```
####################################################### 1. HEADER
#######################################################
# URBANIZATION GROWTH/SHRINKAGE MODEL (UGSM)
#
# Author: Matthew Kotkowski; e-mail: matthew8644@gmail.com; PHD student at FSU
#
# Date: 7/01/2017
#
# Version: 7.01
#
########################################################################################################################

################################################### 1B. VERSION UPDATES
#################################################
# Added:
#
#   1) Parallel computing
#
#   2) Distance to the closest water area
#
################################################ 3. FUNCTIONS
###########################################################

# Replicating function used to replicate rows in a data frame
df.Rep <- function(.data_Frame, .search_Columns, .search_Value, .sub_Value){
  .data_Frame[, .search_Columns] <- ifelse(.data_Frame[, .search_Columns]==.search_Value,.sub_Value/.search_Value,1) *
.data_Frame[, .search_Columns]
  return(.data_Frame)
} # Source: http://stackoverflow.com/questions/14737773/replacing-occurrences-of-a-number-in-multiple-columns-of-data-
frame-with-another

work_dir <- function(directory){
  path <- "/home/hubcamp/RProjects/UGSM/"
  return(paste(path, directory, sep="/", collapse=NULL))
}
################################################ 2. PACKAGES & LIBRARIES
################################################

# Check if the necessary packages are installed, and if not - install them with all dependencies

# For your convinience I have saved packages I have used in a CSV file. The system is loading it into Your memory to
check
# if You have them. If not - they will be installed with all necessary dependencies.

# Following two lines of code are for an author to save his packages that he used
#packages_loaded <- (.packages())
#write.csv(packages_loaded, file='mk_packages.csv', quote=FALSE,row.names=F)

packages_needed = read.csv(work_dir('mk_packages.csv'),header=TRUE, stringsAsFactors = F)
packages_needed <- unlist(packages_needed)
list.of.packages <- c(packages_needed)
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages, dependencies = TRUE)

# Now we can load the packages
suppressMessages(library(maptools))
suppressMessages(library(FedData))
suppressMessages(library(raster))
suppressMessages(library(rasterVis))
suppressMessages(library(rgeos))
suppressMessages(library(sp))
suppressMessages(library(dismo))
suppressMessages(library(mapplots))
suppressMessages(library(spdep))
suppressMessages(library(ggplot2))
suppressMessages(library(ggmap))
suppressMessages(library(randomForest))
suppressMessages(library(geosphere))
suppressMessages(library(cleangeo))
suppressMessages(library(grid))
suppressMessages(library(foreach))
suppressMessages(library(fields))
suppressMessages(library(grid))
suppressMessages(library(beepr))
suppressMessages(library(shapefiles))
suppressMessages(library(doParallel))
suppressMessages(library(fields))
suppressMessages(library(spatial))
suppressMessages(library(gdalUtils))
suppressMessages(library(lattice))
suppressMessages(library(latticeExtra))
suppressMessages(library(maps))
suppressMessages(library(rgdal))

# FOR PARALLEL COMPUTING - not used yet but in FUTURE WORK
#suppressMessages(library(doMC))
```

```
#cl <- makeCluster(detectCores())
#registerDoMC(cl)

beep(6)

############################################### 4. THE CODE - DATA CREATION ##########################################

### GEOGRAPHIC COORDINATES SYSTEM FOR ALABAMA - used to convert between different systems
crs_utmStr <- CRS("+proj=tmerc +datum=NAD83 +lon_0=-87d30 +lat_0=30 +k=.9999333333333333 +x_0=600000 +y_0=0 +no_defs
<>")
crs_utmStr_e <- CRS("+proj=tmerc +datum=NAD83 +lon_0=-74d50 +lat_0=30 +k=.9999333333333333 +x_0=600000 +y_0=0 +no_defs
<>")

crs_nad27 <- CRS("+proj=longlat +datum=NAD27")
crs_wgs84 <- CRS("+proj=longlat +datum=WGS84")
crs_nad83 <- CRS("+proj=tmerc +lat_0=30 +lon_0=-87.5 +k=0.999933333 +x_0=600000 +y_0=0 +ellps=GRS80
+towgs84=0,0,0,0,0,0,0 +units=m +no_defs  <>")
crs_nad83e <- CRS("+proj=tmerc +lat_0=31 +lon_0=-74.5 +k=0.999933333 +x_0=600000 +y_0=0 +ellps=GRS80 +datum=NAD83
+units=us-ft +no_defs  <>")
### LOADING CENSUS 2000 DATA AND POLYGONS
census2000 <- shapefile(work_dir("tracts/TRACTS2000.shp"))
census2000 <- spTransform(census2000, crs_nad27)
census2000_data <- census2000@data
census2000_data <- data.frame(census2000_data)

### loading CENSUS 2010 DATA AND POLYGONS
census2010 <- shapefile(work_dir("tracts/TRACTS2010.shp"))
census2010 <- spTransform(census2010, crs_nad27)
census2010_data <- census2010@data
census2010_data <- data.frame(census2010_data)

### CREATING A GRID FOR A MOBILE COUNTY
# The x_lon represents the width (LONGITUDE) of the GRID
# The y_lat represents the height (LATITUDE) of the GRID
# Range represents the distance between each point on the grid

Range <- 0.03 # Setting how spreaded should be the cells
x_lon<- seq(-88.433,-87.923,by=Range)
y_lat <-seq(30.144,31.174,by=Range)
grid_lon_lat <- expand.grid(x = x_lon, y = y_lat) # and the GRID has been created
colnames(grid_lon_lat)[1:2] <- c("lon","lat") # setting the colnames into "lon" and "lat"

### Calculating the size and area of each cell of the GRID
# Picking the first and second highest values to calculate later the distances between each pair of lon lat values
x_lon_1st_highest <- sort(x_lon, partial=(length(x_lon))-1)[(length(x_lon))-1]
x_lon_2nd_highest <- sort(x_lon, partial=(length(x_lon)))[(length(x_lon))]
y_lat_1st_highest <- sort(y_lat, partial=(length(y_lat))-1)[(length(y_lat))-1]
y_lat_2nd_highest <- sort(y_lat, partial=(length(y_lat)))[(length(y_lat))]

# Creating a data frame of four pairs of lon lat values
x_lon_2points <- rbind(x_lon_1st_highest, x_lon_2nd_highest, x_lon_1st_highest, x_lon_1st_highest)
y_lat_2points <- rbind(y_lat_1st_highest, y_lat_1st_highest, y_lat_1st_highest, y_lat_2nd_highest)
two_points <- cbind(x_lon_2points, y_lat_2points)
colnames(two_points) <- c("lon","lat")
rownames(two_points) <- NULL
two_points <- data.frame(two_points)

# Calculating the distance when the lon is the same and lat value changes and vice versa
lon_distance <- rdist.earth(two_points[1,c('lon','lat')], two_points[2,c('lon','lat')], miles = FALSE, R = NULL)*
(1000/2)
# Times 1000 to convert into meters and dividing by 2 to have a cell size
lat_distance <- rdist.earth(two_points[3,c('lon','lat')], two_points[4,c('lon','lat')], miles = FALSE, R = NULL)*
(1000/2)
area_2points <- lon_distance*lat_distance # calculating the area of each cell

### CALCULATE DISTANCE BETWEEN LON LAG PAIRS OF POINTS TO THE NEAREST ROAD
set.seed(1)
# Loading datasets - CSV is better for calculating the distancec while SHP file is better for plotting
p_road = read.csv(work_dir('roads/csv_road.csv'),header=TRUE, stringsAsFactors = T) # loading the road dataset for
finding the nearest road
p_road2 <- shapefile(work_dir("roads/MAJORROADPLAN.shp")) # loading the road dataset for plotting all roads
p_road3 <- readShapeLines(work_dir("roads/tl_2010_01097_roads.shp")) # loading the detailed road dataset for plotting
all roads

# create a vector of Longitude coordinates
Long <- grid_lon_lat[,1]
# create a vector of Latitude coordinates
Lat <- grid_lon_lat[,2]
# bring them together in a data frame
coords <- as.data.frame(cbind(Long, Lat))
points<- coords
# turn this data frame into a Spatial Point object
coordinates(points) <- ~Long + Lat
summary(points)
pts <- points
proj4string(pts) <- crs_nad27
```

```
#proj4string(p_road) <- crs_nad27

n <- length(pts)
distance_road <- character(n)

#p_road <- p_road[,2:3]

cores=detectCores()
cl <- makeCluster(cores[1]-1) #not to overload your computer
registerDoParallel(cl)

system.time(
  # This loop checks for the minimal distance from each point in the grid to the closest road within a limit described
by
  # +/- long and lat values. This way the system is faster because it is not possible that he closest road is further
than +/- value
  # away from a particular grid point. This value depends and has been chosen by an author manually.

distance_road <- foreach(i=1:length(pts), .combine=cbind, .packages='fields') %dopar% {
#  for (i in seq(1, length(pts), by = 1)){
    x.sub1 <- subset(p_road, Longitude <= (coords[i,1]+0.5) & Longitude >= (coords[i,1]-0.5))
    x.sub1 <- subset(x.sub1, Latitude <= (coords[i,2]+0.5) & Latitude >= (coords[i,2]-0.5))
    distance_road[i] <- unique(min(rdist.earth(coords[i,], x.sub1[,c('Longitude','Latitude')], miles = FALSE, R =
NULL)))
  }
)
stopCluster(cl)

distance_road <- data.frame(distance_road)
distance_road <- data.frame(t(distance_road))

### CALCULATE DISTANCE BETWEEN LON LAG PAIRS OF POINTS TO THE NEAREST WATER AREA
# EVERYTHING IS READY - JUST COULD NOT TO FIND GOOD SHP FILE OF WATER AREAS
p_water = read.csv(work_dir('water/water.csv'),header=TRUE, stringsAsFactors = T) # loading the road dataset for
finding the nearest road
p_water_2 <- shapefile(work_dir('water/WATERBODY.shp'))

distance_water <- character(n)

cores=detectCores()
cl <- makeCluster(cores[1]-1) #not to overload your computer
registerDoParallel(cl)

system.time(
  # This loop checks for the minimal distance from each point in the grid to the closest road within a limit described
by
  # +/- long and lat values. This way the system is faster because it is not possible that he closest road is further
than +/- value
  # away from a particular grid point. This value depends and has been chosen by an author manually.

  distance_water <- foreach(i=1:length(pts), .combine=cbind, .packages='fields') %dopar% {
    #  for (i in seq(1, length(pts), by = 1)){
    x.sub1 <- subset(p_water, x <= (coords[i,1]+0.5) & x >= (coords[i,1]-0.5))
    x.sub1 <- subset(x.sub1, y <= (coords[i,2]+0.5) & y >= (coords[i,2]-0.5))
    distance_water[i] <- unique(min(rdist.earth(coords[i,], x.sub1[,c('x','y')], miles = FALSE, R = NULL)))
  }
)
stopCluster(cl)

distance_water <- data.frame(distance_water)
distance_water <- data.frame(t(distance_water))

### CALCULATE DISTANCE BETWEEN LON LAG PAIRS OF POINTS TO THE NEAREST SCHOOL
schools = read.csv(work_dir('schools/schools.csv'),header=TRUE, stringsAsFactors = T) # loading the road dataset for
finding the nearest road
schools_2 <- shapefile(work_dir("schools/SCHOOLPNT.shp"))

n <- length(pts)
schools_distance <- character(n)

p_school <- schools[,8:9]

cores=detectCores()
cl <- makeCluster(cores[1]-1) #not to overload your computer
registerDoParallel(cl)

system.time(
  # This loop checks for the minimal distance from each point in the grid to the closest school.
  schools_distance <- foreach(i=1:length(pts), .combine=cbind, .packages='fields') %dopar% {
 # for (i in seq(1, length(pts), by = 1)){
  schools_distance[i] <- unique(min(rdist.earth(coords[i,], p_school[,c('Longitude','Latitude')], miles = FALSE, R =
NULL)))
  }
)

stopCluster(cl)
```

```
schools_distance <- data.frame(schools_distance)
schools_distance <- data.frame(t(schools_distance))

### ELEVATION LAYER
elevations_xy <- matrix(c(grid_lon_lat[,1], grid_lon_lat[,2]), nc=2)
elevations_xy <- data.frame(elevations_xy)

m <- data.frame(lon = c(elevations_xy[,1]), lat = c(elevations_xy[,2]))
x1 <-  raster(work_dir('USA1_msk_alt.grd'), state="01", county="1097")
x1a <- x1[[1]]
xm <- terrain(x1a, opt=c('slope', 'aspect'), unit='degrees')
plot(xm)
elevation <- cbind(m, alt = extract(x1a, m))

### CREATTING DATA FROM CENSUS 2000
points_in_2000 <-NULL
tract_points_2000 <- NULL
for (i in seq(1,nrow(census2000_data),by=1)){
  coord_2000 <- census2000@polygons[[i]]@Polygons[[1]]@coords
  points_in_2000 <- point.in.polygon(grid_lon_lat[,1], grid_lon_lat[,2], coord_2000[,1], coord_2000[,2],
mode.checked=FALSE)
  tract_points_2000$a <- points_in_2000
  tract_points_2000 <- data.frame(tract_points_2000)
  colnames(tract_points_2000)[i] <- census2000_data$TRACT[i]
}
tract_points_2000 <- data.frame(tract_points_2000)


grid_final_2000<- cbind(x_lon, y_lat, elevation[,3], distance_road, distance_water, schools_distance,
tract_points_2000)
grid_final_2000 <- grid_final_2000[apply(tract_points_2000[,-1], 1, function(x) !all(x==0)),]
colnames(grid_final_2000)[3] <- c("elevation")
grid_final_2000 <- grid_final_2000[complete.cases(grid_final_2000$elevation),]
#grid_final_2000 <- subset(grid_final_2000, elevation >= 1)
colnames(grid_final_2000) <- gsub("\\X", "", colnames(grid_final_2000))
for (i in seq(1,(ncol(grid_final_2000)), by=1)){
  grid_final_2000[,c(i)]<-as.numeric(as.character(grid_final_2000[,c(i)]))
}

#The following code checks if one point is inside (or on the border of) few polygons. If so delete.
ones_2000 <- apply(grid_final_2000[,6:ncol(grid_final_2000)], 1, function(x) sum(x==1, na.rm=TRUE))
grid_final_2000$ones_2000 <- ones_2000
grid_final_2000 <- grid_final_2000[grid_final_2000$ones_2000 == 1, ] # where 1 exist more than once
grid_final_2000 <- grid_final_2000[,c(1:120)]

### CREATTING DATA FROM CENSUS 2010

points_in_2010 <-NULL
tract_points_2010 <- NULL
for (i in seq(1,nrow(census2010_data),by=1)){
  coord_2010 <- census2010@polygons[[i]]@Polygons[[1]]@coords
  points_in_2010 <- point.in.polygon(grid_lon_lat[,1], grid_lon_lat[,2], coord_2010[,1], coord_2010[,2],
mode.checked=FALSE)
  tract_points_2010$a <- points_in_2010
  tract_points_2010 <- data.frame(tract_points_2010)
  colnames(tract_points_2010)[i] <- census2010_data$TRACT[i]
}
tract_points_2010 <- data.frame(tract_points_2010)

grid_final_2010<- cbind(x_lon, y_lat, elevation[,3], distance_road, distance_water, schools_distance,
tract_points_2010)
grid_final_2010<- grid_final_2010[apply(tract_points_2010[,-1], 1, function(x) !all(x==0)),]
colnames(grid_final_2010)[3] <- c("elevation")
grid_final_2010 <- grid_final_2010[complete.cases(grid_final_2010$elevation),]
#grid_final_2010 <- subset(grid_final_2010, elevation >= 1)
colnames(grid_final_2010) <- gsub("\\X", "", colnames(grid_final_2010))
for (i in seq(1,(ncol(grid_final_2010)), by=1)){
  grid_final_2010[,c(i)]<-as.numeric(as.character(grid_final_2010[,c(i)]))
}

ones_2010 <- apply(grid_final_2010[,6:ncol(grid_final_2010)], 1, function(x) sum(x==1, na.rm=TRUE))
grid_final_2010$ones_2010 <- ones_2010
grid_final_2010 <- grid_final_2010[grid_final_2010$ones_2010 == 1, ] # where 1 exist more than once
grid_final_2010 <- grid_final_2010[,c(1:121)]

### CREATING A FINAL GRID with tract codes for 2000

grid_final2_2000 <- grid_final_2000
for (i in seq(1, (ncol(grid_final2_2000)), by=1)){
  grid_final2_2000[,c(i)]<-as.numeric(as.character(grid_final2_2000[,c(i)]))
}
for (i in seq(1, (ncol(grid_final2_2000)-6), by=1)){
  grid_final2_2000[,6+i][grid_final2_2000[,6+i] == "1"] <- colnames(grid_final2_2000[5+i])
}
tracts_2000 <- grid_final2_2000[,(7:ncol(grid_final2_2000))]
```

```
tracts_2000<-  apply(tracts_2000,1, function(x) head(x[x!=0],1))
grid_final2_2000 <- cbind(grid_final2_2000[,c(1:6)], tracts_2000)
colnames(grid_final2_2000)[7] <- c("Tract")

### CREATING A FINAL GRID with tract codes for 2010
grid_final2_2010 <- grid_final_2010
for (i in seq(1, (ncol(grid_final2_2010)), by=1)){
  grid_final2_2010[,c(i)]<-as.numeric(as.character(grid_final2_2010[,c(i)]))
}
for (i in seq(1, (ncol(grid_final2_2010)-6), by=1)){
  grid_final2_2010[,5+i][grid_final2_2010[,6+i] == "1"] <- colnames(grid_final2_2010[5+i])
}
tracts_2010 <- grid_final2_2010[,(7:ncol(grid_final2_2010))]
tracts_2010<-  apply(tracts_2010,1, function(x) head(x[x!=0],1))
grid_final2_2010 <- cbind(grid_final2_2010[,c(1:6)], tracts_2010)
colnames(grid_final2_2010)[7] <- c("Tract")

#### CREATING TEST AND TRAIN DATA (still adding statistical data - difficult to obtain)
#### Data acquired from:
# income for each zipcode
# rest of the data - for each US Census tract

train<-NULL
train <- grid_final2_2000
colnames(train)[1:7] <- (c("lon","lat", "elevation", "distance_road", "distance_water", "distance_school", "TRACT"))
train<- merge(train, census2000_data, by="TRACT", all.x=T, sort=T)
train <- subset(train, select = -c(TRACT, ID, COUNTY, STATE, NAME, LOGRECNO, Shape_area, Shape_len))
for(i in 1:7)
{
  train[,c(i)]<-as.numeric(as.character(train[,c(i)]))
}

# INCOME DATA - ZIPCODE LEVEL

zipcode <- shapefile(work_dir("zipcodes/ZIPCODE.shp"))
plot(zipcode)
gIsValid(zipcode, byid = FALSE, reason=FALSE)
zipcode <- clgeo_Clean(zipcode)
gIsValid(zipcode, byid = FALSE, reason=FALSE)

zipcode <- spTransform(zipcode, crs_wgs84)

zipcode_data <- zipcode@data
zipcode_data <- data.frame(zipcode_data)
AL_097_zipcodes = read.csv(work_dir('zipcodes/AL_097_zipcodes2.csv'),header=TRUE, stringsAsFactors = T)
AL_097_zipcodes <-AL_097_zipcodes[,1:7]
colnames(AL_097_zipcodes)[1] <- c("NAME")
zipcode_data<- merge(zipcode_data, AL_097_zipcodes, by="NAME", all.x=T, sort=T) #merging both data frames by NAME
zipcode@data <- zipcode_data

### CREATTING DATA FROM ZipCode shape file
points_in_train <-NULL
zipcode_points_train <- NULL
for (i in seq(1,nrow(zipcode_data),by=1)){
  coord_train <- zipcode@polygons[[i]]@Polygons[[1]]@coords
  points_in_train <- point.in.polygon(train[,1], train[,2], coord_train[,1], coord_train[,2], mode.checked=FALSE)
  zipcode_points_train$a <- points_in_train
  zipcode_points_train <- data.frame(zipcode_points_train)
  colnames(zipcode_points_train)[i] <- as.character(zipcode@data$NAME[i])
}
zipcode_points_train <- data.frame(zipcode_points_train)
train_final = NULL
train_final<- cbind(train, zipcode_points_train)

for(i in seq(66,ncol(train_final),by=1)){
  train_final[,c(i)]<-as.numeric(train_final[,c(i)])
}

train_final<- train_final[apply(zipcode_points_train[,-1], 1, function(x) !all(x==0)),]
colnames(train_final) <- gsub("\\X", "", colnames(train_final))

# Again checking if a point belongs to one or more zipcodes (if so - delete duplicating rows)
ones_train_final <- apply(train_final[,66:ncol(train_final)], 1, function(x) sum(x==1, na.rm=TRUE))
train_final$ones_train_final <- ones_train_final
train_final <- train_final[train_final$ones_train_final == 1, ] # where 1 exist more than once
train_final <- train_final[,c(1:99)]

### CREATING A FINAL GRID with tract codes for TRAIN
train_final2 <- train_final
train_final2 <- train_final2[,c(1:(ncol(train_final)-1))]

for (i in seq(1, (ncol(train_final2) - ncol(train)), by=1)){
  train_final2[,65+i][train_final2[,65+i] == "1"] <- as.character(colnames(train_final2[65+i]))
}
zipcodes_train <- train_final2
zipcodes_train <- zipcodes_train[apply(zipcodes_train[c(66:(ncol(train_final2)))],1,function(z) any(z!=0)),]
```

```
for(i in seq(66,ncol(zipcodes_train),by=1)){
  zipcodes_train[,c(i)]<-as.numeric(zipcodes_train[,c(i)])
}
zipcodes_train_zipcode <- zipcodes_train[,c(66:(ncol(train_final2)))]
zipcodes_train_zipcode <-apply(zipcodes_train_zipcode,1, function(x) head(x[x!="0"],1))
zipcodes_train_zipcode<-data.frame(zipcodes_train_zipcode)
colnames(zipcodes_train_zipcode)[1] <- c("ZIPCODE")

zipcodes_train <- cbind(zipcodes_train[,c(1:65)], zipcodes_train_zipcode)
zipcodes_train$ZIPCODE[zipcodes_train$ZIPCODE == 36603.1] <- 36303

zipcode_data_train <- zipcode_data[,c(1, 14,16)]
zipcode_data_train$ChangeInIncome1990_2000 <- (zipcode_data_train$MHI2000_value2015 -
zipcode_data_train$MHI1990_value2015)
colnames(zipcode_data_train)[1] <- c("ZIPCODE")
zipcode_data_train <- zipcode_data_train[, c(1,4)]
zipcode_data_train <- zipcode_data_train[!duplicated(zipcode_data_train), ]
zipcodes_train<- merge(zipcodes_train, zipcode_data_train, by="ZIPCODE", all.x=F, sort=F) #merging both data frames by
NAME
train <- zipcodes_train[,c(2:66)]

## CALCULATE DISTANCE BETWEEN LON LAG PAIRS OF POINTS TO URBANIZED AREA IN 2000
set.seed(1)
p_2000 = read.csv(work_dir('urban/p_2000.csv'),header=TRUE, stringsAsFactors = T) # loading the road dataset for
finding the nearest road
p_2000_2 <- shapefile(work_dir("urban/p_2000.shp"))

gIsValid(p_2000_2, byid = FALSE, reason=FALSE)
p_2000_2 <- clgeo_Clean(p_2000_2)
gIsValid(p_2000_2, byid = FALSE, reason=FALSE)

# create a vector of Longitude coordinates
Long_train <- train[,1]
# create a vector of Latitude coordinates
Lat_train <- train[,2]
# bring them together in a data frame
coords_train <- as.data.frame(cbind(Long_train, Lat_train))

## Set up container for results
n <- nrow(train)
distance_2000 <- character(n)

p_urban2000 <- p_2000[,2:3]

system.time(
  # This loop checks for the minimal distance from each point in the grid to the urbanized area 2000.
for (i in seq(1, nrow(train), by = 1)){
    distance_2000[i] <- unique(min(rdist.earth(coords_train[i,], p_urban2000[,c('Longitude','Latitude')], miles =
FALSE, R = NULL)))
  }
)

distance_2000 <- data.frame(distance_2000)

#URBANIZED IN 2010 for a train set
p_2010 = read.csv(work_dir('urban/p_2010.csv'),header=TRUE, stringsAsFactors = T) # loading the road dataset for
finding the nearest road
p_2010_2 <- shapefile(work_dir("urban/p_2010.shp"))

gIsValid(p_2010_2, byid = FALSE, reason=FALSE)
p_2010_2 <- clgeo_Clean(p_2010_2)
gIsValid(p_2010_2, byid = FALSE, reason=FALSE)

urbanized2010 <- NULL
n <- character(length=nrow(train))

urbanized2010 <- list(n)
nPolys <- sapply(p_2010_2@polygons, function(x)length(x@Polygons))
region <- p_2010_2[which(nPolys==max(nPolys)),]

for (i in seq(1,nrow(train),by=1)){
  urbanized2010[i] <- gContains(region,SpatialPoints(train[i,1:2],proj4string=CRS(proj4string(region))))
}
urbanized2010 <-unlist(urbanized2010)
urbanized2010 <- data.frame(urbanized2010)

for(i in 1)
{
  urbanized2010[,c(i)]<-as.numeric(urbanized2010[,c(i)])
}
urbanized2010 <- data.frame(urbanized2010)

#URBANIZED IN 2000 for a train set
p_2000 = read.csv(work_dir('urban/p_2000.csv'),header=TRUE, stringsAsFactors = T) # loading the road dataset for
finding the nearest road
```

```
p_2000_2 <- shapefile(work_dir("urban/p_2000.shp"))

gIsValid(p_2000_2, byid = FALSE, reason=FALSE)
p_2000_2 <- clgeo_Clean(p_2000_2)
gIsValid(p_2000_2, byid = FALSE, reason=FALSE)

urbanized2000 <- NULL
n <- character(length=nrow(train))
urbanized2000 <- list(n)
nPolys <- sapply(p_2000_2@polygons, function(x)length(x@Polygons))
region <- p_2000_2[which(nPolys==max(nPolys)),]

for (i in seq(1,nrow(train),by=1)){
  urbanized2000[i] <- gContains(region,SpatialPoints(train[i,1:2],proj4string=CRS(proj4string(region))))
}
urbanized2000<-unlist(urbanized2000)
urbanized2000 <- data.frame(urbanized2000)

for(i in 1)
{
  urbanized2000[,c(i)]<-as.numeric(urbanized2000[,c(i)])
}
urbanized2000 <- data.frame(urbanized2000)
unique(urbanized2000)

### CALCULATE DISTANCE BETWEEN LON LAG PAIRS OF POINTS TO DOWNTOWN MOBILE TRAIN
mobile_downtown_lon <- c("-88.043056") # coordinates of Downtown Mobile
mobile_downtown_lat <- c("30.694444") # coordinates of Downtown Mobile
mobile_downtown <- cbind(mobile_downtown_lon, mobile_downtown_lat)
mobile_downtown <- data.frame(mobile_downtown)

for (i in range(1:ncol(mobile_downtown))){
  mobile_downtown[,i] <- as.numeric(as.character(mobile_downtown[,i]))
}

colnames(mobile_downtown) <- c("Longitude","Latitude")

n <- nrow(train)
downtown_distance_train <- character(n)

system.time(
  # This loop checks for the minimal distance from each point in the grid to Downtown Mobile.

  for (i in seq(1, nrow(train), by = 1)){
    downtown_distance_train[i] <- unique(min(rdist.earth(coords_train[i,],
mobile_downtown[,c('Longitude','Latitude')], miles = FALSE, R = NULL)))
  }
)
downtown_distance_train <- data.frame(downtown_distance_train)

############################################ CREATING FINAL TRAIN DATA
train<- cbind(train, downtown_distance_train, distance_2000, urbanized2000, urbanized2010)
colnames(train)[66:69] <- c("downtown_distance", "distance_2000", "urbanized2000", "urbanized2010")
for(i in seq(1,ncol(train),by=1))
{
  train[,c(i)]<-as.numeric(as.character(train[,c(i)]))
}

write.csv(train, file='train.csv', quote=FALSE,row.names=F)

############################################ TEST DATA
test<-NULL
test <- grid_final2_2010
colnames(test)[1:7] <- (c("lon","lat", "elevation", "distance_road", "distance_water", "distance_school", "TRACT"))
test<- merge(test, census2010_data, by="TRACT", all.x=T, sort=T)
test <- subset(test, select = -c(TRACT, ID, COUNTY, STATE, NAME, LOGRECNO, AREAWATR, Shape_area, Shape_len))
for(i in 1:7)
{
  test[,c(i)]<-as.numeric(as.character(test[,c(i)]))
}

### CREATTING DATA FROM ZipCode shape file
points_in_test <-NULL
zipcode_points_test <- NULL
for (i in seq(1,nrow(zipcode_data),by=1)){
  coord_test <- zipcode@polygons[[i]]@Polygons[[1]]@coords
  points_in_test <- point.in.polygon(test[,1], test[,2], coord_test[,1], coord_test[,2], mode.checked=FALSE)
  zipcode_points_test$a <- points_in_test
  zipcode_points_test <- data.frame(zipcode_points_test)
  colnames(zipcode_points_test)[i] <- as.character(zipcode@data$NAME[i])
}
zipcode_points_test <- data.frame(zipcode_points_test)

test_final = NULL
test_final<- cbind(test, zipcode_points_test)
test_final<- test_final[apply(zipcode_points_test[,-1], 1, function(x) !all(x==0)),]
```

```
colnames(test_final) <- gsub("\\X", "", colnames(test_final))
ones_test_final <- apply(test_final[,66:ncol(test_final)], 1, function(x) sum(x==1, na.rm=TRUE))
test_final$ones_test_final <- ones_test_final
test_final <- test_final[test_final$ones_test_final == 1, ] # where 1 exist more than once
test_final <- test_final[,c(1:99)]

### CREATING A FINAL GRID with tract codes for TEST

test_final2 <- test_final
test_final2 <- test_final2[,c(1:(ncol(test_final)-1))]

for (i in seq(1, (ncol(test_final2) - ncol(test)), by=1)){
  test_final2[,65+i][test_final2[,65+i] == "1"] <- as.character(colnames(test_final2[64+i]))
}
zipcodes_test <- test_final2
zipcodes_test <- zipcodes_test[apply(zipcodes_test[c(65:(ncol(test_final2)))],1,function(z) any(z!=0)),]

for(i in seq(65,ncol(zipcodes_test),by=1)){
  zipcodes_test[,c(i)]<-as.numeric(zipcodes_test[,c(i)])
}
zipcodes_test_zipcode <- zipcodes_test[,c(65:(ncol(test_final2)))]
zipcodes_test_zipcode <-apply(zipcodes_test_zipcode,1, function(x) head(x[x!="0"],1))
zipcodes_test_zipcode<-data.frame(zipcodes_test_zipcode)
colnames(zipcodes_test_zipcode)[1] <- c("ZIPCODE")

zipcodes_test <- cbind(zipcodes_test[,c(1:64)], zipcodes_test_zipcode)
zipcodes_test$ZIPCODE[zipcodes_test$ZIPCODE == 36603.1] <- 36303 # fixing zipcode

zipcode_data_test <- zipcode_data[,c(1, 12,14)]
zipcode_data_test$ChangeInIncome2000_2010 <- (zipcode_data_test$MHI2010_value2015 -
zipcode_data_test$MHI2000_value2015)
colnames(zipcode_data_test)[1] <- c("ZIPCODE")
zipcode_data_test <- zipcode_data_test[, c(1,4)]
zipcode_data_test <- zipcode_data_test[!duplicated(zipcode_data_test), ]
zipcodes_test<- merge(zipcodes_test, zipcode_data_test, by="ZIPCODE", all.x=F, sort=F) #merging both data frames by
NAME
test <- zipcodes_test[,c(2:66)]

## CALCULATE DISTANCE BETWEEN LON LAG PAIRS OF POINTS TO URBANIZED AREA IN 2010 FOR A TEST SET

# create a vector of Longitude coordinates
Long_test <- test[,1]
# create a vector of Latitude coordinates
Lat_test <- test[,2]
# bring them together in a data frame
coords_test <- as.data.frame(cbind(Long_test, Lat_test))

## Set up container for results
n <- nrow(test)
distance_2010 <- character(n)

p_urban2010 <- p_2010[,2:3]

system.time(
  # This loop checks for the minimal distance from each point in the grid to the urbanized area 2000.

  for (i in seq(1, nrow(test), by = 1)){
    distance_2010[i] <- unique(min(rdist.earth(coords_test[i,], p_urban2010[,c('Longitude','Latitude')], miles =
FALSE, R = NULL)))
  }
)

distance_2010 <- data.frame(distance_2010)

trueCentroids_2010 = gCentroid(p_2010_2,byid=TRUE)

#URBANIZED IN 2010 for a test set
urbanized2010_test <- NULL
n <- character(length=nrow(test))
urbanized2010_test <- list(n)
nPolys_10test <- sapply(p_2010_2@polygons, function(x)length(x@Polygons))
region_10test <- p_2010_2[which(nPolys_10test==max(nPolys_10test)),]

for (i in seq(1,nrow(test),by=1)){
  urbanized2010_test[i] <-
gContains(region_10test,SpatialPoints(test[i,1:2],proj4string=CRS(proj4string(region_10test))))
}
urbanized2010_test<-unlist(urbanized2010_test)
urbanized2010_test <- data.frame(urbanized2010_test)

for(i in 1)
{
  urbanized2010_test[,c(i)]<-as.numeric(urbanized2010_test[,c(i)])
}
urbanized2010_test <- data.frame(urbanized2010_test)
```

```
### CALCULATE DISTANCE BETWEEN LON LAG PAIRS OF POINTS TO DOWNTOWN MOBILE TEST

n <- nrow(test)
downtown_distance_test <- character(n)

system.time(
  # This loop checks for the minimal distance from each point in the grid to Downtown Mobile.

  for (i in seq(1, nrow(test), by = 1)){
    downtown_distance_test[i] <- unique(min(rdist.earth(coords_test[i,], mobile_downtown[,c('Longitude','Latitude')],
miles = FALSE, R = NULL)))
  }
)
downtown_distance_test <- data.frame(downtown_distance_test)


test<- cbind(test, downtown_distance_test, distance_2010, urbanized2010_test)
colnames(test)[67:69] <- c("downtown distance","distance_2010", "urbanized2010")

write.csv(test, file='test.csv', quote=FALSE,row.names=F)

############################################## 4. THE CODE - MAPS AND PLOTS ########################################
mobile_county <- readShapeSpatial(work_dir("county/mobile_county.shp")) # county border

#PLOT ZIP CODES
map_mobile_zipcodes_z9 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom = 9,
                                  maptype = "satellite", scale = 2)
ggmap(map_mobile_zipcodes_z9) + geom_path(data=zipcode,  aes(long,lat, group=group), size=1, color="red")
grid.text("ZipCodes", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
#text(getSpPPolygonsLabptSlots(zipcode), labels=zipcode$NAME, cex=0.7)
ggsave("map_mobile_zipcodes_z9.pdf")
dev.off()

#PLOT MEDIAN HOUSEHOLD INCOME 2000 - VALUE AS OF 2015
spplot(zipcode, z="MHI2000_value2015", cuts= 33)
grid.text("MEDIAN HOUSEHOLD INCOME 2000 - VALUE AS OF 2015", x=unit(0.9, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_medianincome2000.pdf")
dev.off()

#PLOT MEDIAN HOUSEHOLD INCOME 2010 - VALUE AS OF 2015
spplot(zipcode, z="MHI2010_value2015", cuts= 33)
grid.text("MEDIAN HOUSEHOLD INCOME 2010 - VALUE AS OF 2015", x=unit(0.9, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_medianincome2010.pdf")
dev.off()

#PLOT TRACTS 2000 on a google map
map_mobile_tracts2000_z9 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom =
9,
                                    maptype = "satellite", scale = 2)
ggmap(map_mobile_tracts2000_z9) + geom_path(data=census2000,  aes(long,lat, group=group), size=1, color="red")
grid.text("TRACTS 2000", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_tracts2000_z9.pdf")
dev.off()

#PLOT TRACTS 2010
map_mobile_tracts2010_z9 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom =
9,
                                    maptype = "satellite", scale = 2)
ggmap(map_mobile_tracts2010_z9) + geom_path(data=census2010,  aes(long,lat, group=group), size=1, color="red")
grid.text("TRACTS 2010", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_tracts2010_z9.pdf")
dev.off()

# Population graphs for 2000 and 2010
#1)
spplot(census2000, z="TOTAL", cuts= 7)
grid.text("Population 2000", x=unit(0.9, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_population2000.pdf")
dev.off()

#2)
spplot(census2010, z="TOTAL", cuts= 7)
grid.text("Population 2010", x=unit(0.9, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_population2010.pdf")
dev.off()

#PLOT SCHOOLS
#1)
map_mobile_schools_z9 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom = 9,
                                 maptype = "satellite", scale = 2)
ggmap(map_mobile_schools_z9) + geom_point(data=schools,  aes(Longitude,Latitude), size=3, color="blue")+
geom_path(data=mobile_county,  aes(long,lat, group=group), size=1, color="red")
grid.text("Schools - zoom 9", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_schools_z9.pdf")
dev.off()
```

```
#2)
map_mobile_schools_z11 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom = 11,
                                  maptype = "satellite", scale = 2)
ggmap(map_mobile_schools_z11) + geom_point(data=schools,  aes(Longitude,Latitude), size=3, color="blue")+
geom_path(data=mobile_county,  aes(long,lat, group=group), size=1, color="red")
grid.text("Schools - zoom 11", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_schools_z11.pdf")
dev.off()

#PLOT MAJOR_ROADS, county boarder on a google map
map_mobile_r2_z9 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom = 9,
                      maptype = "satellite", scale = 2)
ggmap(map_mobile_r2_z9) + geom_path(data=p_road2,  aes(long,lat, group=group), size=1, color="black") +
geom_path(data=mobile_county,  aes(long,lat, group=group), size=1, color="red")
grid.text("Major Road Network - zoom 11", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_r2_z9.pdf")
dev.off()

#PLOT DETAILED ROADS, county boarder on a google map with a different ZOOM value
#1)
map_mobile_r3_z9 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom = 9,
                       maptype = "satellite", scale = 2)
ggmap(map_mobile_r3_z9) + geom_path(data=p_road3,  aes(long,lat, group=group), size=1, color="black") +
geom_path(data=mobile_county,  aes(long,lat, group=group), size=1, color="red")
grid.text("Big Road Network - zoom 9", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_r3_z9.pdf")
dev.off()

#2)
map_mobile_r3_z11 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom = 11,
                       maptype = "satellite", scale = 2)
ggmap(map_mobile_r3_z11) + geom_path(data=p_road3,  aes(long,lat, group=group), size=1, color="black") +
geom_path(data=mobile_county,  aes(long,lat, group=group), size=1, color="red")
grid.text("Big Road Network - zoom 11", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_r3_z11.pdf")
dev.off()

#3)
map_mobile_r3_z13 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom = 13,
                       maptype = "satellite", scale = 2)
ggmap(map_mobile_r3_z13) + geom_path(data=p_road3,  aes(long,lat, group=group), size=1, color="black") +
geom_path(data=mobile_county,  aes(long,lat, group=group), size=1, color="red")
grid.text("Big Road Network - zoom 13", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_r3_z13.pdf")
dev.off()

#PLOT urbanized 2000, county boarder on a google map
map_mobile_urbanized2000 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom =
9,
                      maptype = "satellite", scale = 2)
ggmap(map_mobile_urbanized2000) + geom_polygon(data=p_2000_2,  aes(long,lat, group=group), size=1, color="black",
fill="grey") + geom_path(data=mobile_county,  aes(long,lat, group=group), size=1, color="red")
grid.text("Urbanized Area in 2000", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_urbanized2000.pdf")
dev.off()

#PLOT urbanized 2010, county boarder on a google map
map_mobile_urbanized2010 <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom =
9,
                                    maptype = "satellite", scale = 2)
ggmap(map_mobile_urbanized2010) + geom_polygon(data=p_2010_2,  aes(long,lat, group=group), size=1, color="black",
fill="grey") + geom_path(data=mobile_county,  aes(long,lat, group=group), size=1, color="red")
grid.text("Urbanized Area in 2010", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_mobile_urbanized2010.pdf")
dev.off()

################################### 5. SAVE THE WORKSPACE & CLEAR ALL DATA###########################################
save.image(file="UGSM_Data.RData") # saving everything into RData file

#rm(list=ls()) # remove everything from Environment

############################################### 5. PREDICTION ######################################################
# Loading data for the prediction
#train = read.csv('train.csv',header=TRUE, stringsAsFactors = F)
#test = read.csv('test.csv',header=TRUE, stringsAsFactors = F)

#Setting the same colnames for train and test datasets
colnames(train)[c(65,67,68)] <- c("deltaIncome","distance_to_urban","previouslyUrbanized")
colnames(test)[c(65,67,68)] <- c("deltaIncome","distance_to_urban","previouslyUrbanized")

### ORGANIZING DATA FOR MODELS
dat_train = data.frame(x = train[,1:68], y = as.factor(train[,69]))
for (i in seq(1, (ncol(dat_train)-1), by=1)){
  dat_train[,c(i)]<-as.numeric(as.character(dat_train[,c(i)]))
}
dat_test = data.frame(x = test[,1:68])
```

```
for (i in seq(1, ncol(dat_test), by=1)){
  dat_test[,c(i)]<-as.numeric(as.character(dat_test[,c(i)]))
}

# Creating smaller random data
trainPortion <- floor(nrow(dat_train)*0.8)
train_sm_gs <- dat_train[1:floor(trainPortion/2),1:ncol(dat_train)]
test_sm_gs <- dat_train[(floor(trainPortion/2)+1):trainPortion,]
portion_train <- round(nrow(train_sm_gs)*0.25)
portion_test <- round(nrow(test_sm_gs)*0.25)
dat_train_sm <- train_sm_gs[sample(nrow(train_sm_gs), portion_train), ]
dat_test_sm <- test_sm_gs[sample(nrow(test_sm_gs), portion_test), ]

### FEATURE IMPORTANCE USING RANDOM FOREST
set.seed(4543)
train.rf <- randomForest(y ~ .,
                         data=dat_train[,3:69], #without the first 2 columns - lon and lat
                         ntree=1000,
                         keep.forest=FALSE,
                         importance=TRUE,
                         cv.fold=10)

importance(train.rf)
importance(train.rf, type=1)
varImpPlot <- varImpPlot(train.rf)
varImpPlot(train.rf)
write.csv(varImpPlot, file='varImpPlot.csv', quote=FALSE,row.names=T)
dev.off()

############################################### 4. PREDICTING URBANIZATION GROWTH USING H2O MODEL ##################
# For the prediction an H2O model has been used.
# max_mem_size = the amount of memory is going to be used for H2O package
# nthread =-1 all CPU threads
suppressMessages(library(h2o))
localH2O <- h2o.init(nthread=-1,max_mem_size="50g")

### FINDING the best parameters mix to perform a full prediction. Here the model is assignig all possible
# mixes of parameters and looking for the minimal MSE (MEAN SQUARED ERROR). It is possible because I am
# using the validation frame which is a randomly chosen soubset of an original train set.

#Parameters for H2O model:
list_models <- matrix(0, nrow=0, ncol=8) # creating an empty list to save models
hidden_opt <- list(c(1024,512,256), c(500,500,500), c(100,300,100))
l1_opt <- c(1e-5 ,1e-7)
epochs_opt <- c(50,60)
activation_opt <- c("Rectifier", "Tanh", "RectifierWithDropout", "TanhWithDropout")
hyper_params <- list(hidden = hidden_opt, l1 = l1_opt, epochs = epochs_opt, activation = activation_opt)

set.seed(825)

print(sprintf("White Eagle: Loading training data for a grid search"))
train_sm.hex = as.h2o(dat_train_sm, destination_frame = "train")
print(sprintf("White Eagle: Loading testing data for a grid search"))
test_sm.hex <- as.h2o(dat_test_sm, destination_frame = "test")

print(sprintf("White Eagle: Performing a grid search"))
model_grid <- h2o.grid("deeplearning",
                       x = c(3 : 68),
                       y = 69,
                       training_frame = train_sm.hex,
                       validation_frame = test_sm.hex,
                       hyper_params = hyper_params,
                       nfolds = 10,
                       score_training_samples = 0,
                       variable_importances = TRUE,
                       #    train_samples_per_iteration = 2000,
                       max_w2 = 10,
                       seed = 1

)
for (model_id in model_grid@model_ids) {
  gc()
  model <- h2o.getModel(model_id)
 # mse <- h2o.mse(model, valid = TRUE)
  auc <- h2o.auc(model, valid = TRUE)

  list_models <- rbind(list_models,c(model@model_id,
                                     model@allparameters$hidden[1],
                                     model@allparameters$hidden[2],
                                     model@allparameters$hidden[3],
                                     model@allparameters$l1,
                                     model@allparameters$epochs,
                                     model@allparameters$activation,
                                     auc))
}
```

```
colnames(list_models) <- c("model_id", "model_hidden1", "model_hidden2", "model_hidden3", "model_l1", "model_epochs",
"activation", "auc")

best_model <- NULL
# The best model will have the highest AUC:
best_model <- as.matrix(list_models[which(abs(as.numeric(list_models[,8])) == max(abs(as.numeric(list_models[,8])))),
])
write.csv(best_model, file='best_model.csv', quote=FALSE,row.names=F)
write.csv(list_models, file='list_models.csv', quote=FALSE,row.names=F)

### Now we can perform the full prediction with the best parameter's set chosen previously
print(sprintf("White Eagle: Final Prediction for: %f", i))
print(sprintf("White Eagle: Loading training data for a Final Prediction"))
train.hex = as.h2o(dat_train[,3:ncol(dat_train)], destination_frame = "train") #without long lat values which are like
an id here
print(sprintf("White Eagle: Loading testing data for a Final Prediction"))
test.hex <- as.h2o(dat_test[,3:ncol(dat_test)], destination_frame = "test")

print(sprintf("White Eagle: Performing a Final Prediction"))
model_main <- h2o.deeplearning(
  x = c(1 : 66),
  y = 67,
  training_frame = train.hex,
  nfolds = 10,
  epochs = as.numeric(best_model[6]),
  hidden = c(as.numeric(best_model[2]), as.numeric(best_model[3]), as.numeric(best_model[4])),
  l1 = as.numeric(best_model[5]),
  activation = best_model[7],
#  score_training_samples = 0,
  variable_importances = TRUE,
  #     train_samples_per_iteration = 2000,
  max_w2 = 10,
  seed = 1
)

# Save a model:
model_main_save <- h2o.saveModel(object = model_main, path=getwd(), force = TRUE)
print(model_main_save)
# Load the model
#saved_model_main <- h2o.loadModel(model_main_save)

# PREDICTION
predictions = as.data.frame(h2o.predict(model_main, test.hex))
predictions2<- predictions[,1]
# Creating a data frame with coordinates and values of urbanization for 2020:
final_prediction <- cbind(test[,c(1:2)],predictions2)
### Comparing the results with the previously urbanized area(in 2010) and checking which cells have
# changed their urbanization status

urbanization_frame <- cbind(final_prediction, test[,68])
colnames(urbanization_frame)[c(1:4)]<-c("lon","lat","urban2020","urban2010")
urbanization_frame$change <- match(urbanization_frame$urban2010,urbanization_frame$urban2020)
#Deleting rows in the urbanization change frame with values 1 (matching values) and
#creating a frame with only different values
urbanization_frame <- urbanization_frame[urbanization_frame$change!=1,]

#Creating data frames in_plus and in_minus.
#In_plus - cells which have become urbanized (growth)
#In_minus - shrinkage of urbanization
change_in_plus <- urbanization_frame[urbanization_frame$urban2020=="1",]
change_in_plus <- change_in_plus[,1:2] # getting just coordinates to plot it
change_in_minus <- urbanization_frame[urbanization_frame$urban2020=="0",]
change_in_minus <- change_in_minus[,1:2] # getting just coordinates to plot it
### PLOTTING THE PREDICTIONS ON THE MAP

# Save the Workspace
save.image("UGSM_DataPrediction.RData")

# Plotting on the 2010 urbanzied layer
#PLOT urbanized 2010, county boarder on a google map and urbanization changes (points in red and/or blue)
#1) Force of change (in plus, in minus or neutral)
map_final_prediction_force <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom =
10,
                                    maptype = "satellite", scale = 2)
ggmap(map_final_prediction_force) + geom_polygon(data=p_2010_2,  aes(long,lat, group=group), size=1, color="black",
fill="grey") + geom_point(data=change_in_minus,  aes(lon,lat), size=0.1, color="red") + geom_path(data=mobile_county,
aes(long,lat, group=group), size=1, color="red")+ geom_point(data=change_in_plus,  aes(lon,lat), size=0.1,
color="blue")
grid.text("Final Prediction for 2020 - Force of change, Zoom 9", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_final_prediction_force.pdf")
dev.off()

#2) Change in plus
map_final_prediction_plus <- get_map(location = c(lon = mean(grid_lon_lat$lon), lat = mean(grid_lon_lat$lat)), zoom =
10,
                                maptype = "roadmap", scale = 1)
```

```
ggmap(map_final_prediction_plus) + geom_point(data=change_in_plus,  aes(lon,lat), size=2, color="blue") +
geom_polygon(data=p_2010_2,  aes(long,lat, group=group), size=1, color="black", fill="grey") +
geom_path(data=mobile_county,  aes(long,lat, group=group), size=1, color="red")
grid.text("Final Prediction for 2020 - Change in plus, Zoom 11", x=unit(0.8, "npc"), y=unit(0.50, "npc"), rot=-90)
ggsave("map_final_prediction_plus.pdf")
dev.off()

# Final sound
beep("fanfare")

# Save the Workspace
save.image("UGSM_DataPredictionFMaps.RData")

###################################################### 1. THE END
#####################################################
# THANK YOU FOR YOUR ATTENTION
#
#######################################################################################################################
```