##                 ID["EPSG",9122]]],
##         AXIS["latitude",north,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433,
##                 ID["EPSG",9122]]]]
```

```
crs(arm)
```

```
## [1] "PROJCRS[\"WGS 84 / UTM zone 38N\",\n    BASEGEOGCRS[\"WGS 84\",\n        DATUM[\"World Geodetic System 1984\",\n            EL
```

Geospatial Data Visualization

1. Save long lat format as a string
2. Change the CRS for raster object
3. Create the dataframe again

```
arm_sf <- st_transform(arm_sf, crs(arm_solar))

arm_df <- arm_sf %>% group_by(MarzID) %>%
    st_cast("POINT")


arm_df = data.frame(Marz = arm_df$Name_Eng, st_coordinates(arm_df))
```

Geospatial Data Visualization

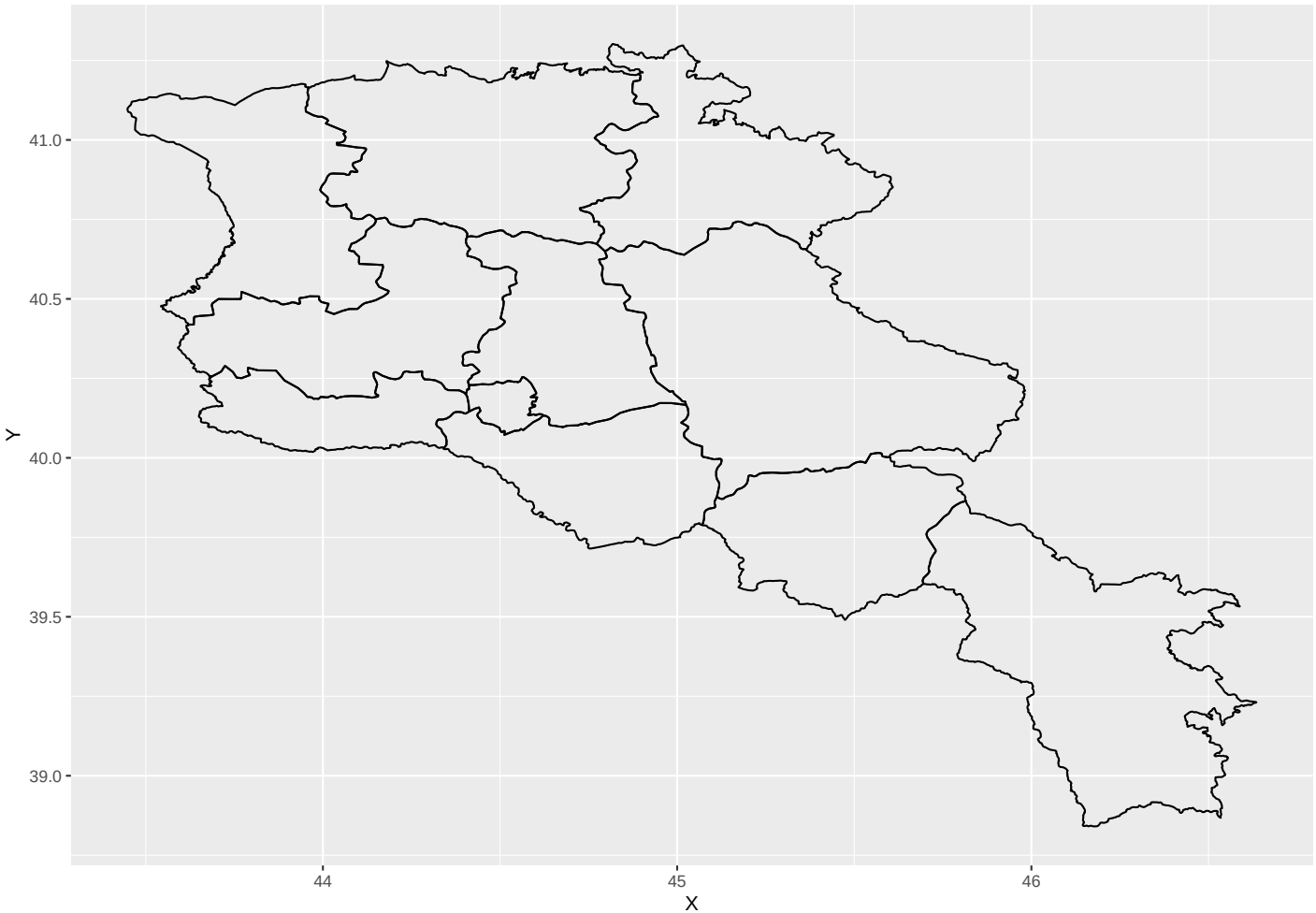
```
summary(arm_df)
```

```
##      Marz                X               Y        
##  Length:9460        Min.   :43.45   Min.   :38.84  
##  Class :character   1st Qu.:44.40   1st Qu.:39.61  
##  Mode  :character   Median :45.10   Median :40.23  
##                     Mean   :45.10   Mean   :40.23  
##                     3rd Qu.:45.88   3rd Qu.:40.90  
##                     Max.   :46.64   Max.   :41.30
```

Geospatial Data Visualization

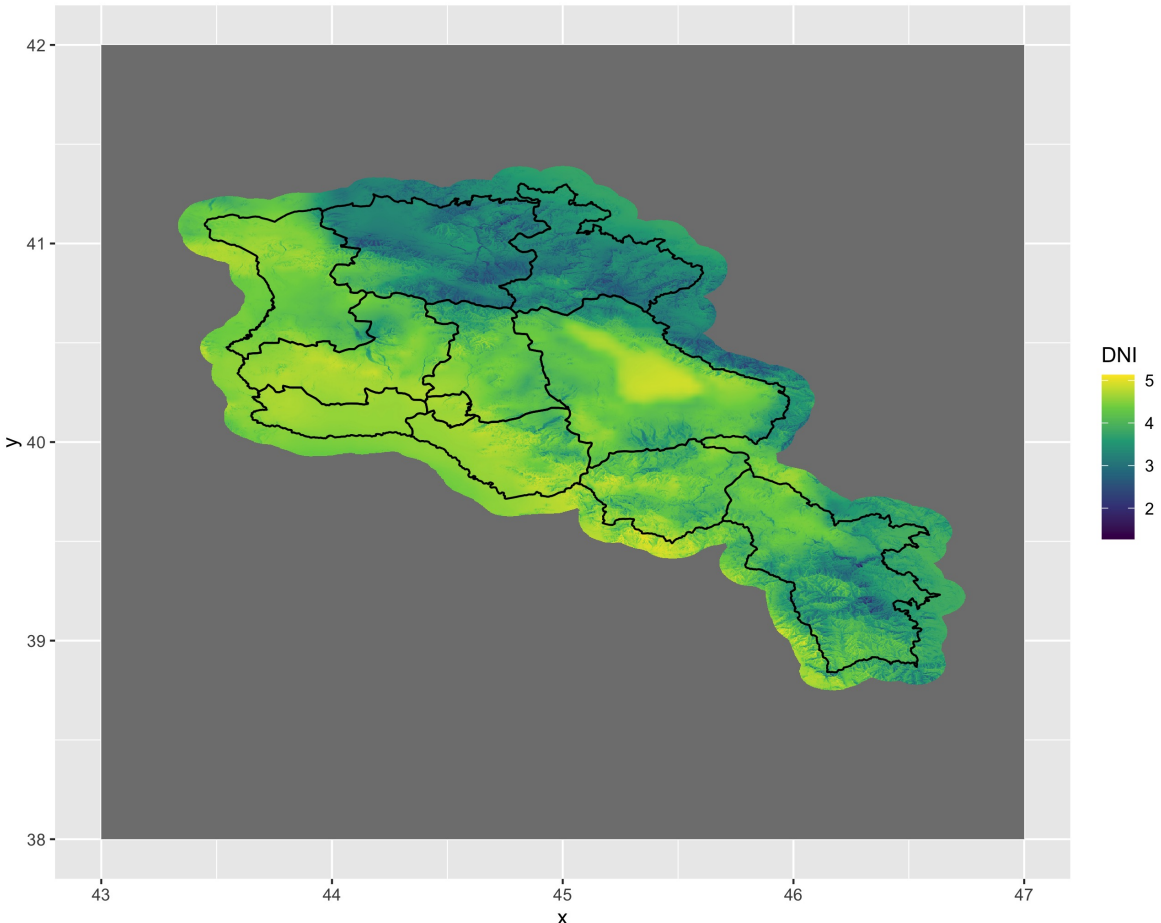
Use geom_path on top of geom_raster()

```
ggplot()  +
  geom_path(data = arm_df, aes(x = X, y = Y, group = Marz))
```



Geospatial Data Visualization

```
ggplot()  +
  geom_raster(data = arm_solar_df, aes(x =x, y = y, fill = DNI)) +
  geom_path(data = arm_df, aes(x = X, y = Y, group = Marz)) +
  scale_fill_viridis_c()
```



Geospatial Data Visualization

- ggplot has a special layer geom_sf() to deal with spatial data - shapefiles.
- In order to use it, you need to read shapefile using library(sf) function st_read
- Also change the CRS to long-lat format as with arm_soalr, but using function st_transform

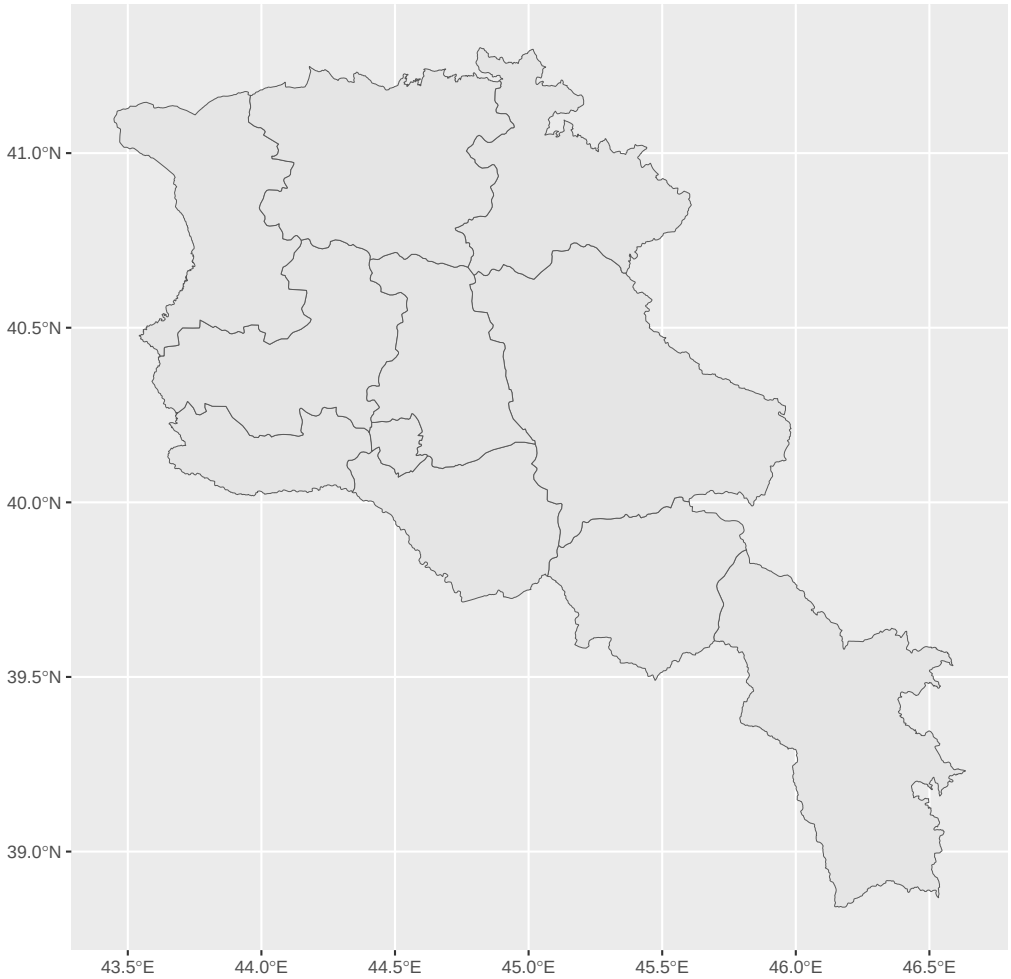
The result is an object with class sf, data.frame

```
arm <- st_read('Data/Armenia_Marzes/Armenia_Marzes.shp', quiet = T)
arm <- st_transform(arm, crs(arm_solar))
```

Geospatial Data Visualization

All you need at this stage is just to give ggplot the sf object for the **data** argument

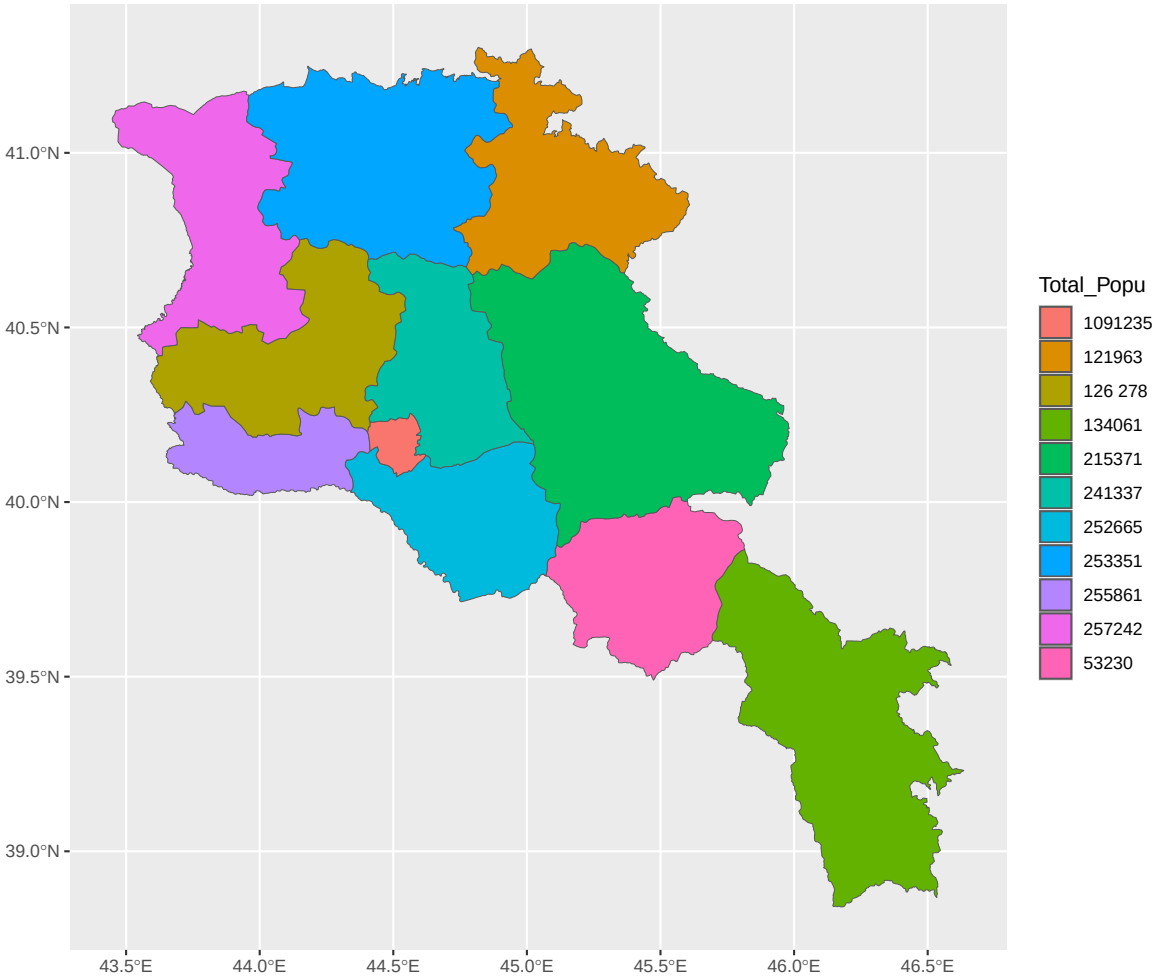
```
ggplot(data = arm) + geom_sf()
```



Geospatial Data Visualization

Our shape object has attributes tied to given polygons, such as Population, Name, etc, we can use them to visualize as well.

```
ggplot(data = arm, aes(fill = Total_Popu)) + geom_sf()
```



Geospatial Data Visualization

The variable is a character, need to transform to numeric first.

```
arm$Total_Popu
```

```
##  [1] "134061"  "53230"   "252665"  "255861"  "215371"  "1091235" "126 278"
##  [8] "241337"  "257242"  "121963"  "253351"
```

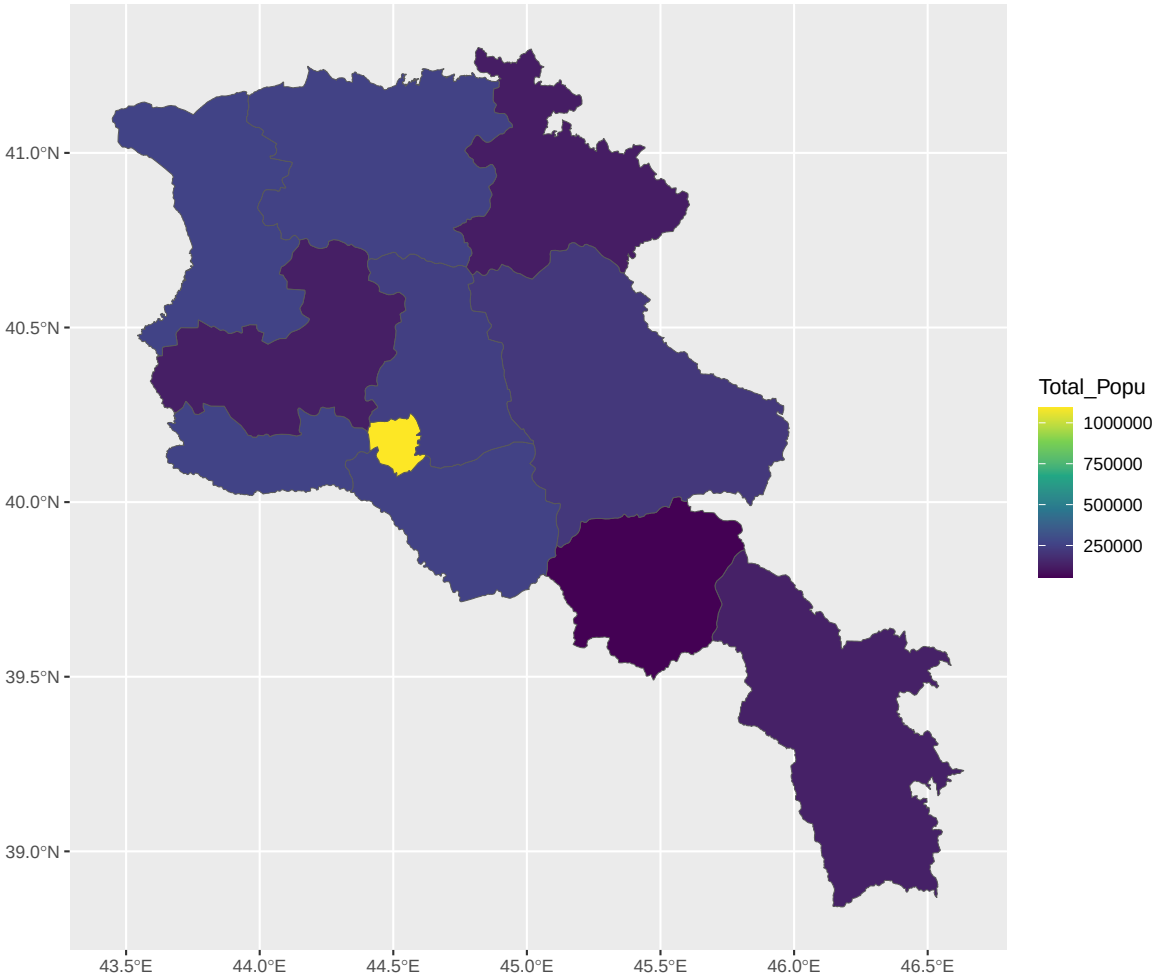
Geospatial Data Visualization

Remove white spaces

```
arm$Total_Popu <- gsub(pattern = " ", x = arm$Total_Popu,
                       replacement = "") %>%  as.numeric()
```

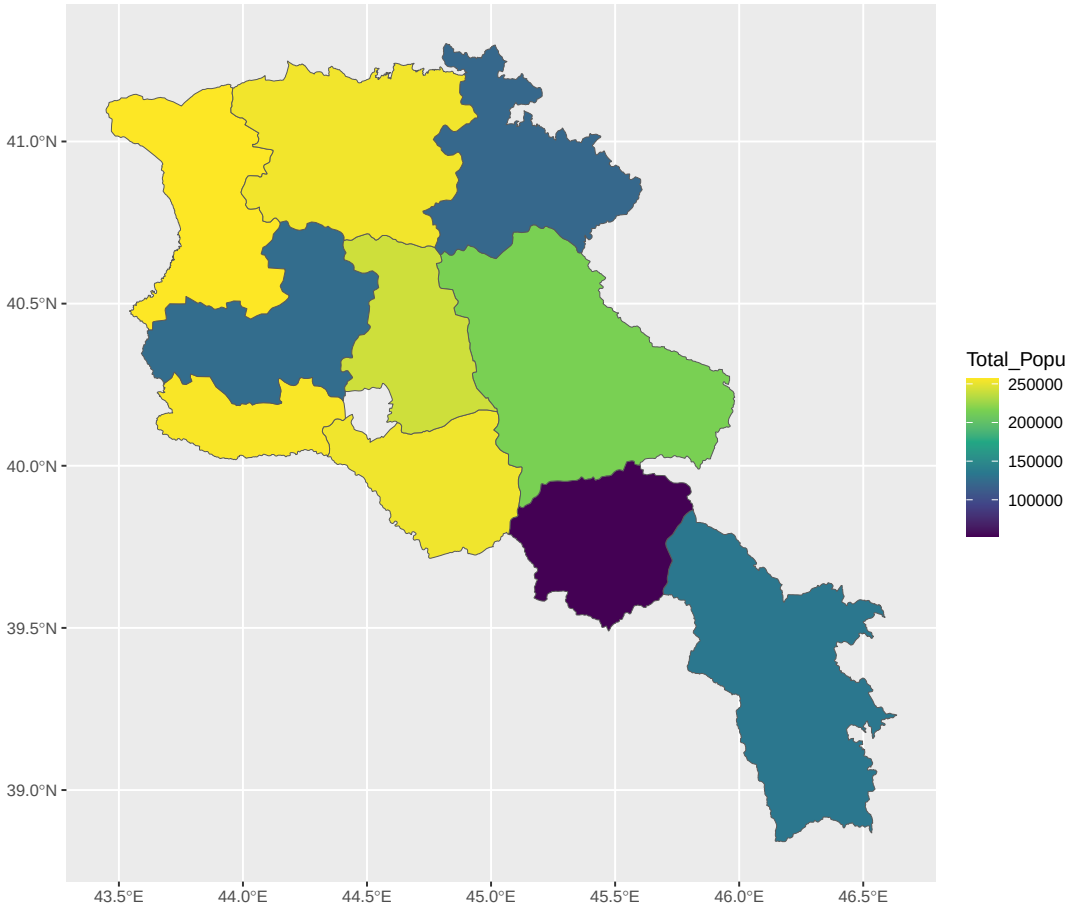
Geospatial Data Visualization

```
ggplot(data = arm, aes(fill = Total_Popu)) + geom_sf() +
  scale_fill_viridis_c()
```



Geospatial Data Visualization

```
arm_state <- arm[-6, ]

ggplot(data = arm_state, aes(fill = Total_Popu)) + geom_sf() +
  scale_fill_viridis_c()
```



Geospatial Data Visualization

You can add your own variable to the sf object and use it You can do simple operations with sf object

Add

```
arm$test <- 1:11
```

Delete

```
arm$test <- NULL
```

Geospatial Data Visualization

Read GDP data

```
gdp <- read.csv('Data/gdp by regions.csv')
names(gdp)
```

```
## [1] "Region"        "GDP_mln"       "Per_Capita_gdp"
```

Geospatial Data Visualization

Use dplyr to join two datasets together

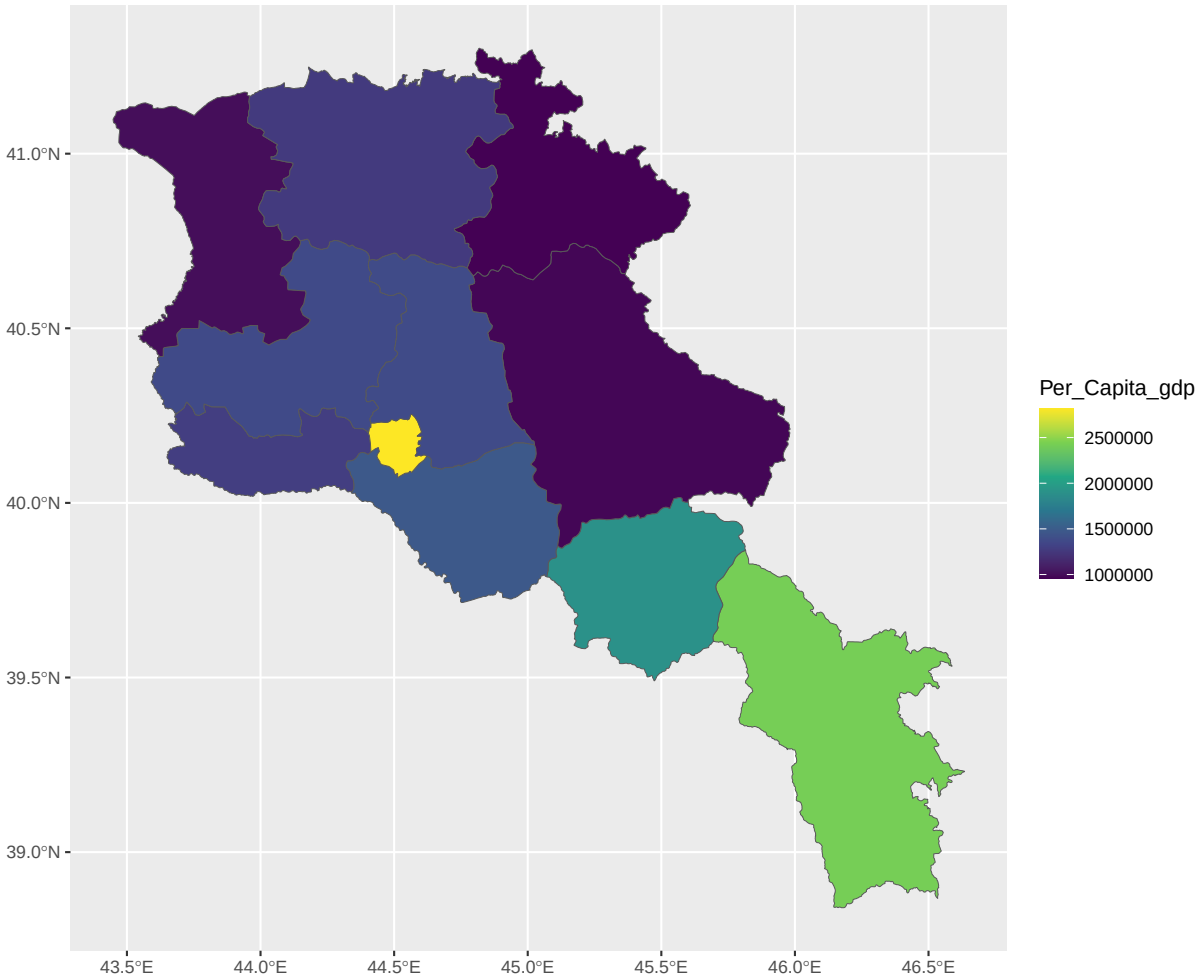
```
arm <- arm %>% left_join(gdp, by = c('Name_Eng' = "Region"))
arm$Per_Capita_gdp
```

```
##  [1] 2417001 1892716 1478727 1284327  974118 2816433 1371121 1375270 1002954
## [10]  956908 1263372
```

Geospatial Data Visualization

GDP per capita

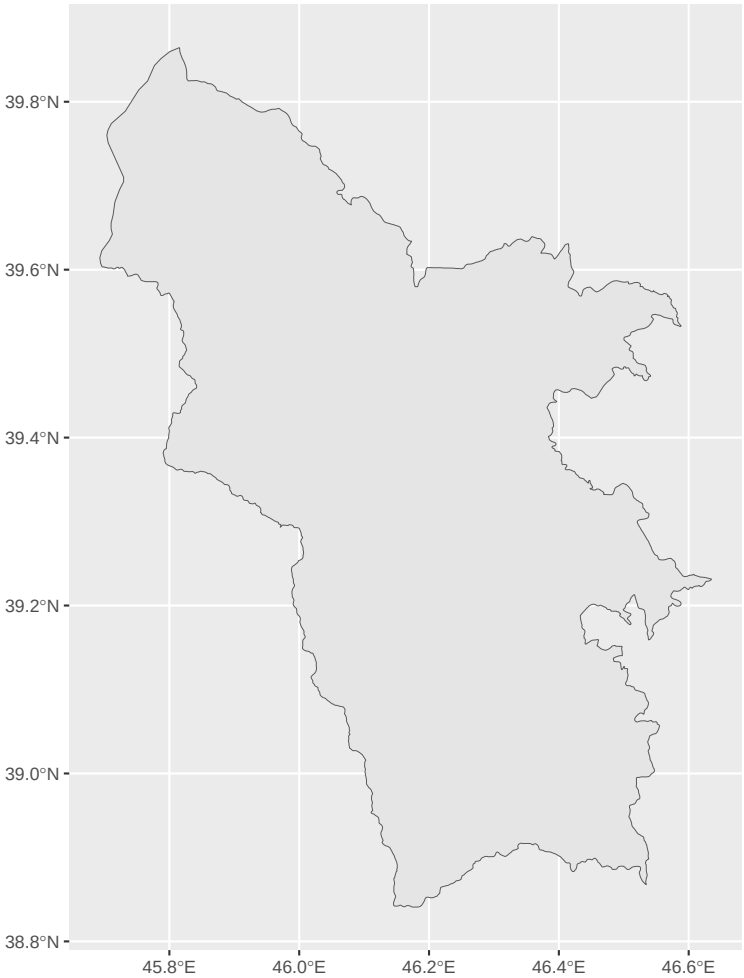
```
ggplot(data = arm, aes(fill = Per_Capita_gdp)) + geom_sf() +
  scale_fill_viridis_c()
```



Geospatial Data Visualization

sf object can be filtered and slices using dplyr, just like any other object

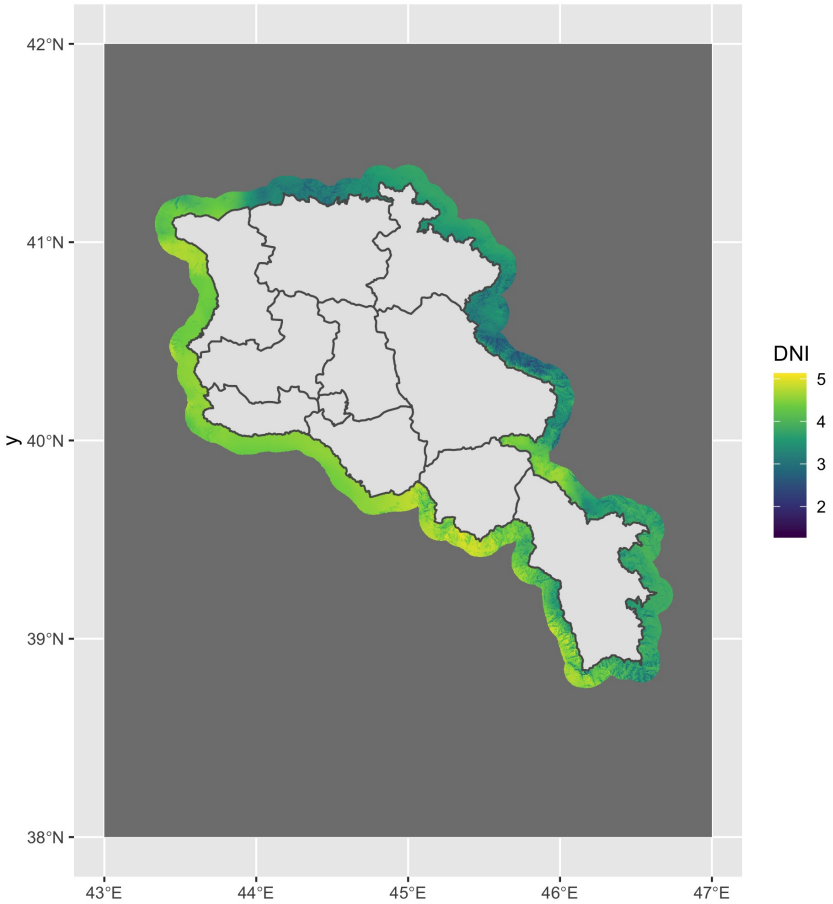
```
arm %>% filter(Name_Eng == 'SYUNIK') %>%
  ggplot() + geom_sf()
```



Geospatial Data Visualization

Using geom_sf() and geom_raster() together

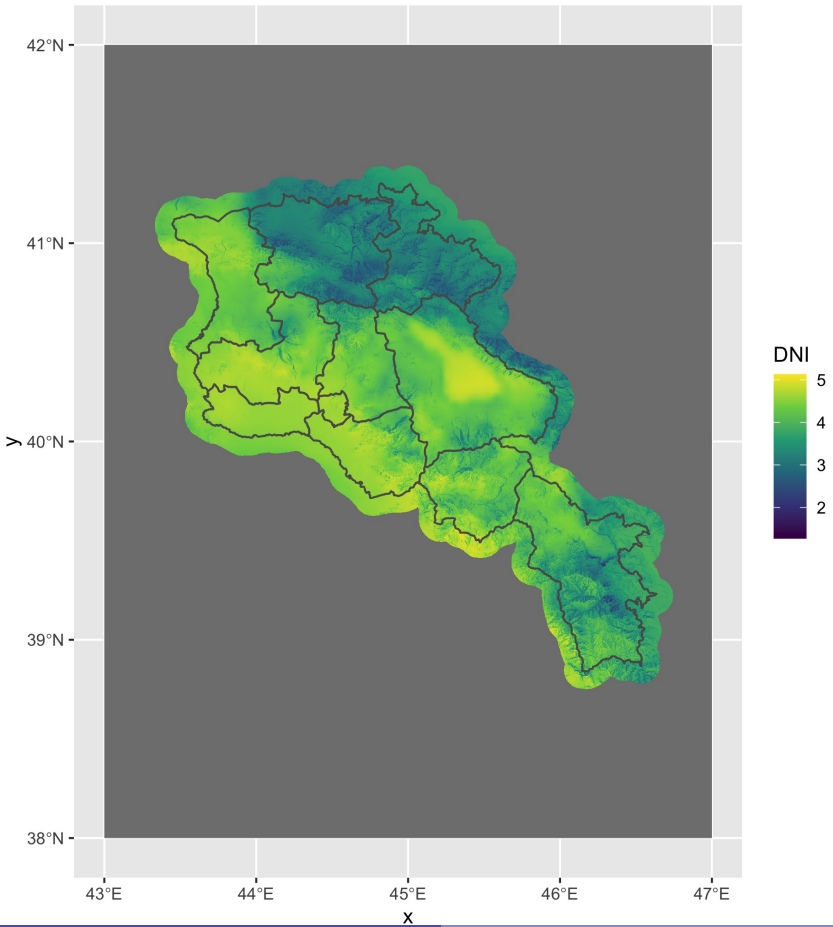
```
ggplot()  +
  geom_raster(data = arm_solar_df, aes(x =x, y = y, fill = DNI)) +
  geom_sf(data = arm) + scale_fill_viridis_c()
```



Geospatial Data Visualization

We will do a trick, set fill = NA

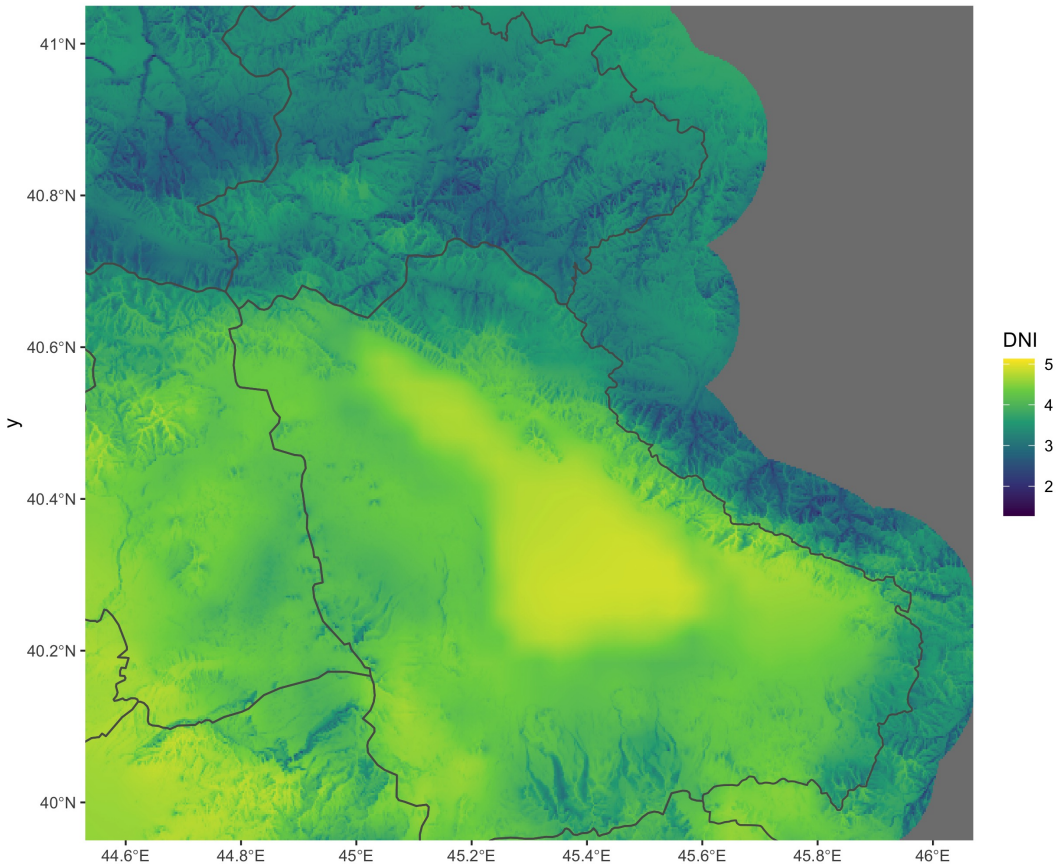
```
ggplot()  +
  geom_raster(data = arm_solar_df, aes(x =x, y = y, fill = DNI)) +
  geom_sf(data = arm, fill = NA) + scale_fill_viridis_c()
```



Geospatial Data Visualization

You can use ggplot coord_layer to zoom in, you need to use coord_sf

```
ggplot()  +
  geom_raster(data = arm_solar_df, aes(x =x, y = y, fill = DNI)) +
  geom_sf(data = arm, fill = NA) + scale_fill_viridis_c() +
  coord_sf(xlim = c(44.6,46), ylim = c(40,41))
```

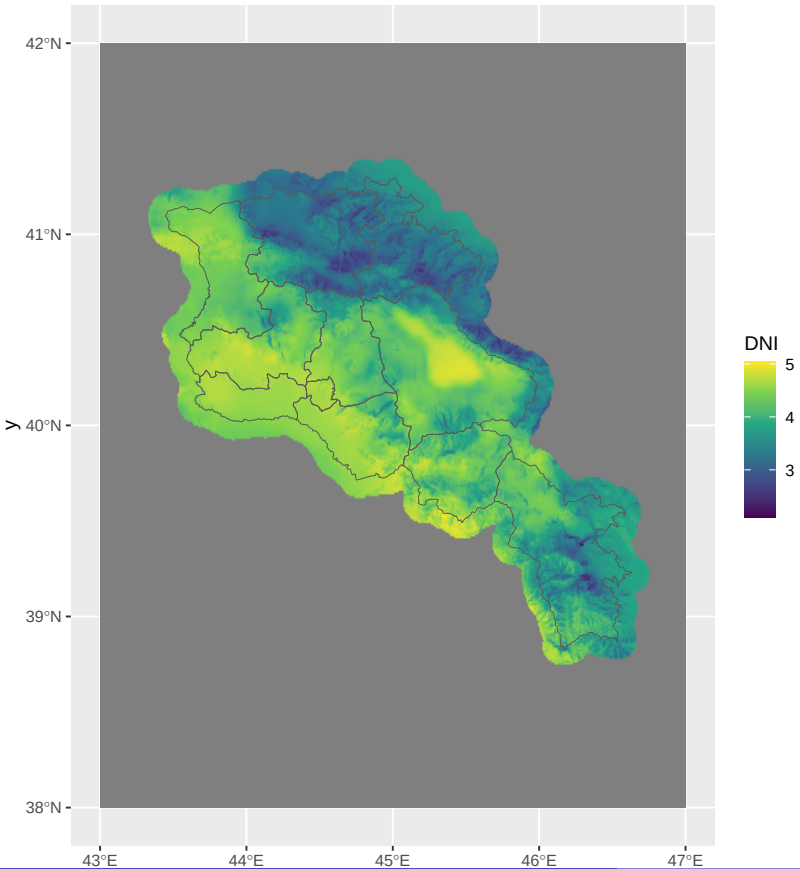


Geospatial Data Visualization

When raster resolution is high, it can create problems such as memory usage, time needed to draw the map, etc.
To decrease the resolution of the raster image, you can use function raster::aggregate().
The function aggregates grid cells into larger grid cells using a user-defined function (for example, mean or max).
The additional argument to provide is **fact** and **fun**.
fact is a positive integer. Aggregation factor is expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor)

Geospatial Data Visualization

```
arm_solar_df <- arm_solar %>% raster::aggregate(fact = 4) %>%
  as.data.frame(xy = T)
ggplot()  +
  geom_raster(data = arm_solar_df, aes(x =x, y = y, fill = DNI)) +
  geom_sf(data = arm, fill = NA) + scale_fill_viridis_c()
```



Case: John Snow and Cholera

- In the beginning of 19th century miasma theory was dominant, stating that diseases such as cholera and bubonic plague were caused by pollution or a noxious form of “bad air”.
- John Snow was a well known physicist who was a proponent of germ theory.
- In 1854 he mapped the cholera cases and water pumps in Soho district of London. He did statistical analysis that showed that cholera cases were clustered around one water pump.

Case: John Snow and Cholera

On proceeding to the spot, I found that nearly all the deaths had taken place within a short distance of the Broad Street pump. There were only ten deaths in houses situated decidedly nearer to another street-pump. In five of these cases the families of the deceased persons informed me that they always sent to the pump in Broad Street, as they preferred the water to that of the pumps which were nearer. In three other cases, the deceased were children who went to school near the pump in Broad Street.

Case: John Snow and Cholera

Lets look at the Snow's cholera data

```
deaths <- st_read('Data/SnowGIS_SHP/Cholera_Deaths.shp', quiet = T)
deaths
```

```
## Simple feature collection with 250 features and 2 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 529160.3 ymin: 180857.9 xmax: 529655.9 ymax: 181306.2
## Projected CRS: OSGB36 / British National Grid
## First 10 features:
##    Id Count                  geometry
## 1   0     3 POINT (529308.7 181031.4)
## 2   0     2 POINT (529312.2 181025.2)
## 3   0     1 POINT (529314.4 181020.3)
## 4   0     1 POINT (529317.4 181014.3)
## 5   0     4 POINT (529320.7 181007.9)
## 6   0     2   POINT (529336.7 181006)
## 7   0     2 POINT (529290.1 181024.4)
## 8   0     2   POINT (529301 181021.2)
## 9   0     3   POINT (529285 181020.2)
## 10  0     2 POINT (529288.4 181031.8)
```

Case: John Snow and Cholera

```
pumps <- st_read('Data/SnowGIS_SHP/Pumps.shp', quiet = T)
pumps
```

```
## Simple feature collection with 8 features and 1 field
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 529183.7 ymin: 180660.5 xmax: 529748.9 ymax: 181193.7
## Projected CRS: OSGB36 / British National Grid
##   Id                 geometry
## 1  0 POINT (529396.5 181025.1)
## 2  0 POINT (529192.5 181079.4)
## 3  0 POINT (529183.7 181193.7)
## 4  0 POINT (529748.9 180924.2)
## 5  0 POINT (529613.2 180896.8)
## 6  0 POINT (529453.6 180826.4)
## 7  0 POINT (529593.7 180660.5)
## 8  0 POINT (529296.1 180794.8)
```

Case: John Snow and Cholera

Check if coordinate reference system is the same for both

```
crs(pumps)
```

```
## [1] "PROJCRS[\"OSGB36 / British National Grid\",\n    BASEGEOGCRS[\"OSGB36\",\n        DATUM[\"Ordnance Survey of Great Britain 193
```

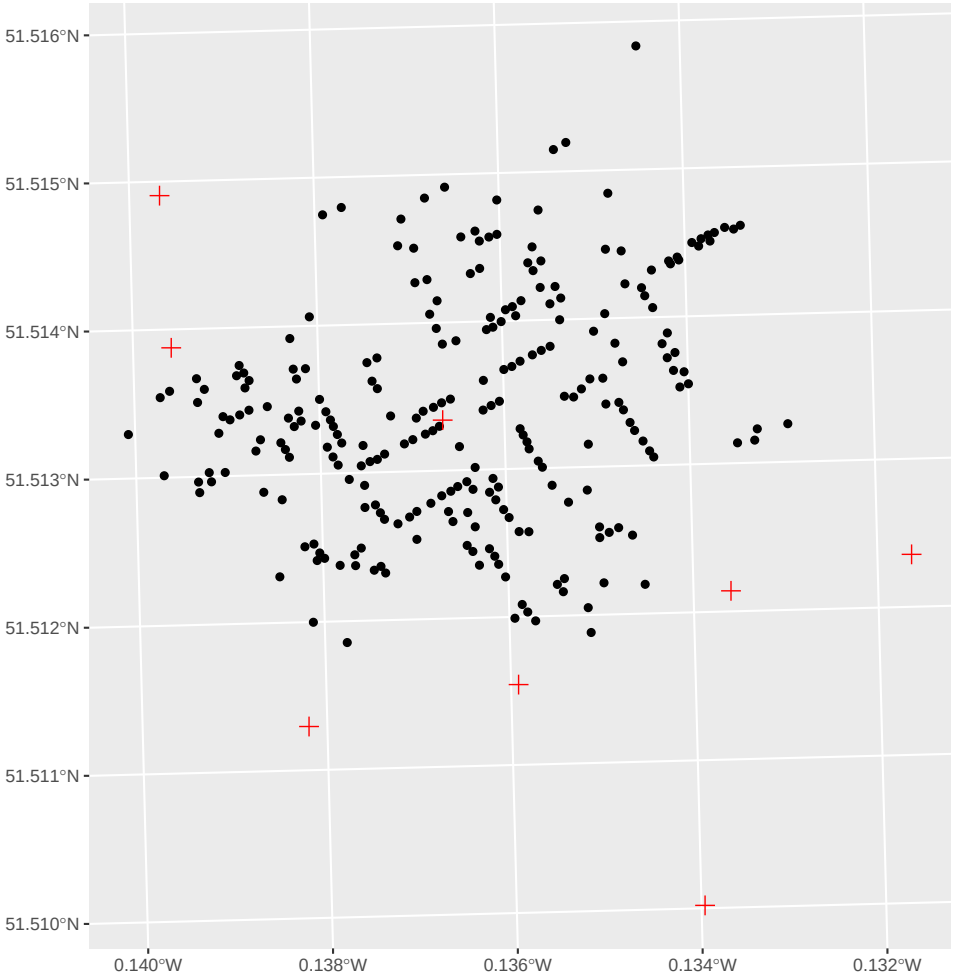
```
crs(deaths)
```

```
## [1] "PROJCRS[\"OSGB36 / British National Grid\",\n    BASEGEOGCRS[\"OSGB36\",\n        DATUM[\"Ordnance Survey of Great Britain 193
```

Case: John Snow and Cholera

Plot deaths and pumps together

```
ggplot() +  geom_sf(data = deaths) +
  geom_sf(data = pumps, color = 'red', shape= 3, size = 3)
```



Case: John Snow and Cholera

We can also use the actual map of Soho.

```
img <- readJPEG('Data/SnowGIS_SHP/SnowMap.jpg')
ggplot() +   background_image(img) +
  geom_sf(data = deaths) +
  geom_sf(data = pumps, color = 'red', shape= 3, size = 3) +
  theme(axis.text = element_blank(), axis.ticks = element_blank())
```

Case: John Snow and Cholera

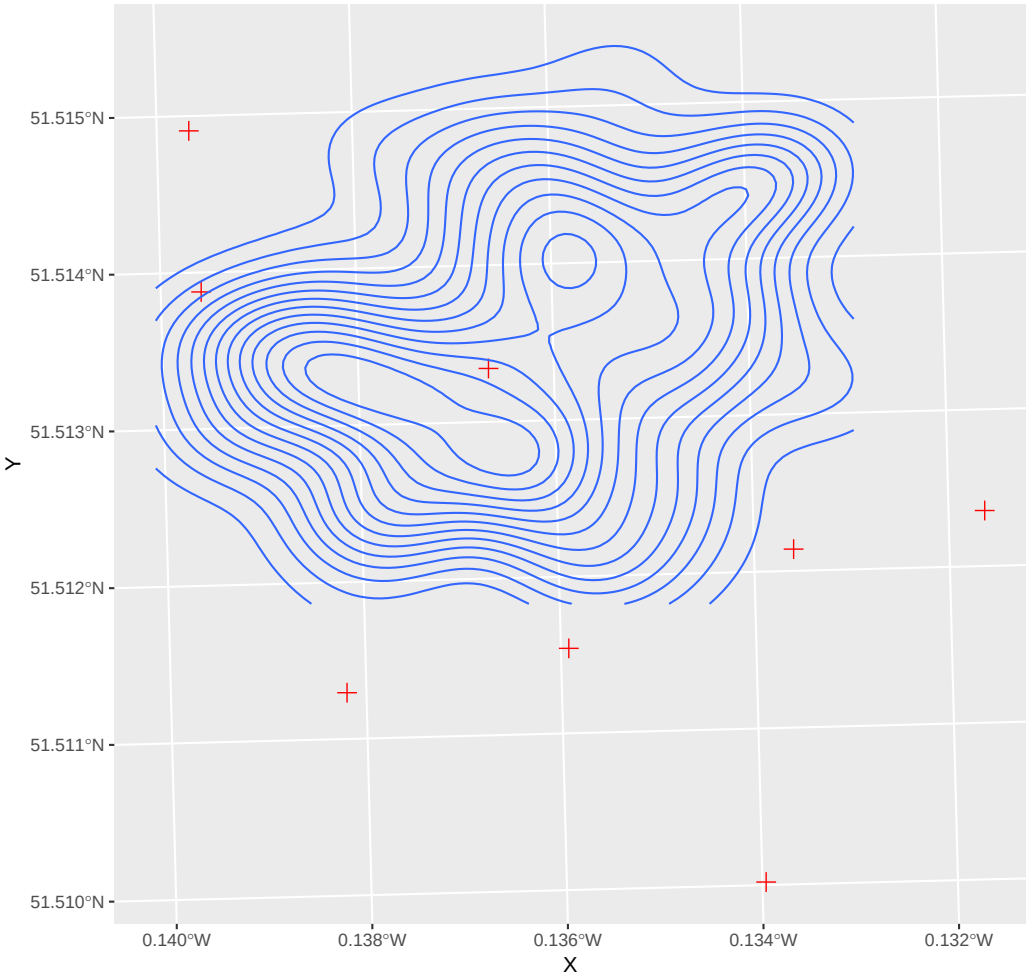
Case: John Snow and Cholera

- To visualize how the deaths are concentrated around the Broad Street pump, we can draw 2d density plots.
- First we need to extract coordinates.

```
deaths_coords <- deaths %>%  st_coordinates() %>% as.data.frame()
```

Case: John Snow and Cholera

```
ggplot() +
  geom_sf(data = pumps, color = 'red', shape= 3, size = 3) +
  geom_density_2d(data = deaths_coords, aes(X,Y))
```



Choropleth maps

- Choropleth maps are popular thematic maps used to represent statistical data through various shading patterns or symbols on predetermined geographic areas (i.e. countries).
- They are used to visualize some statistical measure like mean, sum, median, across geographical regions.
- They are good at utilizing data to easily represent variability of the desired measurement, across a region.

Choropleth maps

There are different packages in R that you can use to get the shapefiles of geographic regions.

```
sf_world <- ne_countries(returnclass='sf')
ggplot() + geom_sf(data = sf_world)
```

Choropleth maps

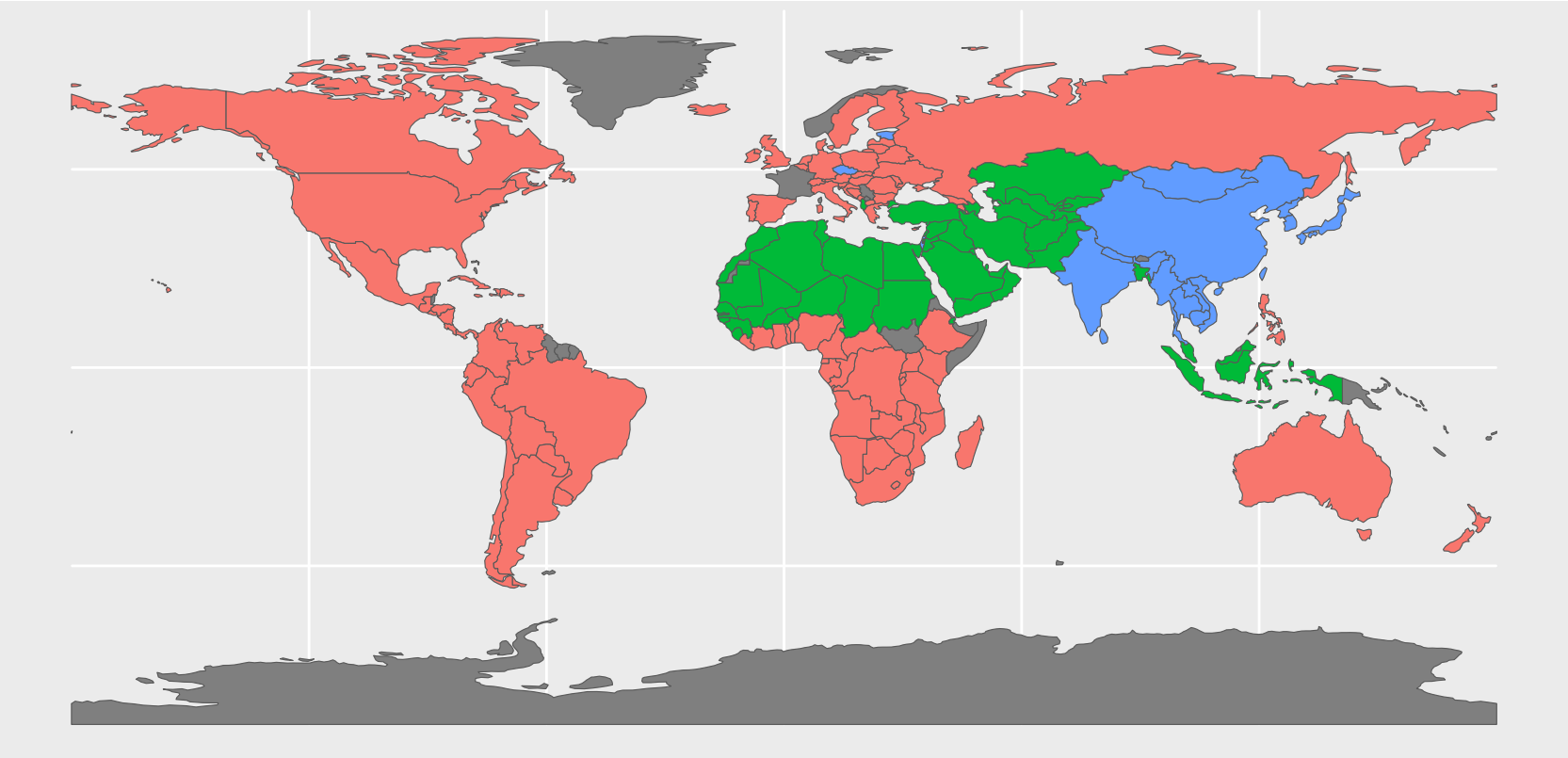
Read the fertility rate and join with the **sf** data

```
fertility <- read.csv('Data/fertility.csv')
sf_world <- sf_world %>%
  left_join(fertility[,c("Code", "Fertility_rate", 'Religion', 'GDP_PPP')],
                                        by = c('iso_a3' = 'Code'))
```

Choropleth maps

Categorical variable: Religion

```
ggplot() + geom_sf(data = sf_world, aes(fill = Religion))
```

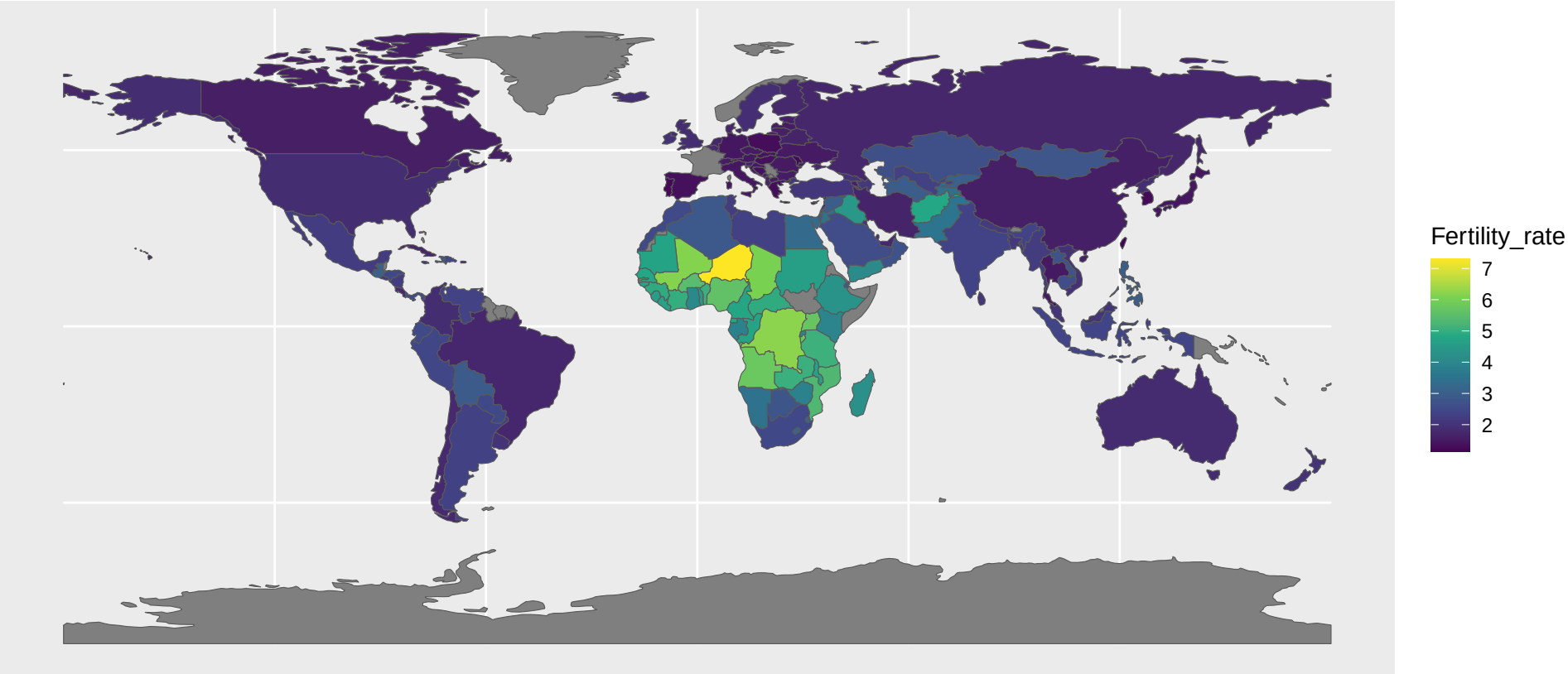


Choropleth maps

Continuous variable: Fertility Rate

```
ggplot() + geom_sf(data = sf_world, aes(fill = Fertility_rate)) +
  scale_fill_viridis_c()
```

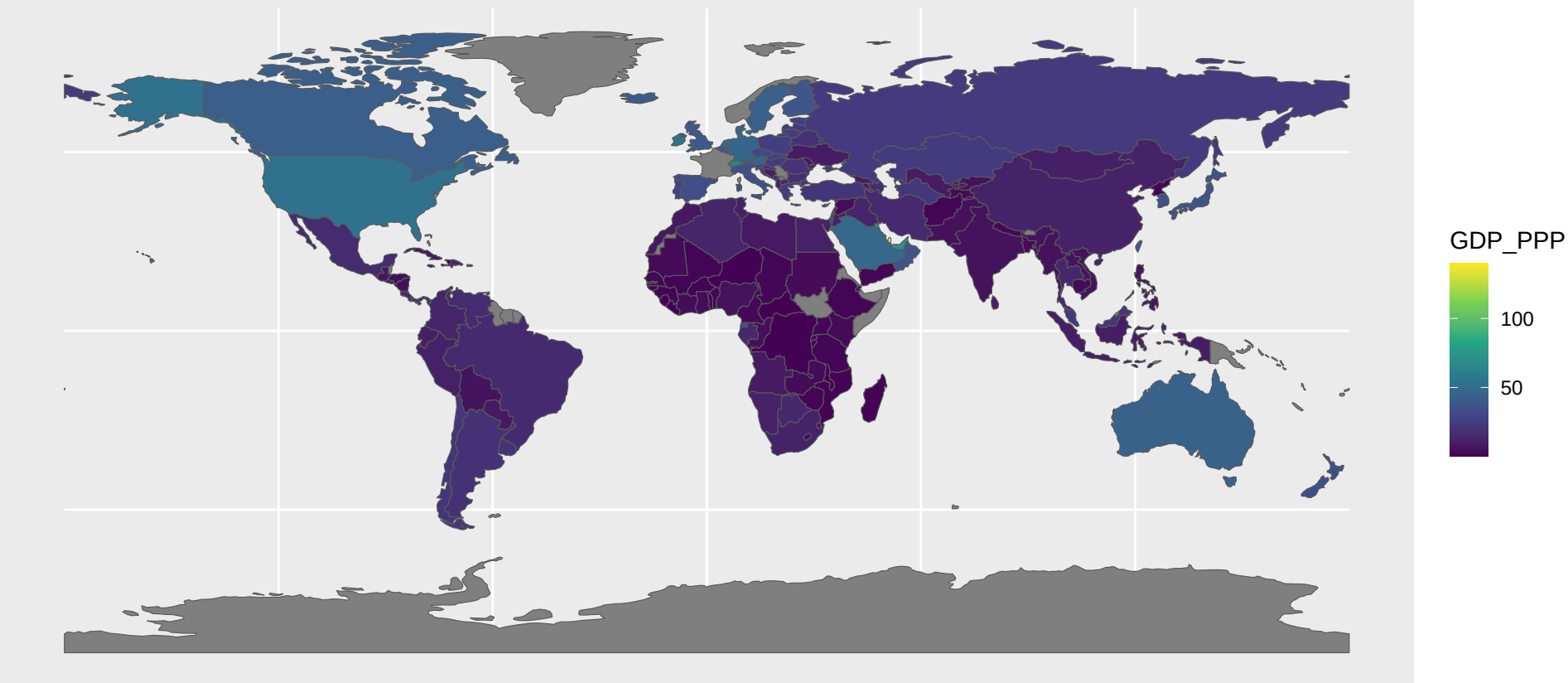
Choropleth maps



Choropleth maps

```
ggplot() + geom_sf(data = sf_world, aes(fill = GDP_PPP)) +
  scale_fill_viridis_c()
```

Choropleth maps



Choropleth maps

- When mapping with continuous variable, sometimes (almost always) it is hard to see the differences and individual results.
- The solution is to transform the continuous variable into categorical/ordinal variable

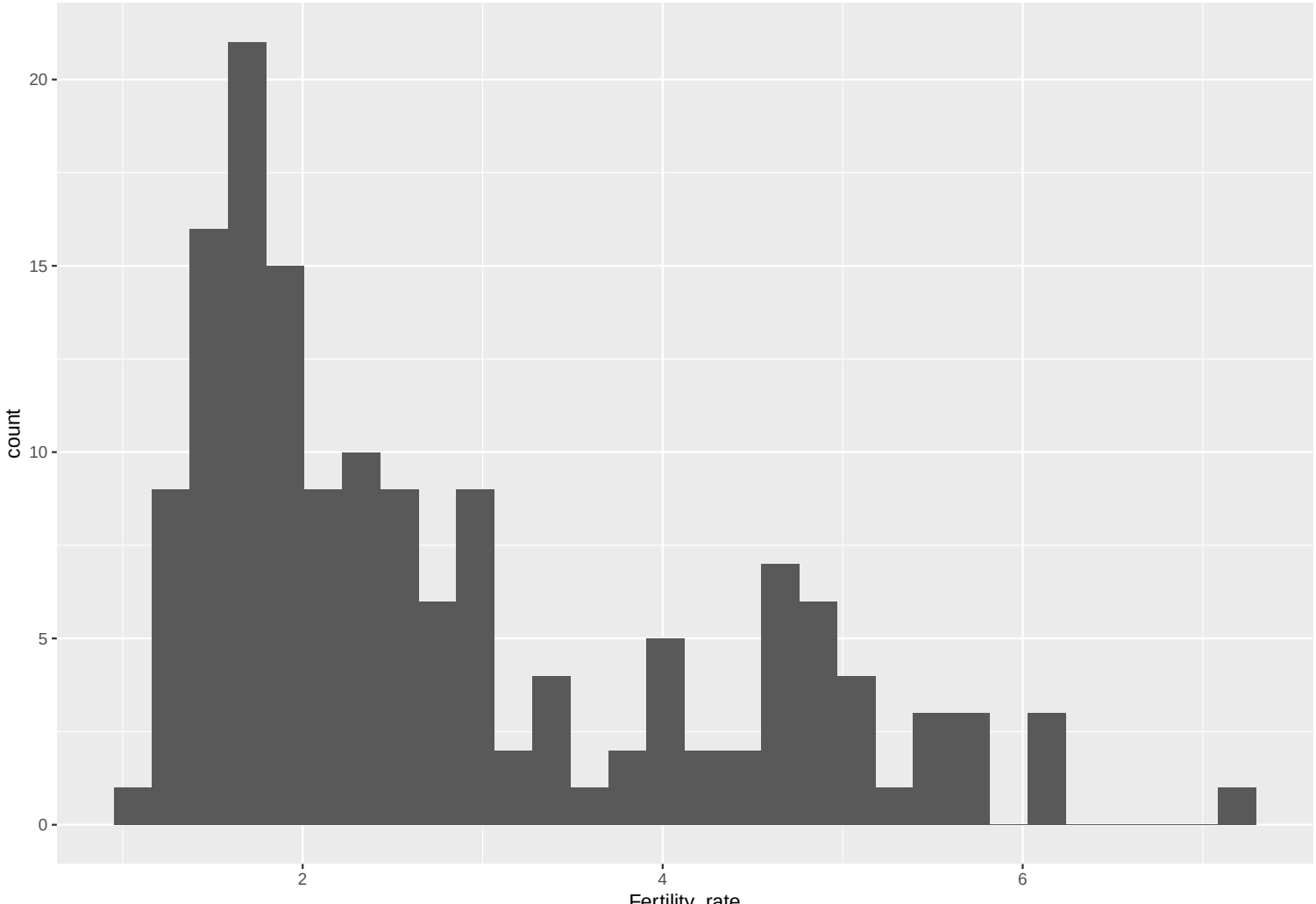
There are several methods to do so, we will look at some of them

- Equal length intervals
- Quantiles
- Natural breaks optimization methods
- Standard deviation

Choropleth maps: Equal length intervals

Look at the distribution of fertility rate

```
ggplot(data = sf_world, aes(x = Fertility_rate)) + geom_histogram()
```



Choropleth maps: Equal length intervals

Let's say we want to transform the Fertility variable to a categorical one with 5 categories/intervals.

```
summary(sf_world$Fertility_rate)
```

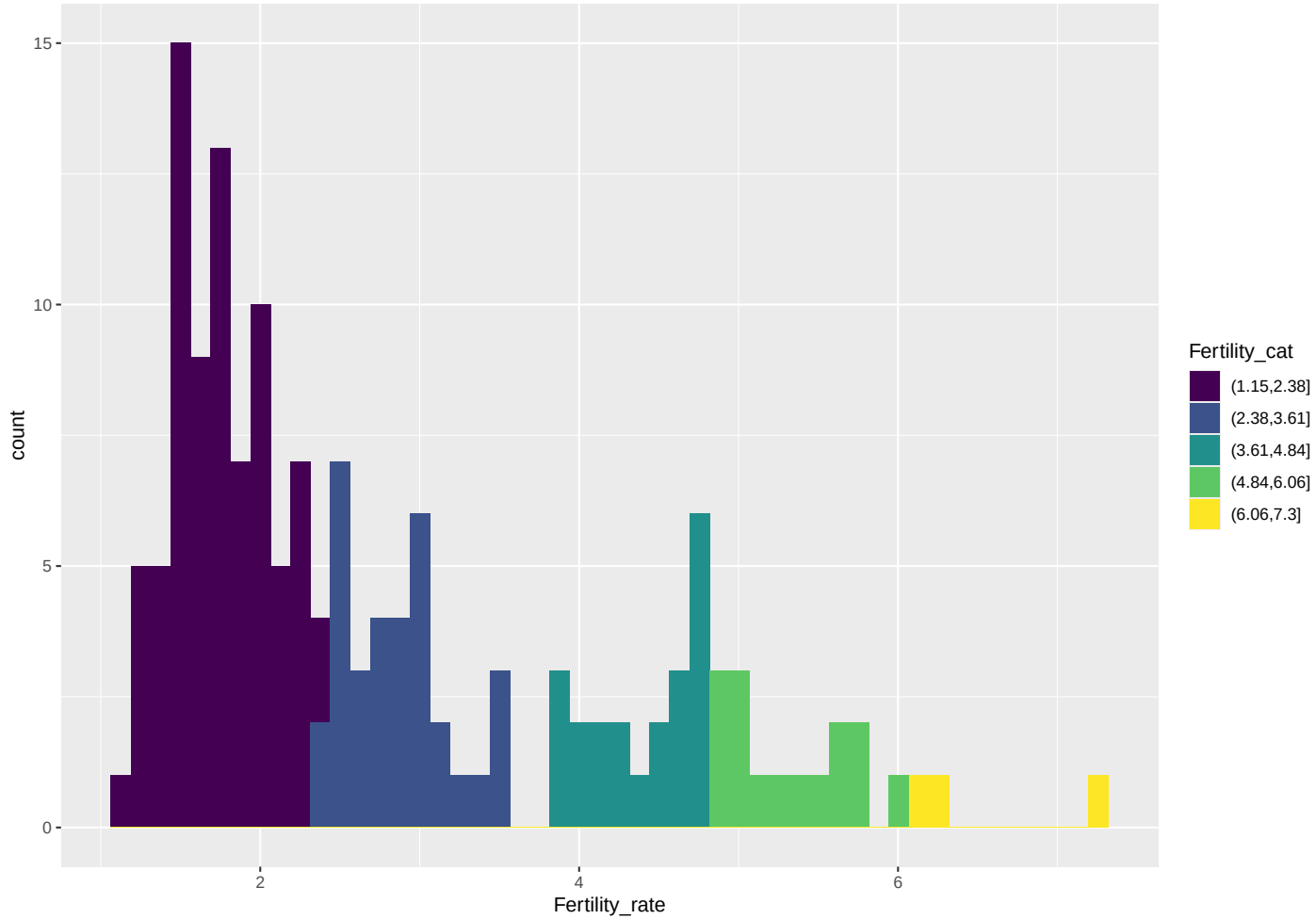
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   1.158   1.728   2.312   2.814   3.843   7.290      26
```

```
sf_world$Fertility_cat <- cut(sf_world$Fertility_rate,
                              breaks = 5, ordered_result = T)
```

Choropleth maps: Equal length intervals

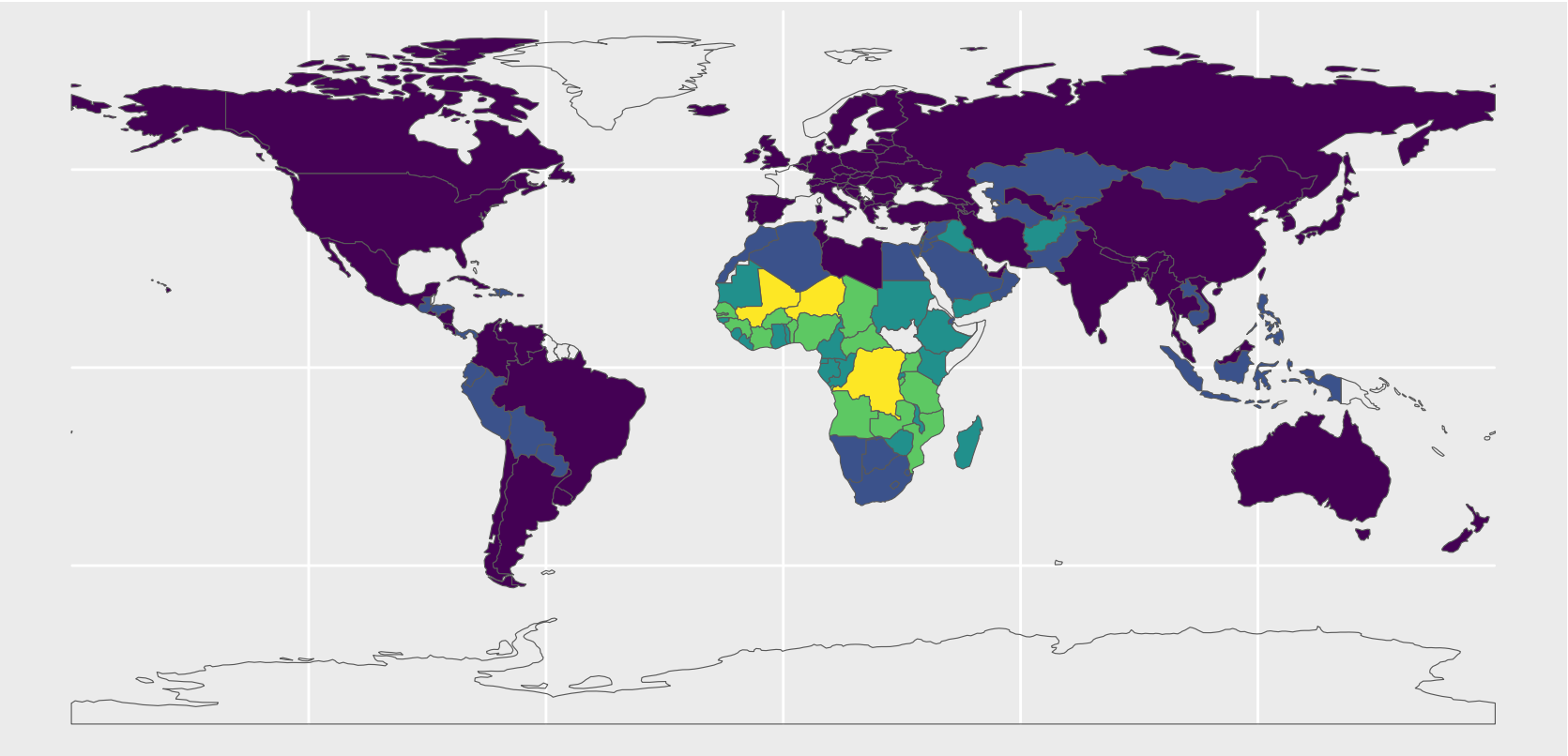
Let's look at the distribution

```
ggplot(data = sf_world, aes(x = Fertility_rate, fill = Fertility_cat)) +
  geom_histogram(bins = 50)
```



Choropleth maps: Equal length intervals

```
ggplot() + geom_sf(data = sf_world, aes(fill = Fertility_cat))
```

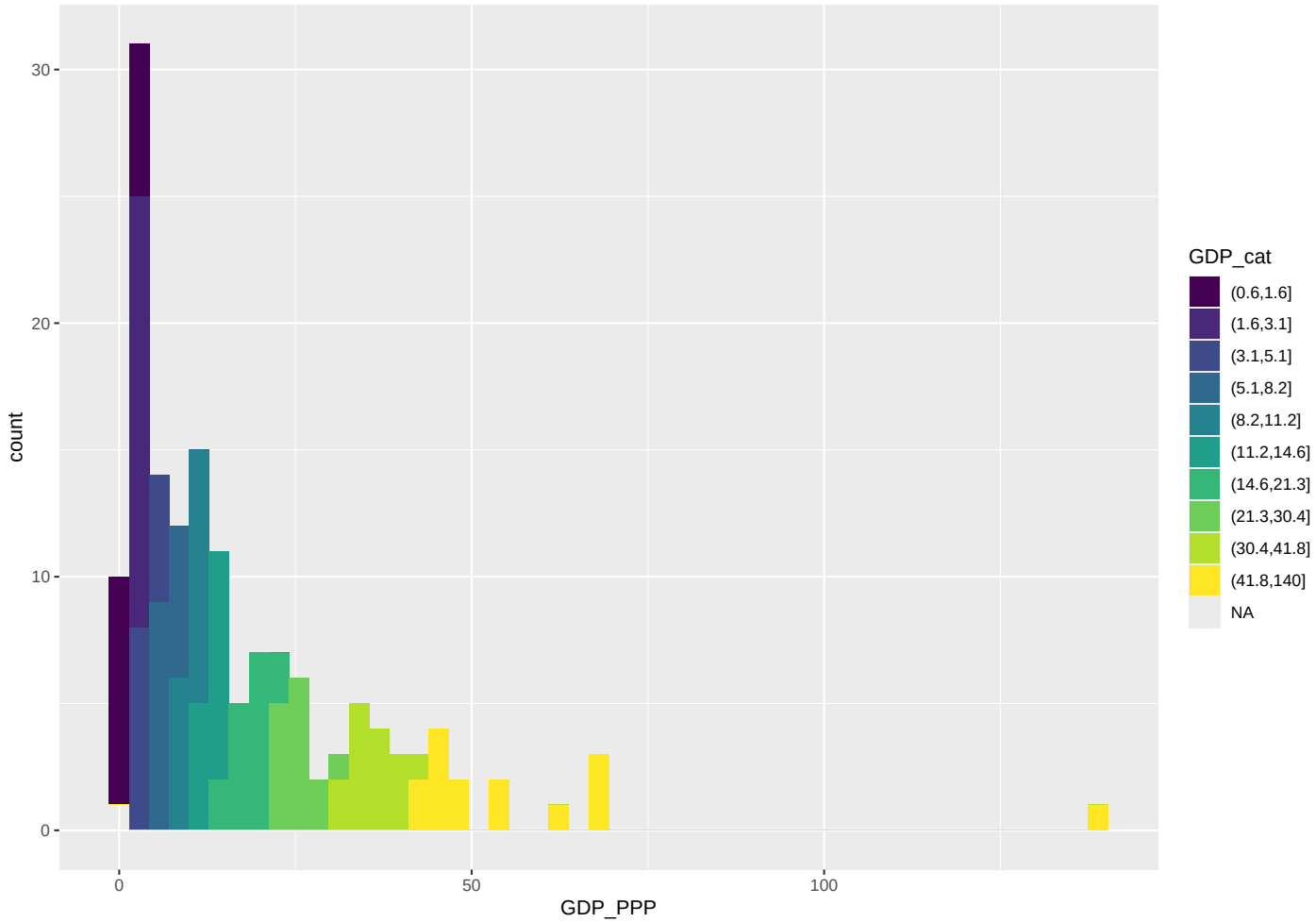


Choropleth maps: Quantiles

```
quant <- quantile(x = sf_world$GDP_PPP,
                  probs = seq(0,1, 0.1), na.rm = T)
sf_world$GDP_cat <- cut(sf_world$GDP_PPP,
                        breaks = quant, ordered_result = T)
```

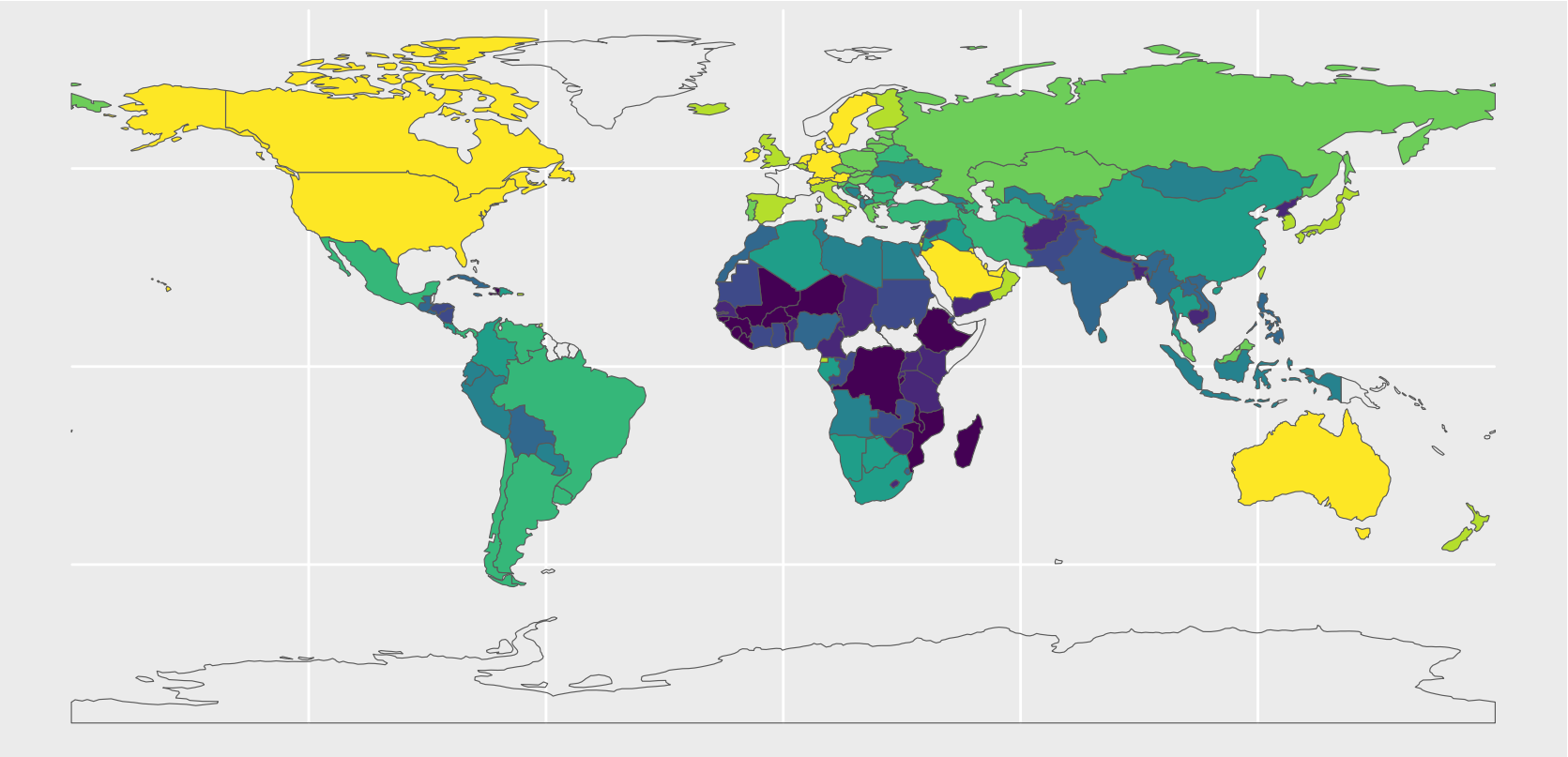
Choropleth maps: Equal length intervals

```
ggplot(data = sf_world, aes(x = GDP_PPP, fill = GDP_cat)) +
  geom_histogram(bins = 50)
```



Choropleth maps: Quantiles

```
ggplot() + geom_sf(data = sf_world, aes(fill = GDP_cat))
```



Choropleth maps: Quantiles

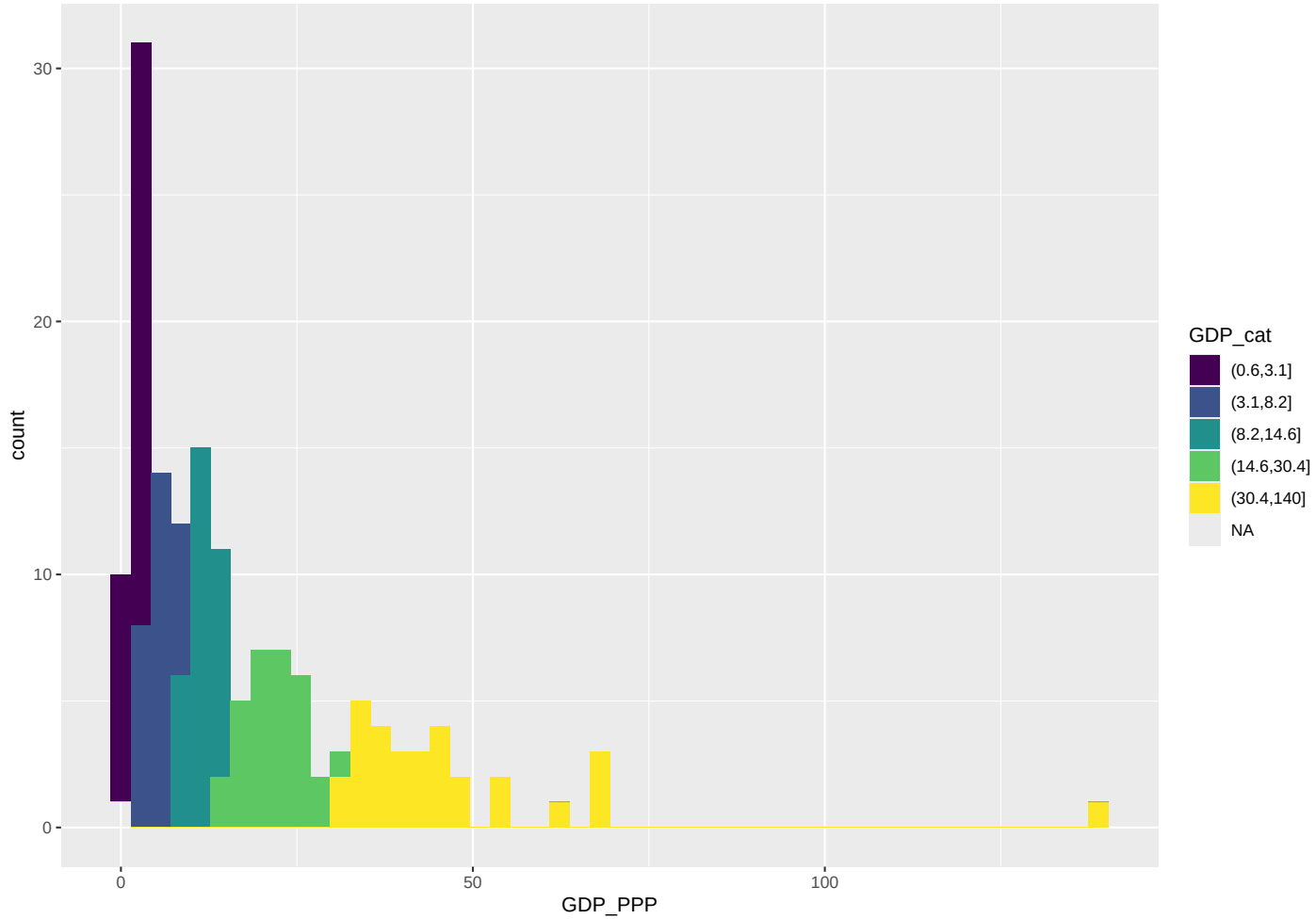
If you are lost in colors, change the number of quantiles

```
quant <- quantile(x = sf_world$GDP_PPP,
                  probs = seq(0,1, 0.2), na.rm = T)
sf_world$GDP_cat <- cut(sf_world$GDP_PPP,
                        breaks = quant, ordered_result = T)
```

Choropleth maps: Quantiles

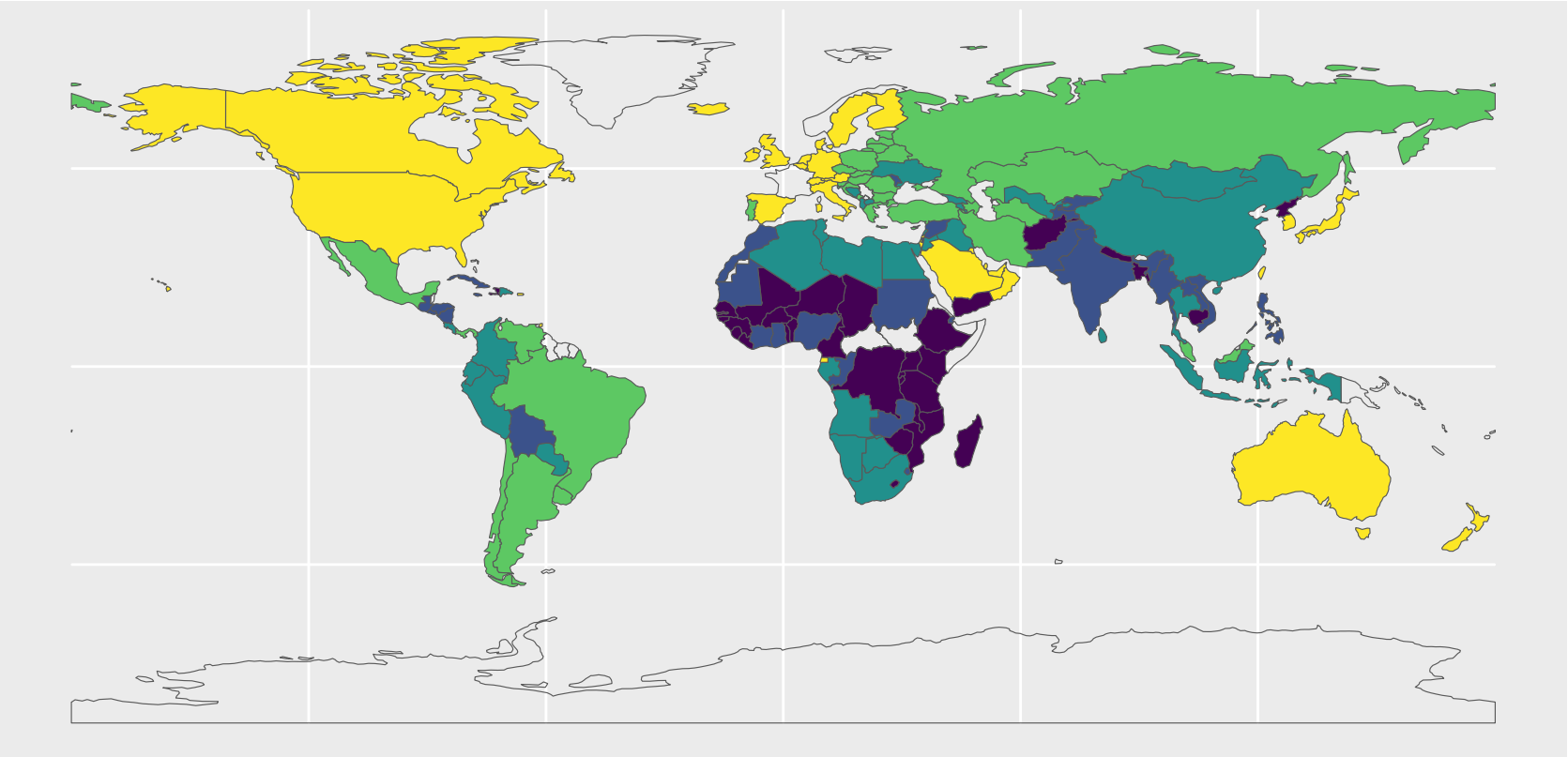
The distribution

```
ggplot(data = sf_world, aes(x = GDP_PPP, fill = GDP_cat)) +
  geom_histogram(bins = 50)
```



Choropleth maps: Quantiles

```
ggplot() + geom_sf(data = sf_world, aes(fill =GDP_cat))
```



Choropleth maps: Natural Breaks (Optimization)

- Jenks optimization algorithm is widely used within GIS packages, these are forms of variance-minimization classification.
- Breaks are typically uneven, and are selected to separate values where large changes in value occur.
- May be significantly affected by the number of classes selected and tends to have unusual class boundaries.
- Does not work well when the variance in the variable is small

Choropleth maps: Natural Breaks (Optimization)

```
jenks_int <- classInt::classIntervals(sf_world$Fertility_rate, n = 5,
                                      style = 'jenks')
jenks_int
```

```
## style: jenks
##   one of 18,671,940 possible partitions of this variable into 5 classes
## [1.158,1.876] (1.876,2.594]  (2.594,3.55]  (3.55,5.079]  (5.079,7.29]
##            52            38            22            28            11
```

```
jenks_int$brks
```

```
## [1] 1.158 1.876 2.594 3.550 5.079 7.290
```

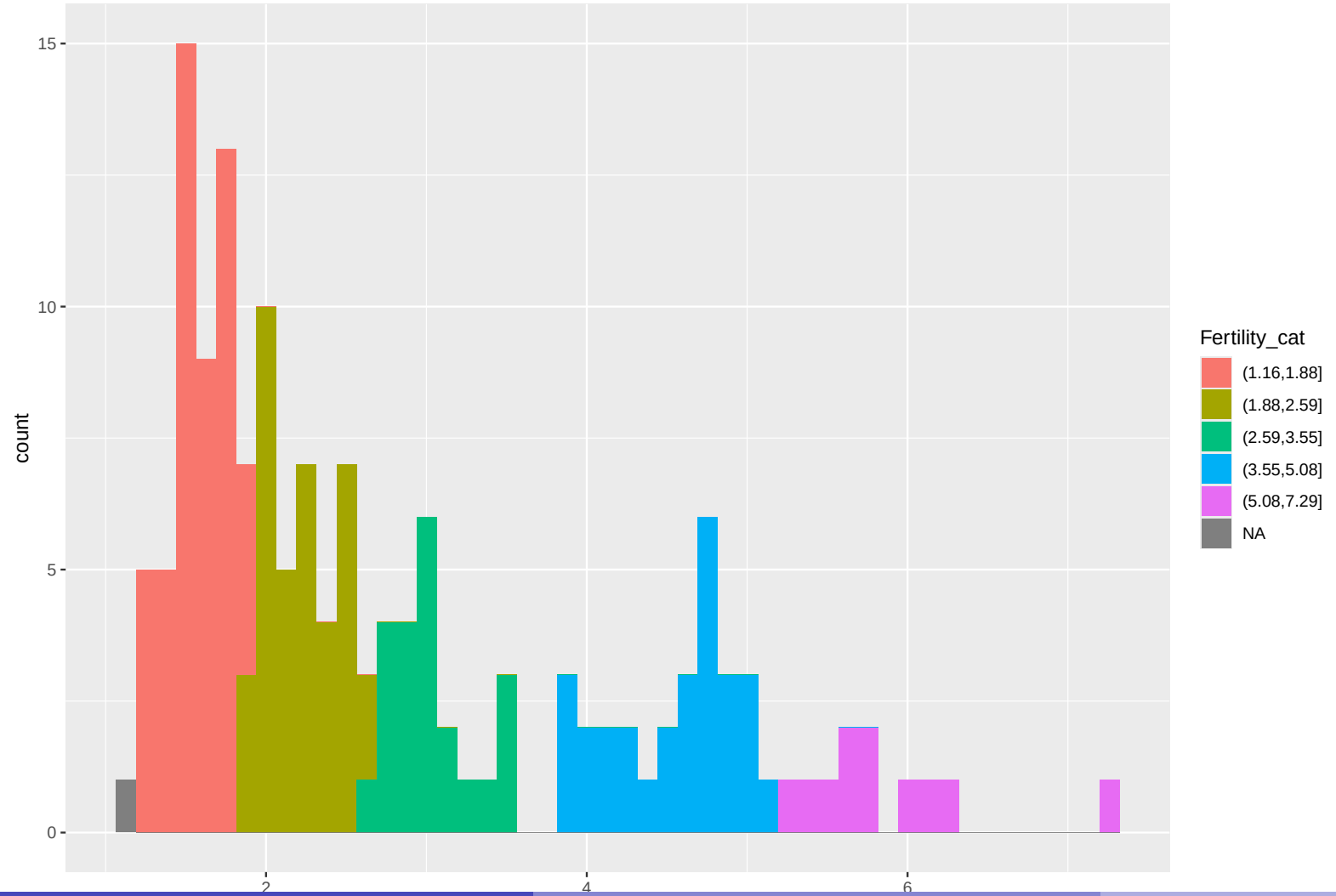
Choropleth maps: Natural Breaks (Optimization)

Create the variable

```
sf_world$Fertility_cat <- cut(x = sf_world$Fertility_rate,
                              breaks = jenks_int$brks,
                              right = T)
```

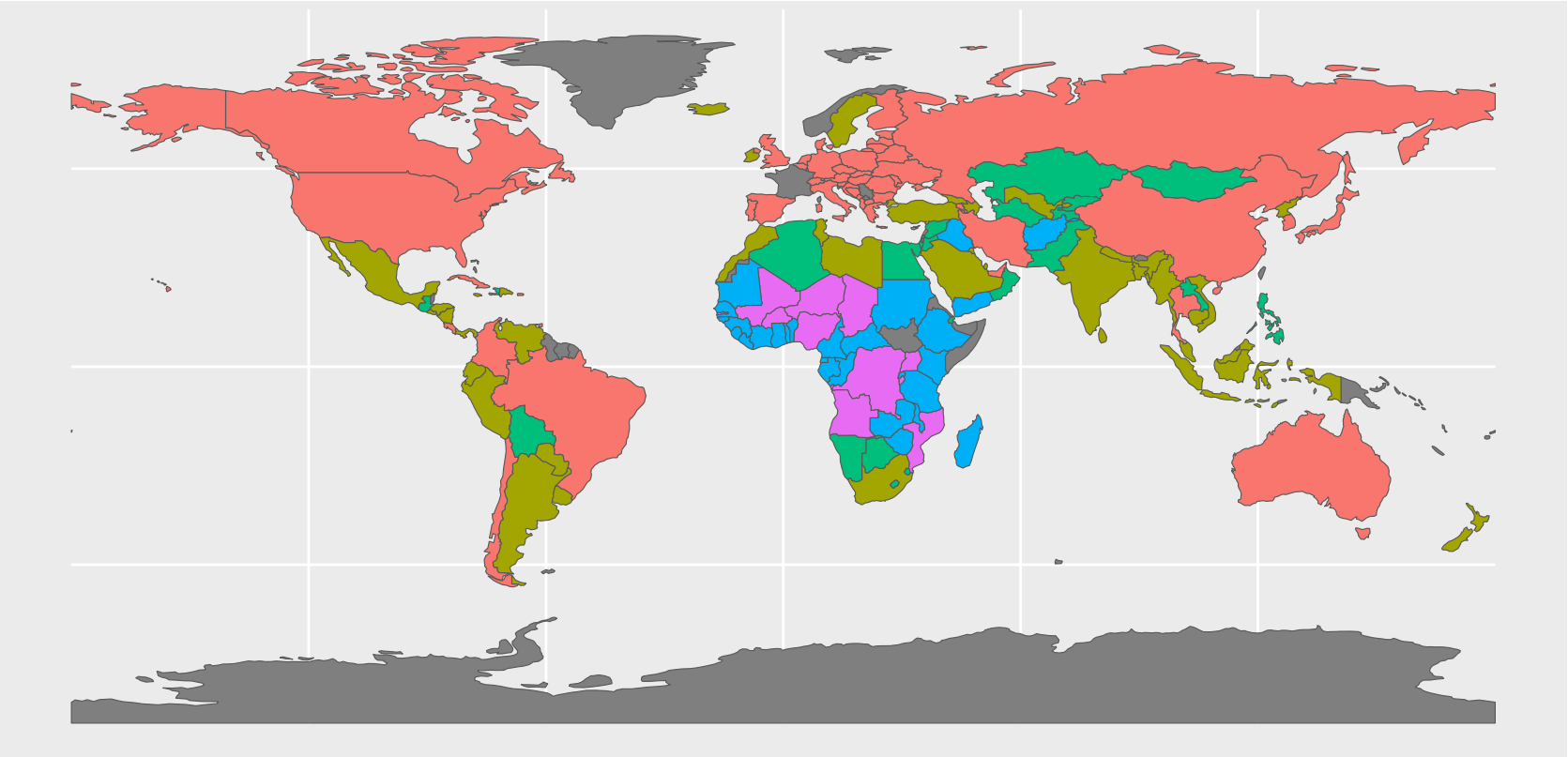
Choropleth maps: Natural Breaks (Optimization)

```
ggplot(data = sf_world, aes(x = Fertility_rate, fill = Fertility_cat)) +
  geom_histogram(bins = 50)
```



Choropleth maps: Natural Breaks (Optimization)

```
ggplot() + geom_sf(data = sf_world, aes(fill = Fertility_cat))
```



Choropleth maps: Standard deviation

- The Standard Deviation Classification method finds the mean value of the observations then places class breaks above and below the mean at intervals of either .25, .5, or 1 standard deviation until all the data values are contained within the classes.
- This classification method shows how much the feature's attribute value varies from the mean. Using a diverging color scheme to illustrate these values is useful to emphasize which observations are above the mean and which observations are below the mean.

Choropleth maps

So how many classes ?

$$7 \pm 2$$

Five is better

Choropleth maps: Elections in USA

The dataset usa_elections.rda contains info on US presidential elections from 1976 by state.

```
load(file = 'Data/usa_elections.rda')


usa_elections
```

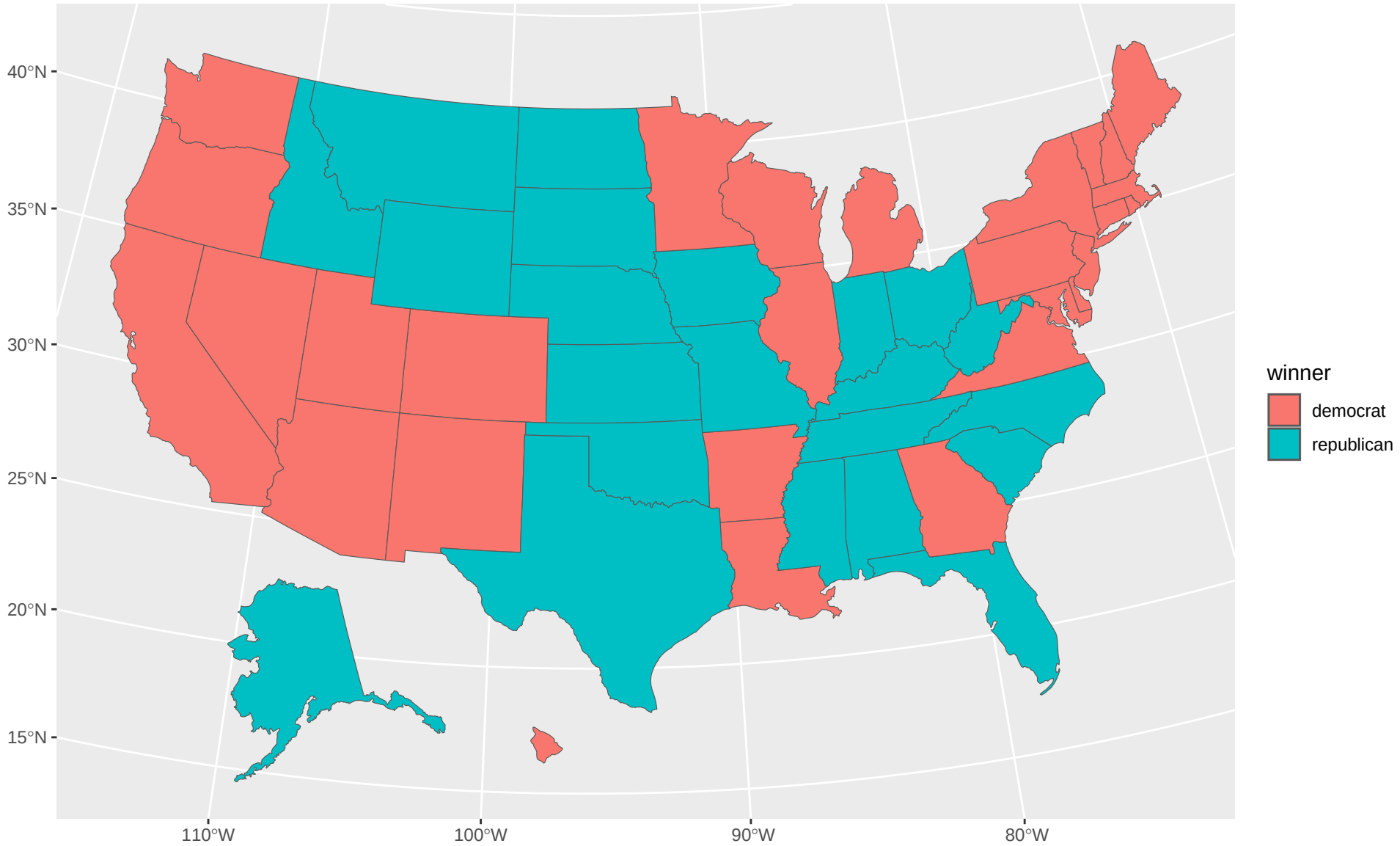
```
## Simple feature collection with 799 features and 6 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: -2031905 ymin: -2462297 xmax: 2516293 ymax: 731649.6
## Projected CRS: +proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=37.5 +lon_0=-96 +x_0=0 +y_0=0 +ellps=GRS80 +datum=NAD83 +units=m +no_defs
## First 10 features:
##    state_po year     winner     margin fips    full
## 1        AL 1976   democrat  0.1311240   01 ALABAMA
## 2        AK 1976 republican -0.2225144   02  ALASKA
## 3        AK 1976 republican -0.2225144   02  ALASKA
## 4        AK 1976 republican -0.2225144   02  ALASKA
## 5        AK 1976 republican -0.2225144   02  ALASKA
## 6        AK 1976 republican -0.2225144   02  ALASKA
## 7        AK 1976 republican -0.2225144   02  ALASKA
## 8        AK 1976 republican -0.2225144   02  ALASKA
## 9        AK 1976 republican -0.2225144   02  ALASKA
## 10       AK 1976 republican -0.2225144   02  ALASKA
##                           geometry
## 1  POLYGON ((1091779 -1380695,...
## 2  POLYGON ((-1533714 -1824318...
## 3  POLYGON ((-1251799 -2255311...
## 4  POLYGON ((-718556 -2186838,...
## 5  POLYGON ((-819355.2 -211594...
## 6  POLYGON ((-747402.1 -214869...
## 7  POLYGON ((-788796.7 -208382...
## 8  POLYGON ((-1621769 -1916804
```

Choropleth maps: Elections in USA

Plot the results of the latest elections

```
usa_elections %>% filter(year == 2020) %>%
  ggplot(aes(fill = winner)) + geom_sf()
```

Choropleth maps: Elections in USA

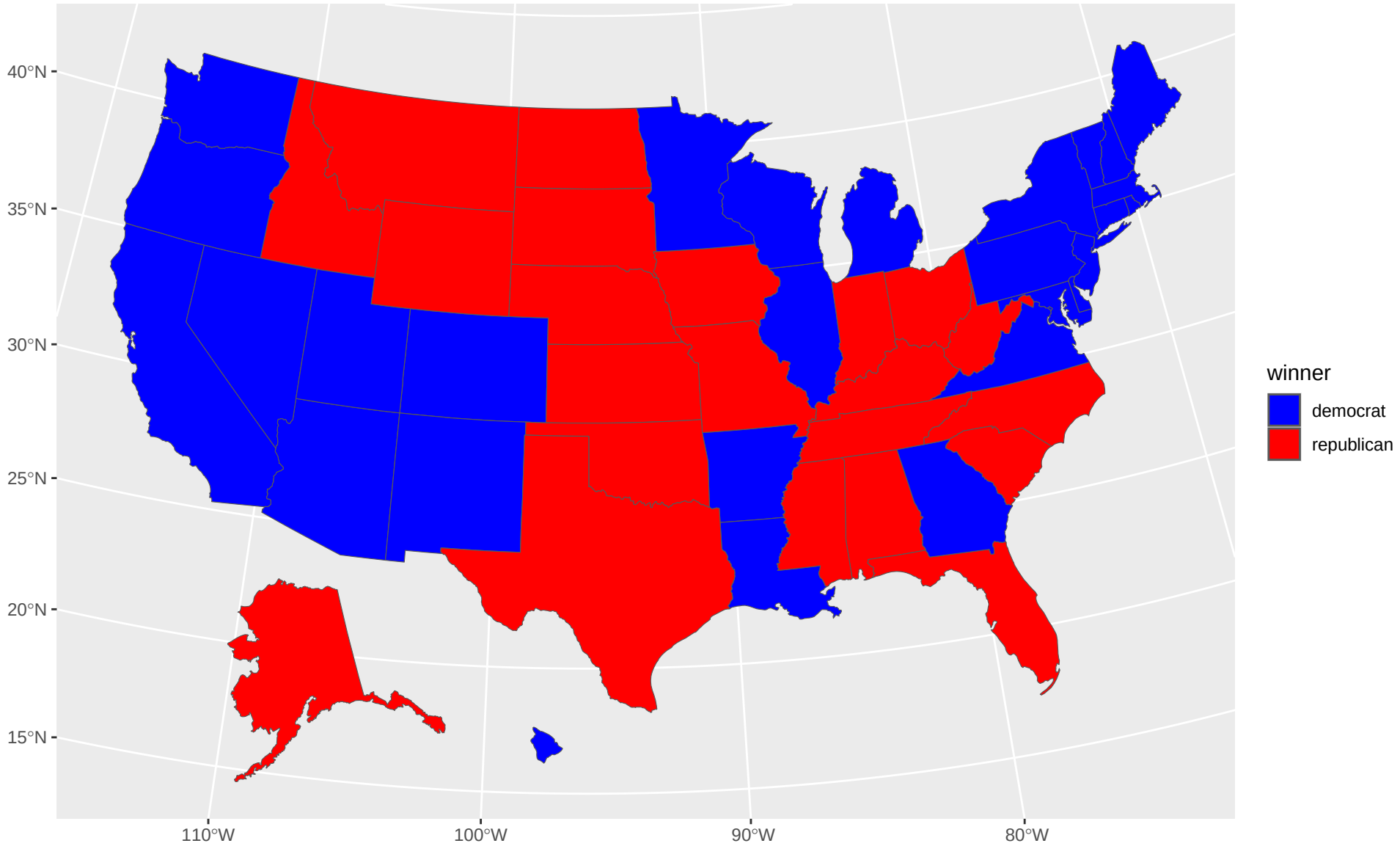


Choropleth maps: Elections in USA

Set up appropriate colors for democrats and republicans

```
usa_elections %>% filter(year == 2020) %>%
  ggplot(aes(fill = winner)) + geom_sf() +
  scale_fill_manual(values = c('blue', 'red'))
```

Choropleth maps: Elections in USA

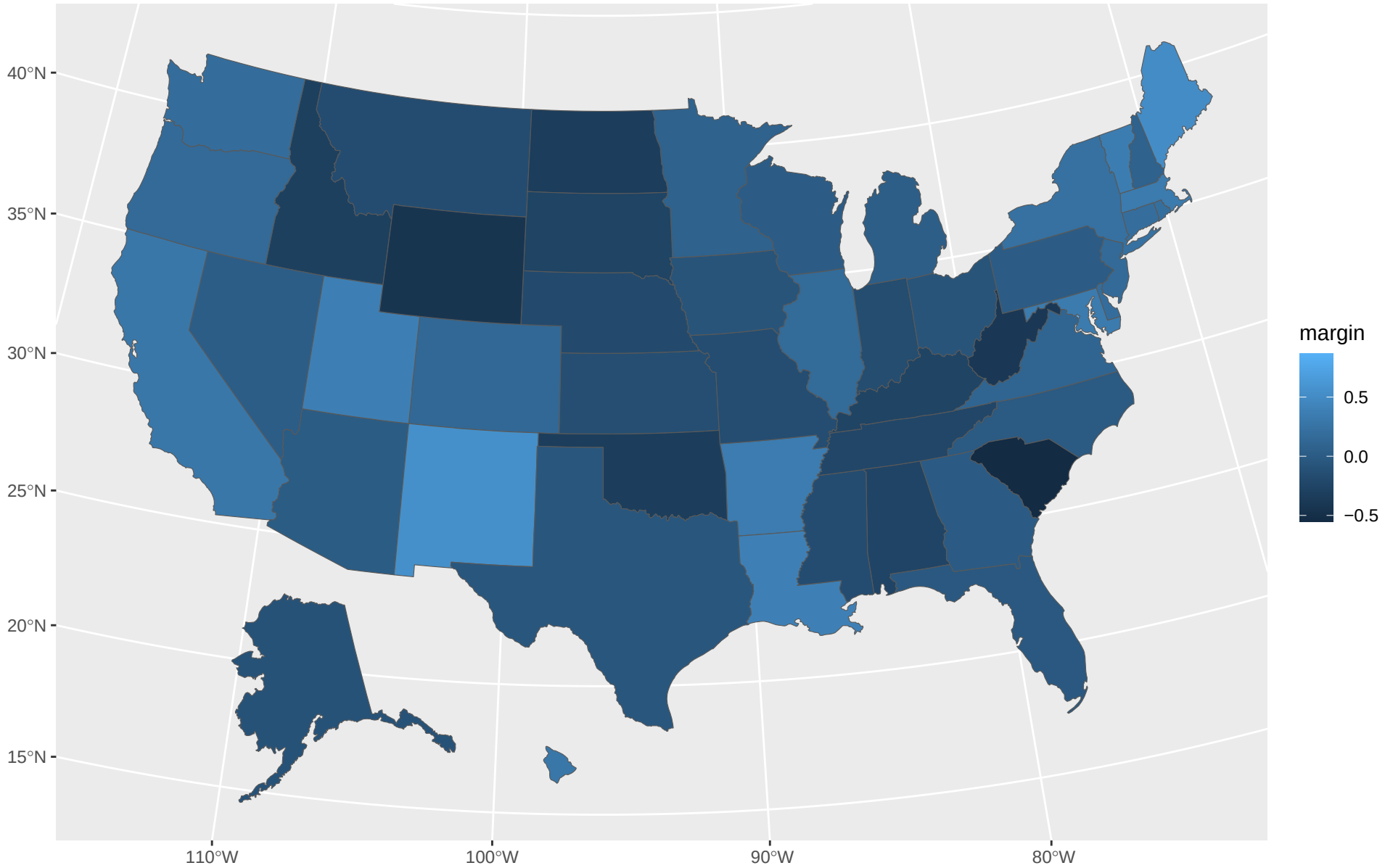


Choropleth maps: Elections in USA

Using margin for fill aesthetics

```
usa_elections %>% filter(year == 2020) %>%
  ggplot(aes(fill = margin)) + geom_sf()
```

Choropleth maps: Elections in USA



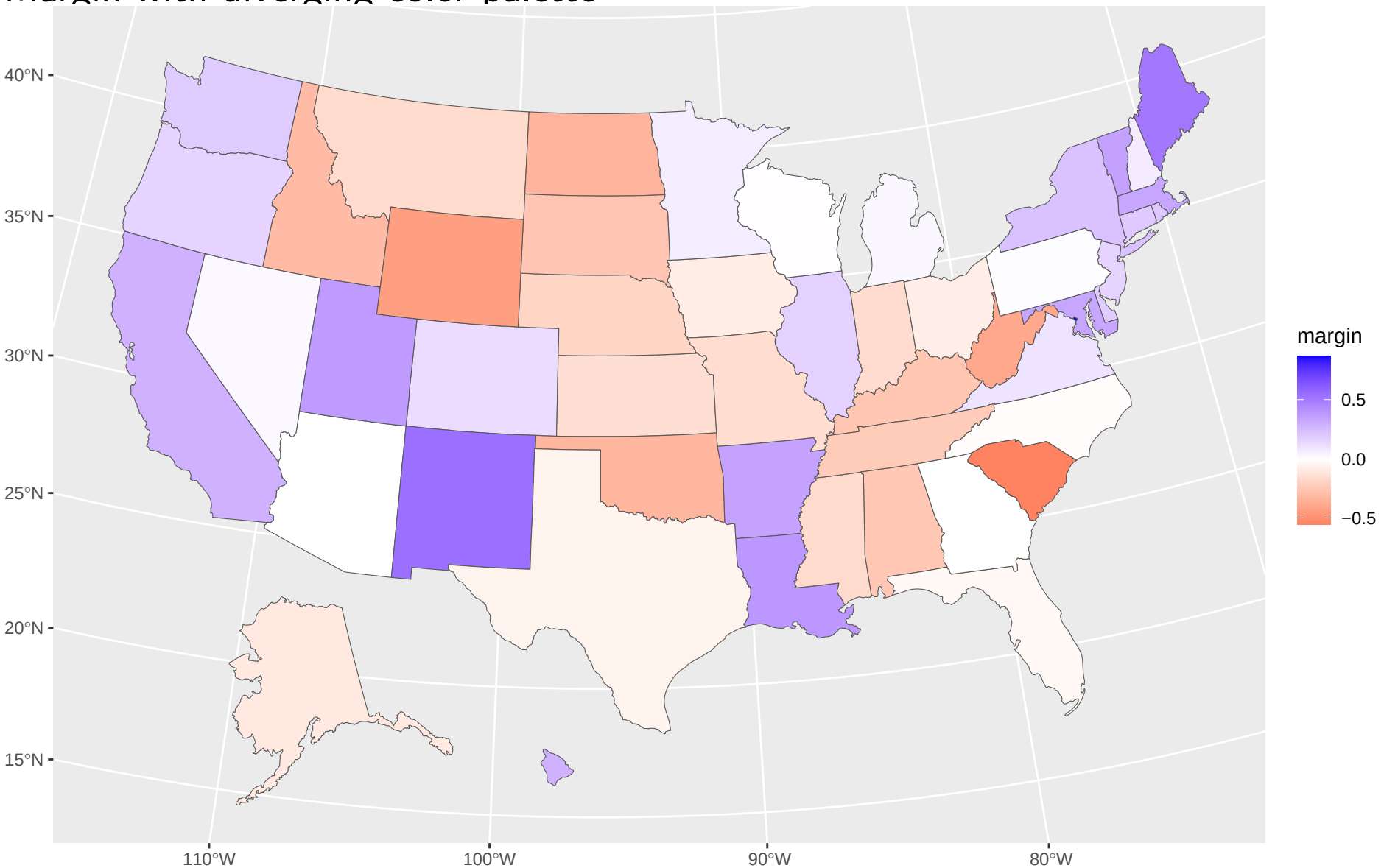
Choropleth maps: Elections in USA

Margin with diverging color palette

```
usa_elections %>% filter(year == 2020) %>%
  ggplot(aes(fill = margin)) + geom_sf() +
  scale_fill_gradient2(low = 'red', high = 'blue')
```

Choropleth maps: Elections in USA

Margin with diverging color palette

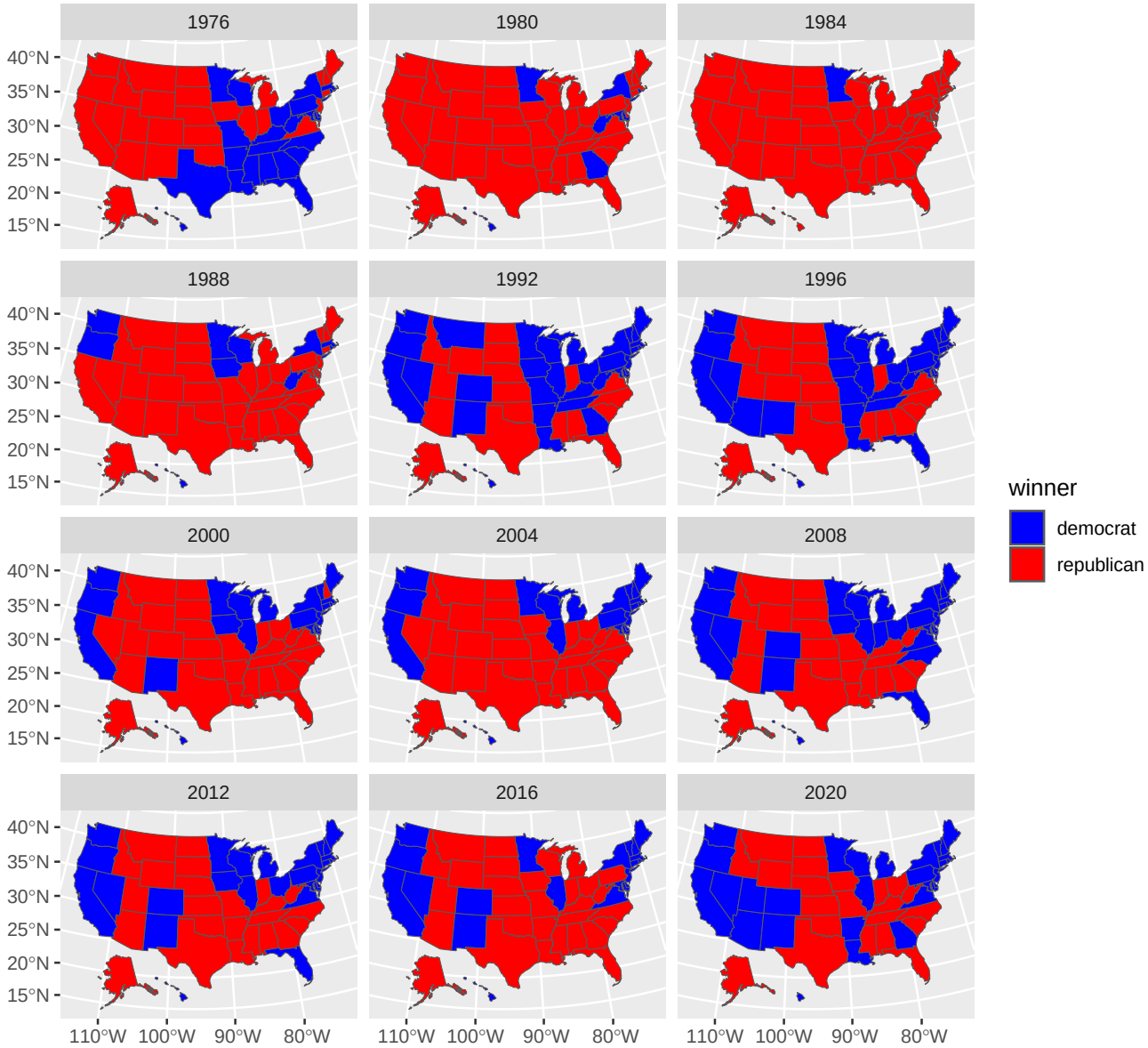


Choropleth maps: Elections in USA

Looking at the election results faceted by time

```
ggplot(usa_elections, aes(fill = winner)) + geom_sf() +
  facet_wrap(~year,nrow=4) +
  scale_fill_manual(values = c('blue', 'red'))
```

Choropleth maps: Elections in USA



Choropleth maps: Elections in USA

Fix few things:

- We don't need coordinates on axis
- Take out the gridlines

```
ggplot(usa_elections, aes(fill = winner)) + geom_sf() +
  facet_wrap(~year,nrow=4) +
  scale_fill_manual(values = c('blue', 'red')) +
  theme(axis.text = element_blank(),
        axis.ticks = element_blank(),
        panel.grid = element_blank(),
        panel.spacing.y = unit(1.5,'lines'))
```

Choropleth maps: Elections in USA