Problem Set 2

Shisham Adhikari

Math 345

Due: Before class on Thursday, 02/11/2021

Collaborators: Insert names of anyone you talked about this assignment with

Goals of this lab

1. Start working with spatial objects.
2. Calculate distance matrices and an adjacency object.
3. Practice your Data Viz.

Problem 1

Let's practice creating spatial data objects using the "sp" package.

a) Load any packages you need here:

```
pacman::p_load(sp, tidyverse, ggplot2, datasets, spdep)
```

b) Create a "SpatialPoints" object from the "state.center" dataset that is in the datasets package. It should be preloaded. Make sure to set the CRS (use datum = WGS84). This may be helpful: https://rspatial.org/raster/spatial/3-vectordata.html#spatialpoints

```
pts <- SpatialPoints(state.center)
pts
```

```
## class       : SpatialPoints
## features    : 50
## extent      : -127.25, -68.9801, 27.8744, 49.25  (xmin, xmax, ymin, ymax)
## crs         : NA
```

```
#Now, we can set the CRS
crdref <- CRS('+proj=longlat +datum=WGS84')
pts <- SpatialPoints(state.center, proj4string=crdref)
pts
```

```
## class       : SpatialPoints
## features    : 50
## extent      : -127.25, -68.9801, 27.8744, 49.25  (xmin, xmax, ymin, ymax)
## crs         : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

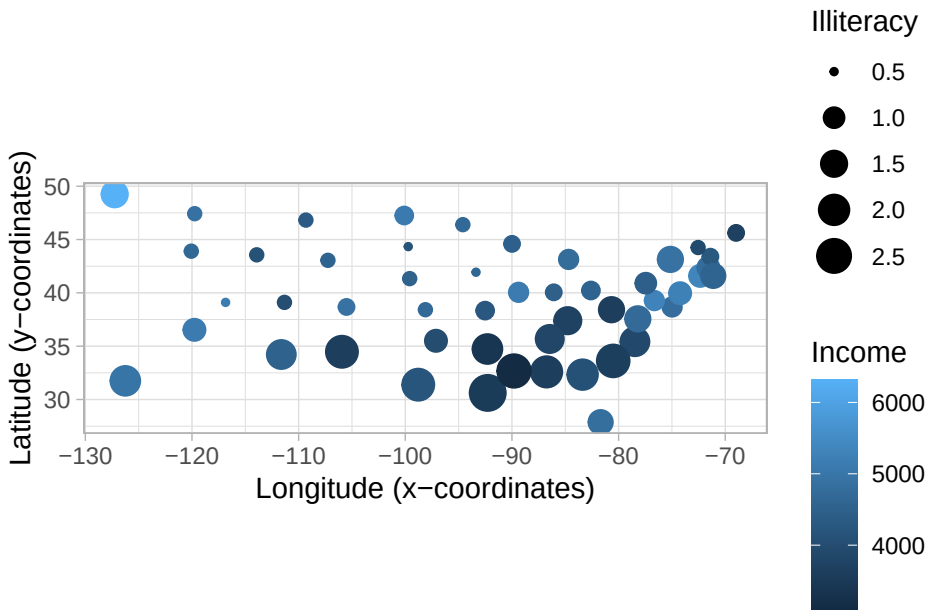
c) Create a "SpatialPointsDataFrame" object that adds to your "SpatialPoints" object using the data in the matrix "state.x77".

```
df <- data.frame(ID=1:50, state.x77)
ptsdf <- SpatialPointsDataFrame(pts, data=df)
ptsdf
```

```
## class       : SpatialPointsDataFrame
## features    : 50
## extent      : -127.25, -68.9801, 27.8744, 49.25  (xmin, xmax, ymin, ymax)
## crs         : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## variables   : 9
## names       : ID, Population, Income, Illiteracy, Life.Exp, Murder, HS.Grad, Frost,   Area
## min values  :  1,        365,   3098,        0.5,    67.96,    1.4,    37.8,     0,   1049
## max values  : 50,      21198,   6315,        2.8,     73.6,   15.1,    67.3,   188, 566432
```

d) Graph the points, making the size vary using one attribute and the color vary using another attribute from state.x77.

```
# Convert spatial data to a dataframe which ggplot can read
plot_df = as(ptsdf, "data.frame")
ggplot(plot_df, aes(x, y,
                    size = Illiteracy,
                    color = Income)) +
  geom_point() +
  labs(x = "Longitude (x-coordinates)",
       y = "Latitude (y-coordinates)") +
  coord_equal() +
  theme_light()
```



Problem 2

Let's see what the difference in distances are when we account for curvature and when we do not.

a) Calculate the distance matrix accounts for the curvature of the earth. One function that can do this is "spDists()".

```
M_curvature <- spDists(pts)
```

b) Calculate a distance matrix that does not account for the curvature of the earth. It is okay if the units of distance are not the same as in part (a) - this version of the distance matrix could consider the points to be in (x,y) space.

```
M_euclidean <- spDists(pts, longlat = FALSE)
```

c) Normalize both matrices from parts (a) and (b) so that the rows add up to 1. Find the percent error for each entry by taking $(M_{euclidean} - M_{curvature})/M_{curvature}$.

```
#Normalizing M_curvature
M_curvature[!is.finite(M_curvature)] <- NA
rtot_c <- rowSums(M_curvature,
                  na.rm = TRUE)
M_norm_c <- M_curvature / rtot_c
rowSums(M_norm_c,
        na.rm = TRUE)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1
```

```
#Normalizing M_euclidean
M_euclidean[!is.finite(M_euclidean)] <- NA
rtot_e <- rowSums(M_euclidean,
                  na.rm = TRUE)
M_norm_e <- M_euclidean / rtot_e
rowSums(M_norm_e,
        na.rm = TRUE)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1
```

```
#Required percent_error matrix
percent_error <- sweep(M_norm_e, 1, M_norm_c)/M_norm_c
```

d) Discuss the magnitude of the errors.

```
#First let's replace all NAs with 0s so that we can run max and min functions without error
percent_error[!is.finite(percent_error)] =  0
max(percent_error)
```

```
## [1] 0.1947149
```

```
min(percent_error)
```

```
## [1] -0.2618669
```

We see that there are not consistent positive or negative errors, this is possibly because degrees of longitude can vary greatly from the actual distance. However,we do notice that the magnitude of both the minimum and maximum errors are close to 0.2. We can even look at how the errors vary based across rows and columns of the error matrix.

```
#Across rows
max(rowSums(percent_error[, 1:50]))
```

```
## [1] -0.3354578
```

```
min(rowSums(percent_error[, 1:50]))
```

```
## [1] -1.725792
```

```
#Across columns
max(colSums(percent_error[1:50,]))
```

```
## [1] 3.754025
```

```
min(colSums(percent_error[1:50,]))
```

```
## [1] -6.25895
```

We see that the magnitude of the errors are larger across columns than across rows. Also, all the errors are negative across rows whereas they are both positive and negative across columns.

Problem 3

Time for nearest neighbors and one more data viz. a) Create the 3-nearest neighbor matrix.

```
Ak3.sp <- knearneigh(pts, k = 3)
class(Ak3.sp)
```

```
## [1] "knn"
```

```
#We can directly grab the nearest neighbor matrix
Ak3.sp$nn
```

```
##       [,1] [,2] [,3]
##  [1,]   24   10   42
##  [2,]   47   37   12
##  [3,]   31   44   28
##  [4,]   24   25   36
##  [5,]   28    3   44
##  [6,]   31   44   50
##  [7,]   39   21   29
##  [8,]   30   20   46
##  [9,]   10   40    1
## [10,]   40    1   42
## [11,]    5   28    3
## [12,]   37   26   44
## [13,]   14   25   15
## [14,]   13   35   17
## [15,]   13   49   25
## [16,]   36   27   25
## [17,]   42   14   35
## [18,]   24    4    1
## [19,]   29   45   21
## [20,]    8   38   30
## [21,]   39    7   29
## [22,]   14   35   49
## [23,]   49   34   41
## [24,]    1    4   18
## [25,]   13    4   15
## [26,]   50   12   34
## [27,]   41   16   15
## [28,]    5   44   12
## [29,]   21   45   39
## [30,]    8   20    7
## [31,]    6    3   44
## [32,]   45    7   21
```

```
## [33,]   46   40   48
## [34,]   41   23   27
## [35,]   48   14   17
## [36,]   16    4   43
## [37,]   47   12   28
## [38,]   20   30   32
## [39,]   21    7   29
## [40,]   33   10   46
## [41,]   34   27   23
## [42,]   17    1   24
## [43,]   36   18    4
## [44,]   28    6   12
## [45,]   29   21   32
## [46,]   20   48   33
## [47,]   37    2   12
## [48,]   46   35   20
## [49,]   15   23   22
## [50,]   26    6   12
```

```
#Or, we can use following functions
Ak3.nb <- knn2nb(Ak3.sp)
Ak3.m <- nb2mat(Ak3.nb)
```

b) Graph the nearest neighbors as lines between your state centroids.

```
par(mar = c(0, 0, 1, 0))
plot(Ak3.nb, pts, col = "grey")
title(main = "The nearest neighbors as lines between state centroids",
      cex.main = 0.9)
```

The nearest neighbors as lines between state centroids