Spatial distribution map

```
# Load required libraries
library(rgdal)  # For reading geospatial data
library(raster)
library(plyr)
library(dplyr)
library(openxlsx)
library(data.table) # For working with data tables
library(tmap)  # For creating thematic maps

# Load district shapefile
ai = readOGR("./2011_Dist/2011_Dist.shp")  # Reading district-level shapefiles
plot(ai)

# View column names of shapefile
t(names(ai))

# Load input outbreak data
df = read.xlsx("./input.xlsx")

# Aggregate outbreak data at the district level
df1 = aggregate(df[8], by = df[c("State_Name", "District")], FUN = sum, na.rm = TRUE)
colnames(df1)[1:2] = c("ST_NM", "DISTRICT")  # Rename columns to match shapefile

# Perform a left join to merge outbreak data with shapefile data
ai@data = join(data.frame(ai@data), data.frame(df1), type = "left", match = "first")
View(ai@data)

# Save merged data to CSV files
write.csv(ai@data, "ai.csv")
write.csv(df1, "districtwise_attack.csv")

# Function to classify outbreak data into categories
f = function(m) {
  if (m <= 0.0 | is.na(m)) i = 1
  else if (m >= 1 && m <= 3) i = 2
  else if (m >= 4 && m <= 6) i = 3
  else if (m >= 7 && m <= 9) i = 4
  else if (m >= 10 && m <= 12) i = 5
  else i = 6
}

# Assign labels for outbreak range
ai$lb = factor(mapply(f, ai$outbreak), levels = 1:6,
           labels = c("0", "1 - 3", "4 - 6", "7 - 9", "10 - 12", ">12"))

# Generate spatial polygon plot
spplot(ai, "lb", main = "Avian influenza outbreaks",
      col.regions = c("white", "pink2", "yellow", "orange", "cyan", "darkred"),
      cut = 5, scales = list(draw = TRUE))

# Generate intensity map
t_map =
  tm_shape(ai, unit = "mi") +
  tm_polygons(col = "lb", style = "pretty", title = "Outbreaks",
            border.alpha = 0, palette = c("lightblue", "hotpink", "green", "yellow", "orange", "red")) +
  tm_scale_bar(breaks = c(0, 5, 10, 15, 20), size = 0.5, position = c("left", "bottom")) +
  tm_compass(type = "arrow", position = c("right", "bottom")) +
  tm_layout(legend.text.size = 0.4, frame = FALSE,
          main.title = "Outbreaks of Avian influenza",
          main.title.size = 1) +
  tmap_options(check.and.fix = TRUE)

# Save the intensity map
tmap_save(tm = t_map, filename = "AI_OB_district_wise.jpeg", width = 6, dpi = 600, height = 4, units
= 'in')
```

Temporal distribution map

```
# Load required library
library(openxlsx)

# Read input data
df = read.xlsx("input.xlsx")

# Convert numeric months to factor with proper month names
df$Month <- factor(month.name[df$Month], levels = month.name)

# Aggregate outbreak data month-wise
df1 = aggregate(df[8], by = df[c("Month")], FUN = sum, na.rm = TRUE)

# Save month-wise outbreak data to CSV
write.csv(df1, "AI_monthwise_OB.csv")

# Plot month-wise outbreaks as a barplot
par(cex.axis = 0.9, las = 2)  # Adjust axis text size and orientation
options(scipen = 999)  # Disable scientific notation
barplot(height = df1$outbreak, names.arg = df1$Month, col = "#0097A7",
        ylim = c(0, 200), xlab = "Month", ylab = "Number of outbreaks",
        main = "Avian Influenza Outbreak Plot")

# Aggregate outbreak data year-wise
df2 = aggregate(df[8], by = df[c("Year")], FUN = sum, na.rm = TRUE)

# Save year-wise outbreak data to CSV
write.csv(df2, "AI_yearwise_OB.csv")

# Plot year-wise outbreaks as a barplot
barplot(height = df2$outbreak, names.arg = df2$Year, col = "#0097A7",
        ylim = c(0, 160), xlab = "Year", ylab = "Number of outbreaks",
        main = "Avian Influenza Outbreak Plot", space = 0.2)
```

Getis ord index hotspot map

```
# Required Libraries
library(openxlsx)
library(tmap)  # For visualizing spatial data distributions
library(rgdal)  # For handling spatial data
library(spdep)  # For creating spatial weights matrix objects
library(plyr)
library(BAMMtools)  # For creating percentile value

# Load dataset
df = read.xlsx("input.xlsx")
df$Long = round(df$Long, 2)  # Round longitude to 2 decimal places
df$Lat = round(df$Lat, 2)    # Round latitude to 2 decimal places

# Read shapefile
as = readOGR("./2011_Dist/2011_Dist.shp")
as$long = round(as$long, 2)
as$lat = round(as$lat, 2)

# Join outbreak data to shapefile data
df1 = join(data.frame(as@data), df, match = "first", type = "left")
df1[is.na(df1$Outbreaks), "Outbreaks"] = 0  # Replace NA values with 0
as$Outbreaks = df1$Outbreaks  # Assign outbreak values to shapefile

# Handle disjoint polygons
sf::sf_use_s2(FALSE)  # Disable spherical geometry
sacb = poly2nb(as)  # Create neighbor index list
as = as[-c(638, 639, 640, 641), ]  # Remove disjoint polygons
sacb = poly2nb(as)
sacw = nb2listw(sacb, style = "W")  # Assign weights to neighboring polygons

# Calculate Getis-Ord G index
localg = localG(as$Outbreaks, sacw)
as@data = cbind(as@data, localg_outbreak = as.numeric(localg))
write.csv(as@data, "IND_AI_Getis-Ord_Index.csv")

# Break values into categories and assign labels
breaks = c(-Inf, -1.96, 1.96, Inf)
ob = cut(as$localg_outbreak, breaks = breaks, include.lowest = TRUE,
         labels = c("Cold spot", "None", "Hot spot"))
as$class_ob = ob

# Save classification labels
write.csv(ob, "ob_new.csv")

# Generate map visualization
tm.plot = tm_shape(as, unit = "mi") +
  tm_polygons(col = "class_ob", title = "Gi value",
           palette = c("#FFFFB2", "lightblue", "red"), border.alpha = 0) +
  tm_scale_bar(breaks = c(0, 10, 20), size = 1, position = "left") +
  tm_layout(frame = FALSE, main.title = "Avian Influenza Outbreak Getis Index",
         title.size = 0.4, legend.outside = TRUE)

# Save map to file
tmap_save(tm.plot, "IND_AI_map.jpg", width = 6, height = 4, dpi = 200)

# Close any open graphical devices
while (!is.null(dev.list())) {
  dev.off()
}
```

Environmental Risk Factors identified through LDA

```
library(foreign) # to read DBF files
library(openxlsx) # for working with Excel files
library(plyr) # for data manipulation
library(data.table) # for efficient data processing
library(geosphere) # for geographical distance calculations
library(MASS) # for Linear Discriminant Analysis (LDA)

# Reading SaTScan result DBF file
df = read.dbf("./12/12.col.dbf")

# Extracting significant clusters based on p-value
df.sig = df[df$P_VALUE <= 0.05, "CLUSTER"]

# Reading GIS output DBF file
df1 = read.dbf("./12/12.gis.dbf")

# Filtering GIS data to include only significant clusters
df1.sig = df1[df1$CLUSTER %in% df.sig, ]

# Converting LOC_ID to numeric type
df1.sig$LOC_ID = as.numeric(df1.sig$LOC_ID)

# Reading Excel file with additional data
d = read.xlsx("./input.xlsx")

# Adding a new column "classify" and initializing with 1
d$classify = 1

# Renaming the first column to "LOC_ID" for consistency
colnames(d)[1] = "LOC_ID"

# Joining GIS data with the Excel data
d1 = join(data.frame(d), df1.sig, match = "first", type = "left")

# Assigning zero to "classify" for rows where CLUSTER is NA
d1[is.na(d1$CLUSTER), "classify"] = 0

# Writing the resulting dataset to a CSV file
write.csv(d1, "AI_lda.csv")

# Integrating parameter data
df = read.csv("AI_lda.csv")
pars = c("EVI", "LST", "NDVI", "LAI", "PET", "Potential evaporation rate",
        "Surface Pressure", "Specific humidity", "Rain precipitation rate",
        "Soil moisture", "Air Temperature", "Wind speed")

# Retrieving path of CSV files in the "da" folder
files = list.files(path = "da", pattern = ".csv", full.names = TRUE, recursive = TRUE)
pars = sort(pars)

# Iterating through each file to extract parameter data
for (k in 1:length(files)) {
  d = fread(files[k], header = TRUE, check.names = FALSE, data.table = FALSE)
  d = d[, c(1:7, grep("2006-01", names(d)):ncol(d))] # Filtering relevant columns

  fdf = NULL
  for (i in 1:nrow(df)) {
    y = df[i, "Year"]
    mn = df[i, "Month"]
    ym = glob2rx(paste0(y, "*", mn))
    tmp = d[, c(1:7, grep(ym, names(d)))]
    dist.ll = c()
    tmp.dist = tmp[tmp$DISTRICT == df[i, "District"], ]

    if (nrow(tmp.dist) != 0) {
      for (j in 1:nrow(tmp.dist)) {
        v = distm(c(df[i, "Long"], df[i, "Lat"]), c(tmp.dist[j, "long"], tmp.dist[j, "lat"]), fun = distHaversine)
        dist.ll = append(dist.ll, v)
      }
      dist.ll[which(dist.ll == 0)] = Inf
      ind = which(min(dist.ll) == dist.ll)
      ls.val = tmp.dist[ind, ncol(tmp.dist)]
      fdf = rbind(fdf, ls.val)
    } else {
      fdf = rbind(fdf, c(NA, NA))
    }
  }

  df = cbind(df, fdf)
  colnames(df)[ncol(df)] = pars[k]
}

# Writing the updated dataset to a CSV file
write.csv(df, "AI_LDA_parameters_NEW_.csv")

# Linear Discriminant Analysis (LDA)
df = read.xlsx("Book1.xlsx") # Created Excel file with classify & 12 parameters columns
df$Rain.precipitation.rate = df$Rain.precipitation.rate * 86400 # Converting values into millimeters

df[is.na(df)] = 0

# Generating formula
cols = names(df)
formula = paste(cols[-1], collapse = " + ", sep = "") # Remove classify column and insert + for all parameters
lda_formula = as.formula(paste("classify", "~", formula)) # Creating formula

# Performing LDA
lda_1 = lda(lda_formula, data = df)

# Extracting group means
mean_df = data.frame(lda_1$means)
mean_df = data.frame(t(mean_df)) # Transposing rows to columns

# Performing ANOVA
lda_2 = aov(lda_formula, data = df)
temp = data.frame(anova(lda_2))
```

```
# Saving results
fwrite(mean_df, "AI_groupmean_.csv", row.names = TRUE) # Saving LDA group means
fwrite(temp, "AI_LDA__anova_.csv", row.names = TRUE) # Saving ANOVA results
```

Risk assessment and estimation

```
# Dates and parameters aggregation
Sys.setenv(JAVA_HOME='C:\\Program Files\\Java\\jre1.8.0_341\\') # Setting JAVA environment

dates_df = read.csv("./DatesMonth.csv", header = TRUE, check.names = FALSE)
dates_df = dates_df[c(17:35),] # Subset for years 2014 to 2018

files = list.files(path = "da/", pattern = glob2rx("*_*.csv"), full.names = TRUE, recursive = FALSE)
pars = c("Air Temperature", "EVI", "LAI", "LST", "NDVI", "PET", "Potential evaporation rate", "Rain
precipitation rate", "Soil moisture", "Specific humidity", "Surface Pressure", "Wind speed")

d = fread(files[1], header = TRUE, check.names = FALSE, data.table = FALSE)
d = d[, 1:7] # Extract places and geocoordinates columns
yearRange = "2006-24"

for (j in 1:length(files)) {
  df = fread(files[j], header = TRUE, check.names = FALSE, data.table = FALSE)
  col_df = colnames(df)
  mnp = d

  for (i in 1:12) {
    vp = paste(dates_df[, i], sep = "", collapse = "|")
    col_ind = grep(vp, col_df)
    temp_df = if (length(col_ind) == 1) rowMeans(df[, c(col_ind, col_ind)], na.rm = TRUE)
            else if (length(col_ind) > 1) rowMeans(df[, col_ind], na.rm = TRUE)
            else {
              print(vp[i])
              next
            }
    mnp = cbind(mnp, temp_df)
  }

  colnames(mnp) = c(col_df[1:7], 1:12)
  df_ag = rowMeans(mnp[, 8:ncol(mnp)], na.rm = TRUE)
  mnp = cbind(mnp[, 1:7], df_ag)
  colnames(mnp)[8] = pars[j]

  file_name = paste("da/AI_", pars[j], yearRange, ".csv", sep = "")
  fwrite(mnp, file = file_name, col.names = TRUE, sep = ',', row.names = FALSE, nThread = 40)
}

files = list.files(path = "da/", pattern = glob2rx(paste0("AI*", yearRange, ".csv")), full.names = TRUE,
recursive = TRUE)
df = fread(files[1], header = TRUE, check.names = FALSE, data.table = FALSE, select = 1)
avg_df = data.frame(1:nrow(df))

for (j in 1:length(files)) {
  df = fread(files[j], header = TRUE, check.names = FALSE, data.table = FALSE)
  colstr = char2end(substr(beg2char(basename(files[j]), "_", 2), 1, nchar(beg2char(basename(files[j]),
"2", 1))), "_")
  df_row = data.frame(df[, ncol(df)])
  df_row = setNames(df_row, colstr)
  avg_df = cbind(avg_df, df_row)
}

avg_df = cbind(d, avg_df[, 2:ncol(avg_df)])
fwrite(avg_df, file = paste0("India_AI", yearRange, "_pars.csv"), col.names = TRUE, sep = ',',
row.names = FALSE)

##################### Risk Modelling - Average #####################

# Load required libraries
library(biomod2)  # Species distribution model
library(dismo)
library(mgcv)
library(gbm)
library(randomForest)
library(mda)
library(earth)
library(nnet)
library(rpart)
library(caret)
library(raster)
library(kernlab)
library(imputeMissings)
library(e1071)
library(psych)
library(SDMTools)
library(BIOMOD)
library(pROC)
library(ada)
library(rgdal)
library(plyr)
library(openxlsx)
library(MASS)
library(data.table)
library(Metrics)
library(ModelMetrics)
library(geosphere)

Sys.setenv(JAVA_HOME='C:\\Program Files\\Java\\jre1.8.0_341\\')

# Load India grid shapefile and associated data
ais <- readOGR("./0.4_Inida_grid/India_grid_0.4_shp.shp")
data <- fread("./India_AI2006-24_pars.csv", header = TRUE, check.names = FALSE, data.table =
FALSE)
names(data) <- gsub(pattern = " ", replacement = "_", x = names(data))  # Replace spaces with
underscores
ais@data <- data

# Convert precipitation values to millimeters
ais$Rain_precipitation_rate <- ais$Rain_precipitation_rate * 86400
```

```
# Create empty raster object
r <- raster(extent(ais))
projection(r) <- proj4string(ais)  # Set projection properties
res(r) <- 0.01745  # Set resolution

# Generate raster files for each parameter
cols <- names(ais)[8:length(names(ais))]
for (col in cols) {
  resize <- rasterize(ais, field = col, r)  # Create raster object
  filename <- paste0(col, ".tif")
  writeRaster(resize, filename, format = "GTiff", overwrite = TRUE)
}

# Read significant parameter values
pars_sigf <- read.csv("./AI_LDA__anova_.csv")
pars_sigf <- pars_sigf[pars_sigf$P <= 0.05, "pars"]  # Filter significant parameters
pars_sigf <- na.omit(pars_sigf)
pars_sigf <- gsub(pattern = "\\.", replacement = "_", x = pars_sigf)

# Create formula for modeling
formula <- paste(c(pars_sigf), collapse = "+")
formula <- as.formula(paste("pb ~", formula))

# Load presence data
files <- "./Long_lat_.csv"
DataSpecies <- read.csv(files, sep = ",", header = TRUE)
DataSpecies <- DataSpecies[!duplicated(DataSpecies), ]  # Remove duplicates
DataSpecies <- na.omit(DataSpecies)  # Remove NA values
myRespXY <- DataSpecies[c("long", "lat")]
myResp <- rep(1, nrow(myRespXY))

# Stack raster data
raster_data <- list.files(path = "./", pattern = glob2rx("*.tif$"), full.names = TRUE)
myExpl <- stack(raster_data)

# Format data for SDM
myBiomodData <- BIOMOD_FormatingData(
  resp.var = myResp,
  expl.var = myExpl,
  resp.xy = myRespXY,
  resp.name = "Species_Model",
  PA.nb.rep = 2,
  PA.nb.absences = 300,
  PA.strategy = "user"
)

# Extract presence-absence data
coor <- myBiomodData@coord
lat <- coor$lat[(length(myResp) + 1):nrow(coor)]
lon <- coor$long[(length(myResp) + 1):nrow(coor)]
latlon <- cbind(lat, lon)

presvals <- extract(myExpl, myRespXY)
backgr <- cbind(lat, lon)
absvals <- extract(myExpl, backgr[, c(2, 1)])
```

```
pb <- c(rep(1, nrow(presvals)), rep(0, nrow(absvals)))
sdmdata <- data.frame(cbind(coor, pb, rbind(presvals, absvals)))

# Prepare presence-absence data
count_one <- sum(pb == 1)
nr <- nrow(sdmdata)
cut_abs <- sdmdata[(count_one + 1):nr, ]
ss <- replicate(10, sample(nrow(cut_abs), 100, replace = TRUE))
pp <- round(rowMeans(ss), 0)
gg <- cut_abs[pp, ]
pre_abs <- rbind(sdmdata[1:count_one, ], gg)
pre_abs <- data.frame(impute(pre_abs))
pre_abs <- pre_abs[!duplicated(pre_abs), ]

# Plot presence-absence data
case <- "Case_Data.png"
png(case)
plot(myExpl$NDVI, main = "Case Data over NDVI", ylab = "Latitude (\u00b0N)", xlab = "Longitude
(\u00b0E)", sub = "o - Case Data")
points(myRespXY, col = "darkred")
dev.off()

control <- "Control_Data.png"
png(control)
plot(myExpl$NDVI, main = "Control Data over NDVI", ylab = "Latitude (\u00b0N)", xlab =
"Longitude (\u00b0E)", sub = ". - Control data")
points(backgr, col = "blue", pch = 17, cex = 0.5)
dev.off()

case_control <- "Case_Control_Data.png"
png(case_control)
plot(myExpl$NDVI, main = "Case Control Data over NDVI", ylab = "Latitude (\u00b0N)", xlab =
"Longitude (\u00b0E)", sub = "o - Case Data; . - Control data")
points(myRespXY, col = "darkred")
points(backgr, col = "blue", pch = 17, cex = 0.5)
dev.off()

# GLM Model (Logistic Regression)
m1 <- glm(formula, family="binomial", data=pre_abs)
p1 <- predict(myExpl, m1, type="response")  # Predict using raster data
summary(m1)
plot(p1)
glmresult <- extract(p1, cbind(pre_abs$lon, pre_abs$lat))  # Extract values
glkappa <- cohen.kappa(data.frame(pre_abs$pb, glmresult))  # Kappa for accuracy
kappa[1] <- round(glkappa[[2]], 3)

# ROC and AUC
glroc <- roc(pre_abs$pb, glmresult)
glroc1 <- as.numeric(glroc$auc)
roc[1] <- round(glroc1[[1]], 3)

# Threshold and confusion matrix
glvv <- optim.thresh(pre_abs$pb, glmresult)
max_thres = max(glvv[[8]])
```

```
glxx <- confusion.matrix(pre_abs$pb, glmresult, max_thres)
tss[1] <- round(TSS.Stat(glxx), 3)

# Extract confusion matrix metrics
cm_glm = as.data.frame.table(glxx)
tn_glm = cm_glm$Freq[1]
fn_glm = cm_glm$Freq[2]
fp_glm = cm_glm$Freq[3]
tp_glm = cm_glm$Freq[4]
total_glm = sum(cm_glm$Freq)

# Calculate model performance metrics
accuracy[1] = (tp_glm + tn_glm) / total_glm
error_rate[1] = (fp_glm + fn_glm) / total_glm
precision[1] = precision(pre_abs$pb, glmresult)
sensitivity[1] = sensitivity(pre_abs$pb, glmresult)
specificity[1] = specificity(pre_abs$pb, glmresult)
f1score[1] = f1Score(pre_abs$pb, glmresult)
logloss[1] = logLoss(pre_abs$pb, glmresult)
gini_coefficient[1] = gini(pre_abs$pb, glmresult)

# GAM Model
g1 <- gam(formula, family="binomial", data=pre_abs)
p2 <- predict(myExpl, g1, type="response")
plot(p2)
gamresult <- extract(p2, cbind(pre_abs$lon, pre_abs$lat))
gakappa <- cohen.kappa(data.frame(pre_abs$pb, gamresult))
kappa[2] <- round(gakappa[[2]], 3)

# ROC and AUC for GAM
garoc <- roc(pre_abs$pb, gamresult)
garoc1 <- as.numeric(garoc$auc)
roc[2] <- round(garoc1[[1]], 3)

# Threshold and confusion matrix for GAM
gavv <- optim.thresh(pre_abs$pb, gamresult)
max_thres = max(gavv[[8]])
gaxx <- confusion.matrix(pre_abs$pb, gamresult, max_thres)
tss[2] <- round(TSS.Stat(gaxx), 3)

# Extract confusion matrix metrics for GAM
cm_gam = as.data.frame.table(gaxx)
tn_gam = cm_gam$Freq[1]
fn_gam = cm_gam$Freq[2]
fp_gam = cm_gam$Freq[3]
tp_gam = cm_gam$Freq[4]
total_gam = sum(cm_gam$Freq)

# Calculate GAM model performance metrics
accuracy[2] = (tp_gam + tn_gam) / total_gam
error_rate[2] = (fp_gam + fn_gam) / total_gam
precision[2] = precision(pre_abs$pb, gamresult)
sensitivity[2] = sensitivity(pre_abs$pb, gamresult)
specificity[2] = specificity(pre_abs$pb, gamresult)
f1score[2] = f1Score(pre_abs$pb, gamresult)
```

```
logloss[2] = logLoss(pre_abs$pb, gamresult)
gini_coefficient[2] = gini(pre_abs$pb, gamresult)

# Random Forest Model
rf2 <- randomForest(model, data=na.omit(pre_abs), ntree=500, mtry=6, importance=T)
varImpPlot(rf2, sort=T)  # Plot variable importance
importance(rf2)
pr1 <- predict(myExpl, rf2)
plot(pr1)
rfresult <- extract(pr1, cbind(pre_abs$lon, pre_abs$lat))
rfkappa <- cohen.kappa(data.frame(pre_abs$pb, rfresult))
kappa[3] <- round(rfkappa[[2]], 3)

# ROC and AUC for RF
rfroc <- roc(pre_abs$pb, rfresult)
rfroc1 <- as.numeric(rfroc$auc)
roc[3] <- round(rfroc1[[1]], 3)

# Threshold and confusion matrix for RF
rfvv <- optim.thresh(pre_abs$pb, rfresult)
max_thres = max(rfvv[[8]])
rfxx <- confusion.matrix(pre_abs$pb, rfresult, max_thres)
tss[3] <- round(TSS.Stat(rfxx), 3)

# Extract confusion matrix metrics for RF
cm_rf = as.data.frame.table(rfxx)
tn_rf = cm_rf$Freq[1]
fn_rf = cm_rf$Freq[2]
fp_rf = cm_rf$Freq[3]
tp_rf = cm_rf$Freq[4]
total_rf = sum(cm_rf$Freq)

# Calculate RF model performance metrics
accuracy[3] = (tp_rf + tn_rf) / total_rf
error_rate[3] = (fp_rf + fn_rf) / total_rf
precision[3] = precision(pre_abs$pb, rfresult)
sensitivity[3] = sensitivity(pre_abs$pb, rfresult)
specificity[3] = specificity(pre_abs$pb, rfresult)
f1score[3] = f1Score(pre_abs$pb, rfresult)
logloss[3] = logLoss(pre_abs$pb, rfresult)
gini_coefficient[3] = gini(pre_abs$pb, rfresult)

# GBM Model
gbm_model = gbm.step(data=pre_abs, gbm.x = 4:ncol(pre_abs), gbm.y = 3, family="bernoulli",
tree.complexity=1, learning.rate=0.01,
            bag.fraction=0.5, n.trees=5, keep.fold.fit=T, tolerance.method="fixed", step.size=5,
n.folds=10)
predictions <- predict(myExpl, gbm_model, n.trees=gbm_model$gbm.call$best.trees,
type="response")
plot(predictions)
gbresult <- extract(predictions, cbind(pre_abs$lon, pre_abs$lat))
gbkappa <- cohen.kappa(data.frame(pre_abs$pb, gbresult))
kappa[4] <- round(gbkappa[[2]], 3)

# ROC and AUC for GBM
```

```
gbroc <- roc(pre_abs$pb, gbresult)
gbroc1 <- as.numeric(gbroc$auc)
roc[4] <- round(gbroc1[[1]], 3)

# Threshold and confusion matrix for GBM
gbvv <- optim.thresh(pre_abs$pb, gbresult)
max_thres = max(gbvv[[8]])
gbxx <- confusion.matrix(pre_abs$pb, gbresult, max_thres)
tss[4] <- round(TSS.Stat(gbxx), 3)

# Extract confusion matrix metrics for GBM
cm_gbm = as.data.frame.table(gbxx)
tn_gbm = cm_gbm$Freq[1]
fn_gbm = cm_gbm$Freq[2]
fp_gbm = cm_gbm$Freq[3]
tp_gbm = cm_gbm$Freq[4]
total_gbm = sum(cm_gbm$Freq)

# Calculate GBM model performance metrics
accuracy[4] = (tp_gbm + tn_gbm) / total_gbm
error_rate[4] = (fp_gbm + fn_gbm) / total_gbm
precision[4] = precision(pre_abs$pb, gbresult)
sensitivity[4] = sensitivity(pre_abs$pb, gbresult)
specificity[4] = specificity(pre_abs$pb, gbresult)
f1score[4] = f1Score(pre_abs$pb, gbresult)
logloss[4] = logLoss(pre_abs$pb, gbresult)
gini_coefficient[4] = gini(pre_abs$pb, gbresult)

# Neural Network Model
gg <- cbind(na.omit(pre_abs[, 3:ncol(pre_abs)]))
model <- nnet(gg, gg$pb, data=gg, size=30, linout=T, maxit=700)
nn1 <- predict(myExpl, model)
plot(nn1)
nnresult <- extract(nn1, cbind(pre_abs$lon, pre_abs$lat))
nnkappa <- cohen.kappa(data.frame(pre_abs$pb, nnresult))
kappa[5] <- round(nnkappa[[2]], 3)

# ROC and AUC for Neural Network
nnroc <- roc(pre_abs$pb, nnresult)
nnroc1 <- as.numeric(nnroc$auc)
roc[5] <- round(nnroc1[[1]], 3)

# Threshold and confusion matrix for NN
nnvv <- optim.thresh(pre_abs$pb, nnresult)
max_thres = max(nnvv[[8]])
nnxx <- confusion.matrix(pre_abs$pb, nnresult, max_thres)
tss[5] <- round(TSS.Stat(nnxx), 3)

# Extract confusion matrix metrics for NN
cm_nn = as.data.frame.table(nnxx)
tn_nn = cm_nn$Freq[1]
fn_nn = cm_nn$Freq[2]
fp_nn = cm_nn$Freq[3]
tp_nn = cm_nn$Freq[4]
total_nn = sum(cm_nn$Freq)
```

```
# Calculate NN model performance metrics
accuracy[5] = (tp_nn + tn_nn) / total_nn
error_rate[5] = (fp_nn + fn_nn) / total_nn
precision[5] = precision(pre_abs$pb, nnresult)
sensitivity[5] = sensitivity(pre_abs$pb, nnresult)
specificity[5] = specificity(pre_abs$pb, nnresult)
f1score[5] = f1Score(pre_abs$pb, nnresult)
logloss[5] = logLoss(pre_abs$pb, nnresult)
gini_coefficient[5] = gini(pre_abs$pb, nnresult)

gc()

# Additional Line for Results Summary
results <- data.frame(
  Model = c("GLM", "GAM", "RF", "GBM", "NN"),
  Accuracy = accuracy,
  ErrorRate = error_rate,
  Precision = precision,
  Sensitivity = sensitivity,
  Specificity = specificity,
  F1Score = f1score,
  LogLoss = logloss,
  GiniCoefficient = gini_coefficient,
  Kappa = kappa,
  AUC = roc,
  TSS = tss
)

print(results)

# Multinomial Logistic Regression (Earth)
mar <- earth(formula, glm=list(family=binomial), data=na.omit(pre_abs))
mar1 <- predict(myExpl, mar, type="response")
marresult <- extract(mar1, cbind(pre_abs$lon, pre_abs$lat))
markappa <- cohen.kappa(data.frame(pre_abs$pb, marresult))
kappa[6] <- round(markappa[[2]], 3)
marroc <- roc(pre_abs$pb, marresult)
marroc1 <- as.numeric(marroc$auc)
roc[6] <- round(marroc1[[1]], 3)
marvv <- optim.thresh(pre_abs$pb, marresult)
max_thres <- max(marvv[[8]])
marxx <- confusion.matrix(pre_abs$pb, marresult, max_thres)
tss[6] <- round(TSS.Stat(marxx), 3)
cm_mars <- as.data.frame.table(marxx)
tn_mars <- cm_mars$Freq[1]
fn_mars <- cm_mars$Freq[2]
fp_mars <- cm_mars$Freq[3]
tp_mars <- cm_mars$Freq[4]
total_mars <- sum(cm_mars$Freq)
auc[6] <- marroc$auc
accuracy[6] <- (tp_mars + tn_mars) / total_mars
error_rate[6] <- (fp_mars + fn_mars) / total_mars
precision[6] <- precision(pre_abs$pb, marresult)
sensitivity[6] <- sensitivity(pre_abs$pb, marresult)
```

```
specificity[6] <- specificity(pre_abs$pb, marresult)
f1score[6] <- f1Score(pre_abs$pb, marresult)
logloss[6] <- logLoss(pre_abs$pb, marresult)
gini_coefficient[6] <- gini(pre_abs$pb, marresult)
gc()

# Flexible Discriminant Analysis (FDA)
fd <- fda(as.factor(pb) ~ Air_Temperature + NDVI + Surface_Pressure, data=pre_abs)
fda1 <- predict(myExpl, fd)
fdaresult <- extract(fda1, cbind(pre_abs$lon, pre_abs$lat))
fdakappa <- cohen.kappa(data.frame(pre_abs$pb, fdaresult))
kappa[7] <- round(fdakappa[[2]], 3)
fdaroc <- roc(pre_abs$pb, fdaresult)
fdaroc1 <- as.numeric(fdaroc$auc)
roc[7] <- round(fdaroc1[[1]], 3)
fdavv <- optim.thresh(pre_abs$pb, fdaresult)
max_thres <- max(fdavv[[8]])
fdaxx <- confusion.matrix(pre_abs$pb, fdaresult, max_thres)
tss[7] <- round(TSS.Stat(fdaxx), 3)
cm_fda <- as.data.frame.table(fdaxx)
tn_fda <- cm_fda$Freq[1]
fn_fda <- cm_fda$Freq[2]
fp_fda <- cm_fda$Freq[3]
tp_fda <- cm_fda$Freq[4]
total_fda <- sum(cm_fda$Freq)
auc[7] <- fdaroc$auc
accuracy[7] <- (tp_fda + tn_fda) / total_fda
error_rate[7] <- (fp_fda + fn_fda) / total_fda
precision[7] <- precision(pre_abs$pb, fdaresult)
sensitivity[7] <- sensitivity(pre_abs$pb, fdaresult)
specificity[7] <- specificity(pre_abs$pb, fdaresult)
f1score[7] <- f1Score(pre_abs$pb, fdaresult)
logloss[7] <- logLoss(pre_abs$pb, fdaresult)
gini_coefficient[7] <- gini(pre_abs$pb, fdaresult)
gc()

# Classification Tree (rpart)
ct <- rpart(formula, data=pre_abs)
ct1 <- predict(myExpl, ct)
ctresult <- extract(ct1, cbind(pre_abs$lon, pre_abs$lat))
ctkappa <- cohen.kappa(data.frame(pre_abs$pb, ctresult))
kappa[8] <- round(ctkappa[[2]], 3)
ctroc <- roc(pre_abs$pb, ctresult)
ctroc1 <- as.numeric(ctroc$auc)
roc[8] <- round(ctroc1[[1]], 3)
ctvv <- optim.thresh(pre_abs$pb, ctresult)
max_thres <- max(ctvv[[8]])
ctxx <- confusion.matrix(pre_abs$pb, ctresult, max_thres)
tss[8] <- round(TSS.Stat(ctxx), 3)
cm_ct <- as.data.frame.table(ctxx)
tn_ct <- cm_ct$Freq[1]
fn_ct <- cm_ct$Freq[2]
fp_ct <- cm_ct$Freq[3]
tp_ct <- cm_ct$Freq[4]
total_ct <- sum(cm_ct$Freq)
```

```
auc[8] <- ctroc$auc
accuracy[8] <- (tp_ct + tn_ct) / total_ct
error_rate[8] <- (fp_ct + fn_ct) / total_ct
precision[8] <- precision(pre_abs$pb, ctresult)
sensitivity[8] <- sensitivity(pre_abs$pb, ctresult)
specificity[8] <- specificity(pre_abs$pb, ctresult)
f1score[8] <- f1Score(pre_abs$pb, ctresult)
logloss[8] <- logLoss(pre_abs$pb, ctresult)
gini_coefficient[8] <- gini(pre_abs$pb, ctresult)
gc()

# Support Vector Machine (ksvm)
svm <- ksvm(formula, data=pre_abs)
svm1 <- predict(myExpl, svm)
svmresult <- extract(svm1, cbind(pre_abs$lon, pre_abs$lat))
svkappa <- cohen.kappa(data.frame(pre_abs$pb, svmresult))
kappa[9] <- round(svkappa[[2]], 3)
svroc <- roc(pre_abs$pb, svmresult)
svroc1 <- as.numeric(svroc$auc)
roc[9] <- round(svroc1[[1]], 3)
svvv <- optim.thresh(pre_abs$pb, svmresult)
max_thres <- max(svvv[[8]])
svxx <- confusion.matrix(pre_abs$pb, svmresult, max_thres)
tss[9] <- round(TSS.Stat(svxx), 3)
cm_svm <- as.data.frame.table(svxx)
tn_svm <- cm_svm$Freq[1]
fn_svm <- cm_svm$Freq[2]
fp_svm <- cm_svm$Freq[3]
tp_svm <- cm_svm$Freq[4]
total_svm <- sum(cm_svm$Freq)
auc[9] <- svroc$auc
accuracy[9] <- (tp_svm + tn_svm) / total_svm
error_rate[9] <- (fp_svm + fn_svm) / total_svm
precision[9] <- precision(pre_abs$pb, svmresult)
sensitivity[9] <- sensitivity(pre_abs$pb, svmresult)
specificity[9] <- specificity(pre_abs$pb, svmresult)
f1score[9] <- f1Score(pre_abs$pb, svmresult)
logloss[9] <- logLoss(pre_abs$pb, svmresult)
gini_coefficient[9] <- gini(pre_abs$pb, svmresult)
gc()

# Naive Bayes (naiveBayes)
nb <- naiveBayes(formula, data=pre_abs)
nb1 <- predict(myExpl, nb, type="raw")
nbresult <- extract(nb1, cbind(pre_abs$lon, pre_abs$lat))
nbkappa <- cohen.kappa(data.frame(pre_abs$pb, nbresult))
kappa[10] <- round(nbkappa[[2]], 3)
nbroc <- roc(pre_abs$pb, nbresult)
nbroc1 <- as.numeric(nbroc$auc)
roc[10] <- round(nbroc1[[1]], 3)
nbvv <- optim.thresh(pre_abs$pb, nbresult)
max_thres <- max(nbvv[[8]])
nbxx <- confusion.matrix(pre_abs$pb, nbresult, max_thres)
tss[10] <- round(TSS.Stat(nbxx), 3)
cm_nb <- as.data.frame.table(nbxx)
```

```
tn_nb <- cm_nb$Freq[1]
fn_nb <- cm_nb$Freq[2]
fp_nb <- cm_nb$Freq[3]
tp_nb <- cm_nb$Freq[4]
total_nb <- sum(cm_nb$Freq)
auc[10] <- nbroc$auc
accuracy[10] <- (tp_nb + tn_nb) / total_nb
error_rate[10] <- (fp_nb + fn_nb) / total_nb
precision[10] <- precision(pre_abs$pb, nbresult)
sensitivity[10] <- sensitivity(pre_abs$pb, nbresult)
specificity[10] <- specificity(pre_abs$pb, nbresult)
f1score[10] <- f1Score(pre_abs$pb, nbresult)
logloss[10] <- logLoss(pre_abs$pb, nbresult)
gini_coefficient[10] <- gini(pre_abs$pb, nbresult)
gc()

# AdaBoost (ada)
ada <- ada(formula, data=pre_abs)
ada1 <- predict(myExpl, ada)
adaresult <- extract(ada1, cbind(pre_abs$lon, pre_abs$lat))
adakappa <- cohen.kappa(data.frame(pre_abs$pb, adaresult))
kappa[11] <- round(adakappa[[2]], 3)
adaroc <- roc(pre_abs$pb, adaresult)
adaroc1 <- as.numeric(adaroc$auc)
roc[11] <- round(adaroc1[[1]], 3)
adaxx <- confusion.matrix(pre_abs$pb, adaresult)
tss[11] <- round(TSS.Stat(adaxx), 3)
cm_ada <- as.data.frame.table(adaxx)
tn_ada <- cm_ada$Freq[1]
fn_ada <- cm_ada$Freq[2]
fp_ada <- cm_ada$Freq[3]
tp_ada <- cm_ada$Freq[4]
total_ada <- sum(cm_ada$Freq)
auc[11] <- adaroc$auc
accuracy[11] <- (tp_ada + tn_ada) / total_ada
error_rate[11] <- (fp_ada + fn_ada) / total_ada
precision[11] <- precision(pre_abs$pb, adaresult)
sensitivity[11] <- sensitivity(pre_abs$pb, adaresult)
specificity[11] <- specificity(pre_abs$pb, adaresult)
f1score[11] <- f1Score(pre_abs$pb, adaresult)
logloss[11] <- logLoss(pre_abs$pb, adaresult)
gini_coefficient[11] = gini(pre_abs$pb,adaresult)
gc()

final <- cbind(kappa, roc, tss, auc, accuracy, error_rate, precision, sensitivity, specificity, f1score,
logloss, gini_coefficient)
rownames(final) <- c("GLM", "GAM", "RF", "GBM", "NNET", "MARS", "FDA", "CT", "SVM",
"NB", "ADA")
fname <- paste(dir_name, "/", dir_name, "_evaluation.csv", sep = "")
write.csv(final, fname)

final_2 <- cbind(kappa, roc, tss, auc, accuracy, error_rate, precision, sensitivity, specificity, f1score,
logloss, gini_coefficient)
rownames(final_2) <- c("GLM", "GAM", "RF", "GBM", "NNET", "MARS", "FDA", "CT", "SVM",
"NB", "ADA")
```

```
fname1 <- paste(dir_name, "/", dir_name, "Metrics.csv", sep = "")
write.csv(final_2, fname1)

col5 <- colorRampPalette(c('green', '#c1ff33', 'red'))
color_levels <- 20

models_name <- c("GLM", "GAM", "RF", "GBM", "MARS", "FDA", "CT", "SVM", "NB", "ADA")
data_models <- c(p1, p2, pr1, predictions, mar1, fda1, ct1, svm1, nb1, ada1)

# Plot individual models
for (h in 1:length(models_name)) {
  plot_finame <- paste(dir_name, "/", models_name[h], ".png", sep = "")
  png(plot_finame, width = 5, height = 4, units = 'in', res = 700)
  title_plot <- paste("Disease Risk Prediction (", models_name[h], " model) - ", dir_name, sep = "")
  plot(data_models[[h]], col = col5(n = color_levels), main = list(title_plot, cex = 0.6), ylab =
expression("Latitude"~degree~N), xlab = expression("Longitude"~degree~E))
  dev.off()
}

case <- paste(dir_name, "/Cases", sep = "")
dir.create(path = case)

# Risk map with cases
for (h in 1:length(models_name)) {
  plot_finame <- paste(case, "/", models_name[h], "_1.png", sep = "")
  png(plot_finame, width = 5, height = 4, units = 'in', res = 700)
  title_plot <- paste("Disease Risk Prediction (", models_name[h], " model) - ", dir_name, sep = "")
  plot(data_models[[h]], col = col5(n = color_levels), main = list(title_plot, cex = 0.6), ylab =
expression("Latitude"~degree~N), xlab = expression("Longitude"~degree~E))
  points(myRespXY, col = "darkred", cex = 0.4)
  dev.off()
}

f_name <- paste0(dir_name, "/", dir_name, "_model_predictions.RData")
save(data_models, file = f_name)

# Model selection
modname <- load(f_name)
mod_df <- data.frame(models_name, modname[1:10])
final <- data.frame(final)

# Filtering models based on evaluation criteria
kappa_tdf <- final[which(final$kappa > 0.5),]
roc_tdf <- kappa_tdf[which(kappa_tdf$roc > 0.7),]
tss_tdf <- roc_tdf[which(roc_tdf$tss > 0.7),]
auc_tdf <- tss_tdf[which(tss_tdf$auc > 0.7),]
accuracy_tdf <- auc_tdf[which(auc_tdf$accuracy > 0.7),]
error_rate_tdf <- accuracy_tdf[which(accuracy_tdf$error_rate < 0.2),]
precision_tdf <- error_rate_tdf[which(error_rate_tdf$precision > 0.8),]
sensitivity_tdf <- precision_tdf[which(precision_tdf$sensitivity > 0.8),]
specificity_tdf <- sensitivity_tdf[which(sensitivity_tdf$specificity > 0.8),]
f1score_tdf <- specificity_tdf[which(specificity_tdf$f1score > 0.8),]
logloss_tdf <- f1score_tdf[which(f1score_tdf$logloss < 0.7),]
gini_coefficient_tdf <- logloss_tdf[which(logloss_tdf$gini_coefficient > 0.8),]
```

```
mod_df <- mod_df[which(mod_df$models_name %in% row.names(gini_coefficient_tdf)),]
mod_index <- as.integer(row.names(mod_df))

# Plot averaged model
models_name_filt <- c("GLM", "GAM", "RF", "GBM", "MARS", "FDA", "CT", "SVM", "NB",
"ADA")
data_models_filt <- c(p1, p2, pr1, predictions, mar1, fda1, ct1, svm1, nb1, ada1)

models <- stack(data_models_filt)
names(models) <- models_name_filt
fn <- paste0(dir_name, "/All_prediction.png")
png(fn, width = 5, height = 4, units = 'in', res = 700)
plot(models)
dev.off()

# Averaged model plot
models <- stack(data_models[c(3, 8)]) # select models based on mod_index
mod_mean <- mean(models)

fn <- paste0(dir_name, "/Average_prediction AI_New_1.png")
png(fn, width = 5, height = 4, units = 'in', res = 700)
plot(mod_mean, col = col5(n = color_levels), main = list(paste("Disease Risk Prediction (Average score
model) - ", dir_name, sep = ""), cex = 0.6), ylab = expression("Latitude"~degree~N), xlab =
expression("Longitude"~degree~E))
points(myRespXY, col = "darkred", cex = 0.4)
dev.off()

# ROC curve plot
e <- evaluate(pre_abs[pre_abs$pb == 1,], pre_abs[pre_abs$pb == 0,], rf2)
fn <- paste0(dir_name, "/AUC Curve_1.png")
png(fn, width = 5, height = 4, units = 'in', res = 700)
plot(e, 'ROC', cex = 0.4)
dev.off()

# Storing predicted values into CSV
taluk_df <- data.frame(s@data[, c("DISTRICT")])
models_name_pred <- c("GLM", "GAM", "RF", "GBM", "MARS", "FDA", "CT", "SVM", "NB",
"ADA", "Average Score")
data_models_pred <- c(p1, p2, pr1, predictions, mar1, fda1, ct1, svm1, nb1, ada1, mod_mean)

fname <- paste0(substr(f_name, 1, nchar(f_name) - 6), "1.csv")
dat_df <- data.frame(c(1:nrow(taluk_df)))
for (i in 1:length(data_models_pred))  {
  rfresult_all <- extract(data_models_pred[[i]], coordinates(s))
  rfresult_pre <- data.frame(round(rfresult_all, 2))
  colnames(rfresult_pre) <- paste(dir_name, models_name_pred[i], "Model")
  dat_df <- cbind(dat_df, rfresult_pre)
}
dat_df <- cbind(taluk_df, dat_df[, 2:ncol(dat_df)])
fwrite(dat_df, fname)
```

Transmission dynamics

```
library(data.table)
library(dplyr)
library(plyr)
library(R0) # To calculate the basic reproduction number
library(rgdal)

ml_r0=NULL
df1=fread("./input.csv",header=T,check.names=F,data.table = F) # Reading outbreak data
df_ndr=fread("./districts_latlong.csv",header=T,check.names=F,data.table = F) # Reading district
lat/long data
dist_id=unique(df1$District) # Extracting unique district IDs
dat_df=NULL # Empty dataframe to store R0 values

for (i in 1:length(dist_id)) {
 ml_r0=NULL # Initialize variable to store R0 for each district
 df.0=df1[df1$District==dist_id[i],] # Extract data for the current district
 df.1=aggregate(df.0[8],by = df.0[c("State","District","month")],FUN = sum,na.rm=T) # Aggregating
monthly outbreak data
 disease_rep=rep(12,12) # Replicating disease ID for 12 months
 month=c(1:12) # Assigning month numbers
 df_sm=data.frame(disease_rep,month) # Creating a data frame for disease and month
 df.1=join(data.frame(df_sm),df.1,match="first",type="left") # Merging dataframes
 df.1$outbreak[is.na(df.1$outbreak) | df.1$outbreak==Inf]=0 # Replacing NA or Inf with 0 for
outbreaks

 if(nrow(df.1)>0 & sum(df.1$outbreak,na.rm = T)!=0) {
  ob=df.1$outbreak # Extracting outbreak data
  names(ob)= df.1$month # Assigning month labels to the outbreak data
  ob=abs(ob-mean(ob))/sd(ob) # Standardizing the outbreak values
  mGT<-generation.time("gamma", c(mean(ob), sd(ob))) # Generating time series data
  ml_r0_all=NULL # Initialize list to store R0 estimates
  eg_R0=try(estimate.R(ob, mGT, begin=1, end=12, range=c(0.01,max(ob)),methods =
c("EG")),silent = T)
  if(class(eg_R0)!="try-error"){ml_r0_all=append(ml_r0_all,eg_R0$estimates$EG$R)}
  ML_R0=try(estimate.R(ob, mGT, begin=1, end=12, range=c(0.01,max(ob)),methods =
c("ML")),silent = T)
  if(class(ML_R0)!="try-error"){ml_r0_all=append(ml_r0_all,ML_R0$estimates$ML$R)}
  TD_R0=try(est.R0.TD(ob, mGT, begin=1, end=12, range=c(0.01,max(ob))),silent = T)
  if(class(TD_R0)!="try-error"){ml_r0_all=append(ml_r0_all,mean(TD_R0$R))}
  AR_R0=try(estimate.R(ob, mGT, begin=1, end=12, range=c(0.01,max(ob)),methods =
c("AR")),silent = T)
  if(class(AR_R0)!="try-error"){ml_r0_all=append(ml_r0_all,AR_R0$R)}
  SB_R0=try(est.R0.SB(ob, mGT, begin=1, end=12, range=c(0.01,max(ob))),silent = T)
  if(class(SB_R0)!="try-error"){ml_r0_all=append(ml_r0_all,max(SB_R0$R))}
  ml_r0=append(ml_r0,max(ml_r0_all,na.rm = T)) # Storing the maximum R0 value from different
methods
 } else {
  ml_r0=append(ml_r0,"-") # If no valid data, append "-"
 }

 dist_df=data.frame(district=dist_id[i],R0=ml_r0) # Creating a dataframe for each district's R0
 dat_df=rbind(dat_df,dist_df) # Appending district data to the final dataframe
}

colnames(dat_df)[1]="DISTRICT" # Renaming the first column
temp=join(data.frame(df_ndr),dat_df,match="first",type="left") # Merging district R0 data with
latitude/longitude data
```

```
write.csv(temp,"R0_district_AI.csv",row.names = F)  # Saving the final data to a CSV file

#--------------  R0 on Riskmap   -------------------#
library(openxlsx)
library(data.table)
library(dplyr)
library(raster)
library(rgdal)

# Read data
rdf <- read.csv("./R0_district_AI_dist.csv", na.strings = "")
kl1 <- readOGR("./2011_Dist/2011_Dist.shp")

# Merge R0 data with shapefile
kl1@data <- merge(kl1@data, rdf, by = "DISTRICT", all.x = TRUE)

# Color palette and levels
col5 <- colorRampPalette(c('green', '#c1ff33', 'red'))
color_levels <- 20

# Plot R0 Risk Map with predictions
jpeg("AI_R0_Risk_Final_1.jpg", width = 5, height = 4, units = 'in', res = 700)
plot(mod_mean, col = col5(n = color_levels), ylab = expression("Latitude"~degree~N), xlab =
expression("Longitude"~degree~E))
plot(kl1, add = TRUE, border = "lightgrey", useRaster = TRUE, interpolate = TRUE, lwd = 0.3)
title(main = "Risk Map for AI_2006-24 disease in India", cex.main = 0.7)

# Add R0 labels on map
text(x = as.numeric(rdf$long), y = as.numeric(rdf$lat), labels = round(rdf$R0, 2), col = "blue", cex =
0.2)

# Save plot
dev.off()
```