Clustering High Collision Areas in Toronto

Jason Kim

# Table of Contents

[Extract, Transform, and Loading the Dataset 1](#_Toc531733718)

[Pre-processing the Data 1](#_Toc531733719)

[Data Cleaning 3](#_Toc531733720)

[Exploring the Data 5](#_Toc531733721)

[Clustering Collision Densities by Location 16](#_Toc531733722)

[Kernel Density Estimation (KDE) 17](#_Toc531733723)

[*K-means Clustering* 20](#_Toc531733724)

[Density-based Spatial Clustering and Application with Noise (DBSCAN) 24](#_Toc531733725)

[Classification 28](#_Toc531733726)

[*K-means Clustering Classification Model* 29](#_Toc531733727)

[*DBSCAN Classification* 34](#_Toc531733728)

[*Using Random Forest to Profile Clusters* 37](#_Toc531733729)

# Extract, Transform, and Loading the Dataset

The differences between this dataset and the original collisions.csv dataset are that the following ETL processes were applied to it:

* only car-on-pedestrian and car-on-cyclist collisions were kept; car-on-car, car-on-property collisions were excluded
* spatial joined in QGIS using the Toronto Neighbourhoods shapefile, which added a Neighourhood ID and Neighbourhood name field to each observation (if a collision took place within the boundaries of a neighbourhood, it was given the corresponding Neighbourhood label)
* spatial joined in QGIS using the Toronto Centrelines shapefile, which added a unique street ID (LFN\_ID) and total length in kilometres of the primary road the collision took place on as new fields to the dataset
* street name columns were merged into a single column called street1
* engineered several binary features which check whether an area has above the city’s average for that measure. E.g. businessess\_check checks whether the area has more than the average number of businesses or not.

# Pre-processing the Data

Once the different datasets from different sources are ready, they need to be joined.

library(readr)  
# Main dataset  
  
collisions <- read\_csv("D:/Google Drive/Data Analysis/136/capstone-repo/Datasets/Collisions - Processed.csv",   
 col\_types = cols(collision\_date = col\_date(format = "%m/%d/%Y")))  
  
# Datasets to be joined on Neighbourhood ID to collisions dataframe  
  
hood\_profiles <- read\_csv("D:/Google Drive/Data Analysis/136/capstone-repo/Datasets/Joined Sets (Neighbourhood-level Data)/Processed - Hood Profiles 2016.csv")

income <- read\_csv("D:/Google Drive/Data Analysis/136/capstone-repo/Datasets/Joined Sets (Neighbourhood-level Data)/Processed - Income.csv")

civics <- read\_csv("D:/Google Drive/Data Analysis/136/capstone-repo/Datasets/Joined Sets (Neighbourhood-level Data)/Processed - Civics.csv")

economics <- read\_csv("D:/Google Drive/Data Analysis/136/capstone-repo/Datasets/Joined Sets (Neighbourhood-level Data)/Processed - Economics.csv")

transportation <- read\_csv("D:/Google Drive/Data Analysis/136/capstone-repo/Datasets/Joined Sets (Neighbourhood-level Data)/Processed - Transportation.csv")

language <- read\_csv("D:/Google Drive/Data Analysis/136/capstone-repo/Datasets/Joined Sets (Neighbourhood-level Data)/Processed - Language.csv")

# join the above tables to the collisions dataset  
main.df <- merge(collisions, hood\_profiles, by.x = "AREA\_S\_CD", by.y = "Hood ID", all.x = T)  
main.df <- merge(main.df, income, by.x = "AREA\_S\_CD", by.y = "HOOD ID", all.x = T)  
main.df <- merge(main.df, language, by.x = "AREA\_S\_CD", by.y = "HOOD ID", all.x = T)  
main.df <- merge(main.df, civics, by.x = "AREA\_S\_CD", by.y = "Neighbourhood Id", all.x = T)  
main.df <- merge(main.df, economics, by.x = "AREA\_S\_CD", by.y = "Neighbourhood Id", all.x = T)  
main.df <- merge(main.df, transportation, by.x = "AREA\_S\_CD", by.y = "Neighbourhood Id", all.x = T)  
colnames(main.df)  
  
# turn four-digit integer into time   
library(caret)

summary(main.df$collision\_time)  
main.df$collision\_time <- substr(as.POSIXct(sprintf("%04.0f", main.df$collision\_time), format='%H%M'), 12, 16)  
main.df$collision\_time <- as.POSIXct(main.df$collision\_time, format = '%H:%M')  
head(main.df$collision\_time)  
  
# drop redundant columns  
  
main.df$`HOOD NAME.x` <- NULL  
main.df$`Hood Name`<- NULL  
main.df$`HOOD NAME.y`<- NULL  
main.df$Neighbourhood.x <- NULL  
main.df$Neighbourhood.y <- NULL  
main.df$Neighbourhood <- NULL  
main.df$`Total % In LIM-AT.y` <- NULL  
main.df$`Total % In LIM-AT` <- main.df$`Total % In LIM-AT.x`  
main.df$`Total % In LIM-AT.x` <- NULL  
colnames(main.df)  
  
# remove all rows containing non-pedestrian collisions   
pedestrian.df <- main.df[which(main.df$involved\_class == "PEDESTRIAN"),]  
unique(pedestrian.df$involved\_class)  
  
# drop non-pedestrian columns   
pedestrian.df <- pedestrian.df[,-c(14,24,27,30:31)]  
dim(pedestrian.df)

# Data Cleaning

# Remove variables with 50% or more missing values  
pedestrian.df <- pedestrian.df[, colMeans(is.na(pedestrian.df)) <= .5]  
dim(pedestrian.df)  
  
# Remove variables with zero or near zero variance (aka nearly all rows have same value)  
library(caret)  
nzv <- nearZeroVar(pedestrian.df)  
nzv  
  
# the below columns have near zero variance   
colnames(pedestrian.df)[12]  
colnames(pedestrian.df)[20]  
  
pedestrian.df <- pedestrian.df[,-nzv]  
dim(pedestrian.df)  
  
# how many missing values?  
sum(is.na(pedestrian.df))  
  
# where are the missing values located?   
na\_count <- sapply(pedestrian.df, function(x)  
 sum(length(which(is.na(x)))))  
na\_count <- data.frame(na\_count)  
print(na\_count)  
  
# there are 107 rows with no identifiable neighbourhood ID so these can be removed  
pedestrian.df <- pedestrian.df[-which(is.na(pedestrian.df$AREA\_S\_CD)),]  
  
# px isn't used in our analysis since its a unique id for joining to some other table  
# streets with unknown LFN\_IDs mean no length could be calculated for that street; since street length is important in our analysis to measure collision density, these unknown streets should be removed  
# involved\_age and light are also important variables we'd like to correlate so we can remove records with NAs for these   
  
dim(pedestrian.df)  
  
# prior to cleaning, there are 16665 rows, 84 features  
pedestrian.df <- pedestrian.df[,-5]  
pedestrian.df <- pedestrian.df[-which(is.na(pedestrian.df$LFN\_ID)),]  
pedestrian.df <- pedestrian.df[-which(is.na(pedestrian.df$involved\_age)),]  
pedestrian.df <- pedestrian.df[-which(is.na(pedestrian.df$light)),]  
dim(pedestrian.df)  
  
#for street\_2 and street\_type 2 -- many times when a collision is reported, only the street the collision took place on is reported, not the intersecting street. street\_type\_2 contains the type of street the intersecting street is which is not useful. We will keep street\_2 since it could be useful for human understanding where we tend to think of streets in terms of intersections not GPS coordinates   
  
pedestrian.df <- pedestrian.df[,-9]  
  
#Remove all records where collisions didn't result in any injury   
dim(pedestrian.df)  
pedestrian.df <- pedestrian.df[!pedestrian.df$involved\_injury\_class == "NONE",]  
dim(pedestrian.df)  
  
# missing values in cleaned data set  
na\_count <- sapply(pedestrian.df, function(x)  
 sum(length(which(is.na(x)))))  
na\_count <- data.frame(na\_count)  
na\_count  
  
# location\_desc, intital\_dir, pedestrian\_action, pedestrian\_collision\_type  
# all 4 of these features are qualitative, categorical variables that further describe the collision so they represent either truly unknown or non-applicable situations   
  
unique(pedestrian.df$location\_desc)  
unique(pedestrian.df$initial\_dir)  
unique(pedestrian.df$pedestrian\_action)  
unique(pedestrian.df$pedestrian\_collision\_type)

# Exploring the Data

Prior to modelling, we should obtain an untuitive understanding of the problem of collisions in Toronto.

# Missing Values  
sum(is.na(pedestrian.df))

## [1] 4060

na\_count

## na\_count  
## street\_2 1468  
## location\_desc 41  
## initial\_dir 1331  
## pedestrian\_action 596  
## pedestrian\_collision\_type 624

# 4060 missing values, all of them categorical variables  
  
# Summary of the dataset   
str(pedestrian.df)

# The dataset contains many features that are related to each other so Principal Component Analysis (PCA) will help reduce the dimensionality greatly   
  
# Let's map these collisions to get an intuition for the problem the city is facing  
# install dev build of ggmap library  
#if(!requireNamespace("devtools")) install.packages("devtools")  
#devtools::install\_github("dkahle/ggmap", ref = "tidyup", force = T)  
  
library(maptools)

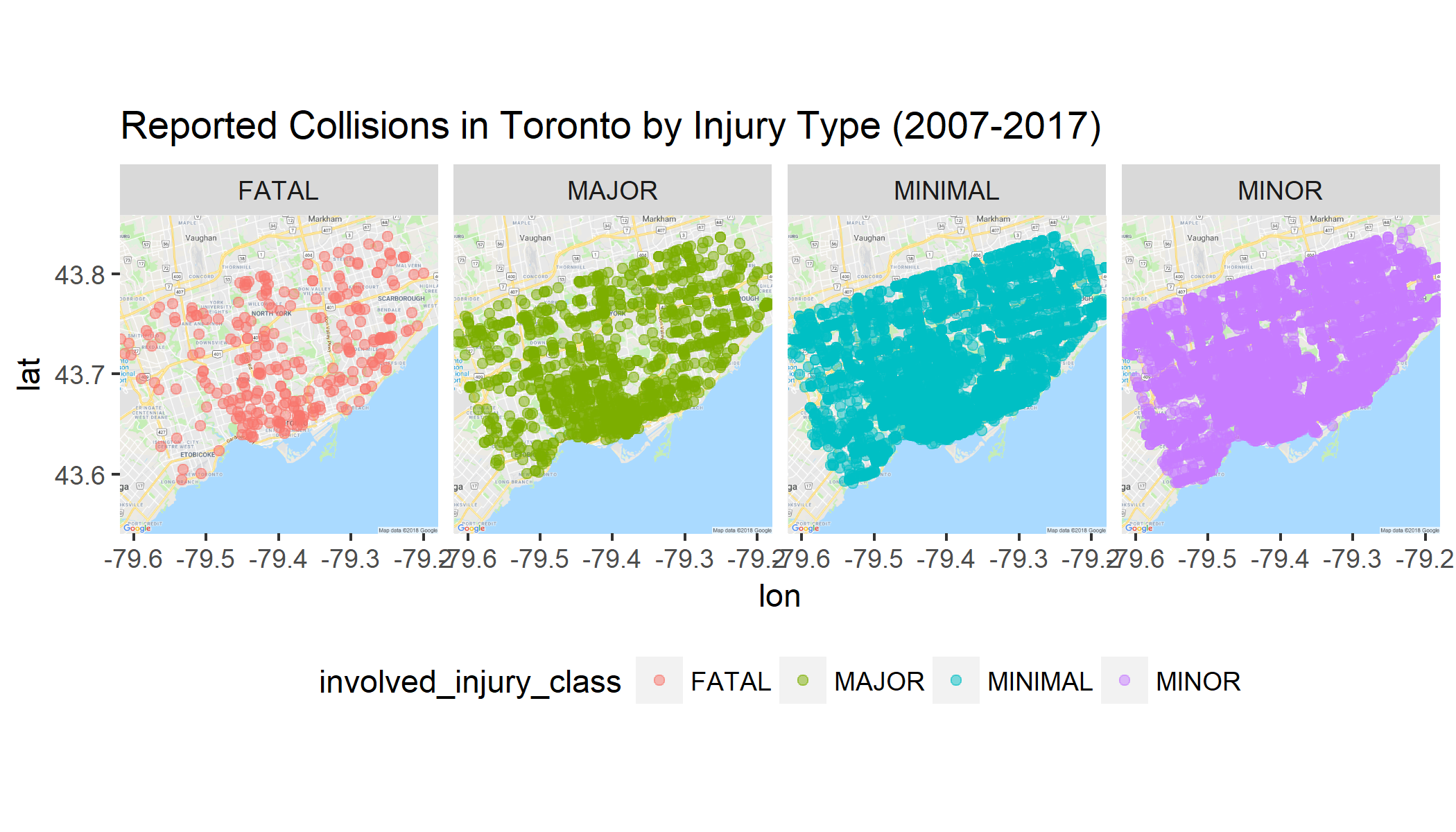
library(ggmap)

library(rgeos)

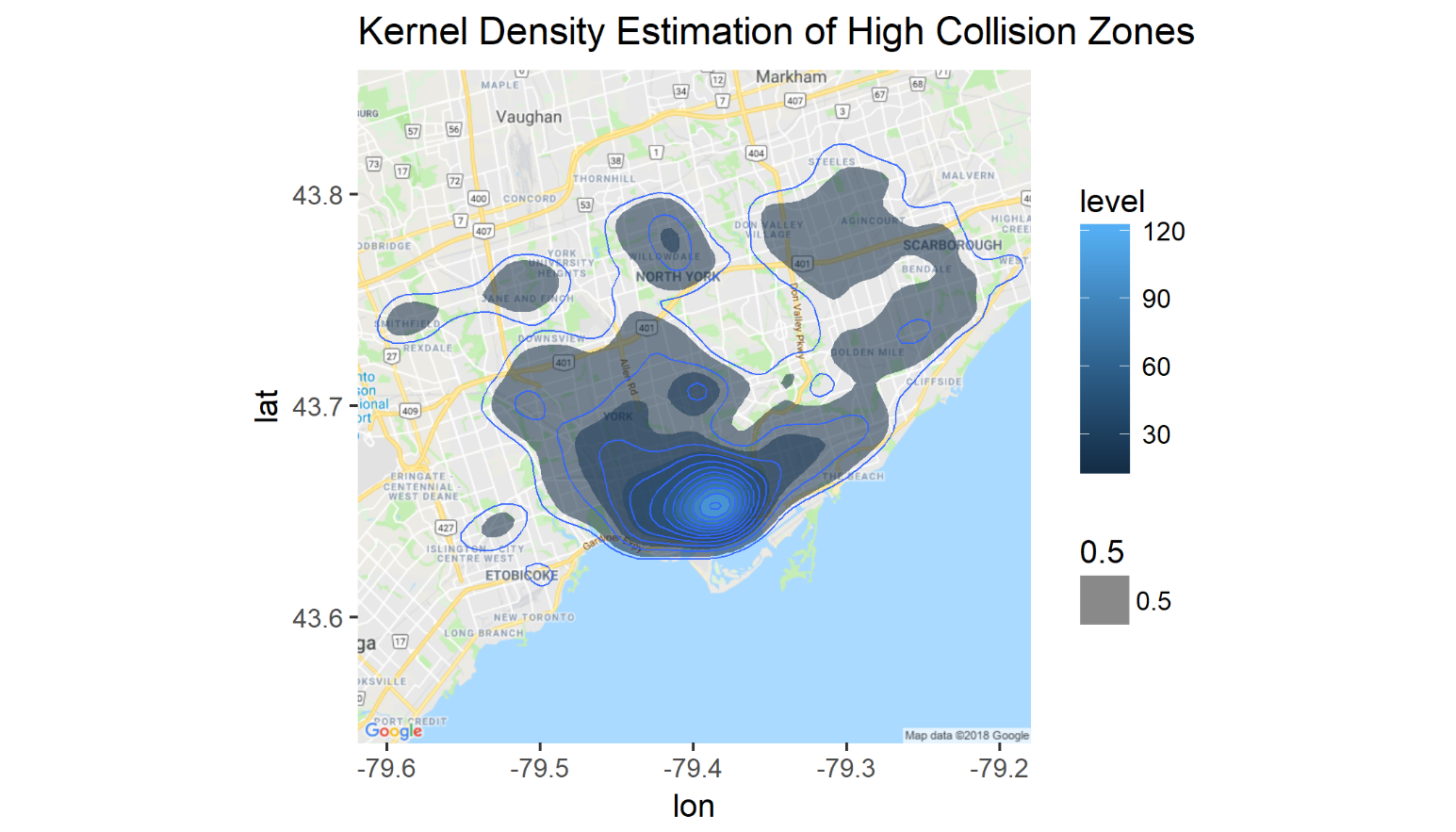
register\_google(key = "AIzaSyBwXArBPS6-g3f2-rzWXJyz0NhcK5I5eUc")  
toronto\_map <- ggmap(get\_googlemap(center = c(-79.4, 43.7), zoom = 11, scale = 1, maptype = "roadmap"))

## Source : https://maps.googleapis.com/maps/api/staticmap?center=43.7,-79.4&zoom=11&size=640x640&scale=1&maptype=roadmap&key=xxx-g3f2-rzWXJyz0NhcK5I5eUc

toronto\_map + geom\_point(aes(x = longitude, y = latitude, color = involved\_injury\_class), data = pedestrian.df, alpha = 0.5, size = 1.5) + theme(legend.position="bottom") + facet\_grid(~ involved\_injury\_class) + labs(title = "Reported Collisions in Toronto by Injury Type (2007-2017)")



# Hard to tell if there are any particular areas that are more prone to collisions which we address below. What we do see is that fatal collisions are rare events  
  
# Use Kernal Density Estimation (KDE) to show the density of collisions in Toronto  
toronto\_map + stat\_density2d(aes(x = longitude, y = latitude, fill = ..level.., alpha = 0.5), data = pedestrian.df, size = 0.1, bins = 10, geom = "polygon") + geom\_density2d(data = pedestrian.df, aes(x = longitude, y = latitude), size = 0.3) + labs(title = "Kernel Density Estimation of High Collision Zones")



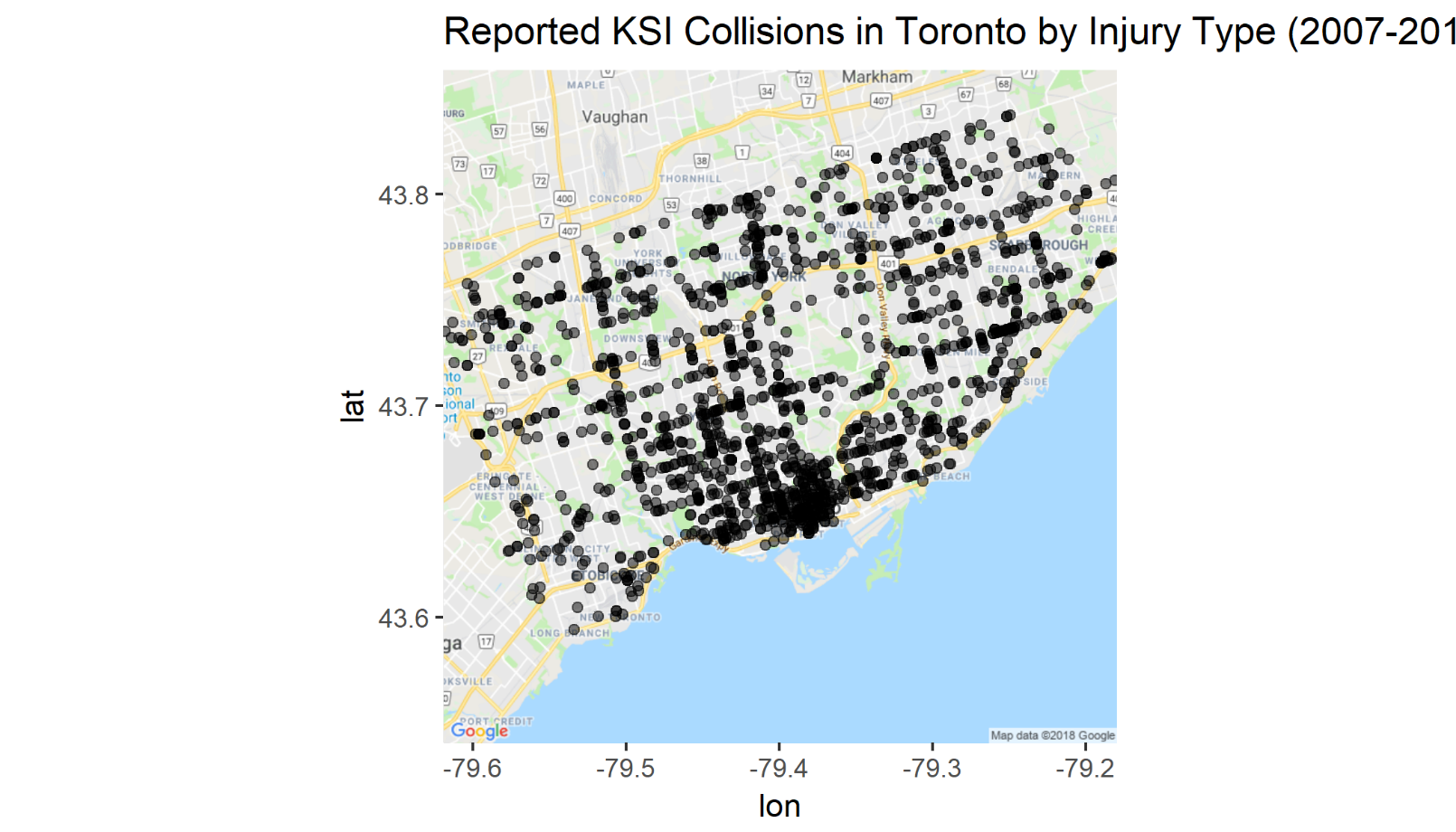
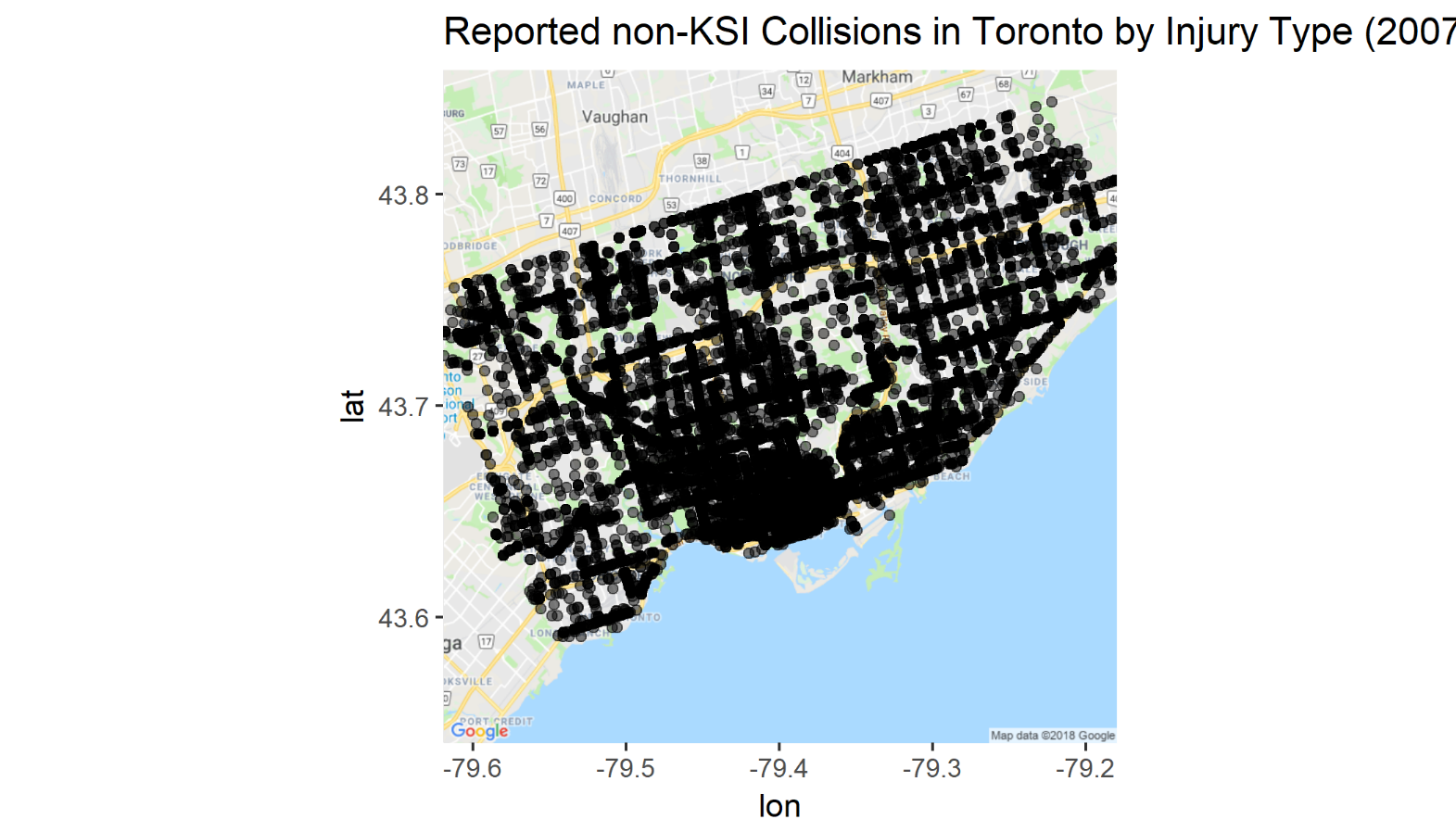
# this does a much better job of showing high colliaion areas   
  
# perhaps a trend can be observed if we differentiate between collisions resulting in death or seriously injury vs. non-KSI collisions so we look at that below  
  
# subset the Killed or Seriously Injured incidents and non-KSIs  
ksi\_df <- pedestrian.df[pedestrian.df$involved\_injury\_class == "FATAL" | pedestrian.df$involved\_injury\_class == "MAJOR",]  
dim(ksi\_df)

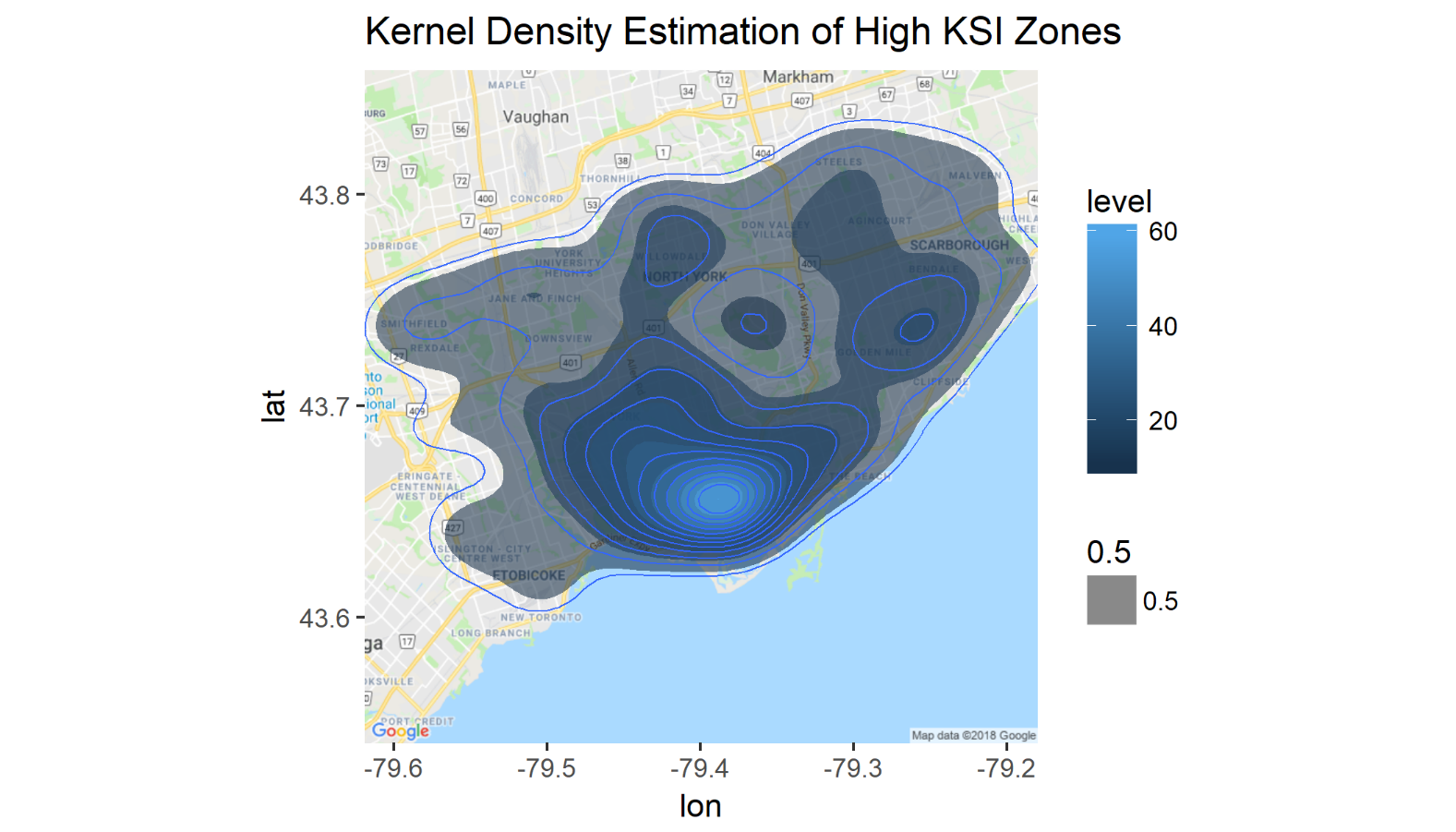
## [1] 1606 87

nonksi\_df <- pedestrian.df[-(pedestrian.df$involved\_injury\_class == "FATAL" | pedestrian.df$involved\_injury\_class == "MAJOR"),]  
dim(nonksi\_df)

## [1] 15532 87

# is there a pattern in where pedestrians were killed or seriously injured?  
toronto\_map + geom\_point(aes(x = longitude, y = latitude), data = ksi\_df, alpha = 0.5, size = 1.5) + theme(legend.position="bottom") + labs(title = "Reported KSI Collisions in Toronto by Injury Type (2007-2017)")

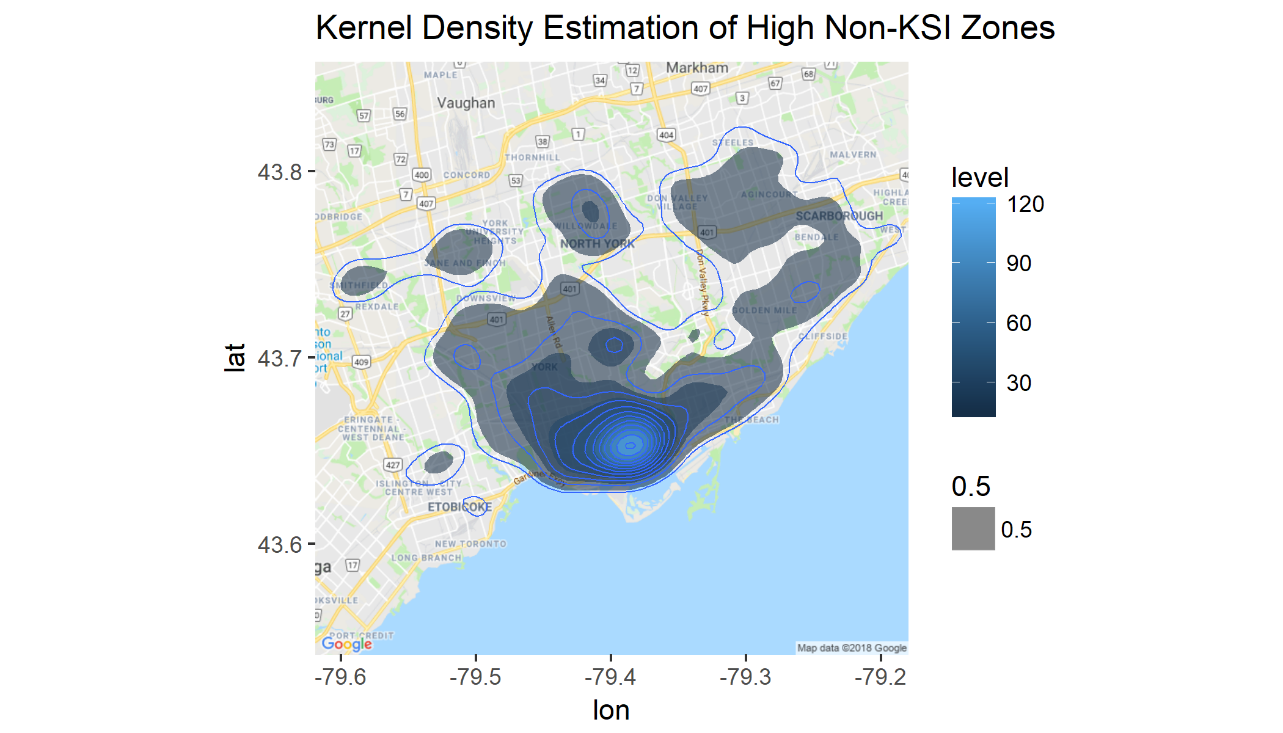
toronto\_map + geom\_point(aes(x = longitude, y = latitude), data = nonksi\_df, alpha = 0.5, size = 1.5) + theme(legend.position="bottom") + labs(title = "Reported non-KSI Collisions in Toronto by Injury Type (2007-2017)")

# KSIs seem to roughly correspond to areas that have high collisions in general.. lets use KDE to see the densities better to compare  
  
# Use Kernal Density Estimation (KDE) to show the density of KSI vs non-KSI collisions in Toronto  
toronto\_map + stat\_density2d(aes(x = longitude, y = latitude, fill = ..level.., alpha = 0.5), data = ksi\_df, size = 0.1, bins = 10, geom = "polygon") + geom\_density2d(data = ksi\_df, aes(x = longitude, y = latitude), size = 0.3) + labs(title = "Kernel Density Estimation of High KSI Zones")

# Use Kernal Density Estimation (KDE) to show the density of non-KSI collisions in Toronto  
toronto\_map + stat\_density2d(aes(x = longitude, y = latitude, fill = ..level.., alpha = 0.5), data = nonksi\_df, size = 0.1, bins = 10, geom = "polygon") + geom\_density2d(data = nonksi\_df, aes(x = longitude, y = latitude), size = 0.3) + labs(title = "Kernel Density Estimation of High Non-KSI Zones")

## Warning: Removed 130 rows containing non-finite values (stat\_density2d).

## Warning: Removed 130 rows containing non-finite values (stat\_density2d).



# The non-KSIs seem to be more diffuse when compared to KSI collisions, but both seem to occur with more frequency in the same locations.. the shape of the kernal densities look very similar. Just in case I want to study this further, I will engineer a new feature called ksi\_check where "1" means the collision is a KSI, "0" not.   
  
pedestrian.df$ksi\_check <- ifelse(pedestrian.df$involved\_injury\_class == "FATAL" | pedestrian.df$involved\_injury\_class == "MAJOR", 1, 0)  
prop.table(table(pedestrian.df$ksi\_check))

##   
## 0 1   
## 0.8966072 0.1033928

pedestrian.df$ksi\_check <- as.factor(pedestrian.df$ksi\_check)  
  
# major imbalance between non-ksi vs. ksi collisions (9:1 ratio)  
  
# pair-wise correlations of numerical variables   
pedestrian.df$AREA\_S\_CD <- as.character(pedestrian.df$AREA\_S\_CD)  
pedestrian.df$collision\_id <- as.character(pedestrian.df$collision\_id)  
pedestrian.df$LFN\_ID <- as.character(pedestrian.df$LFN\_ID)  
pedestrian.df$child\_check <- as.factor(pedestrian.df$child\_check)  
pedestrian.df$senior\_check <- as.factor(pedestrian.df$senior\_check)  
pedestrian.df$minority\_check <- as.factor(pedestrian.df$minority\_check)  
pedestrian.df$immigrants\_check <- as.factor(pedestrian.df$immigrants\_check)  
pedestrian.df$commute\_car\_check <- as.factor(pedestrian.df$commute\_car\_check)  
pedestrian.df$businesses\_check <- as.factor(pedestrian.df$businesses\_check)  
pedestrian.df$childcare\_check <- as.factor(pedestrian.df$childcare\_check)  
pedestrian.df$homeprice\_check <- as.factor(pedestrian.df$homeprice\_check)  
pedestrian.df$localemployment\_check <- as.factor(pedestrian.df$localemployment\_check)  
pedestrian.df$socialasst\_check <- as.factor(pedestrian.df$socialasst\_check)  
pedestrian.df$ttc\_check <- as.factor(pedestrian.df$ttc\_check)  
pedestrian.df$road\_km\_check <- as.factor(pedestrian.df$road\_km\_check)  
pedestrian.df$road\_vol\_check <- as.factor(pedestrian.df$road\_vol\_check)  
pedestrian.df <- unique(pedestrian.df)  
  
library(mlbench)  
library(caret)  
library(corrplot)

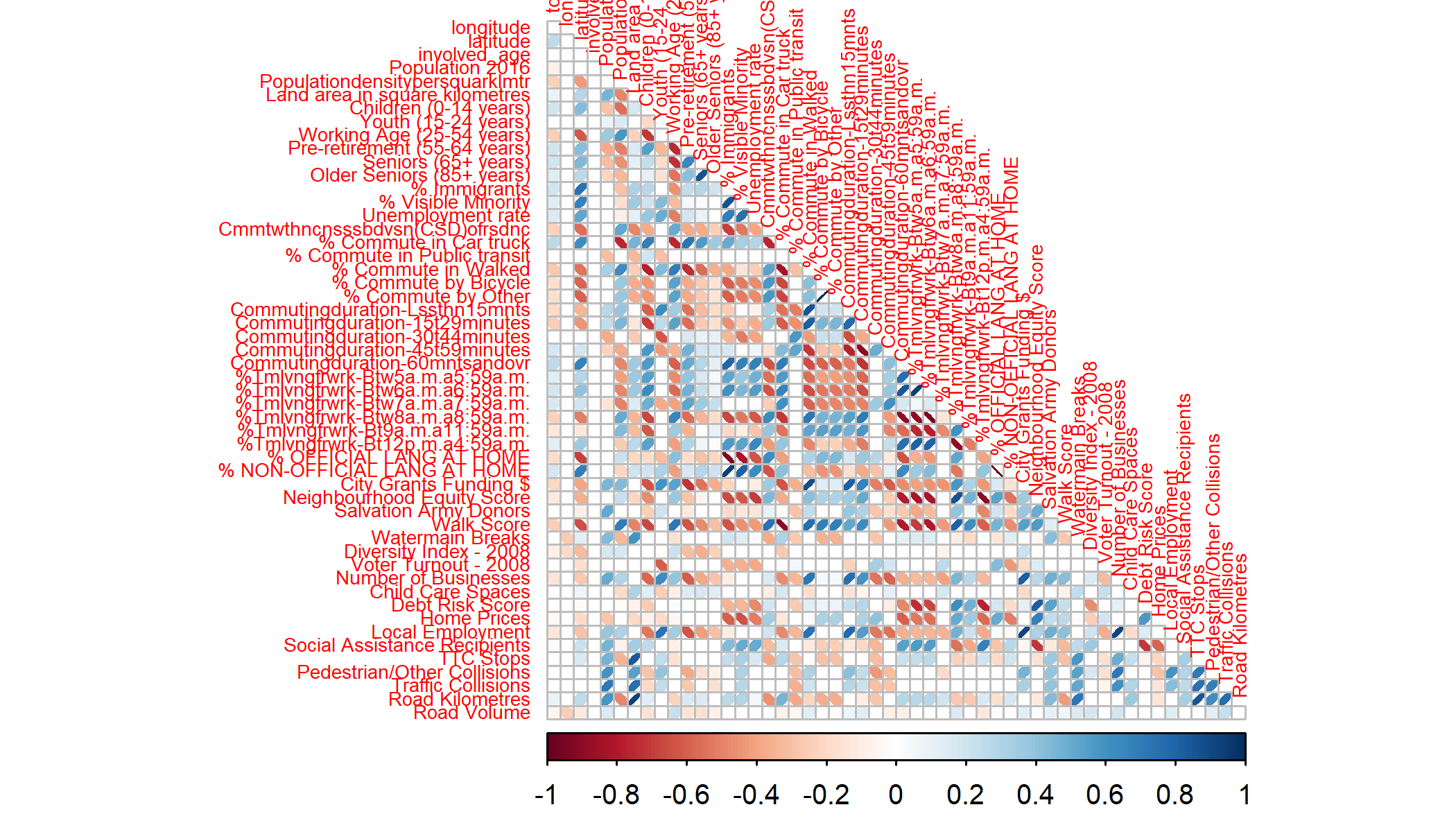
cor\_pedestrian.df <- cor(Filter(is.numeric, pedestrian.df))  
pedestrian.df\_num <- Filter(is.numeric, pedestrian.df)  
  
# looking for very strong pair-wise correlation aka colinearity   
# find attributes that are highly corrected i.e. >|0.9| (candidates for removal due to pair-wise correlations)  
highlyCorrelated <- findCorrelation(cor\_pedestrian.df, cutoff=0.9, verbose = T)

## Compare row 31 and column 29 with corr 0.938   
## Means: 0.434 vs 0.285 so flagging column 31   
## Compare row 29 and column 28 with corr 0.938   
## Means: 0.398 vs 0.28 so flagging column 29   
## Compare row 14 and column 35 with corr 0.935   
## Means: 0.357 vs 0.276 so flagging column 14   
## Compare row 36 and column 47 with corr 0.912   
## Means: 0.357 vs 0.272 so flagging column 36   
## Compare row 43 and column 47 with corr 0.918   
## Means: 0.333 vs 0.269 so flagging column 43   
## Compare row 21 and column 22 with corr 1   
## Means: 0.325 vs 0.266 so flagging column 21   
## Compare row 35 and column 34 with corr 1   
## Means: 0.294 vs 0.264 so flagging column 35   
## Compare row 7 and column 52 with corr 0.923   
## Means: 0.275 vs 0.263 so flagging column 7   
## All correlations <= 0.9

# print names of highly correlated attributes   
highlyCorrelated\_names <- colnames(cor\_pedestrian.df)[highlyCorrelated]  
highlyCorrelated\_names

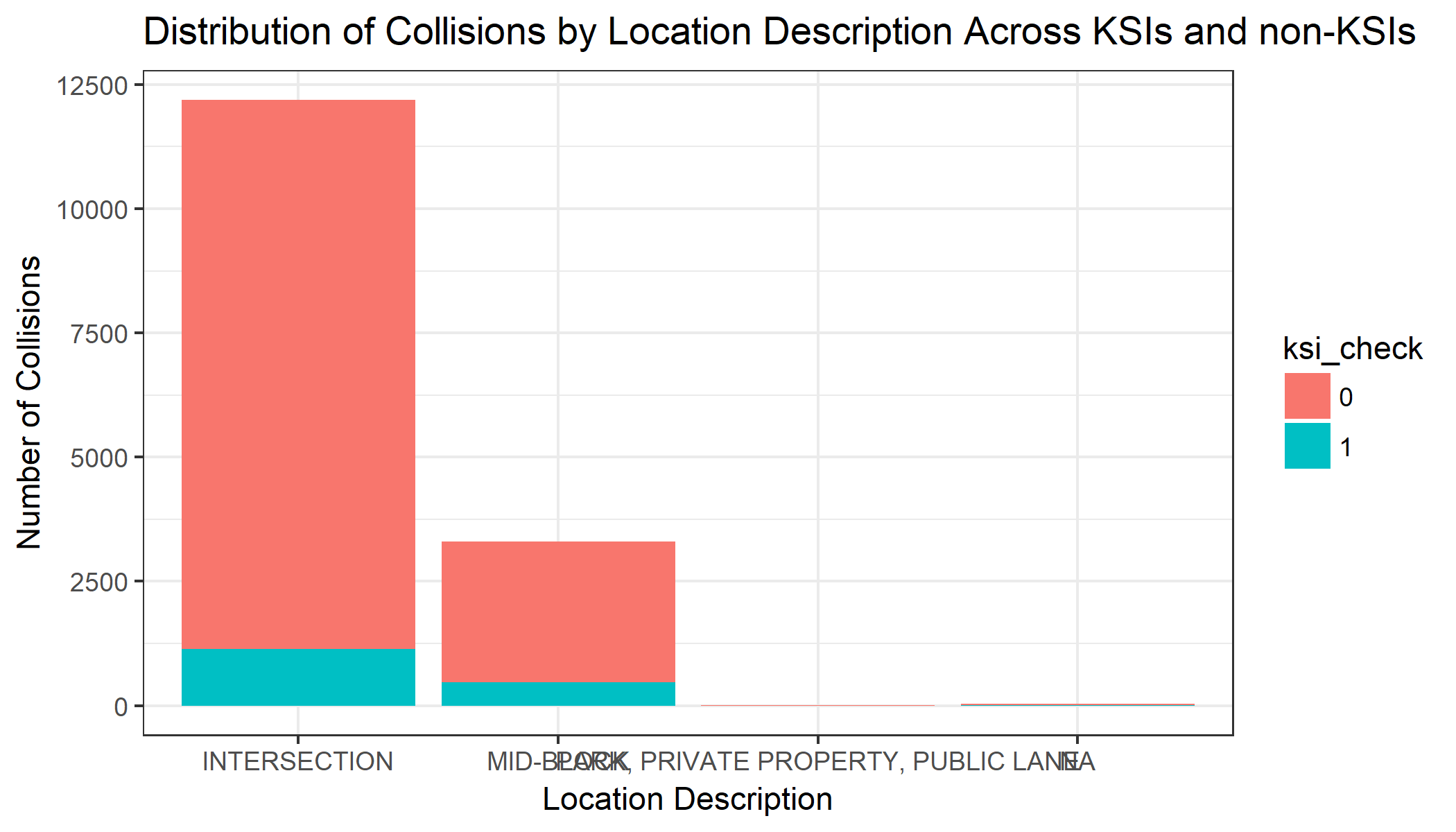
## [1] "% Time leaving for work - Between 8 a.m. and 8:59 a.m."  
## [2] "% Time leaving for work - Between 6 a.m. and 6:59 a.m."  
## [3] "% Immigrants"   
## [4] "City Grants Funding $"   
## [5] "Number of Businesses"   
## [6] "% Commute by Bicycle"   
## [7] "% NON-OFFICIAL LANG AT HOME"   
## [8] "Land area in square kilometres"

# none of these variables having very high correlation seem to matter much for our analysis so they are to be kept in for now.. PCA will combine these attributes anyway   
  
# Function to calc p values in correlation matrix  
cor.mtest <- function(mat, ...) {  
 mat <- as.matrix(mat)  
 n <- ncol(mat)  
 p.mat<- matrix(NA, n, n)  
 diag(p.mat) <- 0  
 for (i in 1:(n - 1)) {  
 for (j in (i + 1):n) {  
 tmp <- cor.test(mat[, i], mat[, j], ...)  
 p.mat[i, j] <- p.mat[j, i] <- tmp$p.value  
 }  
 }  
 colnames(p.mat) <- rownames(p.mat) <- colnames(mat)  
 p.mat  
}  
  
p.mat <- cor.mtest(cor\_pedestrian.df)  
colnames(cor\_pedestrian.df) <- abbreviate(colnames(cor\_pedestrian.df), minlength=30)  
rownames(cor\_pedestrian.df) <- abbreviate(rownames(cor\_pedestrian.df), minlength=30)  
  
corrplot(cor\_pedestrian.df, method = "ellipse", type = "lower", diag = F, insig = "blank", sig.level = 0.05, p.mat = p.mat, tl.cex = 0.55)

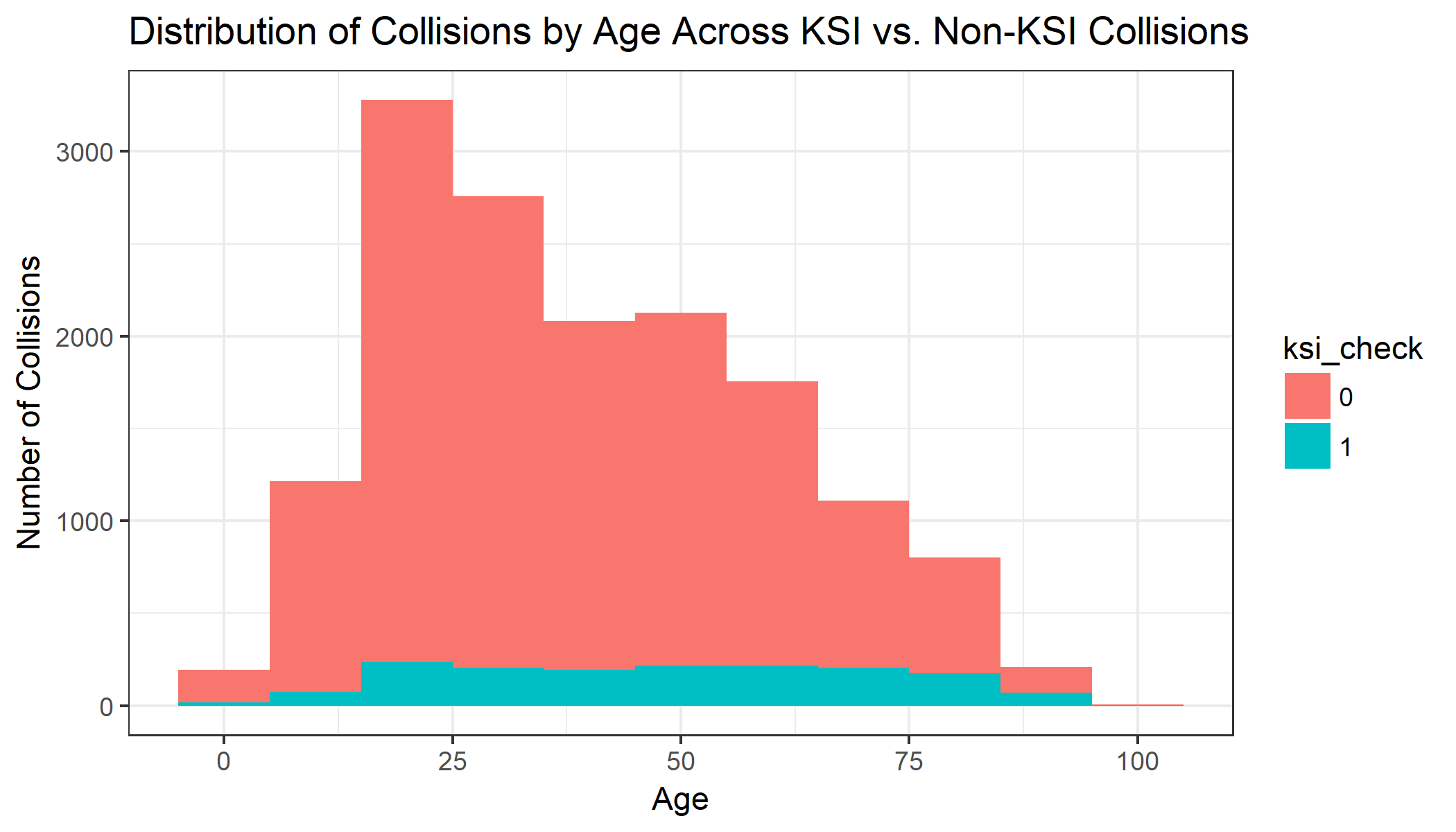


# there are way too many statistically significant correlations between variables so need to reduce dimensions further to make any sense of it

|  |  |
| --- | --- |
| Neighbourhood | Collisions |
| Waterfront Communities-The Island (77) | 608 |
| Bay Street Corridor (76) | 567 |
| Church-Yonge Corridor (75) | 367 |
| Downsview-Roding-CFB (26) | 286 |
| Islington-City Centre West (14) | 286 |
| West Humber-Clairville (1) | 280 |
| Kensington-Chinatown (78) | 272 |
| Annex (95) | 253 |
| Moss Park (73) | 241 |
| York University Heights (27) | 237 |
| Woburn (137) | 236 |
| South Riverdale (70) | 232 |
| Niagara (82) | 226 |
| Weston (113) | 211 |
| Trinity-Bellwoods (81) | 199 |
| Newtonbrook West (36) | 196 |
| Dovercourt-Wallace Emerson-Junction (93) | 195 |
| High Park-Swansea (87) | 187 |
| South Parkdale (85) | 182 |
| Agincourt South-Malvern West (128) | 176 |
|  |  |
| Street | Collisions |
| YONGE ST | 555 |
| DUNDAS ST W | 371 |
| BATHURST ST | 366 |
| BLOOR ST W | 326 |
| EGLINTON AVE E | 319 |
| JANE ST | 293 |
| QUEEN ST W | 268 |
| FINCH AVE W | 266 |
| SHEPPARD AVE E | 254 |
| LAWRENCE AVE E | 245 |
| DUFFERIN ST | 240 |
| EGLINTON AVE W | 232 |
| DANFORTH AVE | 219 |
| FINCH AVE E | 212 |
| VICTORIA PARK AVE | 208 |
| KEELE ST | 203 |
| KING ST W | 192 |
| LAWRENCE AVE W | 185 |
| ST CLAIR AVE W | 184 |
| KINGSTON RD | 182 |
|  |  |
| Road Type | Collisions |
| MAJOR ARTERIAL | 10318 |
| MINOR ARTERIAL | 2645 |
| COLLECTOR | 1294 |
| LOCAL | 1276 |



|  |  |
| --- | --- |
| Location | Collisions |
| INTERSECTION | 12186 |
| MID-BLOCK | 3304 |
| NA | 41 |
| PARK, PRIVATE PROPERTY, PUBLIC LANE | 2 |
|  |  |
| Road Type | Collisions |
| MAJOR ARTERIAL | 10318 |
| MINOR ARTERIAL | 2645 |
| COLLECTOR | 1294 |
| LOCAL | 1276 |
|  |  |
| Light Condition | Collisions |
| DAYLIGHT | 9685 |
| DARK | 5233 |
| DUSK | 391 |
| DAWN | 222 |
| OTHER | 2 |
|  |  |
| Visibility Condition | Collisions |
| CLEAR | 12311 |
| RAIN | 2563 |
| SNOW | 457 |
| OTHER | 72 |
| FREEZING RAIN | 41 |
| FOG, MIST, SMOKE, DUST | 40 |
| DRIFTING SNOW | 29 |
| STRONG WIND | 20 |



# Clustering Collision Densities by Location

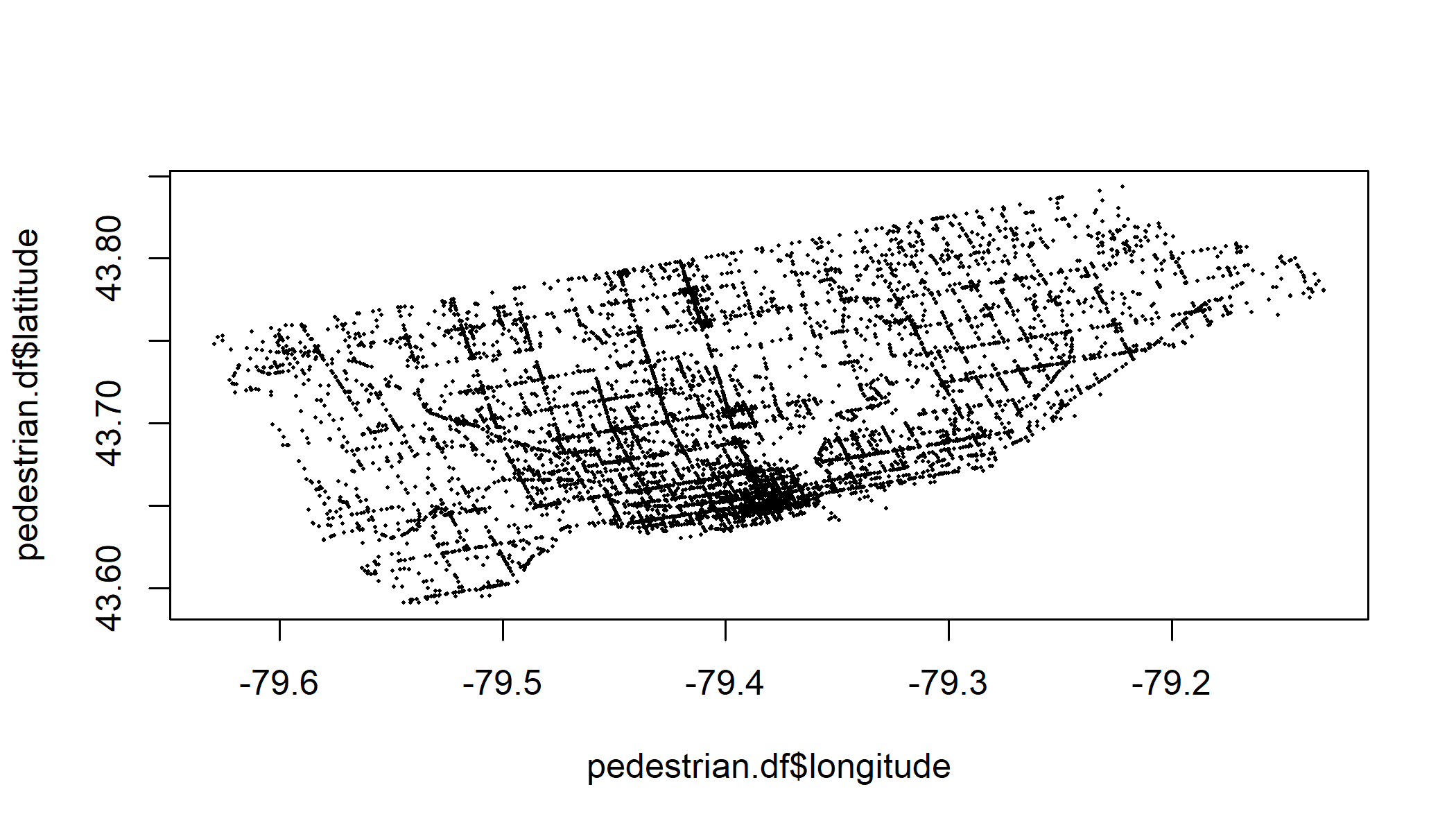
Prior to classification, we need to group the collision densities (where collisions occur within a set radius). We use three methods of grouping collision densities:

1. Kernel Density Estimation (KDE)
2. K-means Clustering
3. Density Based Spatial Clustering and Application with Noise (DBSCAN)

By using unsupervised learning to group these collision densities we can create a map of collision hotspots and then model the data to examine shared chareristics and classify unseen data.

## Kernel Density Estimation (KDE)

#KDE is the standard method used in most of the traffic safety literature. KDE groups densities of points into 3-dimensional "humps" or kernels along a specificed threshold. I used a KDE in the exploratory data analysis stage above, but didn't pull those high collision zones into the data set as a new feature.   
  
# Below, I reconstruct the KDE to my specifications and then create a new feature in the dataset called zone which assigns every collision falling within at least 70% of the surface area of the KDE one of four zones according to the levels set below (70%, 50%, 25%, 10%).   
  
# I can then use the zone attribute to look at shared characteristics of collisions within each zone beyond geographic proximity during classification.   
  
library(MASS)  
library(sp)  
library(reshape2)

# Reference plot showing all collision points plotted   
plot(pedestrian.df$longitude, pedestrian.df$latitude, pch = 16, cex = .25)

# use kde2d function to create kernel density estimates   
x <- pedestrian.df$longitude  
y <- pedestrian.df$latitude  
dens <- kde2d(x, y, n=100)  
  
# create the contours to plot - 70%, 50%, 25%, 10% of density contained in each contour   
prob <- c(0.7, 0.5, 0.25, 0.1)  
dx <- diff(dens$x[1:4])  
dy <- diff(dens$y[1:4])  
sz <- sort(dens$z)  
c1 <- cumsum(sz) \* dx \* dy

levels <- sapply(prob, function(x) {   
 approx(c1, sz, xout = 1 - x)$y  
})  
  
#create the contour plot using smoothScatter which smooths the collisions into kernel densities   
  
smoothScatter(x,y) + contour(dens, levels=levels, labels=prob, col = c("green", "yellow", "orange", "red"), lwd = 1.5, add=T)

# points within polygons to identify which collisions lie within which of the four contours   
# show how many polygons created per level   
ls <- contourLines(dens, level=levels)  
sort(table(sapply(ls, `[[`, "level")))

##   
## 70.2266571229002 29.7050372913424 15.5351922832459 11.5092342683039   
## 1 2 5 7

# there are 15 polygons in total but 4 levels; this is bc each polygon is on a separate layer   
  
setNames(  
 lapply(ls, function(poly) sum(sp::point.in.polygon(pedestrian.df$longitude, pedestrian.df$latitude, poly$x, poly$y))),  
 sapply(ls, `[[`, "level")  
) -> level\_cts  
  
# show sum of collisions per contour level   
sapply(  
 split(level\_cts, names(level\_cts)),  
 function(level) sum(unlist(level))  
) -> pt\_cts  
  
pt\_cts <- as.data.frame(pt\_cts)  
pt\_cts <- t(pt\_cts)  
colnames(pt\_cts) <- c("Zone 4 (70%)", "Zone 3 (50%)", "Zone 2 (25%)", "Zone 1 (10%)")  
rownames(pt\_cts) <- "Number of Collisions"  
pt\_cts <- t(pt\_cts)  
library(knitr)  
kable(pt\_cts)

|  |  |
| --- | --- |
|  | Number of Collisions |
| Zone 4 (70%) | 12017 |
| Zone 3 (50%) | 8469 |
| Zone 2 (25%) | 4325 |
| Zone 1 (10%) | 2054 |
|  |  |

# below, I attempted to add Zone labels as a new feature to the dataset but the function didnt work as expected as it generated more non-duplicate records than the dataset contained. As a result, I wasn't able to test the clustering performance of KDE on location coordinates as of December 4, 2018. I will update the notebook if I'm able to succeed with this later.   
  
#do.call(  
# rbind.data.frame,  
# lapply(ls, function(poly) {   
# which\_pts <- as.logical(sp::point.in.polygon(pedestrian.df$longitude, pedestrian.df$latitude, poly$x, poly$y))  
# tdf <- pedestrian.df[which\_pts,] # assign them to a temp data frame  
# tdf$level <- poly$level # add the level  
# tdf  
# })  
#) -> pedestrian.df2  
  
#library(dplyr)  
#dplyr::glimpse(pedestrian.df2)  
  
#new\_xdf$level\_num <- as.integer(factor(new\_xdf$level, levels, labels=1:length(levels)))  
#new\_xdf$prob <- as.numeric(as.character(factor(new\_xdf$level, levels, labels=prob)))  
  
#pedestrian.kde <- pedestrian.df  
#prop.table(table(pedestrian.kde$zone))

## *K-means Clustering*

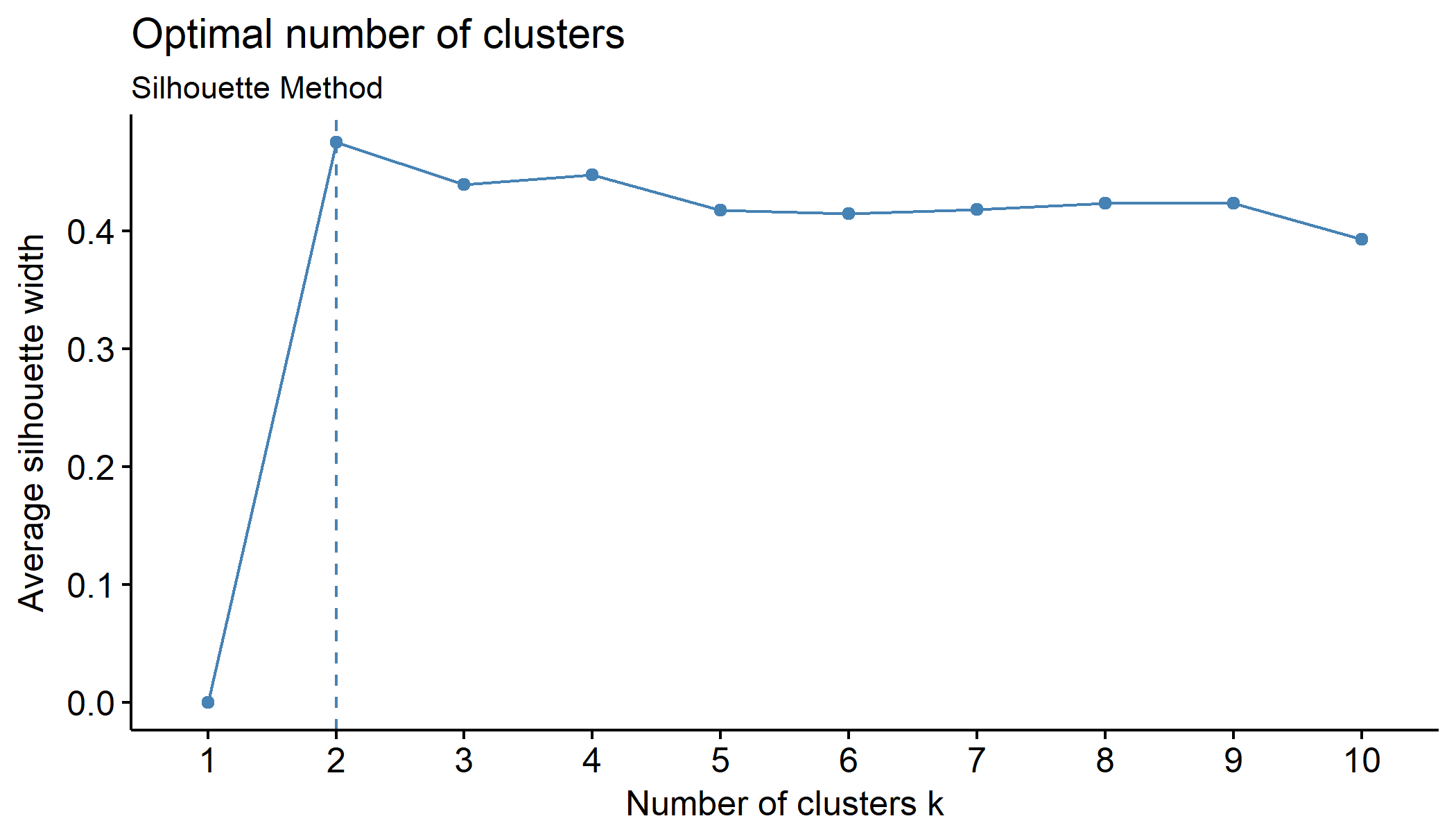
Rather than clustering collisions by kernel densities, we can also cluster using k-means clustering which is an unsupervised learning algorithm. Below we find the optimal value for **k**, which is the number of clusters our model will attempt to fit the points. Effectively, this will cluster collisions into discrete concentrations based on location not unlike KDE, but using the shortest Eucledean distance to the centroid rather than kernel density. Thus, I expect the two models to look somewhat similar to each other, but k-means will have potentially many elliptically-shaped clusters rather than the organic, blob-like shape KDE creates.

# Before applying k-means, we must find k, the optimal number of clusters   
library(factoextra)

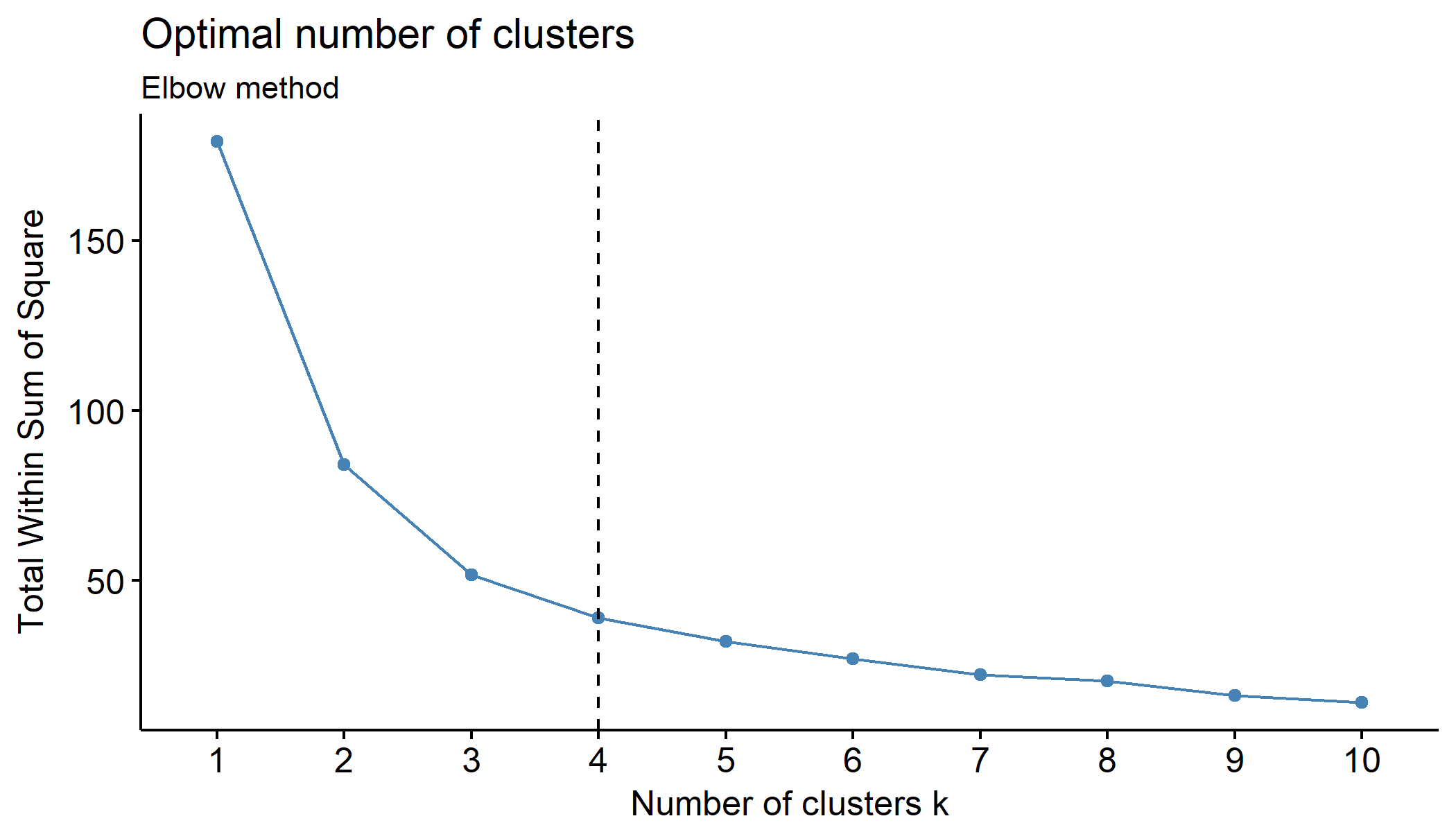
library(NbClust)

library(doSNOW)

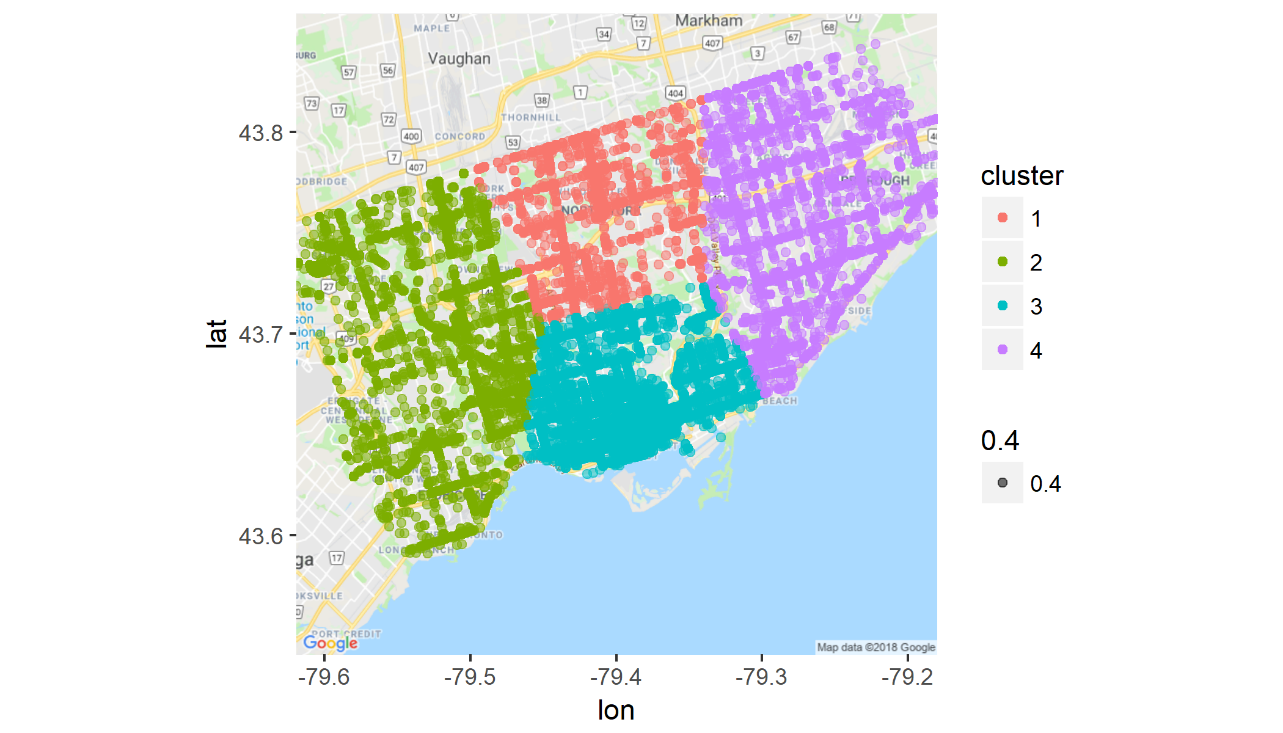
library(caret)  
  
# creating subset of main dataset to only contain lon and lat coordinates of collisions  
pedestrian.coords <- pedestrian.df[,c("longitude", "latitude")]  
pedestrian.kmeans <- pedestrian.df[,c("longitude", "latitude")]  
# create a stratified sample of the above so that the training and test sets contains the same proportion of collisions by location in case this influences collisions   
  
#pedestrian.df$location\_desc <- as.factor(pedestrian.df$location\_desc)  
  
#set.seed(123)  
#index <- createDataPartition(pedestrian.coords$location\_desc, times = 1, p = 0.75, list = F)  
#train\_pedestrian.kmeans <- pedestrian.coords[index,]  
#test\_pedestrian.kmeans <- pedestrian.coords[-index,]  
  
# remove location\_desc since it is a classifier and we are using an unsupervised algorithm   
#train\_pedestrian.kmeans <- train\_pedestrian.kmeans[,-3]  
#test\_pedestrian.kmeans <- test\_pedestrian.kmeans[,-3]  
  
# creating parallel processing clusters to speed up calculations  
# WARNING - these calculations are resource intensive on both CPU and RAM  
# Need at least 3 cores and 16 GB of RAM to run   
  
cl <- makeCluster(3, type = "SOCK")  
registerDoSNOW(cl)  
  
# Using avg silhouette to determine optimal k clusters   
fviz\_nbclust(pedestrian.kmeans, kmeans, method = "silhouette") + labs(subtitle = "Silhouette Method")



# silhouette method suggests 2 clusters  
  
# Elbow method  
fviz\_nbclust(pedestrian.kmeans, kmeans, method = "wss") + geom\_vline(xintercept = 4, linetype = 2) +  
 labs(subtitle = "Elbow method")

# elbow method suggests 4 clusters   
  
#we will try 4 clusters since having only 2 clusters with so many points and noise will lead to a useless outcome   
  
# k-means clustering   
set.seed(123)  
kmeans\_model <- kmeans(pedestrian.kmeans, 4, nstart = 25)  
pedestrian.kmeans$cluster <- as.factor(kmeans\_model$cluster)  
  
stopCluster(cl)  
gc()

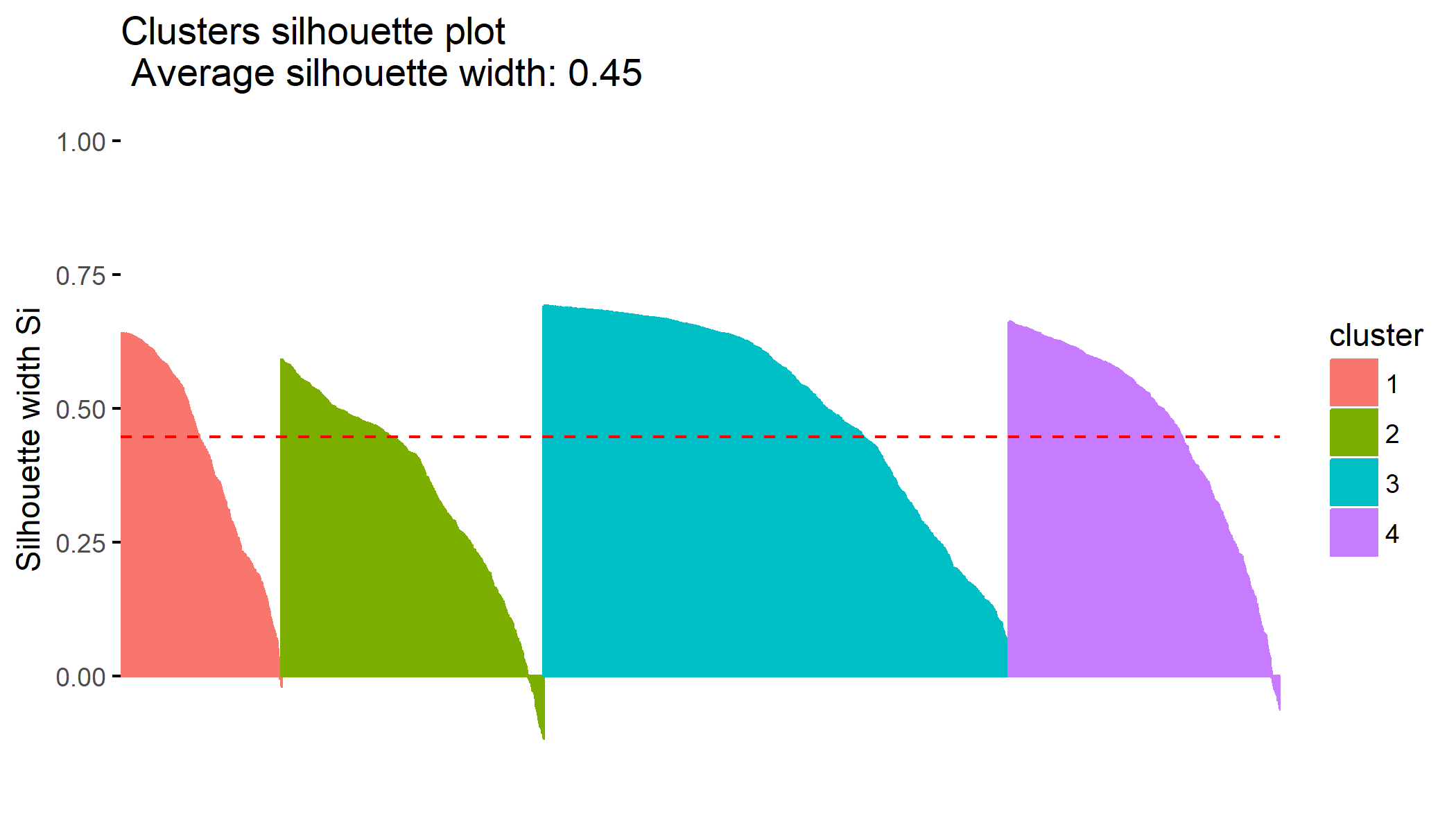
toronto\_map + geom\_point(aes(x = longitude, y = latitude, color = cluster, alpha = 0.4), data = pedestrian.kmeans)

# interestingly, the kmeans clustering grouped the points roughly according to the boundaries of North York, Etobicoke, Downtown, and Scarborough, even though all it had was lat and lon data.   
  
# create a disimilarity matrix for use in silhouette  
dm <- as.matrix(dist(pedestrian.coords))  
  
# use the identified clusters and dissimilarity matrix to calculate the silhouette   
# our cluster labels need to be turned back into numerical values not factor to work   
library(cluster)

## Warning: package 'cluster' was built under R version 3.3.3

pedestrian.kmeans$cluster <- as.numeric(pedestrian.kmeans$cluster)  
silhouette\_kmeans <- silhouette(pedestrian.kmeans$cluster, dm)  
  
# plot results  
fviz\_silhouette(silhouette\_kmeans, print.summary = T)

## cluster size ave.sil.width  
## 1 1 2161 0.41  
## 2 2 3506 0.35  
## 3 3 6237 0.51  
## 4 4 3629 0.46

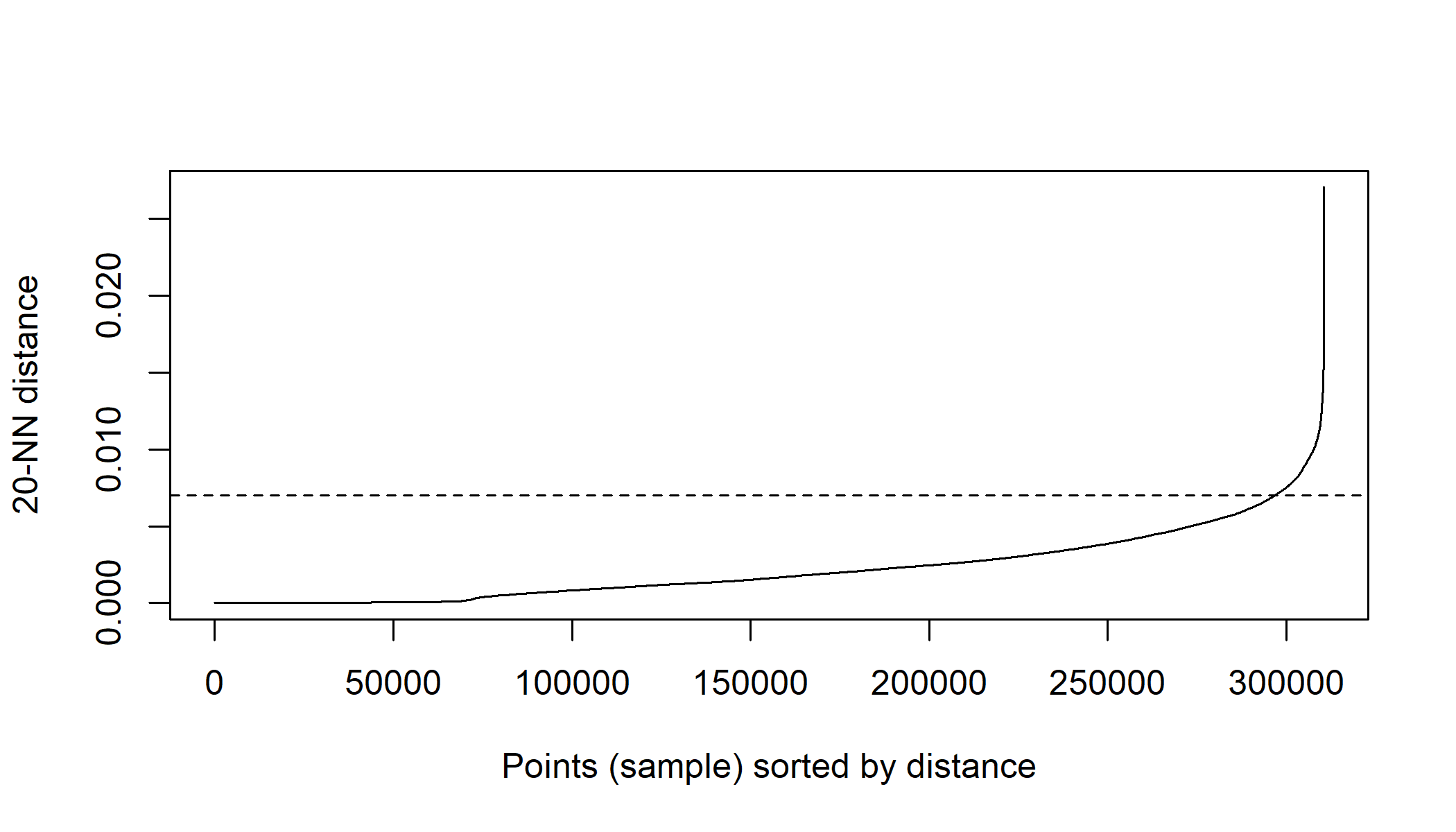


# With an average silhouette of 0.45, the structure of the clusters is acceptable, but not strong. Notably cluster 2 is quite weak with an average silhouette length of 0.35. This perhaps means that clustering by kmeans on collision locations is reasonable, but not necessarily solid.   
  
#add kmeans clusters to main dataset   
pedestrian.df$kmeans\_cluster <- pedestrian.kmeans$cluster

## Density-based Spatial Clustering and Application with Noise (DBSCAN)

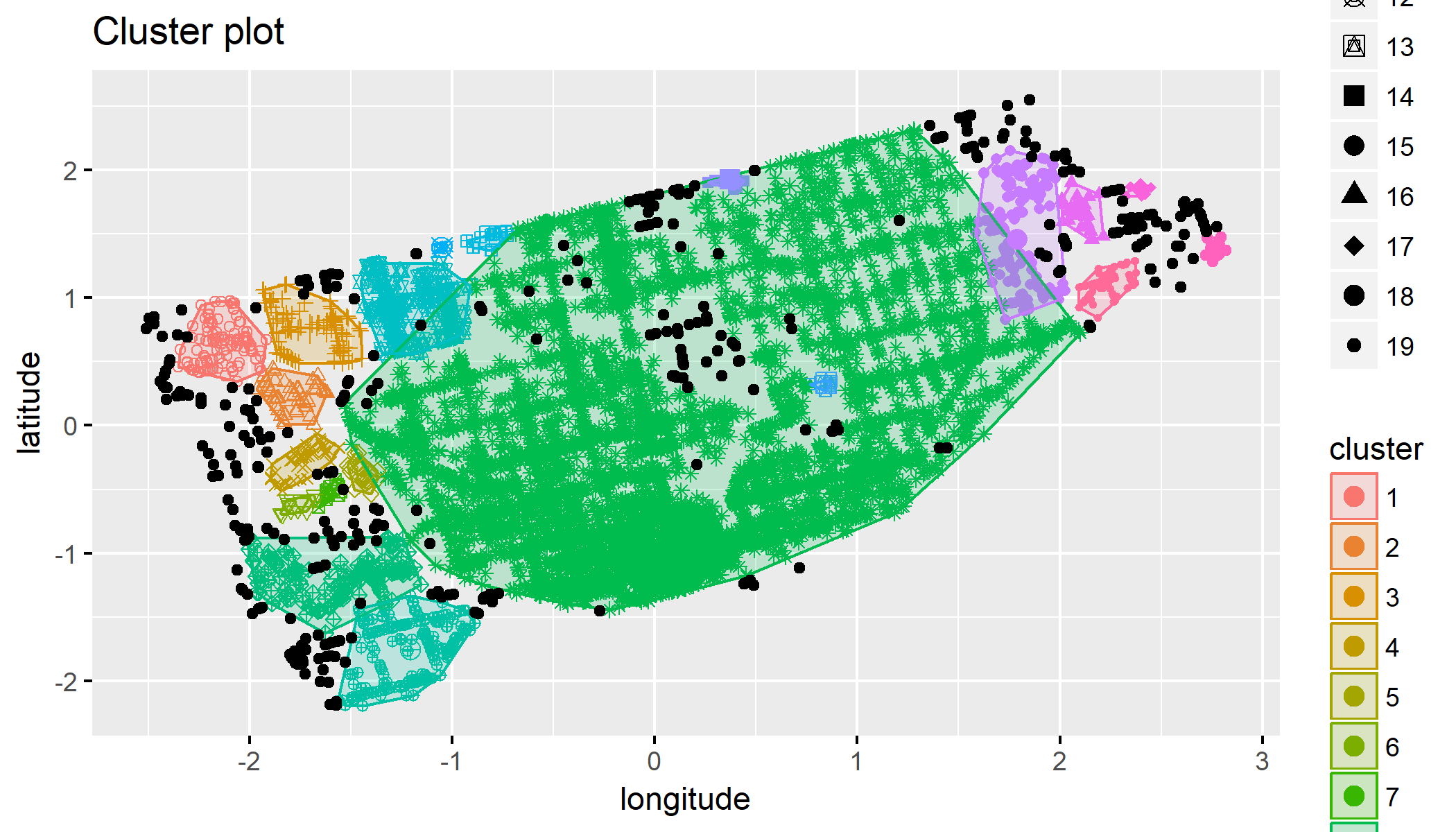
Similar in logic to both KNN and KDE, this clustering algorithm is said to have better performance for clusters of linear shape. Since my data is based on collisions on streets which have a linear shape and a lot of outliers (aka noise), it is expected for DBSCAN to perform well in this case vs. k-means which is effective at clustering points in an eliptical shape and with little noise. Another benefit of DBSCAN is that it requires no **k** to be set ahead of time. Much like KNN, it requires the minimum amount of neighbours to be set, as well as **epsilon**, which is the radius of the neighbourhood. In DBSCAN, it is possible for a point to not belong to any cluster which is beneficial in this case since we want to prioritize areas with high concentrations of collisions.

# for Density-based Clustering (DBSCAN) and visualization of clusters   
library("dbscan")  
library("factoextra")  
library("knitr")  
  
# Determining optimal epsilon  
# Just like k in k-means, there are methods to determine the optimal epsilon in DBSCAN using k-nearest neighbours. The value of k in our KNN corresponds to the min points value in DBSCAN.   
  
# These k-distances are plotted and similar to the elbow method used above for k-means, a sharp bend occurs which corresponds to optimal eps.   
  
pedestrian.dbscan <- pedestrian.coords  
  
  
# prepare our training and test sets   
#pedestrian.dbscan <- pedestrian.coords[index,]  
#test\_pedestrian.dbscan <- pedestrian.coords[-index,]  
  
# remove ksi\_check since it is a classifier and we are using an unsupervised algorithm   
#train\_pedestrian.dbscan <- train\_pedestrian.dbscan[,-3]  
#test\_pedestrian.dbscan <- test\_pedestrian.dbscan[,-3]  
  
# Check for optimal epsilson value using KNN   
dbscan::kNNdistplot(pedestrian.dbscan, k = 20) + abline(h = 0.007, lty = 2)



## numeric(0)

#therefore, for dbscan, we set eps = 0.006 and minPts = 18  
  
# However, keep in mind that an epsilon of 0.001 is equivalent to a radius of 111 metres since the coordinates are using longitude and latitude. This means too high an eps can easily span across the entire city, leading to one giant cluster.  
  
  
set.seed(123)  
dbscan\_model <- dbscan::dbscan(pedestrian.dbscan, 0.007, minPts = 20)  
  
# plot the results  
fviz\_cluster(dbscan\_model, pedestrian.dbscan, geom = "point")

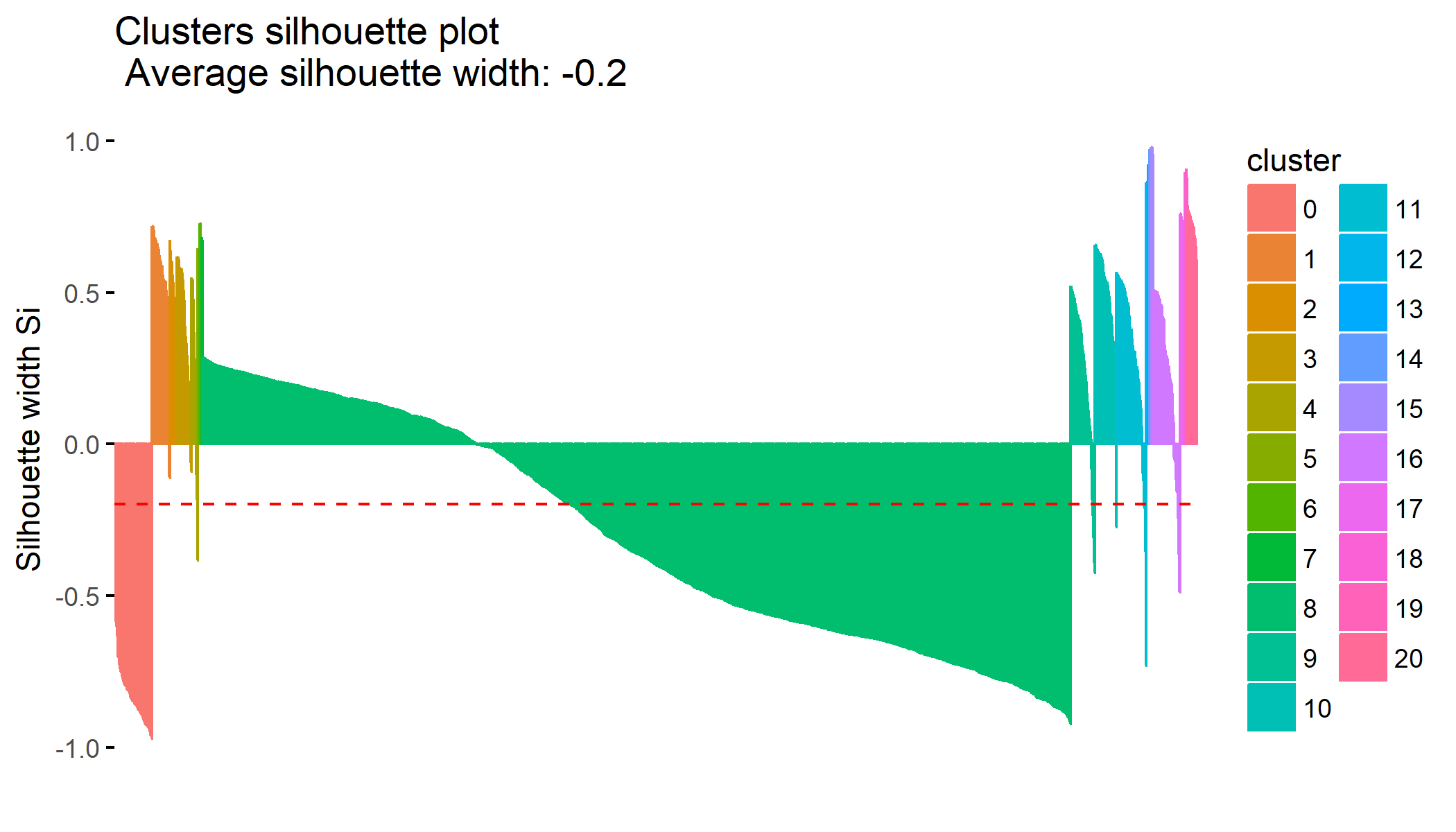


# show results in text form   
dbscan\_model

## DBSCAN clustering for 15533 objects.  
## Parameters: eps = 0.007, minPts = 20  
## The clustering contains 20 cluster(s) and 538 noise points.  
##   
## 0 1 2 3 4 5 6 7 8 9 10 11   
## 538 257 99 210 93 29 26 16 12442 349 310 429   
## 12 13 14 15 16 17 18 19 20   
## 32 18 17 33 386 64 20 20 145   
##   
## Available fields: cluster, eps, minPts

# The column names are cluster numbers - cluster 0 are the black points in the plot. They do not fit in any cluster (considered noise)  
# our model created 19 clusters and the shape of the clusters aren't as intuitive as the kmeans model but they do seem to highlight specific collision hotspots unlike kmeans.   
  
# Now lets see if DBSCAN performed better than kmeans using silhouette   
  
# use the identified clusters and dissimilarity matrix to calculate the silhouette   
# our cluster labels need to be turned back into numerical values not factor to work   
library(cluster)  
dbscan\_clusters <- dbscan\_model$cluster  
   
silhouette\_dbscan <- silhouette(dbscan\_model$cluster, dm)  
  
# plot results  
fviz\_silhouette(silhouette\_dbscan, print.summary = T)

## cluster size ave.sil.width  
## 0 0 538 -0.82  
## 1 1 257 0.57  
## 2 2 99 0.51  
## 3 3 210 0.41  
## 4 4 93 0.25  
## 5 5 29 0.52  
## 6 6 26 0.49  
## 7 7 16 0.61  
## 8 8 12442 -0.30  
## 9 9 349 0.25  
## 10 10 310 0.50  
## 11 11 429 0.30  
## 12 12 32 0.74  
## 13 13 18 0.90  
## 14 14 17 0.96  
## 15 15 33 0.97  
## 16 16 386 0.23  
## 17 17 64 0.63  
## 18 18 20 0.86  
## 19 19 20 0.87  
## 20 20 145 0.70



# With an average silhouette of -0.2, this is a far worse way to cluster the collisions when compared to kmeans. However, certain clusters have very strong fits. Unfortunately, the weak fit of the largest cluster - which spans a large part of the city and is represented by the bright green colour - brings the average silhouette down significantly.   
  
# One thing the City could do is look at the clusters that the model identified that have strong silhouettes (above 0.65 for example) and target those areas for intervention. Because clusters with low silhouette scores could easily belong to other clusters, the clusters that have high silhouettes are quite durable.   
  
# Due to the overall low silhouette score, however, We will exclude the cluster labels from dbscan from further analysis and see if it is better with multiple dimensions rather than just longitude and latitude.

# Classification

We have so far used 3 clustering methods to group collisions based on their geo-spatial attributes. These clusters were then added as new features to the dataset so we can see whether they are good predictors. For classification purposes, we can’t use KDE, but we can use both k-means and DBSCAN in order to see if there are shared characteristics between collisions and come up with a way to profile high collision zones.

*Creating a Training and Test Set* I created a training and test set in case I want to use a supervised machine learning algorithm for classifcation.

library(caret)  
pedestrian.df$kmeans\_cluster <- as.factor(pedestrian.df$kmeans\_cluster)  
  
# create random stratified sample so that the proportion of k-means clusters is similar across the training set and test set  
set.seed(123)  
index <- createDataPartition(pedestrian.df$kmeans\_cluster, times = 1, p = 0.75, list = F)  
train\_pedestrian <- pedestrian.df[index,]  
test\_pedestrian <- pedestrian.df[-index,]  
  
# verifying the stratified samples have similar distributions of collisions based on location   
prop.table(table(train\_pedestrian$kmeans\_cluster))

##   
## 1 2 3 4   
## 0.1391297 0.2257317 0.4015106 0.2336280

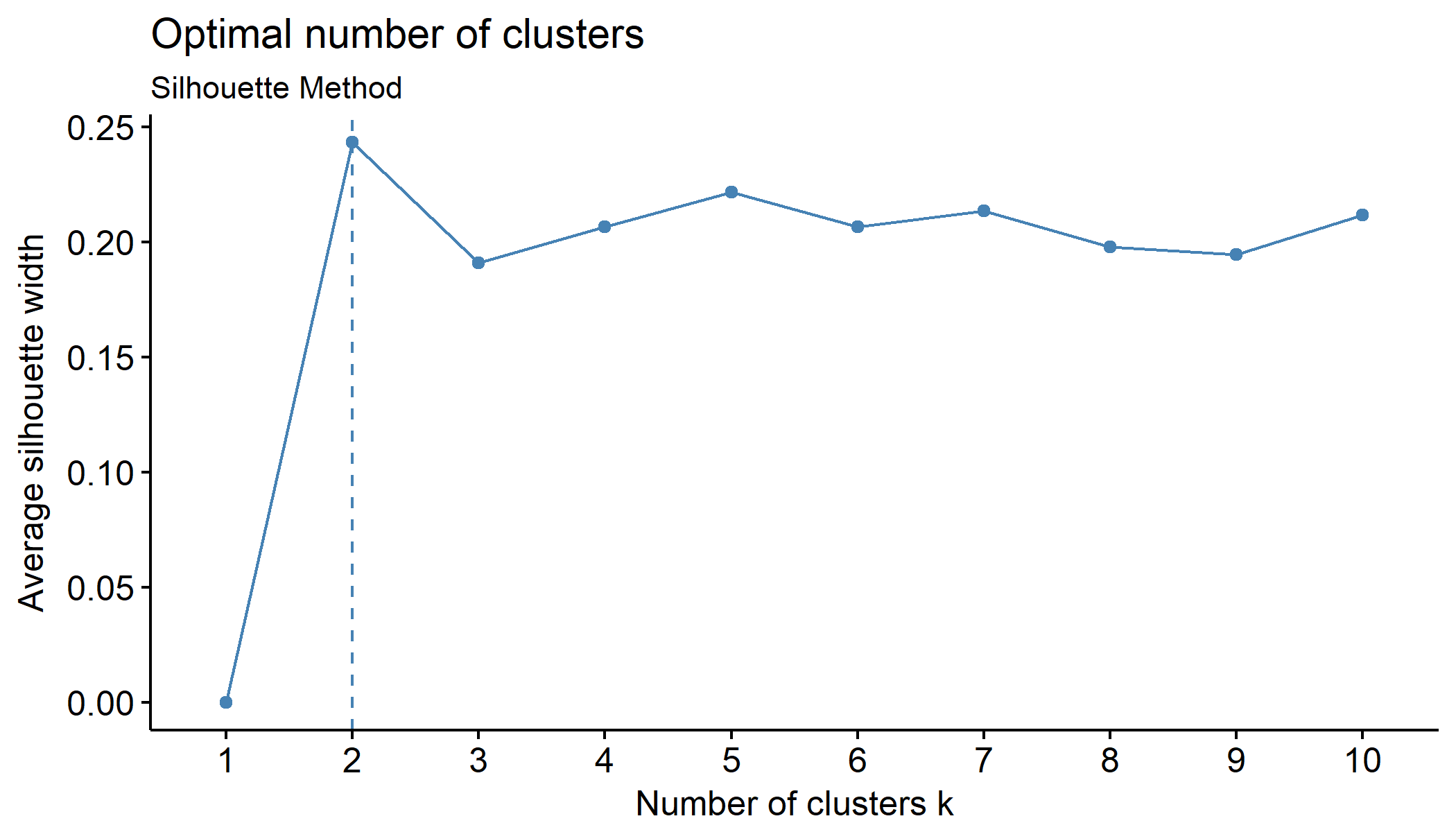
prop.table(table(test\_pedestrian$kmeans\_cluster))

##   
## 1 2 3 4   
## 0.1391036 0.2256569 0.4015971 0.2336425

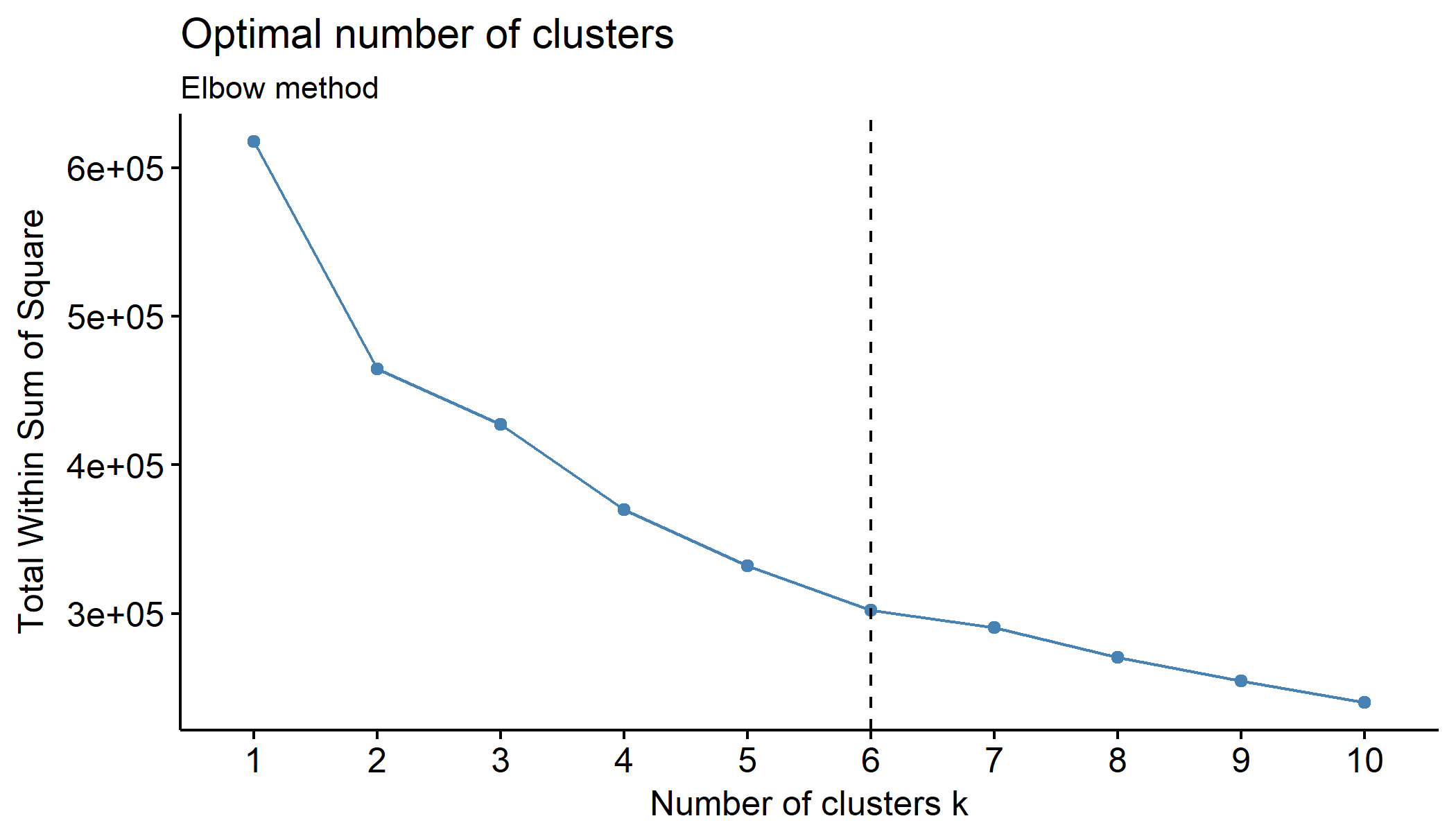
## *K-means Clustering Classification Model*

# Before applying k-means, we must find k, the optimal number of clusters   
library(factoextra)  
library(NbClust)  
library(doSNOW)  
  
# Normalizing numerical values   
train\_pedestrian\_num <- Filter(is.numeric, train\_pedestrian)  
train\_pedestrian\_num\_norm <- scale(train\_pedestrian\_num)  
train\_pedestrian\_num\_norm <- as.data.frame(train\_pedestrian\_num\_norm)

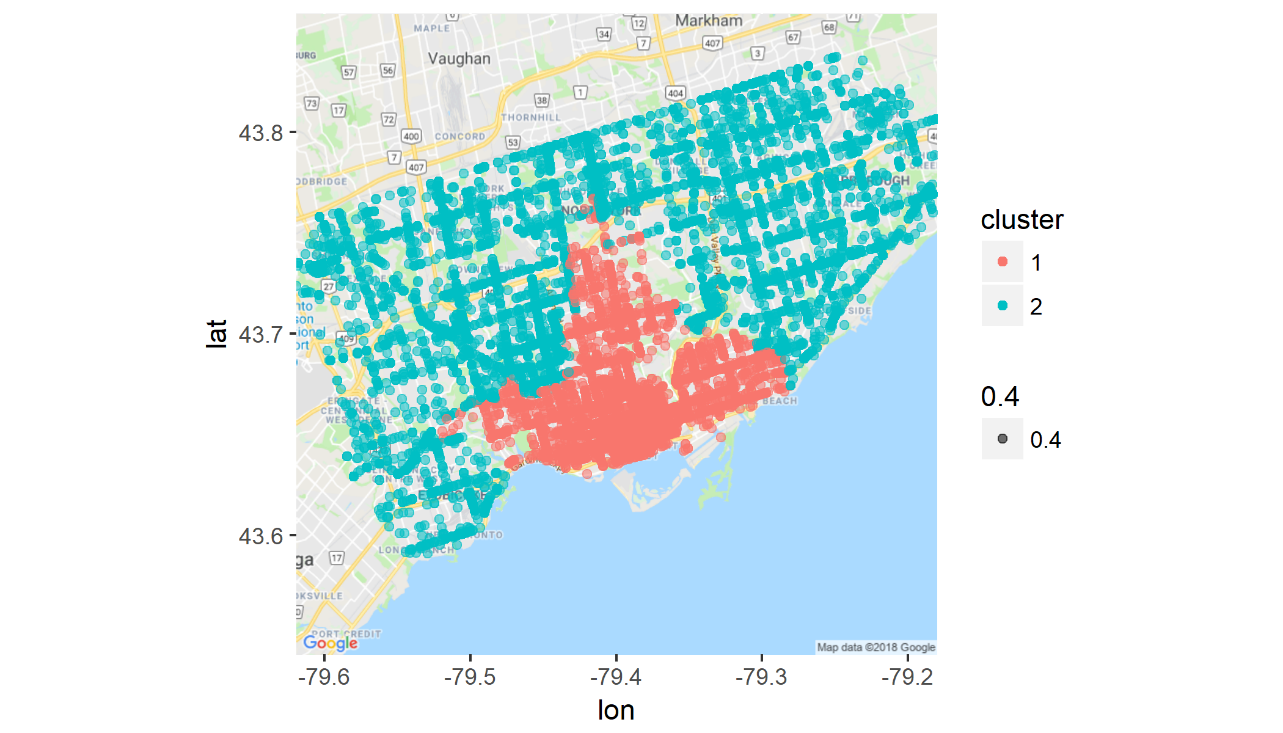
# creating parallel processing clusters   
# WARNING be sure your computer has at least 4 processing cores and 16 GB of RAM  
cl <- makeCluster(3, type = "SOCK")  
registerDoSNOW(cl)  
  
# Using avg silhouette to determine optimal k clusters   
fviz\_nbclust(train\_pedestrian\_num\_norm, kmeans, method = "silhouette") + labs(subtitle = "Silhouette Method")



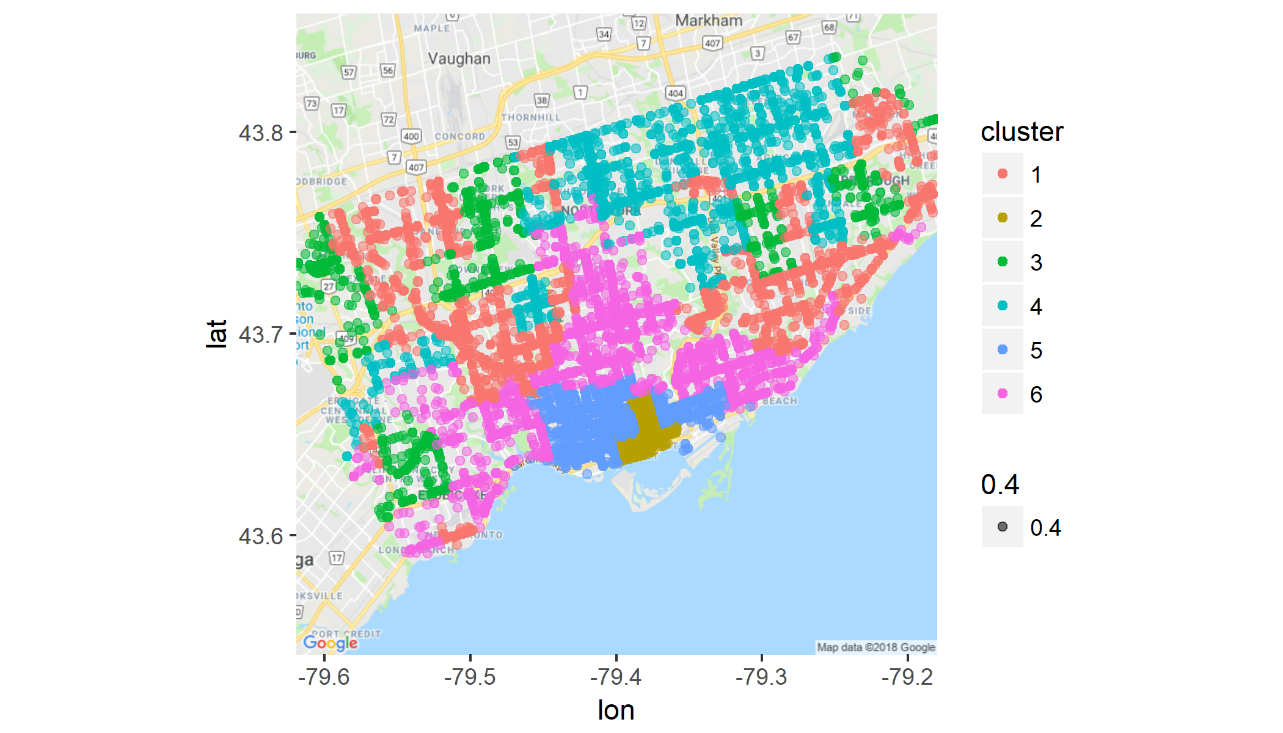
# silhouette method suggests 2 clusters  
  
# Elbow method  
fviz\_nbclust(train\_pedestrian\_num\_norm, kmeans, method = "wss") + geom\_vline(xintercept = 6, linetype = 2) +  
 labs(subtitle = "Elbow method")



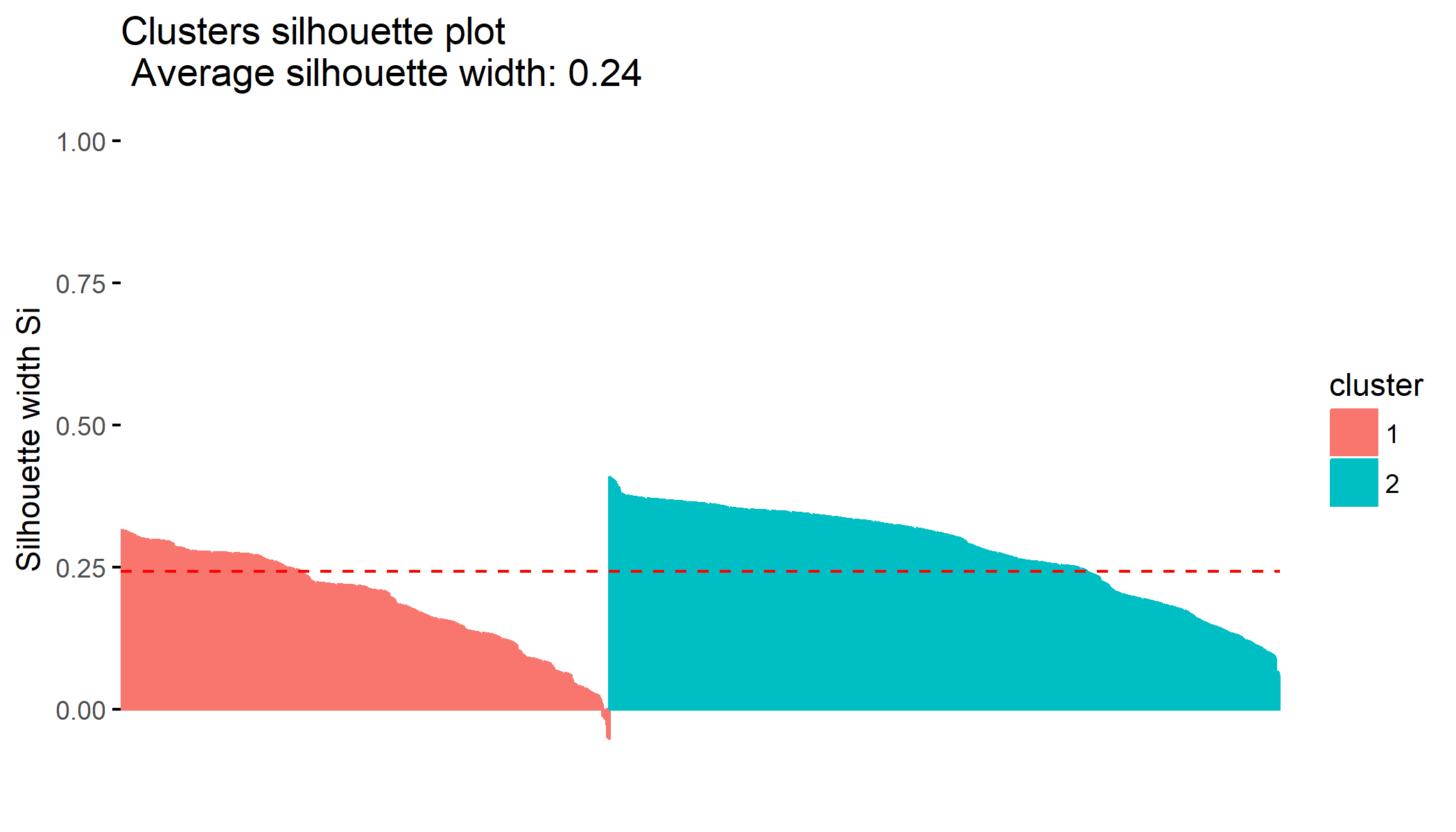
# elbow method suggests 6 clusters   
  
  
# we can go either 2 or 6 clusters - I will try both and see which has better silhouette score   
# k = 2  
set.seed(123)  
train\_kmeans\_k2 <- kmeans(train\_pedestrian\_num\_norm, 2, nstart = 25)  
train\_pedestrian\_num\_norm\_k2 <- train\_pedestrian\_num\_norm   
train\_pedestrian\_num\_norm\_k2$cluster <- as.factor(train\_kmeans\_k2$cluster)  
train\_pedestrian\_num\_norm\_k2$longitude <- train\_pedestrian\_num$longitude  
train\_pedestrian\_num\_norm\_k2$latitude <- train\_pedestrian\_num$latitude  
   
toronto\_map + geom\_point(aes(x = longitude, y = latitude, color = cluster, alpha = 0.4), data = train\_pedestrian\_num\_norm\_k2)



# This cluster pattern is quite intuitive -- cluster 1 is the downtown core, cluster 2 is everything outside the core. Let's see what it looks like when we use 6 as suggested by the Elbow Method.   
  
# k = 6  
set.seed(123)  
train\_kmeans\_k6 <- kmeans(train\_pedestrian\_num\_norm, 6, nstart = 25)  
train\_pedestrian\_num\_norm\_k6 <- train\_pedestrian\_num\_norm   
train\_pedestrian\_num\_norm\_k6$cluster <- as.factor(train\_kmeans\_k6$cluster)  
train\_pedestrian\_num\_norm\_k6$longitude <- train\_pedestrian\_num$longitude  
train\_pedestrian\_num\_norm\_k6$latitude <- train\_pedestrian\_num$latitude  
   
toronto\_map + geom\_point(aes(x = longitude, y = latitude, color = cluster, alpha = 0.4), data = train\_pedestrian\_num\_norm\_k6)

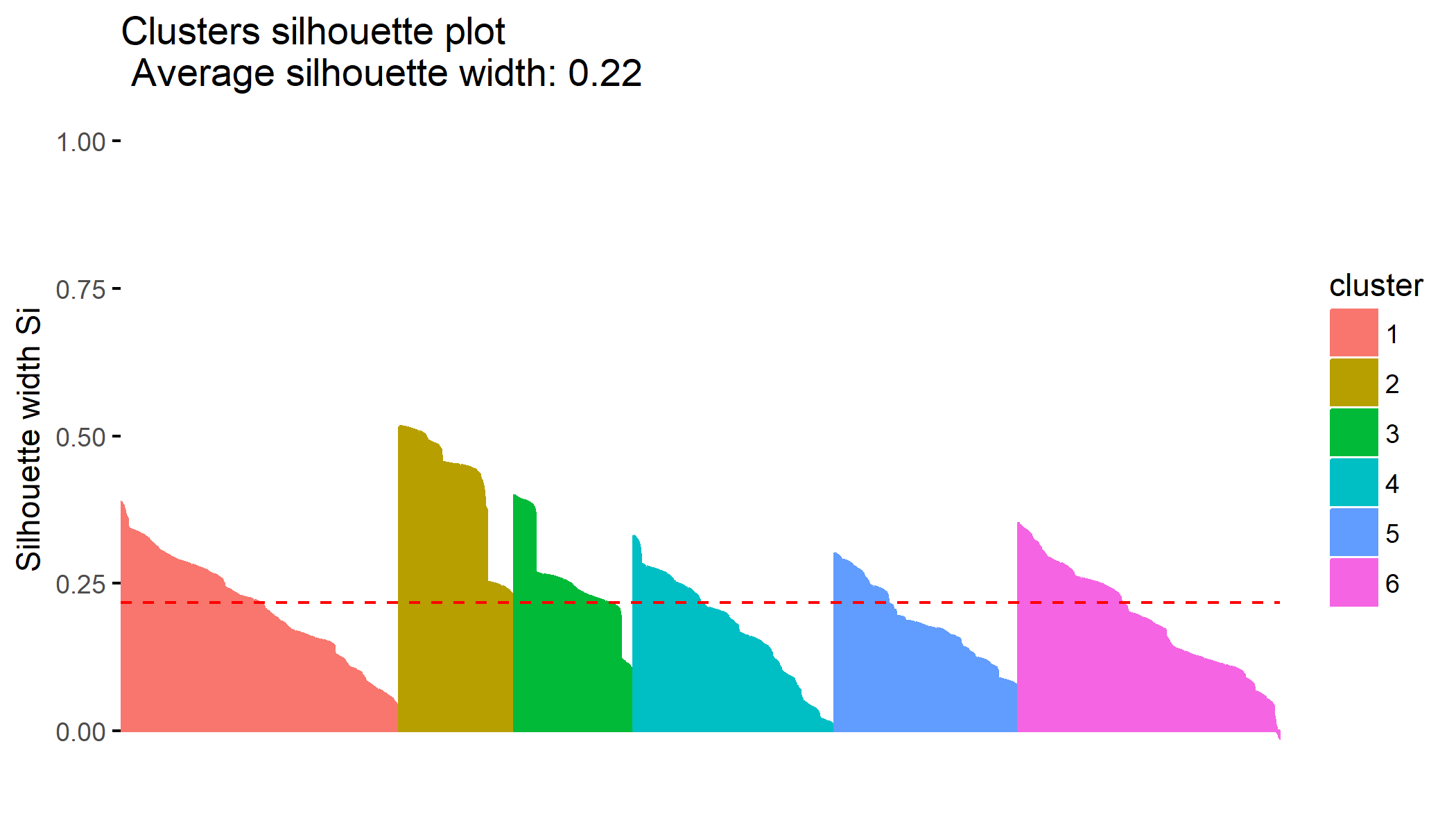
# here too the downtown core is clearly identifiable, but the outlying clusters are mixed together and it is hard to tell from the plot what characteristics the points within these clusters share. Later, we will use Random Forest to see which variables are most important to each cluster to answer this question.   
  
# performance testing using silhouette   
library(cluster)  
# create a disimilarity matrix for use in silhouette  
dm <- as.matrix(dist(train\_pedestrian\_num\_norm))  
train\_pedestrian\_num\_norm\_k2$cluster <- as.numeric(train\_pedestrian\_num\_norm\_k2$cluster)  
train\_pedestrian\_num\_norm\_k6$cluster <- as.numeric(train\_pedestrian\_num\_norm\_k6$cluster)  
  
# calculate and plot silhouette for k = 2 model   
silhouette\_k2 <- silhouette(train\_pedestrian\_num\_norm\_k2$cluster, dm)  
fviz\_silhouette(silhouette\_k2, print.summary = T)

## cluster size ave.sil.width  
## 1 1 4910 0.19  
## 2 2 6741 0.28



# with an average silhouette of 0.24, the clusters are quite weak   
  
# calculate and plot silhouette for k = 6 model   
silhouette\_k6 <- silhouette(train\_pedestrian\_num\_norm\_k6$cluster, dm)  
fviz\_silhouette(silhouette\_k6, print.summary = T)

## cluster size ave.sil.width  
## 1 1 2804 0.21  
## 2 2 1152 0.42  
## 3 3 1199 0.25  
## 4 4 2022 0.17  
## 5 5 1847 0.18  
## 6 6 2627 0.19



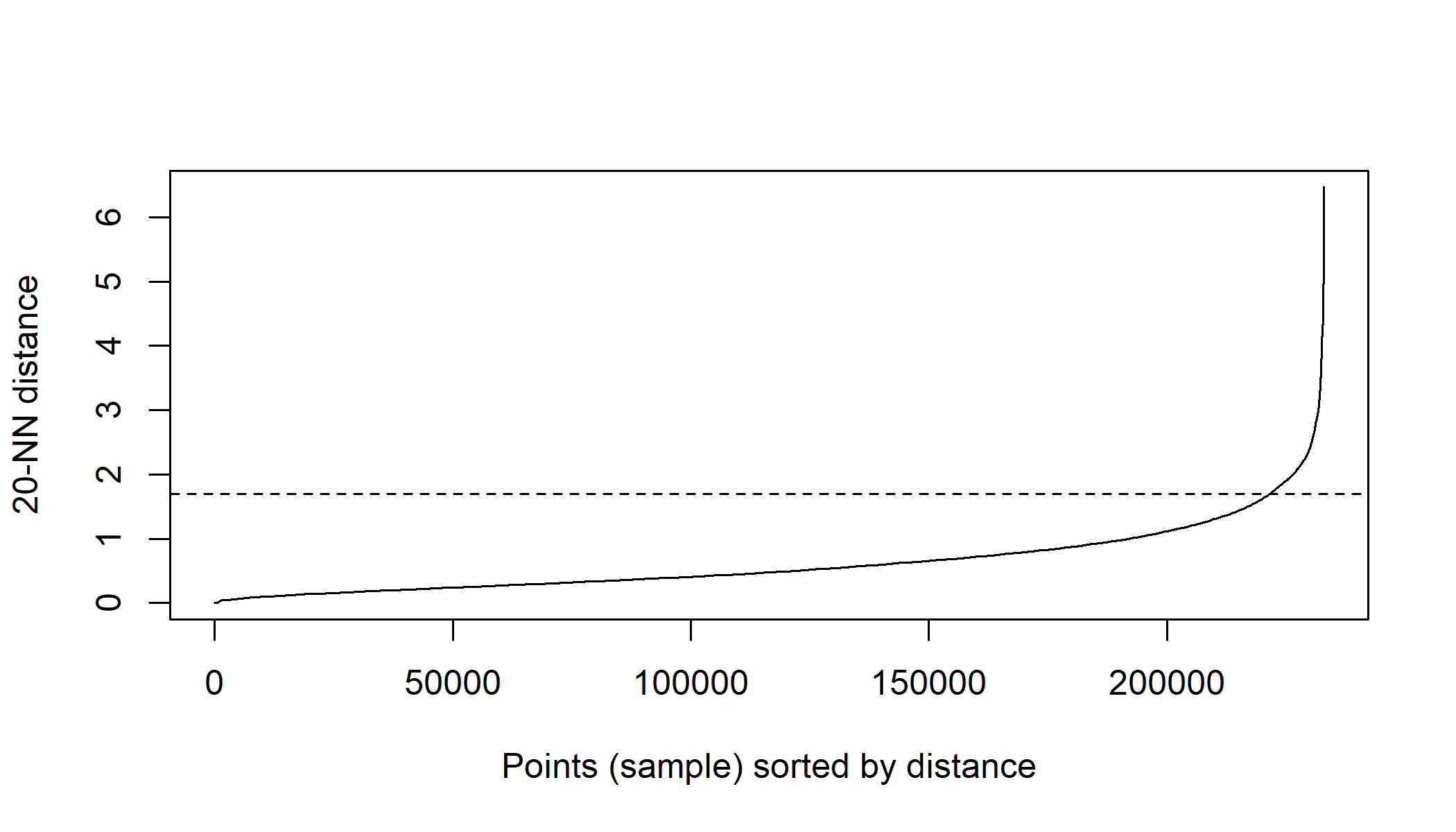
# with an average silhouette of 0.21, this is a very poor model   
stopCluster(cl)  
gc()

## used (Mb) gc trigger (Mb) max used (Mb)  
## Ncells 2362853 126.2 3886542 207.6 3886542 207.6  
## Vcells 149209943 1138.4 670484527 5115.4 1313139997 10018.5

# Considering how both k = 2 and k = 6 let to poor clustering performance, we will not use kmeans any further.

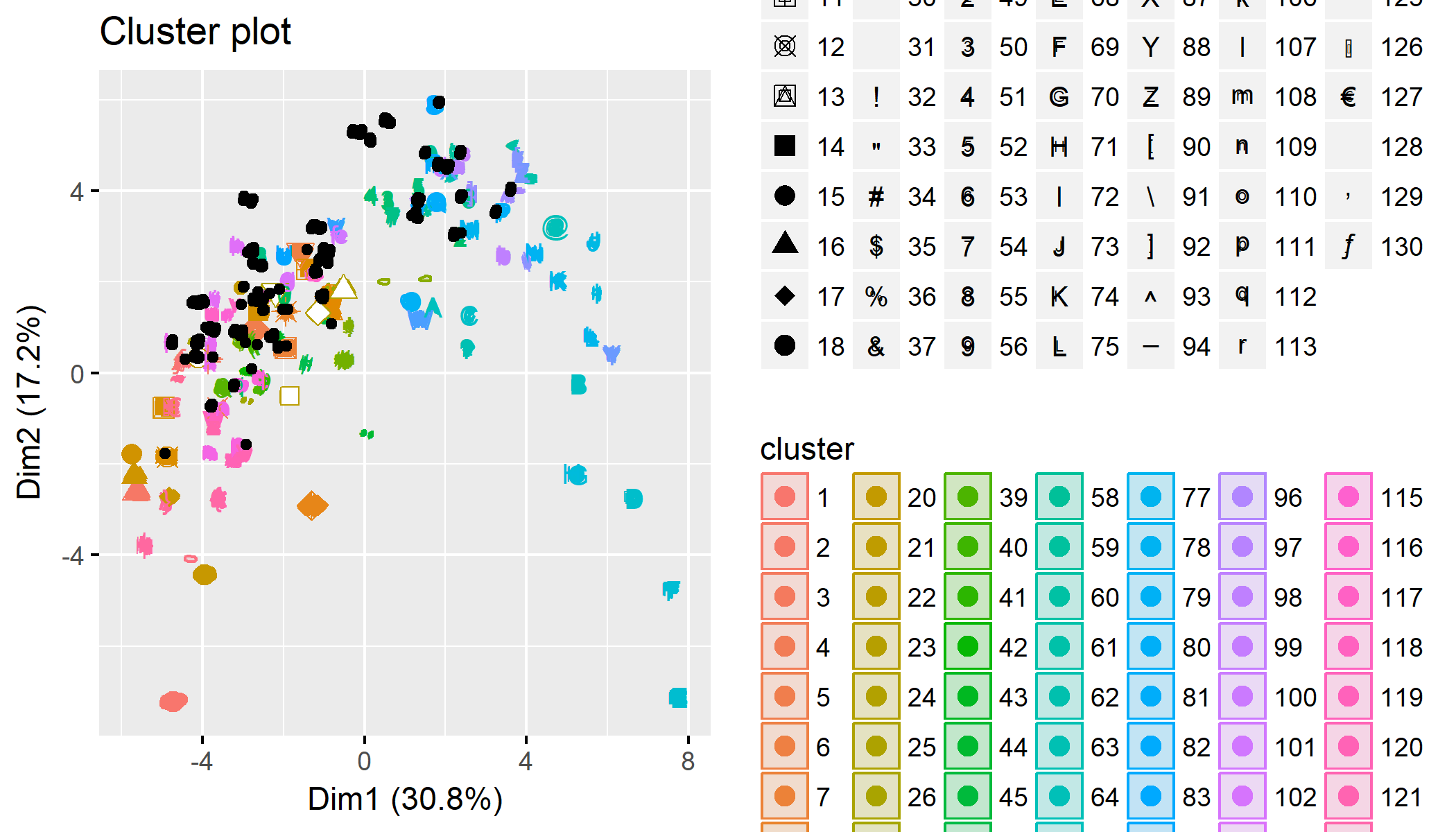
## *DBSCAN Classification*

# for Density-based Clustering (DBSCAN) and visualization of clusters   
library("dbscan")  
library("factoextra")  
library("knitr")  
  
# Determining optimal epsilon  
  
train\_dbscan <- train\_pedestrian\_num\_norm  
  
# Check for optimal epsilson value using KNN   
dbscan::kNNdistplot(train\_dbscan, k = 20) + abline(h = 1.7, lty = 2)

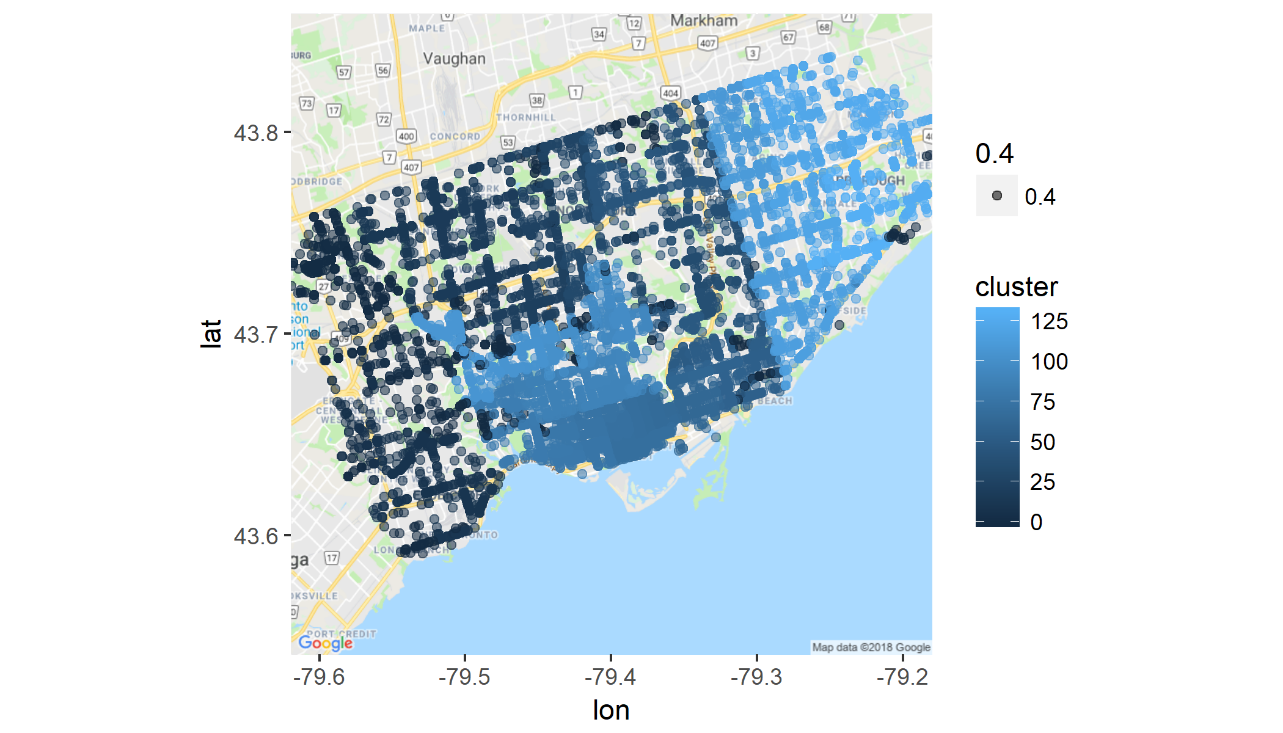


## numeric(0)

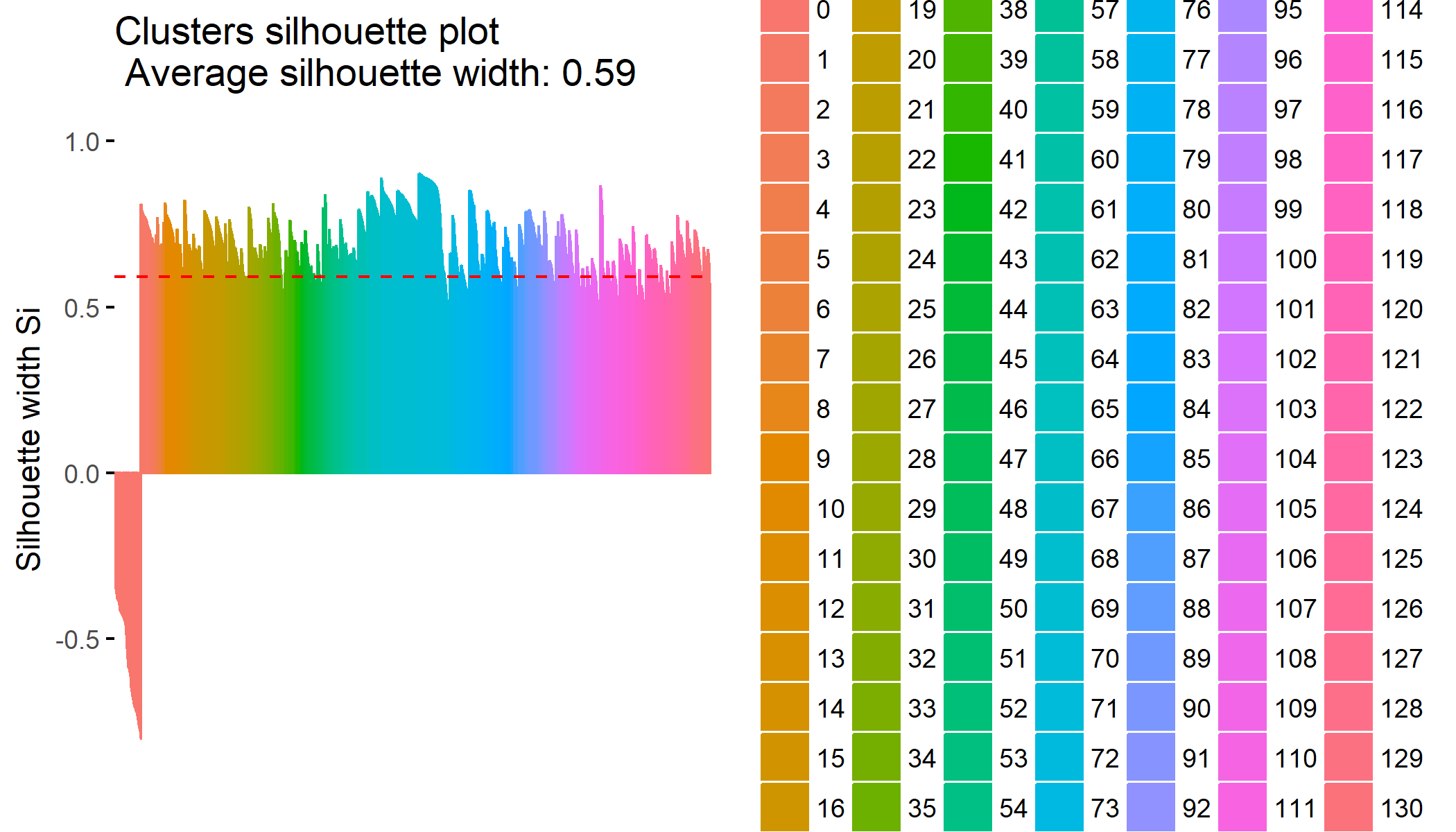
#therefore, for dbscan, we set eps = 1.7 and minPts = 20  
  
set.seed(123)  
train\_dbscan\_model <- dbscan::dbscan(train\_dbscan, 1.7, minPts = 20)  
  
# plot the results  
fviz\_cluster(train\_dbscan\_model, train\_dbscan, geom = "point")



# show results in text form   
train\_dbscan\_model

# our model created 133 clusters and the shape of the clusters aren't as intuitive as the kmeans model since lon and lat were normalized. However, I replot the clusters using actual lon and lat coordinates and we are able to see what the clusters look like in real space:  
  
train\_dbscan\_17 <- train\_pedestrian\_num\_norm  
train\_dbscan\_17$cluster <- train\_dbscan\_model$cluster  
train\_dbscan\_17$longitude <- train\_pedestrian\_num$longitude  
train\_dbscan\_17$latitude <- train\_pedestrian\_num$latitude  
   
toronto\_map + geom\_point(aes(x = longitude, y = latitude, color = cluster, alpha = 0.4), data = train\_dbscan\_17)

# This clustering pattern is very different from kmeans - first, there are a lot more clusters, and second, the clusters that are away from the downtown core are in a grid-like pattern -- a pattern we could expect due to the grid-like layout of toronto's streets!   
  
# Now lets see if DBSCAN performed better than kmeans using silhouette   
  
silhouette\_dbscan\_17 <- silhouette(train\_dbscan\_17$cluster, dm)  
fviz\_silhouette(silhouette\_dbscan\_17, print.summary = T)

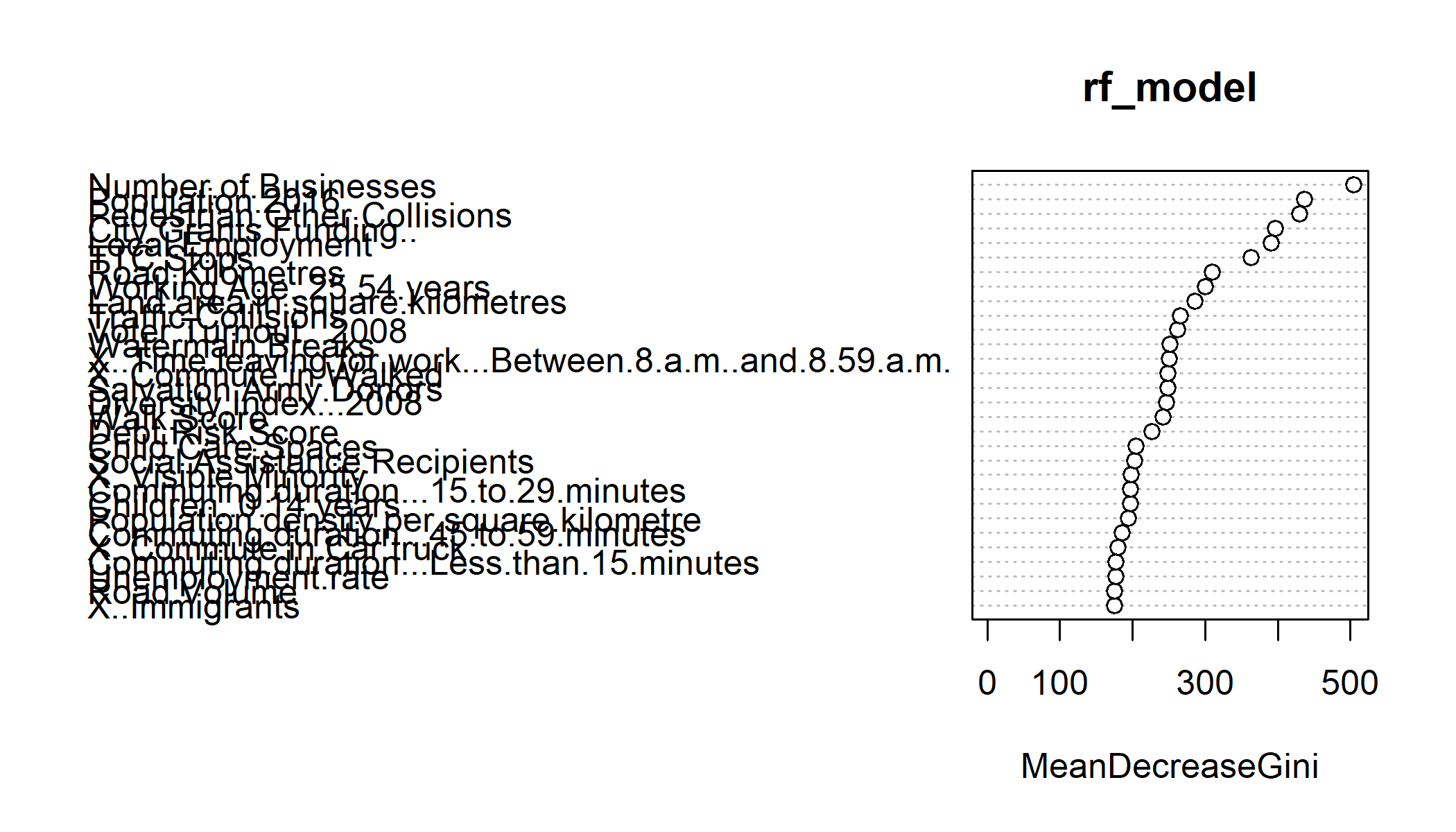


# with an average silhouette of 0.59, the clusters are quite strong. Only cluster 0 is problematic, which we would expect since in DBSCAN, cluster 0 is classified as noise.

## *Using Random Forest to Profile Clusters*

We now have a reliable way to cluster the collision zones in Toronto, but no real idea of what are the shared characteristics of the points within each cluster. The visualization of the clusters hint that location (Scarborough, Downtown Core) play an important role, but what other variables are important to determining a collision’s membership in a cluster? We use Random Forest to find out.

library(randomForest)

# remove cluster 0 since they are classified as noise by DBSCAN - I set all cluster 0 to NA so that I can easily remove them   
train\_dbscan\_17$cluster <- as.numeric(train\_dbscan\_17$cluster)  
train\_dbscan\_17\_clean <- train\_dbscan\_17  
train\_dbscan\_17\_clean$cluster[train\_dbscan\_17\_clean$cluster == 0] <- NA  
train\_dbscan\_17\_clean <- train\_dbscan\_17\_clean[complete.cases(train\_dbscan\_17\_clean),]  
  
# random forest process  
train\_dbscan\_17$cluster <- as.factor(train\_dbscan\_17$cluster)  
names(train\_dbscan\_17) <- make.names(names(train\_dbscan\_17))  
  
# parallel processing  
cl <- makeCluster(3, type = "SOCK")  
registerDoSNOW(cl)  
  
# RF model   
set.seed(123)  
rf\_model <- randomForest(cluster ~ ., data = train\_dbscan\_17)  
varImpPlot(rf\_model)

stopCluster(cl)  
  
# Random Forest suggests that Number of businesses, population of the neighbourhood, city grant funding, TTC stops, total road kilometrage in the neighbourhood, and living in a neighbourhood of working age people all contribute to the cluster characteristics.