I: User’s Guide

cluster_automator()

Description

The *cluster_automator()* function uses point data to simplify the process of: first, applying a DBSCAN algorithm to determine spatial clustering; second, being able to visualise the spatial clustering with a plotted output; and third, determining the optimal distance bands (epsilons) for the DBSCAN, by providing the option to output a K-Nearest Neighbour (KNN) Distance plot (See Appendix). Furthermore, the user is also left with a ‘db’ object with the raw DBSCAN output, for any potential use beyond the function plot.

The function takes three mandatory arguments: the data, and the indices for the two columns containing the point coordinates, in longitude and latitude. Also, the function provides eight optional arguments, to expand the range of analysis and plot style that is possible with the function (see table).

Applying a clustering algorithm, such as DBSCAN, often requires the completion of multiple separate steps in order to successfully output a plot of point patterns in spatial data - sometimes repeatedly, to adjust different values in the data, the DBSCAN, or the plot itself. Moreover, it also often requires a fuller understanding of the specifics of projection systems and optimal epsilons, which may act as a barrier to use. Thus, the *cluster_automator()* function both provides ease of use, by combining these separate steps into a single function, correcting for common errors (see flowchart), while also providing the user with a breadth of further expansion options, to tweak specific parameters at each step in the analysis. Moreover, if the user is not satisfied with the base plot output, they are provided a raw DBSCAN object with which to conduct further analysis.

Usage

*cluster_automator(x, longcol, latcol, proj=*27700*, weightcol =* 0*, eps =* 250*, MinPts =* 4*, knn =* FALSE*, main =* "DBSCAN Output"*, xlab =* "Eastings"*, ylab =* "Northings"*)*

Arguments

Parameters	**Description**
x	Mandatory object of class data.frame. This argument represents the data containing the coordinates for point data, and optionally, a column containing the weighting value associated with entry point coordinate entry.
longcol	Mandatory numeric. The index for the column representing longitude data in WSG84 format (see example).
latcol	Mandatory numeric. The index for the column representing latitude data in WSG84 format (see example).
proj	Optional numeric. The unique coordinate projection system (epsg) for the specific area. Set to 27700 for the British National Grid as default, but can be adjusted for the specific epsg for the land area, by passing a different value in the function.
weightcol	Optional numeric. The column index for weighting (if any) of point data; useful for cases in which point data represent multiple cases e.g. if multiple fatalities recorded in a car crash incident, the analyst may wish to weight each point data to account for severity. The weight col is set to 0 as default, but the user can pass a value to indicate the column index (see example).

I: User's Guide

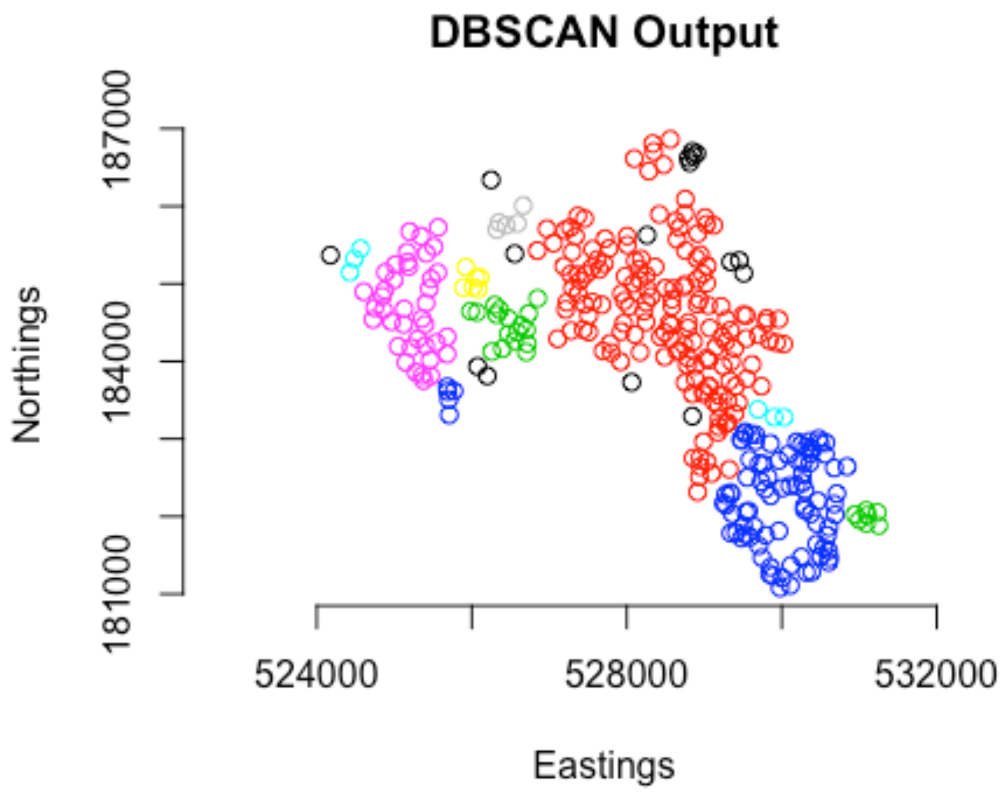
eps	Optional numeric. Represents the distance bands used in applying the DBSCAN algorithm, in metres (for the default epsg (27700)). Set to 250 as default. See: https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/how-density-based-clustering-works.htm for further clarification.
MinPts	Optional numeric. Represents the minimum number of points within distance band to classify a set of points as a 'cluster'. Set to 4 as default. See https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/how-density-based-clustering-works.htm for further information.
knn	Optional logical. Provides user ability to output a KNN distance plot, when set to TRUE. Default value set as FALSE.
main	Optional character. Provides user the ability to adjust the main title of the plot output. Set to "DBSCAN Output" as default.
xlab	Optional character. Provides user the ability to adjust the x-axis title on the plot output. Set to "Eastings" as default.
ylab	Optional character. Provides user the ability to adjust the y-axis title on the plot output. Set to "Northings" as default.

Example

The dataset used below is the Metropolitan Police street crime data for July 2020, and subsetted to only include 'violence and sexual offences' in the 'Camden' area. The data is used for illustration purposes, and many other subsets are possible. The raw data can be acquired from https://data.police.uk/data/.

```
police_data1<- read.csv("2020-07-metropolitan-street.csv")
police_data1<- police_data1[grepl('Camden',
police_data1$LSOA.name),]
crime_count_raw1<- aggregate(police_data1$Crime.ID,
by=list(police_data1$Longitude,
police_data1$Latitude,police_data1$LSOA.code,police_data1$Crime.type
), FUN=length)
crime_count_violence<-crime_count_raw[which(crime_count_raw[,4]==
"Violence and sexual offences"),]


#Tests - Violence
cluster_automator(crime_count_violence, longcol=1, latcol=2,
weightcol = 5)
```

I: User’s Guide



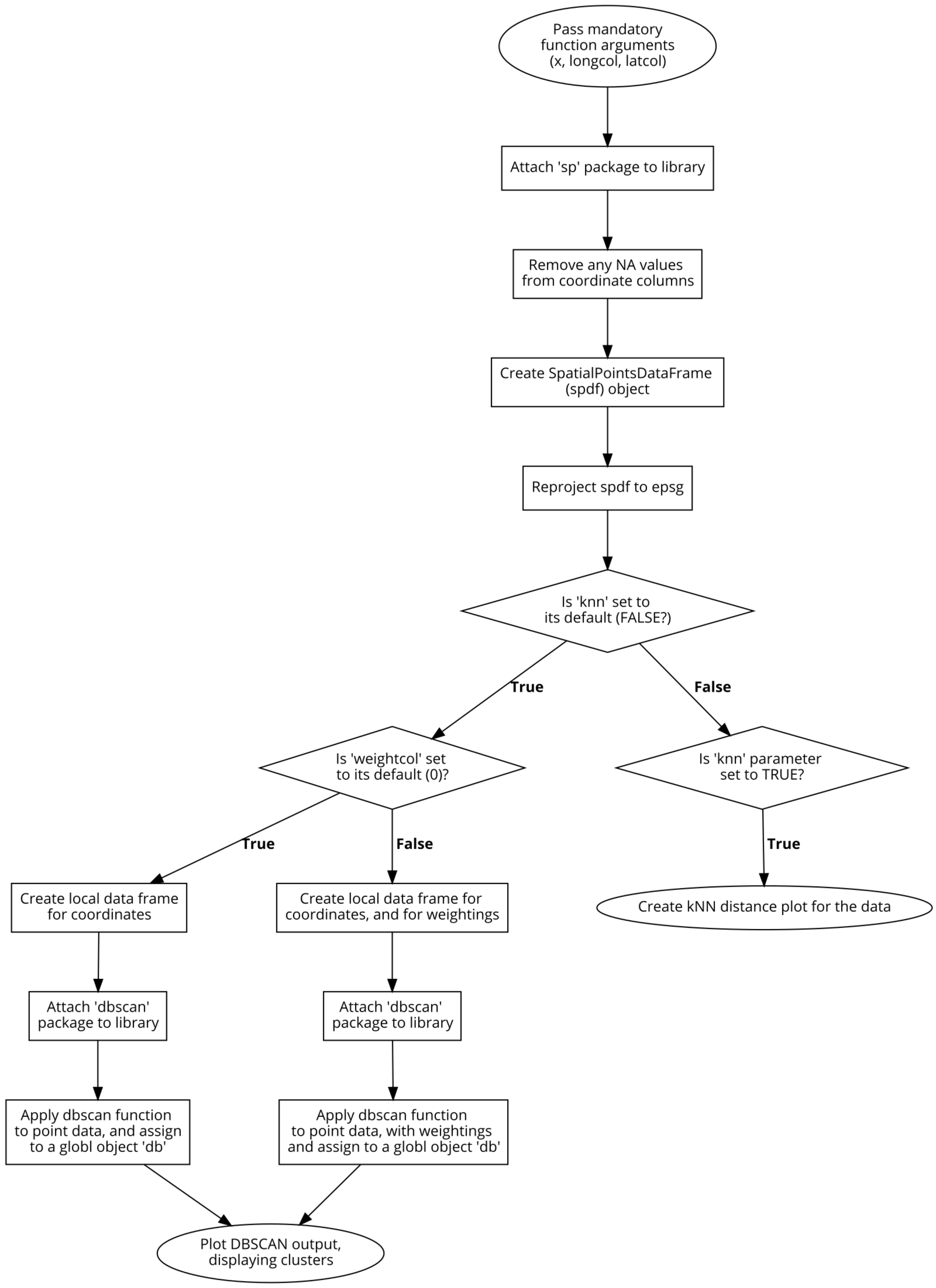
Limitations

While the function combines ease of use with depth of analysis, in applying the DBSCAN clustering algorithm, there are indeed limitations of two types. First, there are limitations in relation to the algorithm itself as a means of identifying spatial clusters, and second, there are limitations to the function itself.

With regards to DBSCAN limitations, the primary disadvantage to the analyst is the algorithm’s sensitivity to the minPts and eps parameters. The analyst must find appropriate values for these two parameters in order to form reliable conclusions about the extent of clustering present in the data, and so this introduces some subjectivity to the types of analysis conducted using DBSCAN. However, given this, I have included the option for a KNN-distance plot to reduce the level of subjectivity in determining appropriate values for both parameters.

For the function itself, some key limitations I have identified are: a) the columns containing the longitude and latitude coordinates must be side-by-side in order to select the correct columns for dbscan application. Multiple methods were attempted to overcome this limit, however, I have thus far been unsuccessful in addressing this; b) the data required for the function must be in the data.frame format, not allowing pre-existing spdf formats, and finally c) the coordinate point data are required to be in the WSG84 projection form, thus limiting the ability to use other epsg formats, including the default British National Grid formats.

II: Flow Diagram


Pass mandatory function arguments (x, longcol, latcol)
Attach 'sp' package to library
Remove any NA values from coordinate columns
Create SpatialPointsDataFrame (spdf) object
Reproject spdf to epsg
Is 'knn' set to its default (FALSE?)
True
False
Is 'weightcol' set to its default (0)?
Is 'knn' parameter set to TRUE?
True
False
True
Create local data frame for coordinates
Create local data frame for coordinates, and for weightings
Create kNN distance plot for the data
Attach 'dbscan' package to library
Attach 'dbscan' package to library
Apply dbscan function to point data, and assign to a globl object 'db'
Apply dbscan function to point data, with weightings and assign to a globl object 'db'
Plot DBSCAN output, displaying clusters

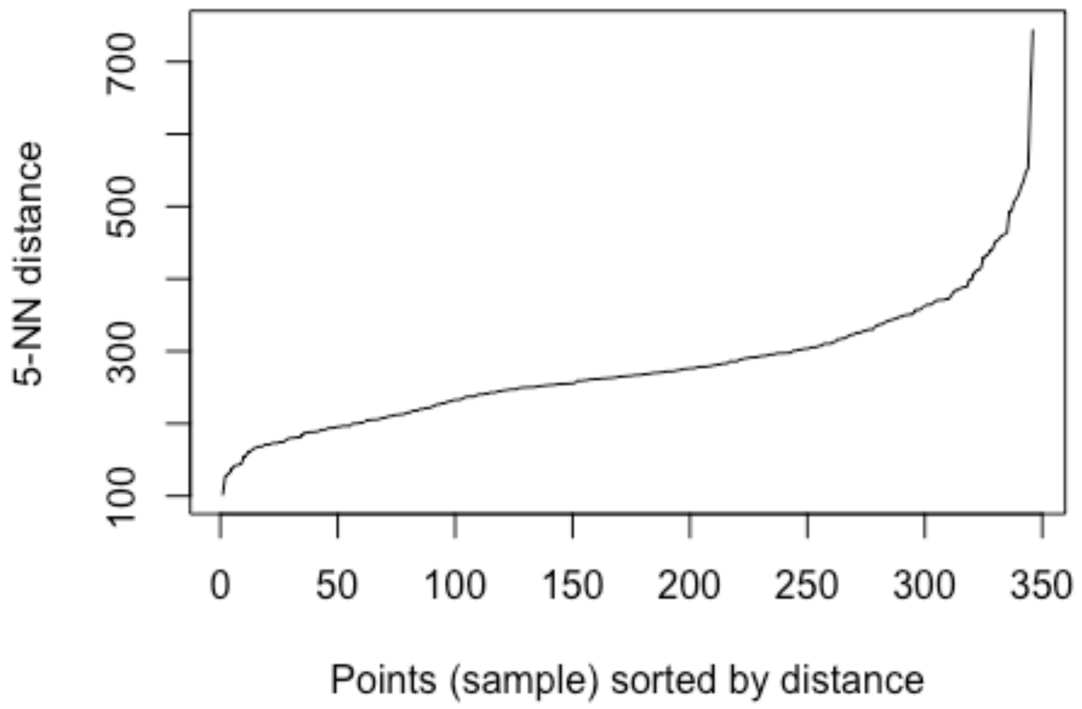
Appendix

```r
cluster_automator <- function(x, longcol, latcol, proj=27700,
weightcol = 0, eps = 250, MinPts = 4,knn = FALSE, main = "DBSCAN
Output", xlab = "Eastings", ylab = "Northings"){
  library(sp)
  x <- x[complete.cases(x[, longcol:latcol]), ]
  spdf<- SpatialPointsDataFrame(x[,longcol:latcol], x, proj4string =
CRS("+init=epsg:4326"))

  b<-spTransform(spdf, CRS(paste0("+init=epsg:",proj)))
  if(knn== FALSE){
    if(weightcol == 0){
      m <- data.frame(b@coords[,longcol:latcol])
      library(dbscan)
      db <<- dbscan(m[,1:2], eps = eps, MinPts = MinPts)
      plot(m[,1:2], col=factor(db$cluster), main = main, frame = F,
asp=T, xlab = xlab, ylab = ylab)
    } else {
      m <- data.frame(b@coords[,longcol:latcol], x[,weightcol])
      library(dbscan)
      db <<- dbscan(m[,1:2], weights = m[,3], eps = eps, MinPts =
MinPts)
      plot(m[,1:2], col=factor(db$cluster), main = main, frame = F,
asp=T, xlab = xlab, ylab = ylab)
    }
  } else if(knn == TRUE){
    m <- data.frame(b@coords[,longcol:latcol])
    kNNdistplot(m, k = MinPts+1)
  }
}

#Test 1
#Kensington Crime Data
setwd("~/Desktop/Y2 Data Analysis/Datasets/Met Pol. Crime Data")
police_data<- read.csv("2019-12-metropolitan-street.csv")
police_data<- police_data[grepl('Kensington',
police_data$LSOA.name),]
crime_count_raw<- aggregate(police_data$Crime.ID,
by=list(police_data$Longitude,
police_data$Latitude,police_data$LSOA.code,police_data$Crime.type),
FUN=length)
crime_count_asb<- crime_count_raw[which(crime_count_raw[,4]=="Anti-
social behaviour"),]

#Tests - Anti-Social Behaviour
cluster_automator(crime_count_asb, longcol=1, latcol=2)
cluster_automator(crime_count_asb, longcol=1, latcol=2, knn = TRUE)
cluster_automator(crime_count_asb, longcol=1, latcol=2, eps = 150,
MinPts = 3)
cluster_automator(crime_count_asb, longcol=1, latcol=2, weightcol =
5)
```

Appendix

```
cluster_automator(crime_count_asb, longcol=1, latcol=2, weightcol =
5, eps = 150, MinPts = 2)


#Test 2
#Kensington Crime Data
setwd("~/Desktop/Y2 Data Analysis/Datasets/Met Pol. Crime Data")
police_data<- read.csv("2019-12-metropolitan-street.csv")
police_data<- police_data[grepl('Kensington',
police_data$LSOA.name),]
crime_count_raw<- aggregate(police_data$Crime.ID,
by=list(police_data$Longitude,
police_data$Latitude,police_data$LSOA.code,police_data$Crime.type),
FUN=length)
crime_count_burg<-
crime_count_raw[which(crime_count_raw[,4]=="Burglary"),]

#Tests - Burglary
cluster_automator(crime_count_burg, longcol=1, latcol=2, weightcol =
5)
cluster_automator(crime_count_burg, longcol=1, latcol=2, knn = TRUE)
cluster_automator(crime_count_burg, longcol=1, latcol=2, eps = 200)



#Test 3
#Camden Crime Data
setwd("~/Desktop/Y2 Data Analysis/Datasets/Met Pol. Crime Data")
police_data1<- read.csv("2020-07-metropolitan-street.csv")
police_data1<- police_data1[grepl('Camden',
police_data1$LSOA.name),]
crime_count_raw1<- aggregate(police_data1$Crime.ID,
by=list(police_data1$Longitude,
police_data1$Latitude,police_data1$LSOA.code,police_data1$Crime.type
), FUN=length)
crime_count_violence<-
crime_count_raw[which(crime_count_raw[,4]=="Violence and sexual
offences"),]

#Tests - Violence
cluster_automator(crime_count_violence, longcol=1, latcol=2,
weightcol = 5)
cluster_automator(crime_count_violence, longcol=1, latcol=2, knn =
TRUE)
cluster_automator(crime_count_violence, longcol=1, latcol=2, eps =
200)
```

Appendix



5-NN distance
100
300
500
700
0
50
100
150
200
250
300
350
Points (sample) sorted by distance