Final Exam

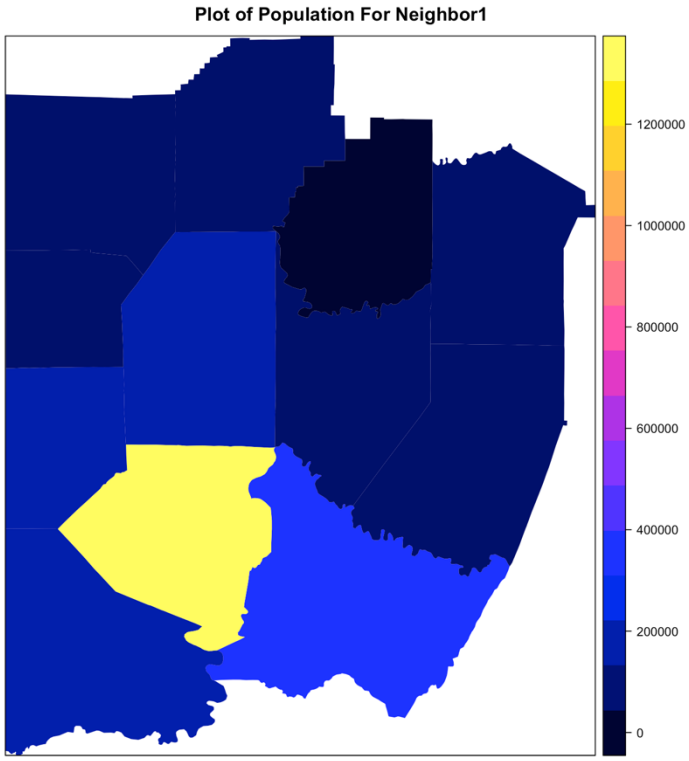
Jiawen Qi (jiq10@pitt.edu)

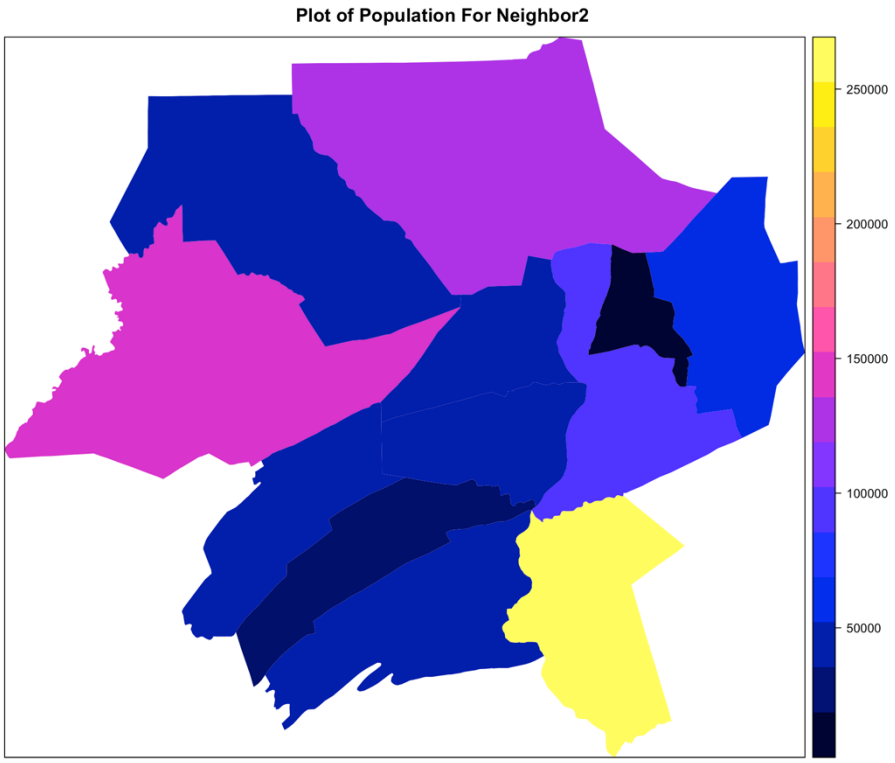
April 22, 2017

Part A

Map

The maps show the polygons and the values of the attribute used for autocorrelation.





Table

This table summaries the results:

Shapefile	Method	Adjacency	I/C value	Expectation	Variance	Standard Deviate	p-value
Neighbor 1	Moran	Rook	0.10081701	-0.09090909	0.03112138	1.0868	0.1386
Neighbor 1	Moran	Queen	0.05015230	-0.09090909	0.02444008	0.90231	0.1834
Neighbor 1	Geary	Rook	0.82080589	1	0.02939741	1.0451	0.148
Neighbor 1	Geary	Queen	0.88672003	1	0.02717464	0.68718	0.246
Neighbor 2	Moran	Rook	-0.11613638	-0.09090909	0.03086932	-0.14358	0.5571

Neighbor 2	Moran	Queen	-0.20617426	-0.09090909	0.02535652	-0.72386	0.7654
Neighbor 2	Geary	Rook	0.91050078	1	0.03290439	0.49339	0.3109
Neighbor 2	Geary	Queen	0.99844253	1	0.02961205	0.0090507	0.4964

Generally speaking, all the p-values are > 0.05, which indicate weak evidence against the null hypothesis. If comment without looking at the p-value, we can generate conclusion according the I and C value.

For Neighbor1 shapefile, the Moran's I >0, but <0.3 which indicates a weak positive autocorrelation. The Geary's C >0 and < 1, which means positive autocorrelation.

For Neighbor2 shapefile, the Moran's I < 0, but > -0.3, which indicates a weak negative autocorrelation. The Geary's C >0 and < 1, gives us the positive autocorrelation. This result is definitely confusing, and when consider the high p-value, this conclusion can not be trusted.

Part B

Table

This table summarizes the IDW results for various values of r and k and the interpolated values of OK at the center locations of each polygon.

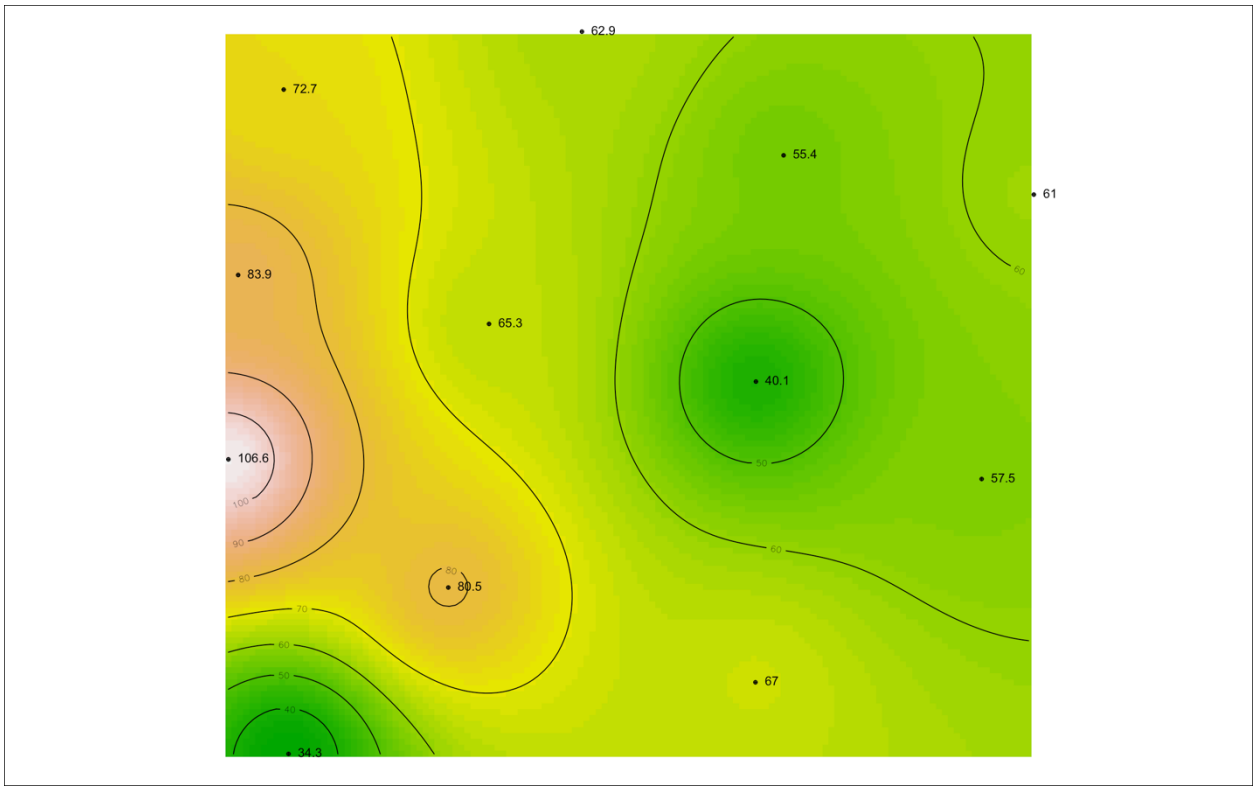
IDW, r = 0.7, k = 1:											
19.0	18.9	19.0	150.0	79.3	33.0	31.0	165.3	114.9	65.0	80.5	34.0
IDW, r = 1.2, k = 1:											
77.1	51.9	33.0	76.9	65.3	41.0	48.0	83.9	106.6	67.0	80.5	34.3
IDW, r = 1.7, k = 1:											
62.9	55.4	61.0	72.7	65.3	40.1	57.5	83.9	106.6	67.0	80.5	34.5
IDW, r = 0.7, k = 2:											

18.9	18.9	18.9	150.0	69.0	21.7	31.5	158.2	143.8	73.3	104.0	10.6
IDW, r = 1.2, k = 2:											
64.7	37.8	29.3	85.9	62.2	23.2	43.2	110.	141.9	73.4	104.0	10.7
IDW, r = 1.7, k = 2:											
57.1	41.2	49.7	81.9	62.2	23.1	49.3	110.1	141.9	73.4	104.0	0.7
IDW, r = 0.7, k = 3:											
18.9	18.8	18.9	150.0	60.0	19.4	31.7	154.0	149.7	74.8	127.6	6.2
IDW, r = 1.2, k = 3:											
52.8	27.9	26.2	96.4	57.1	19.6	39.4	132.8	149.5	74.8	127.6	6.2
IDW, r = 1.7, k = 3:											
49.4	29.7	39.3	93.0	57.1	19.5	42.7	132.8	149.5	74.8	127.6	6.2
OK (Exp)											
77.6	67.8	68.2	81.0	89.3	68.0	68.2	81.8	82.0	68.0	90.2	81.0

IDW Map

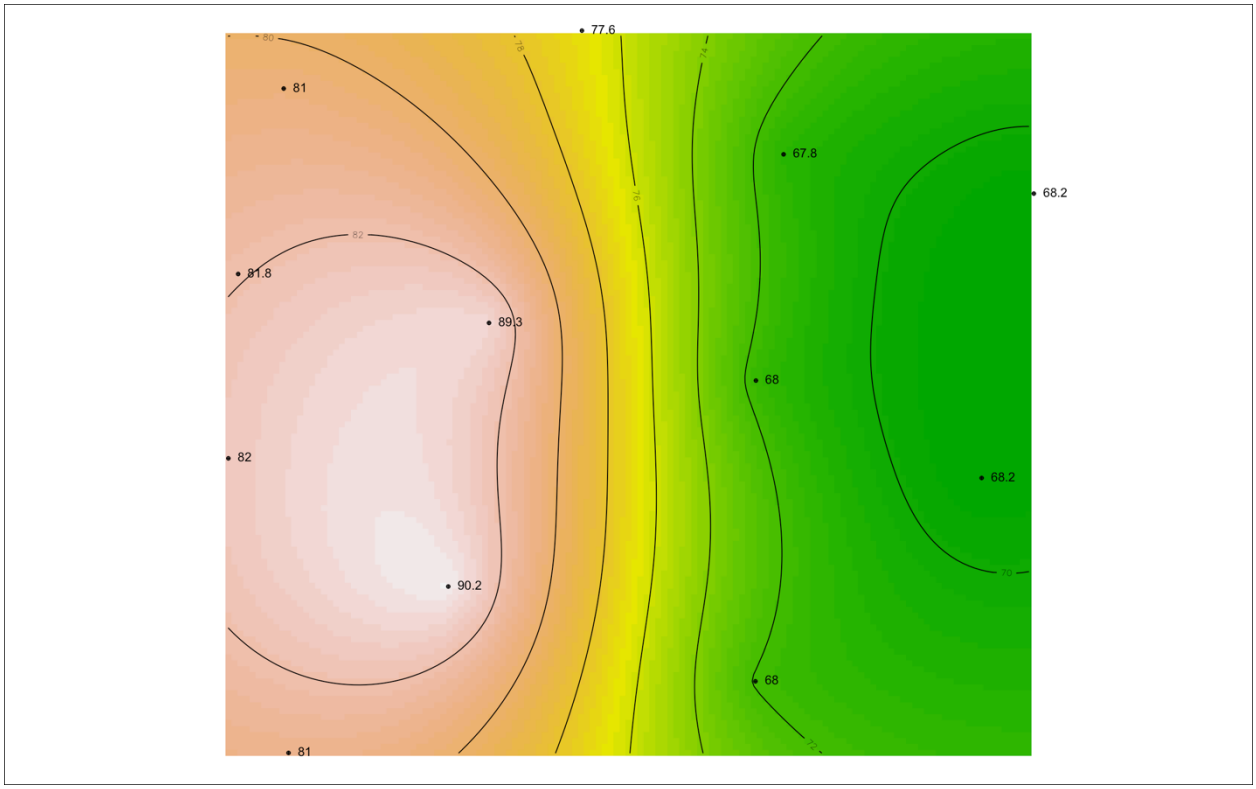
I use the consine similarity method, and the IDW r = 1.7, k = 1 is the most similar one.

This map show the IDW interpolated surface which is most similar to OK surface



OK Map

This map shows the OK interpolated surface:



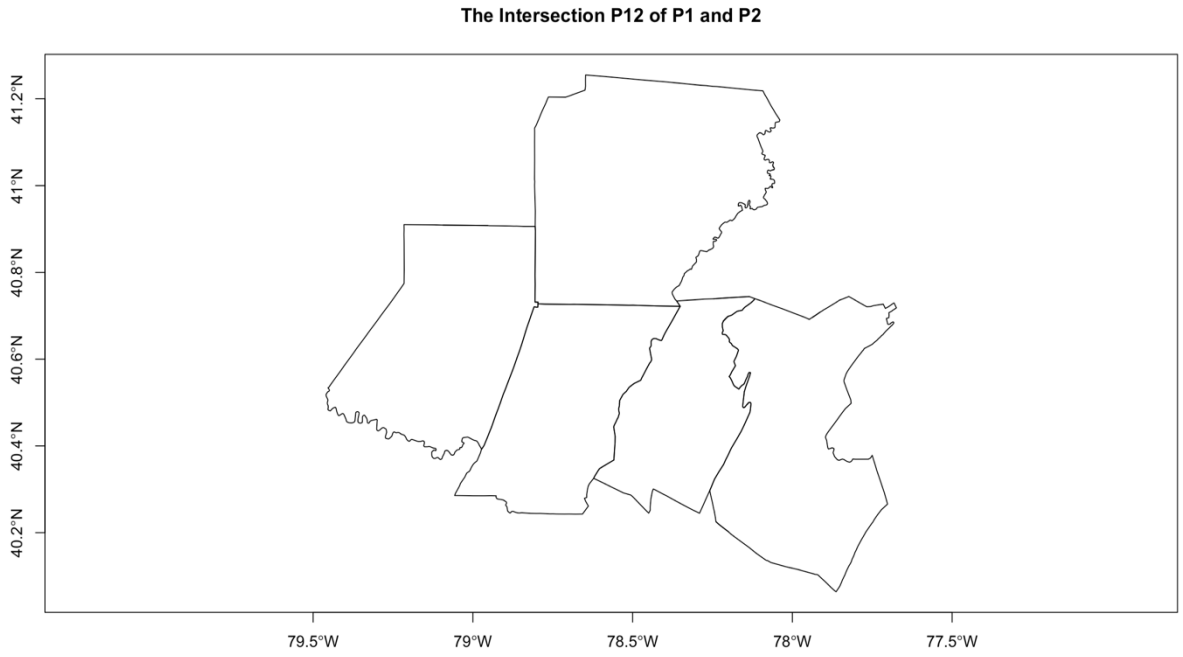
Discussion

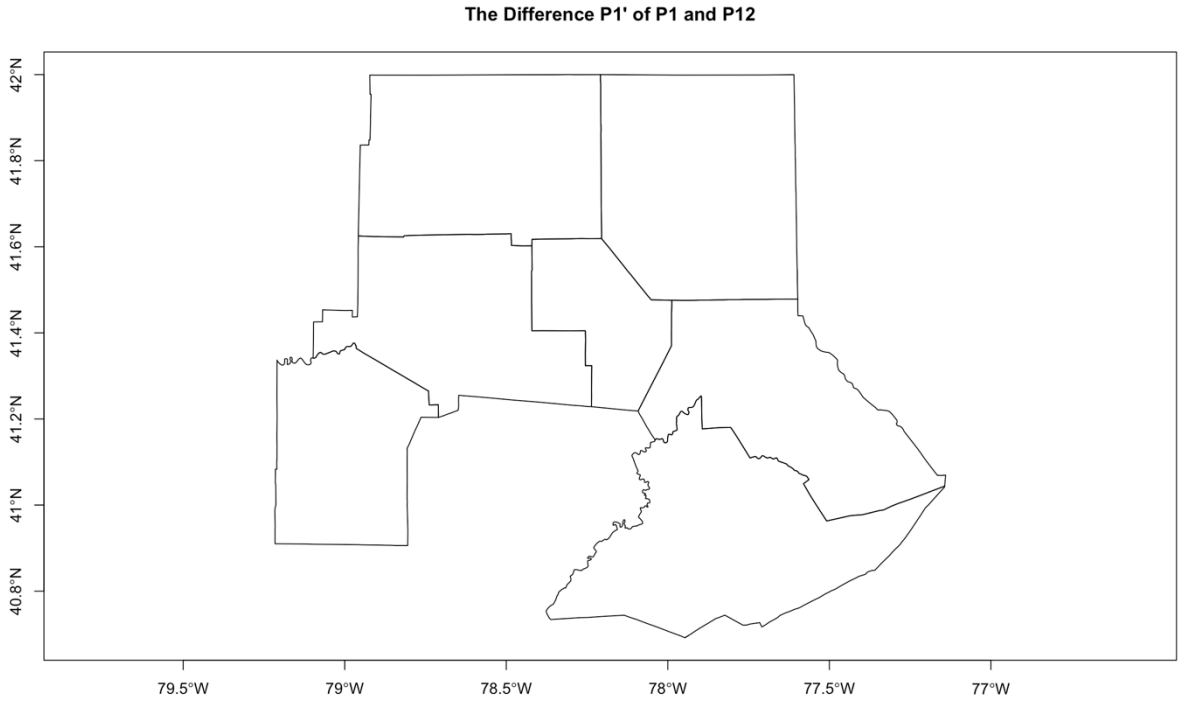
Although IDW r = 1.7, k = 1 is the most similar one with OK surface, there are still some obvious difference on the maps. For the OK Map, there is a obvious “split” at the middle, left part has high z value, right part holds lower z value. In the IDW map, left side is larger than right side, but there is no obvious gap in the middle.

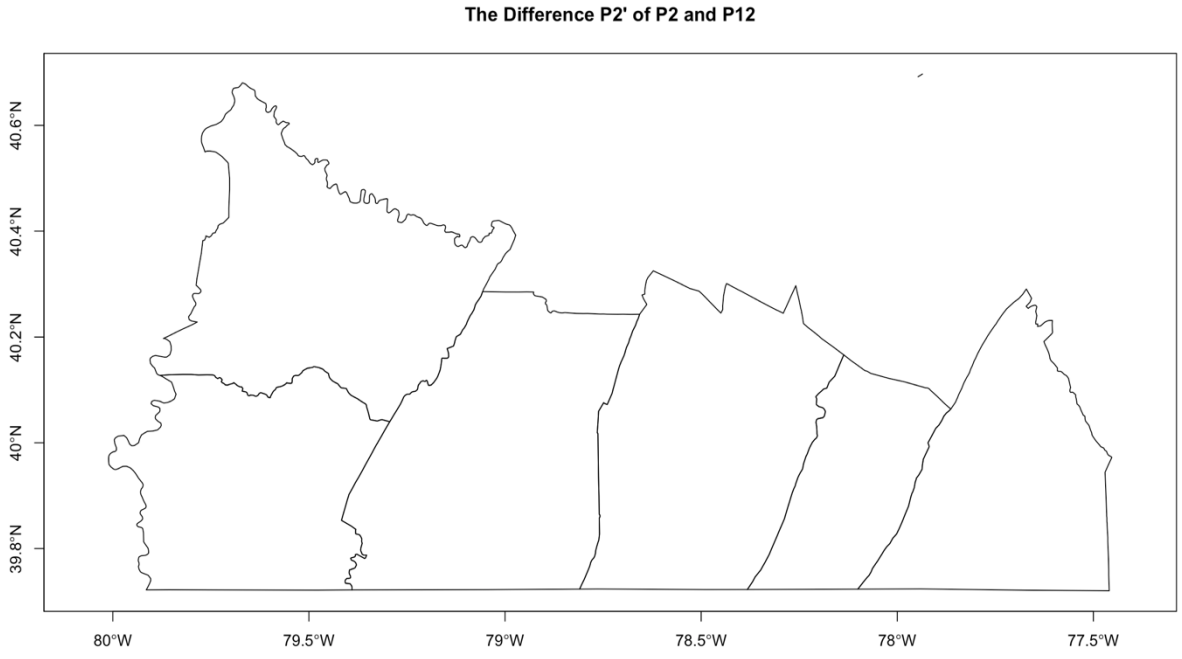
I thinks the most suitable for this dataset is OK, because from the results of different combination of r and k, we can see that there is huge difference from the result. For example , the first point has lower value when r is small. The last point has lower value when k is large. It’s difficult to determine which r and k to use. However, OK uses all the control points and use explainable mathematical function to show the semivariance and use this function to the kriging. So, generally, I’d prefer OK.

Part C

a Polygon-Polygon Overlay

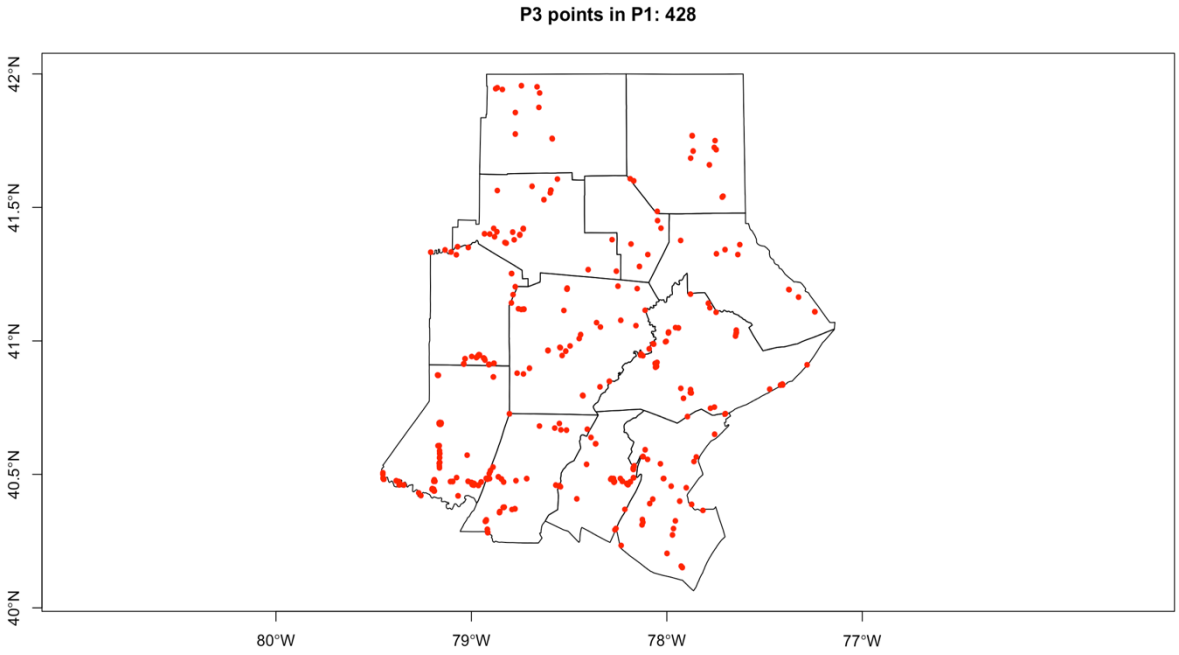




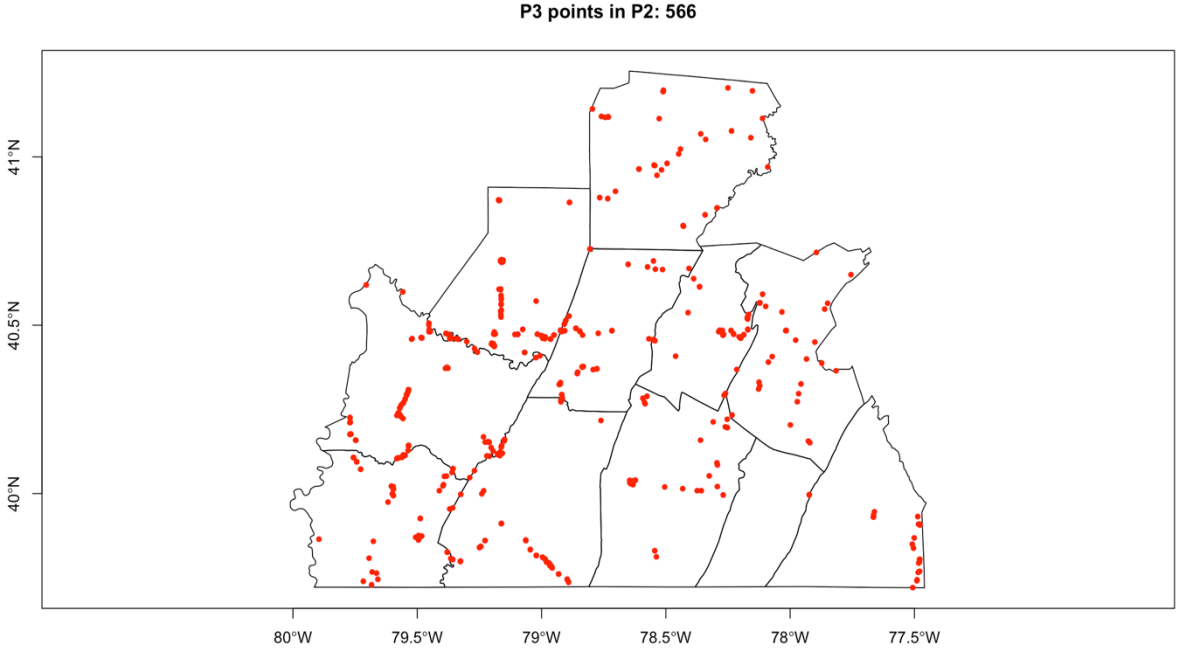


b Polygon-Point Overlay

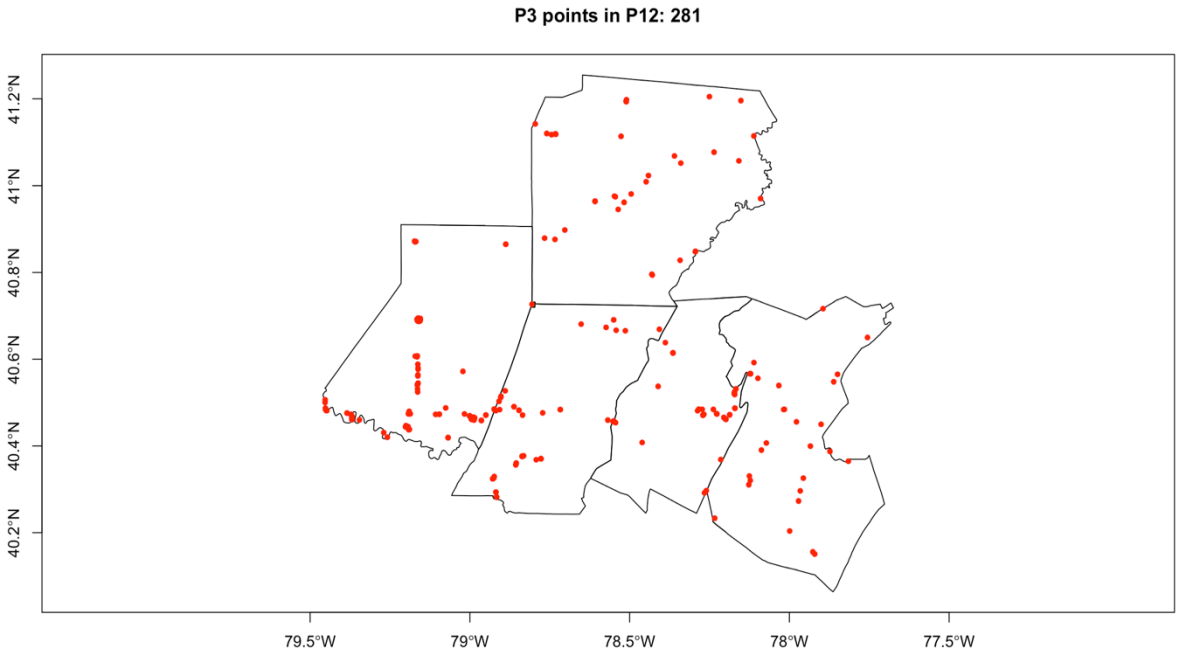
Number of points within P1: 428



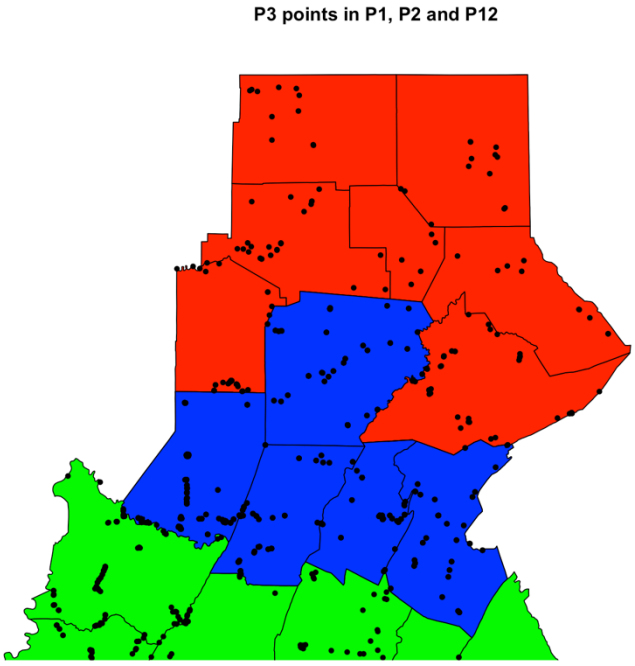
Number of points within P2: 566



Number of points within P12: 281

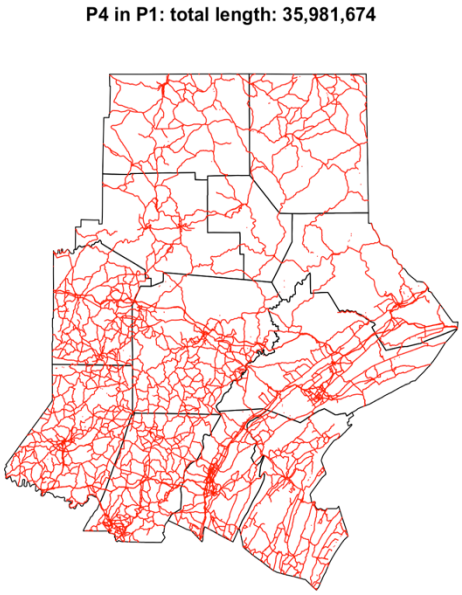


Combination Map:

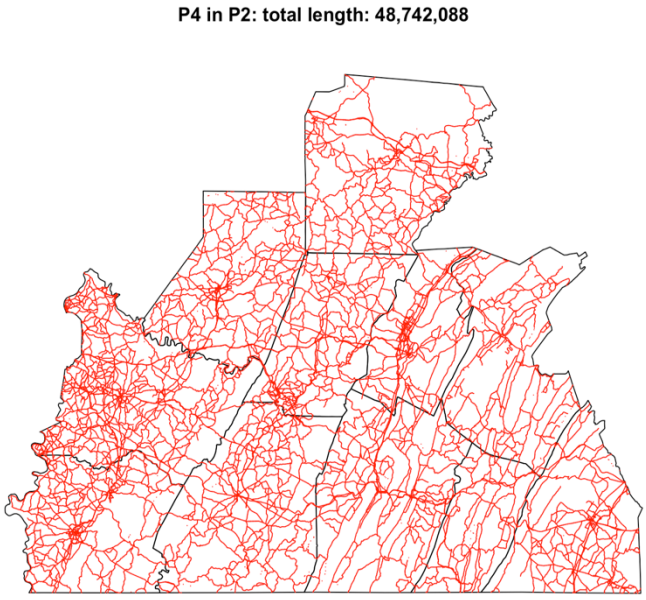


c Polyline-Polygon Overlay

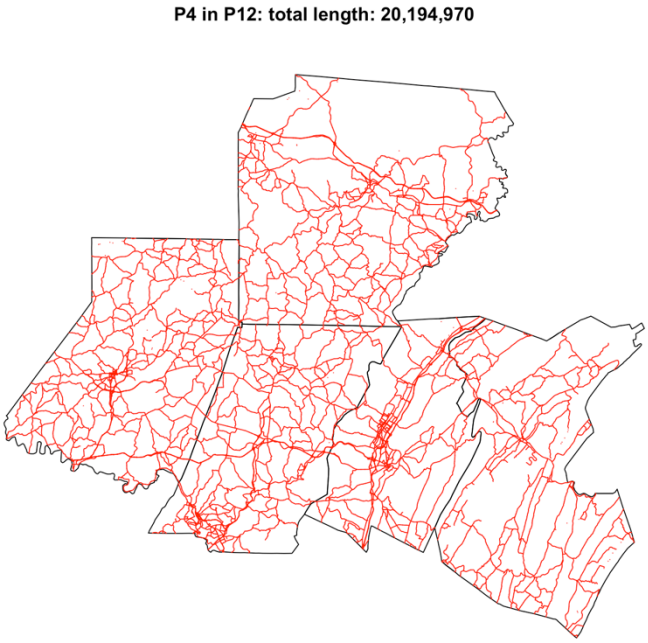
Total length of road segements within P1: 35,981,674



Total length of road segements within P2: 48,742,088



Total length of road segements within P12: 20,194,970



Report

Part C.a is a polygon-polygon overlay analysis. Generally speaking, area1 and area2 share 5 polygons. P1' and P2' were calculated by erase function.

Part C.b is a points-polygons overlay analysis. There are 428 points in P1, 566 points in P2, and 281 points in P12.

Part C.c is a polyline-polygons overlay analysis. The total length of lines in P1 is 35,981,674, in P2 is 48,742,088 and in P12 is 20,194,970.

Code

```
##### Load Libraries #####
library(rgdal)  # Bindings for the Geospatial Data Abstraction Library
library(spdep)  # Spatial Dependence: Weighting Schemes, Statistics and Model
s
library(automap)  # Automatic Interpolation Package
library(lsa)  # Latent Semantic Analysis
library(maps)  # Draw Geographical Maps
library(rgeos)  # Interface to Geometry Engine
library(raster)  # Geographic Data Analysis and Modeling
library(gstat)  # Spatial and Spatio-Temporal Geostatistical Modelling, Predi
ction and Simulation

##### Part A #####
# Import the datasets
Neighbor1 <- readOGR(".", "1_Neighbor1")  # import 1_Neighbor1 shapefile
```

```r
Neighbor2 <- readOGR(".", "1_Neighbor2")  # import 1_Neighbor2 shapefile
# Create Maps for the population attribute (POP_ARR02) for both shapefiles
spplot(Neighbor1, "POP_ARR02", main = "Plot of Population For Neighbor1", col
= "transparent")  # Plot of Population for Neighbor1
spplot(Neighbor2, "POP_ARR02", main = "Plot of Population For Neighbor2", col
= "transparent")  # Plot of Population for Neighbor2

# Construct neighbours list from polygon list
Neighbor1.rook <- poly2nb(Neighbor1, queen = FALSE)  # neighbor1 rook
Neighbor1.queen <- poly2nb(Neighbor1, queen = TRUE)  # neighbor1 queen
Neighbor2.rook <- poly2nb(Neighbor2, queen = FALSE)  # neighbor2 rook
Neighbor2.queen <- poly2nb(Neighbor2, queen = TRUE)  # neighbor2 queen

# change neighbours list to listw (get the spatial weights for neighbours
# lists)
Neighbor1.rook <- nb2listw(Neighbor1.rook)
Neighbor1.queen <- nb2listw(Neighbor1.queen)
Neighbor2.rook <- nb2listw(Neighbor2.rook)
Neighbor2.queen <- nb2listw(Neighbor2.queen)

# Calculate Moran's I & Geary's C
moran.test(Neighbor1$POP_ARR02, Neighbor1.rook, randomisation = FALSE)
moran.test(Neighbor1$POP_ARR02, Neighbor1.queen, randomisation = FALSE)
moran.test(Neighbor2$POP_ARR02, Neighbor2.rook, randomisation = FALSE)
moran.test(Neighbor2$POP_ARR02, Neighbor2.queen, randomisation = FALSE)
geary.test(Neighbor1$POP_ARR02, Neighbor1.rook, randomisation = FALSE)
geary.test(Neighbor1$POP_ARR02, Neighbor1.queen, randomisation = FALSE)
geary.test(Neighbor2$POP_ARR02, Neighbor2.rook, randomisation = FALSE)
geary.test(Neighbor2$POP_ARR02, Neighbor2.queen, randomisation = FALSE)


##### Part B #####

# Import the shapefile
Community <- readOGR(".", "2_Community")  # import the data
ControlPoints <- as.data.frame(Community@data[, c(5:7)])  # get the observed
value at control points
names(ControlPoints) <- c("x", "y", "z")  # rename column
row.names(ControlPoints) <- seq(1:nrow(ControlPoints))  # rename row index
InterpolatePoints <- as.data.frame(coordinates(Community))  # get the x and y
of centroids for each polygon
names(InterpolatePoints) <- c("x", "y")  # rename column
row.names(InterpolatePoints) <- seq(1:nrow(InterpolatePoints))  # rename row
index

# Inverse Distance Weighting change r and k
idw.r.k <- function(r, k) {
    Result.IDW <- NULL
    for (i in 1:length(InterpolatePoints$x)) {
        # iterate the interpolatPoints
```

```
        Distance <- NULL  # initial the distance
        Distance = sqrt((InterpolatePoints$x[i] - ControlPoints$x)^2 + (Inter
polatePoints$y[i] -
            ControlPoints$y)^2)  # calculate the euclidian distance
        ControlPoints.r <- ControlPoints
        ControlPoints.r <- cbind.data.frame(ControlPoints.r, Distance)
        ControlPoints.r <- ControlPoints.r[ControlPoints.r$Distance <= r, ]
#filter the control points by r
        ControlPoints.r$Distance.k <- ControlPoints.r$Distance^k  # give the
exponent k
        ControlPoints.r$InverseDistancek <- 1/ControlPoints.r$Distance.k  # i
nverse
        ControlPoints.r$Weight <- ControlPoints.r$InverseDistancek/sum(Contro
lPoints.r$InverseDistancek)  # normalize the weight
        ControlPoints.r$WeightedValue = ControlPoints.r$Weight * ControlPoint
s.r$z  # w*z
        result <- sum(ControlPoints.r$WeightedValue)  # get the interpolate v
alue
        Result.IDW <- c(Result.IDW, result)  # save the result
    }
    return(Result.IDW)
}

# Diffrent combinations of r and k
Result.IDW.rk1 <- idw.r.k(r = 0.7, k = 1)
Result.IDW.rk2 <- idw.r.k(r = 1.2, k = 1)
Result.IDW.rk3 <- idw.r.k(r = 1.7, k = 1)
Result.IDW.rk4 <- idw.r.k(r = 0.7, k = 2)
Result.IDW.rk5 <- idw.r.k(r = 1.2, k = 2)
Result.IDW.rk6 <- idw.r.k(r = 1.7, k = 2)
Result.IDW.rk7 <- idw.r.k(r = 0.7, k = 3)
Result.IDW.rk8 <- idw.r.k(r = 1.2, k = 3)
Result.IDW.rk9 <- idw.r.k(r = 1.7, k = 3)

# Ordinary Kriging Get the spatial variation by mathematical function
# Reference: https://v8doc.sas.com/sashtml/stat/chap34/sect12.htm y(d) =
# nugget + sill*(1-exp(-d/range))
ControlPoints.sp = ControlPoints
coordinates(ControlPoints.sp) = ~x + y  # create control points spatial objec
t
ControlPoints.variogram <- autofitVariogram(z ~ 1, ControlPoints.sp, model =
"Exp")  # automatically fitting a variogram
plot(ControlPoints.variogram)  # plot the variogram: nugget = 2967, sill = 55
74, range = 0.12
# Create the gamma function
gamma <- function(d) {
    g = 2967 + 5574 * (1 - exp(-d/0.12))
    return(g)
}
# Create the Distance Matrix D for all control points
```

```r
D <- as.data.frame(matrix(data = NA, nrow = nrow(ControlPoints), ncol = nrow(
ControlPoints)))
for (i in 1:nrow(D)) {
    for (j in 1:nrow(D)) {
        D[i, j] = sqrt((ControlPoints$x[i] - ControlPoints$x[j])^2 + (Control
Points$y[i] -
            ControlPoints$y[j])^2)  # Euclidean Distance
    }
}
# Create Matrix A from D
A <- gamma(D)
A <- rbind.data.frame(A, 1)
A <- cbind.data.frame(A, 1)
diag(A) <- 0
names(A) <- c(seq(1:13))
A.inverse <- solve(A)
# A loop to interpolate each point
Result.OK <- NULL
for (i in 1:nrow(InterpolatePoints)) {
    d <- NULL  # initial the vector d
    d <- sqrt((InterpolatePoints$x[i] - ControlPoints$x)^2 + (InterpolatePoin
ts$y -
        ControlPoints$y)^2)
    b <- c(gamma(d), 1)  # calculate matrix b
    w <- A.inverse %*% b  # calculate weight matrix w
    z <- sum(ControlPoints$z * w[1:nrow(w) - 1])  # interpolate
    Result.OK <- c(Result.OK, z)
}
Result.OK

# IDW Interpolated Surface which is most similar to OK surface I use consine
# similarity to determine which result is similar to OK
cosine(Result.OK, Result.IDW.rk1)  # 0.8356637
cosine(Result.OK, Result.IDW.rk2)  # 0.9607648
cosine(Result.OK, Result.IDW.rk3)  # 0.9697523
cosine(Result.OK, Result.IDW.rk4)  # 0.812111
cosine(Result.OK, Result.IDW.rk5)  # 0.8898386
cosine(Result.OK, Result.IDW.rk6)  # 0.9002952
cosine(Result.OK, Result.IDW.rk7)  # 0.8
cosine(Result.OK, Result.IDW.rk8)  # 0.8532501
cosine(Result.OK, Result.IDW.rk9)  # 0.8611911

# The most similar is when r = 1.7, k = 1 Creating the IDW interpolated
# Surface r = 1.7, k = 1
ResultTable <- cbind.data.frame(InterpolatePoints, Result.IDW.rk3)
names(ResultTable) <- c("x", "y", "z")
IDW.map <- as.data.frame(ResultTable)
coordinates(IDW.map) = ~x + y
xRange <- as.numeric(bbox(IDW.map)[1, ])
yRange <- as.numeric(bbox(IDW.map)[2, ])
```

```r
grid <- expand.grid(x = seq(from = xRange[1], to = xRange[2], by = 0.01), y =
seq(from = yRange[1],
    to = yRange[2], by = 0.01))
coordinates(grid) <- ~x + y
gridded(grid) <- TRUE
IDW.data <- gstat::idw(ResultTable$z ~ 1, locations = IDW.map, newdata = grid
)

IDWPlot <- par(mar = c(0, 0, 0, 0))
image(IDW.data, "var1.pred", col = terrain.colors(50))
contour(IDW.data, "var1.pred", add = TRUE, nlevels = 10)
plot(IDW.map, add = TRUE, pch = 10, cex = 0.5)
text(coordinates(Community), as.character(round(ResultTable$z, 1)), pos = 4,
    cex = 0.8, col = "black")
map.axes(cex.axis = 0.8)
par(IDWPlot)

# OK Interpolated Surface
OK.result <- cbind.data.frame(InterpolatePoints, Result.OK)
names(OK.result) <- c("x", "y", "z")
coordinates(OK.result) = ~x + y
variogram <- autofitVariogram(z ~ 1, OK.result, model = "Exp")
variogram
plot(variogram)
nugget = 41
sill = 89
range = 1.1
model <- vgm(psill = sill, model = "Exp", range = range, nugget = nugget)
krige <- krige(OK.result$z ~ 1, OK.result, grid, model = model)
OKPlot <- par(mar = c(0, 0, 0, 0))
image(krige, "var1.pred", col = terrain.colors(50))
contour(krige, "var1.pred", add = TRUE, nlevels = 10)
plot(OK.result, add = TRUE, pch = 10, cex = 0.5)
text(coordinates(OK.result), as.character(round(OK.result$z, 1)), pos = 4, ce
x = 0.8,
    col = "black")
map.axes(cex.axis = 0.8)
par(OKPlot)

##### Part C #####

# Import the shapefiles
P1 <- readOGR(".", "3_Area1")
P2 <- readOGR(".", "3_Area2")
P3 <- readOGR(".", "3_TrailPoints")
P4 <- readOGR(".", "3_State_Roads")

# a. Polygons Overlay
P1@proj4string  # check the coordinate system
P2@proj4string
```

```r
P12 <- intersect(P1, P2)
plot(P12, axes = TRUE, main = "The Intersection P12 of P1 and P2")
P1. <- erase(P1, P12)  # P1' = P1-P12
plot(P1., axes = TRUE, main = "The Difference P1' of P1 and P12")
P2. <- erase(P2, P12)  # P2'= P2 - P12.
plot(P2., axes = TRUE, main = "The Difference P2' of P2 and P12")


# b. Points and Polygons Overlay
P1@proj4string
P3@proj4string  # the system is not the same
P3.new <- spTransform(P3, CRS("+proj=longlat +datum=NAD27 +no_defs +ellps=clr
k66
+ +nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat"))
P3.new@proj4string

# P3 in P1
P3inP1 <- intersect(P3.new, P1)
length(P3inP1)  # 428 points
plot(P1, axes = TRUE, main = "P3 points in P1: 428")
plot(P3inP1, add = TRUE, col = "red", pch = 20)


# P3 in P2
P3inP2 <- intersect(P3.new, P2)
length(P3inP2)  # 566 points
plot(P2, axes = TRUE, main = "P3 points in P2: 566")
plot(P3inP2, add = TRUE, col = "red", pch = 20)


# P3 in P12
P3inP12 <- intersect(P3.new, P12)
length(P3inP12)  # 281 points
plot(P12, axes = TRUE, main = "P3 points in P2: 281")
plot(P3inP12, add = TRUE, col = "red", pch = 20)


# Combine in one Map
plot(P1, col = "red", main = "P3 points in P1, P2 and P3")
plot(P2, col = "green", add = TRUE)
plot(P12, col = "blue", add = TRUE)
plot(P3inP1, add = TRUE, pch = 20)
plot(P3inP2, add = TRUE, pch = 20)
plot(P3inP12, add = TRUE, pch = 20)

# Lines Polygons Overlay
P1@proj4string

P4@proj4string  # the system is not the same
```

```r
P4.new <- spTransform(P4, CRS("+proj=longlat +datum=NAD27 +no_defs +ellps=clr
k66
+ +nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat"))
P4.new@proj4string

# P4 in P1
P4inP1 <- intersect(P4.new, P1)
sum(P4inP1$SEG_LNGTH_)  # 35981674

plot(P1, main = "P4 in P1: total length: 35,981,674")
plot(P4inP1, col = "red", add = TRUE)

# P4 in P2
P4inP2 <- intersect(P4.new, P2)
sum(P4inP2$SEG_LNGTH_)  # 48742088

plot(P2, main = "P4 in P2: total length: 48,742,088")
plot(P4inP2, col = "red", add = TRUE)

# P4 in P12
P4inP12 <- intersect(P4.new, P12)
sum(P4inP12$SEG_LNGTH_)  # 20194970

plot(P12, main = "P4 in P12: total length: 20,194,970")
plot(P4inP12, col = "red", add = TRUE)
```