setwd("C:/Users/user/Desktop/GIS")

library(kohonen)

library(ggplot2)

library(rgdal)

library(gridExtra)

library(grid)

library(readr)

#read in the processed SIMD data for the City of Edinburgh area

data <- read\_csv(file="SIMD2020.csv")

#read in the boundary data for the Edinburgh area from Edinburgh.shp. It has been already matched up by row with SIMD2020 data in QGIS.

edinburgh\_map <- readOGR(dsn="map", layer="Edinburgh")

#check what object types we have already had

class(data)

class(edinburgh\_map)

#what are the column names of the dataframe, and of the spatial points dataframe?

names(data)

names(edinburgh\_map)

#how many rows do each of the datasets contain?

nrow(data)

nrow(edinburgh\_map)

#check the coordinate system of the spatial polygons data frame

proj4string(edinburgh\_map)

#plot the spatial polygons data frame

plot(edinburgh\_map)

#convert the object into latitude and longitude and for easier use with ggmap later

edinburgh\_map <- spTransform(edinburgh\_map, CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no\_defs"))

#convert spatial polygon to dataframe including columns of spatial information

edinburgh\_fort <- fortify(edinburgh\_map, region= "DataZone")

#merge the new dataframe with the edinburgh census data using their shared column (LAName)

edinburgh\_fort <- merge(edinburgh\_fort, data, by="id")

#create a plot of nocentralheat\_rate. Thematic maps in this paper were plotted with QGIS though.

ggplot(data=edinburgh\_fort, aes(x=long, y=lat, fill=nocentralheat\_rate, group=group)) +

scale\_fill\_gradientn(colours=c("#f0f9e8","#bae4bc","#7bccc4","#43a2ca","#0868ac"))+

geom\_polygon(colour="black")+

theme(legend.position="bottom")+

coord\_equal()+

theme()

#SOM TRAINING

#choose the variables from SIMD2020 data, with which to train the SOM by subsetting the dataframe named data

data\_train <- data[, c(6,8,14,35,36)]

#convert to numeric

data\_train<-sapply(data\_train, as.numeric)

#standardise the data creating z-scores and convert to a matrix

data\_train\_matrix <- as.matrix(scale(data\_train))

#keep the column names of data\_train as names in our new matrix

names(data\_train\_matrix) <- names(data\_train)

#define the size and topology of the som grid

som\_grid <- somgrid(xdim = 15, ydim=15, topo="hexagonal")

# Train the SOM model

som\_model <- som(data\_train\_matrix,

grid=som\_grid,

rlen=700,

alpha=c(1,0.01),

keep.data = TRUE )

#SOM VISUALISATION

# Plot of the training progress - how the node distances have stabilised over time.

#mean distance to closes codebook vector during training

plot(som\_model, type = "changes")

#counts within nodes

plot(som\_model, type = "counts", main="Node Counts", palette.name=cm.colors)

#map quality

plot(som\_model, type = "quality", main="Node Quality/Distance", palette.name=cm.colors)

#neighbour distances

plot(som\_model, type="dist.neighbours", main = "SOM neighbour distances", palette.name=grey.colors)

#code spread

plot(som\_model, type = "codes")

# Plot the heatmap for a variable at scaled / normalised values

var <- 1

#define the variable to plot

plot(som\_model, type = "property", property = getCodes(som\_model)[,var],

main=colnames(getCodes(som\_model))[var], palette.name=cm.colors)

#CLUSTERING OF SOM RESULTS.Show the WCSS metric for kmeans for different clustering sizes. It can be used as a "rough" indicator of the ideal number of clusters

mydata <- getCodes(som\_model)

wss <- (nrow(mydata)-1)\*sum(apply(mydata,2,var))

for (i in 2:15) wss[i] <- sum(kmeans(mydata,

centers=i)$withinss)

#Plot the data. Generally minimizing the WCSS will maximise the distance between clusters. You may notice a point of diminishing returns as cluster size increases.

plot(1:15, wss, type="b", xlab="Number of Clusters",

ylab="Within groups sum of squares", main="Within cluster sum of squares (WCSS)")

# Form clusters on grid and use hierarchical clustering to cluster the codebook vectors.

som\_cluster <- cutree(hclust(dist(getCodes(som\_model))), 6)

# define an appealing colour palette

pretty\_palette <- c('#fbb4ae', '#b3cde3', '#ccebc5', '#decbe4', '#fed9a6', '#ffffcc')

#show the same plot with the codes instead of just colours

plot(som\_model, type="codes", bgcol = pretty\_palette[som\_cluster], main = "Clusters")

add.cluster.boundaries(som\_model, som\_cluster)

#MAPPING OF SMALL AREAS

#BACK TO THE GEOGRAPHY

#adding label set to the SOM - grabs a random subset from data

#extracting Edinburgh intermediate zone names

geog\_names <- data$Intermediate\_Zone

#Removing duplicates to gain an idea of the expanse across Edinburgh

geog\_names[duplicated(geog\_names)] <- NA

#searches the index of names which are not NA

naset <- which(!is.na(geog\_names))

#randomly picking 10 of the placenames in the data NA

naset <- sample(naset, length(naset)-10)

geog\_names[naset] <- NA

#replotting the data with added labels=geog\_names

plot(som\_model, type="mapping", bgcol = pretty\_palette[som\_cluster], main = "Clusters", labels=geog\_names)

add.cluster.boundaries(som\_model, som\_cluster)

#create dataframe of the small area id and of the cluster unit

cluster\_details <- data.frame(id=data$id, cluster=som\_cluster[som\_model$unit.classif])

#merge our cluster details onto the fortified spatial polygon dataframe we created earlier

mappoints <- merge(edinburgh\_fort, cluster\_details, by="id")

# Finally map the areas and colour by cluster

ggplot(data=mappoints, aes(x=long, y=lat, group=group, fill=factor(cluster))) +

geom\_polygon(colour="transparent") +

coord\_equal() +

scale\_fill\_manual(values = pretty\_palette)

#combine the clustered data with the original spatial polygons Edinburgh.shp

edinburgh\_map <- merge(edinburgh\_map, cluster\_details, by.x="DataZone", by.y="id")

#write the clustered edinburgh\_map as a new shapefile and plot it in R

writeOGR(obj=edinburgh\_map,

dsn="edinburgh\_map\_clustered",

layer="edinburgh\_map\_clustered",

driver="ESRI Shapefile")