Tracking Community Mobility: An R Program for Cleaning and Creating Constructs from GPS Data

Eugene Brusilovskiy
Ess Jaraha
Louis A. Klein
Mark S. Salzer

1 Photo of child walking a dog on a sidewalk with a light shining down only on them

Temple University Collaborative
On Community Inclusion of Individuals with Psychiatric Disabilities

Acknowledgement: The authors are affiliated with the Temple University Collaborative on Community Inclusion of Individuals with Psychiatric Disabilities. The contents of this document were developed under a grant from the National Institute on Disability, Independent Living and Rehabilitation Research (NIDILRR; Grant Number 901F0065-02-00; Salzer, PI). However, the contents do not necessarily represent the policy of the U.S. Department of Health and Human Services, and endorsement by the Federal government should not be assumed.

Cover photo from https://pixabay.com/photos/person-dog-urban-lamp-post-child-498197/ (Free for commercial use)

For technical assistance and support in using this document and approach please contact Eugene Brusilovskiy at eugeneby@temple.edu.
INTRODUCTION

Global positioning systems (GPS) technology offers one promising approach for expanding measurement of health through a greater understanding of community mobility and participation and physical activity. While GPS technology is not new, decreased costs and increased accessibility have made it a promising method of data collection, and a growing body of research from around the world demonstrates the potential utility of GPS technology.

However, current published studies provide limited detail on their analytical approaches, which may be problematic, as GPS data are numerous and complex. For example, data collection at one-minute intervals could yield 1440 time-stamped data points with latitude and longitude values over the course of a 24 hour period, and using these data in a meaningful way is very complicated. However, there are often inaccuracies and errors associated with GPS data, due to factors including, but not limited to, atmospheric conditions, satellite and receiver errors, and multipath errors (i.e., the GPS signal reflecting off tall objects before it reaches the receiver). Certain types of error may be reduced by post-processing GPS data, however these corrections aren’t helpful with receiver or multipath errors, with the latter being especially likely in urban environments (Trimble, 2004)\(^1\). Error also depends on the GPS device that is being used.

Finally, there has been little discussion about how missing data are addressed. As mentioned above, GPS data collection is sensitive to weather (e.g., Hillier, 2008)\(^2\) and satellite access, which can be blocked when indoors (e.g., Harris et al., 2010)\(^3\), and instead of obtaining an incorrect location at certain time points, data may be absent altogether. Human factors, such as not keeping the tracking device charged, can also create missing data. Therefore, missing data are common in such data collection approaches, and require attention.

All those issues notwithstanding, numerous studies have used GPS to create a wide range of constructs related to community mobility, participation and physical activity. This is often tedious task, as it relies on first cleaning the data and then manually creating these constructs.

This program aims to automate these tasks, and to do them for all study participants at once. First, it provides a thorough cleaning of each participant’s GPS data, and

---


addresses common problems, such as the elimination of duplicate points, smoothing of outliers, and imputation of missing data through linear interpolation. Second, it creates a number of constructs from GPS data, doing additional data cleaning and imputation in the process.

These constructs include:

1) Total number of destinations over the course of the study period
2) Total number of unique destinations
3) Total number of non-home destinations
4) Total number of destinations within 1/2 mile of home
5) Total number of destinations within 1 mile of home
6) Activity space area
7) Total distance traveled
8) Amount of time spent at home
9) Amount of time spent outside of home
10) Amount of time spent in transit
11) Amount of time spent at other destinations
12) Total number of non-missing days (i.e., days with fewer than 50% of the data after cleaning and imputation)

The program was designed to work with GPS data that have been collected with a cellular phone using an app called AccuTracking (www.accutracking.com), however it will work with GPS data collected using other apps and stand-alone trackers, as long as the data have been collected at one minute intervals and have the date-time stamp and latitude and longitude coordinates in decimal degrees.
PACKAGES AND CUSTOM FUNCTIONS

This program uses the following packages and extensions:

```r
#rm(list=ls(all=TRUE))
pckgs <- c("geosphere", "dplyr", "zoo", "rgdal", "dbscan", "cluster", "RColorBrewer", "geometry", "grDevices", "sp")
#install.packages(pckgs)
library(geosphere)    # distVincentyEllipsoid
library(dplyr)        # left_join, groupby, between
library(zoo)          # na.approx
library(rgdal)
library(sf)           # project
library(dbscan)       # dbscan
library(cluster)      # silhouette
library(RColorBrewer) # colorRampPalette
library(geometry)     # polyarea
library(grDevices)    # chull
library(sp)           # polygon
library(Jmisc)        # Add column
memory.limit(size=100000)
```

This file contains custom extensions to be run in conjunction with the program:

```r
source("T:\functions14days.r")
```

In the interest of keeping order in variable sheets, these variables are initialized at the very beginning of each run of this program. These reset each sheet to a blank frame and the future analyses fill these variables.

```r
TimeSheet <- data.frame(matrix(ncol = 7, nrow = 0))
DistanceSheet <- data.frame(matrix(ncol = 4, nrow = 0))
OtherSheet <- data.frame(matrix(ncol = 19, nrow = 0))
TotalMissing <- data.frame(matrix(ncol = 2, nrow = 0))
TotalFrequencies <- data.frame(matrix(ncol = 4, nrow = 0))
colnames(TotalFrequencies) <- c("ID", "WithinHalfMile", "WithinFullMile", "GreaterThanMile")
VarOut <- data.frame(matrix(ncol = 23, nrow = 0))
```
HOME ADDRESSES

Before we proceed any further, we need to take the .csv file with all the geocoded home addresses (presented as latitude and longitude), and project it to the same coordinate system that we used earlier. Here, we are using the same options as in the `project_coordinates()` function.

```r
# CSV file containing geocoded home locations (i.e., latitude and longitude coordinates of geocoded addresses)

HomeAll <- read.csv('T:\Addresses for R.csv', header = TRUE, sep = ',')

out_Home = st_as_sf(HomeAll, coords=c(2,3), crs=4269)  # column 2: Longitude (X); Column 3: Latitude (Y)

out2_Home <- st_transform(out_Home, 6564)

Coord_Home <- as.data.frame(st_coordinates(out2_Home))

Projected_Home <- as.data.frame(append(HomeAll, Coord_Home))

Projected_Home$Long_proj <- Projected_Home$X.1

Projected_Home$Lat_proj <- Projected_Home$Y.1

Projected_Home <- Projected_Home[-c(2:5)]  # Keeping ID and projected coordinates only!
```
IMPORTING AND CLEANING THE GPS DATA

First, we define directory and input path information. Specifically, each of the participants has GPS data saved as a .csv file. The format of the file names is “Participant # Cleaned.csv”, where the # is the 4-digit ID of each participant. Each of the files contains, among other variables, 1) the Time Stamp, 2) Latitude, and 3) Longitude of each GPS point.

Then, after the data have been imported, there are three steps to data cleaning: 1) deduplication of data, 2) smoothing of outliers, and 3) imputing missing points between observed points within 20 minutes apart. Each process is explained in detail below.

Deduplication

Each cell phone recorded longitude/latitude readings at approximately 60 second intervals. There were some occasions during this study, however, where readings may have occurred in slightly fewer than 60 second intervals, such as 59 or 58 seconds apart. In these cases, multiple latitude/longitude readings may have been made during the same minute of time. For example, one reading may have taken place at 1:33 PM on the 1st second and 1:33 PM on the 59th second. Since we are considering only one reading per minute in this study, the first observation (1:33:00) is retained and the second observation (1:33:58) is removed from the analysis. This deduplication did not result in any significant loss of participant tracking data.

Here is an example of the data before deduplication:

<table>
<thead>
<tr>
<th></th>
<th>Date</th>
<th>Long</th>
<th>Lat</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10/25/2011</td>
<td>-75.15544</td>
<td>39.98627</td>
</tr>
<tr>
<td>9</td>
<td>10/25/2011</td>
<td>-75.158</td>
<td>39.98344</td>
</tr>
<tr>
<td>10</td>
<td>10/25/2011</td>
<td>-75.15709</td>
<td>39.98102</td>
</tr>
<tr>
<td>11</td>
<td>10/25/2011</td>
<td>-75.15765</td>
<td>39.9779</td>
</tr>
<tr>
<td>12</td>
<td>10/25/2011</td>
<td>-75.16709</td>
<td>39.97892</td>
</tr>
<tr>
<td>13</td>
<td>10/25/2011</td>
<td>-75.15765</td>
<td>39.9779</td>
</tr>
<tr>
<td>14</td>
<td>10/25/2011</td>
<td>-75.16709</td>
<td>39.97892</td>
</tr>
<tr>
<td>15</td>
<td>10/25/2011</td>
<td>-75.16237</td>
<td>39.97841</td>
</tr>
<tr>
<td>16</td>
<td>10/25/2011</td>
<td>-75.15765</td>
<td>39.9779</td>
</tr>
<tr>
<td>17</td>
<td>10/25/2011</td>
<td>-75.15911</td>
<td>39.97798</td>
</tr>
</tbody>
</table>

After the deduplication, only the first of these points was kept, as can be seen below.
Smoothing Outliers

Some points may have resulted from glitches in cellular signal which placed individuals very far from their actual locations. Although these errant observations were rare occurrences, removing the outlier points was a necessary step in order to accurately calculate the different GPS variables, especially distance traveled. In these occasions, the outlier longitude and latitude point was recalculated as the average of the points immediately before and after the outlier.

Before outlier smoothing, we see a point which had incorrect (or truncated) coordinates:
After outlier smoothing, the data point at 20:18 (8:18 PM) has latitude and longitude coordinates which are averages of the coordinates of the 8:17 PM and 8:18 PM data points.

### Imputing Missing Points

During this study, phones may have had brief signal interruptions when participants went inside buildings, underground subway stations, or tunnels. The phones would not be able to record latitude and longitude points while they lost satellite signal. If there was a gap in the data which was 20 or fewer minutes between consecutive points, missing data were imputed through linear interpolation.

For example, before the linear interpolation, the highlighted points below are three minutes apart.
After the linear interpolation, two new points at 0:51 and 0:52 were created. Their coordinates were weighted averages of the coordinates at 0:50 and 0:53.

<table>
<thead>
<tr>
<th></th>
<th>Date</th>
<th>Long</th>
<th>Lat</th>
</tr>
</thead>
<tbody>
<tr>
<td>370</td>
<td>10/27/2011 0:46</td>
<td>-75.219</td>
<td>39.95014</td>
</tr>
<tr>
<td>371</td>
<td>10/27/2011 0:47</td>
<td>-75.21901</td>
<td>39.95014</td>
</tr>
<tr>
<td>372</td>
<td>10/27/2011 0:48</td>
<td>-75.21901</td>
<td>39.95013</td>
</tr>
<tr>
<td>373</td>
<td>10/27/2011 0:49</td>
<td>-75.21901</td>
<td>39.95013</td>
</tr>
<tr>
<td>374</td>
<td>10/27/2011 0:50</td>
<td>-75.21901</td>
<td>39.95013</td>
</tr>
<tr>
<td>375</td>
<td>10/27/2011 0:51</td>
<td>-75.21901333</td>
<td>39.95013</td>
</tr>
<tr>
<td>376</td>
<td>10/27/2011 0:52</td>
<td>-75.21901667</td>
<td>39.95013</td>
</tr>
<tr>
<td>377</td>
<td>10/27/2011 0:53</td>
<td>-75.21902</td>
<td>39.95013</td>
</tr>
<tr>
<td>378</td>
<td>10/27/2011 0:54</td>
<td>-75.21902</td>
<td>39.95013</td>
</tr>
<tr>
<td>379</td>
<td>10/27/2011 0:55</td>
<td>-75.21901</td>
<td>39.95013</td>
</tr>
</tbody>
</table>

# set working directory
setwd("T:\\Data")

# list of paths to participant data files
data_path_list <- list_data_paths("Participant [[:digit:]]4 Cleaned.csv", FALSE)
set.seed(1234)

for (path1 in data_path_list[2]){         # If we want it to loop, we write data_path_list[1:118]

    # participant number and directory name
    participant_number <- substring(path1,15,18)
    #dir_name <- dirname(sub('.','',(sub('.','',path1))))
dir_name <- dirname(path1)

    #Importing the Data
data <- read.csv(path1, stringsAsFactors = FALSE)
cat("n\n", "Importing Participant Data: ",path1)
nrow_a <- nrow(data)

    #Deduplicating the Data
data_x <- data[!duplicated(data[c("Date")]),]
cat("n\n", "number of duplicate records dropped: ", nrow_a-nrow(data_x))    # cat is a printing function

    #Smoothing Outliers
data_x <- id_subset(df="data_x","Date","Long","Lat")
```r
data_x <- as.POSIXct()
data_x <- time_gap_minutes()
data_x$cum_time <- cumsum(data_x$time_gap)

# This uses Vincenty’s Ellipsoid Distance, and reports in meters. No projection needed.
data_x <- distance_meters()
data_x <- smooth_outliers()

# Imputing Missing Points
attributes(data_x$Date)$tzone<- "America/New_York"
start_date <- data_x[1,"Date"]
# define end date
start_time = "00:00:00"
start_date1 <- paste(format(start_date, "%y/%m/%d"), start_time)

start_date <- strftime(start_date1, "%y/%m/%d %H:%M:%S")
attributes(start_date)$tzone <- "America/New_York"

end_date <- (start_date+86400*14)-1        # Daylight savings time can shift end date by an hour

# When Daylight Savings time starts, last time stamp ends up being 23:00 or 23:01, so we change it to 23:59
# start_time <- "23:59:00"
# end_time <- paste(start_datepart, end_time)
# end_time1 <- strftime(end_time, "%y/%m/%d %H:%M:%S")
# attributes(end_time1)$tzone <- "America/New_York"

# When Daylight Savings time ends, we want to make sure that we don’t have an extra day in there due to changing the
# end_time to 23:59:00, so we check to see if the difference between end_time1 and start_date is more than (14 days + an hour and change). If so, we shift the end_time date back by a day
# if (end_time1 > start_date + (86400*14 + 3700)) {end_time1 = end_time1 - 86000}

# create dataframe
date_time <- data.frame("Date" = seq.POSIXt(start_date,end_date,"min",
tz="America/New_York"))```
date_time <- left_join(date_time, data_x, by=c("Date"="Date"))

print("Imputing longitude and latitude where time_gap <= 20")
date_time <- impute("date_time","Long","Lat")
nrow_a <- nrow(date_time)

date_time$Long <- replace(date_time$Long, is.na(date_time$Long), 0)

date_time <- date_time[which(date_time$Long<0),]

cat("\n\n", "Number of observations missing after imputation: ", nrow_a-nrow(date_time))
date_time <- time_gap_minutes(df="date_time")
date_time$cum_time <- cumsum(date_time$time_gap)
date_time <- distance_meters(df="date_time")
file_name <- paste("Participant",participant_number,"Imputed.csv")
file_path <- paste(getwd(),dir_name,file_name,sep="/\")

#Exporting Cleaned Data
cat("\n\n", "Writing csv of imputed dataset: ", file_path)
write.csv(date_time,file_path, row.names = FALSE)
)
##
##  Importing Participant Data:  ./Participant 1002/Participant 1002 Cleaned.csv
##  number of duplicate records dropped: 150[1] "0 outliers smoothed"
## [1] "Imputing longitude and latitude where time_gap <= 20"
##
##  Number of observations missing after imputation:  3700
##
##  Writing csv of imputed dataset: T:/./Participant 1002/Participant 1002 Imputed.csv
We can see the variables that are part of the exported .csv file:

<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>ID</th>
<th>Long</th>
<th>Lat</th>
<th>time_gap</th>
<th>cum_time</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2020-01-03</td>
<td>1</td>
<td>-75.17187</td>
<td>39.97097</td>
<td>0</td>
<td>0</td>
<td>0.000000</td>
</tr>
<tr>
<td>2</td>
<td>2020-01-03</td>
<td>2</td>
<td>-75.17191</td>
<td>39.97110</td>
<td>1</td>
<td>1</td>
<td>14.833408</td>
</tr>
<tr>
<td>3</td>
<td>2020-01-03</td>
<td>3</td>
<td>-75.17187</td>
<td>39.97107</td>
<td>1</td>
<td>2</td>
<td>4.772099</td>
</tr>
<tr>
<td>4</td>
<td>2020-01-03</td>
<td>NA</td>
<td>-75.17184</td>
<td>39.97107</td>
<td>1</td>
<td>3</td>
<td>2.990046</td>
</tr>
<tr>
<td>5</td>
<td>2020-01-03</td>
<td>4</td>
<td>-75.17180</td>
<td>39.97107</td>
<td>1</td>
<td>4</td>
<td>2.990046</td>
</tr>
<tr>
<td>6</td>
<td>2020-01-03</td>
<td>NA</td>
<td>-75.17185</td>
<td>39.97113</td>
<td>1</td>
<td>5</td>
<td>7.691672</td>
</tr>
</tbody>
</table>
ST-DBSCAN is a spatiotemporal data mining algorithm that takes into consideration the spatial and temporal proximity of GPS points to identify spatiotemporal clusters, or destinations. An extension of the DBSCAN clustering algorithm, ST-DBSCAN has three parameters: time, distance, and minimum number of points. Roughly speaking, this means that when there are at least 10 points which are all within 200 meters and 20 min of each other, the individual was in a cluster, or at a destination (Brusilovskiy, Klein, Salzer, 2016)."4"

First, we define the directory and input path information.

Then, we run a for loop which imports the cleaned files. The loop does a few commands for each participant:

1. It projects the coordinates from decimal degrees to meters.
2. It runs ST-DBSCAN with the aforementioned parameters and saves the results.

---

data_x <- read.csv(path2, stringsAsFactors = FALSE, colClasses = c("Date" = "POSIXct"))

#Projecting coordinates from decimal degrees to meters
cat("\n\n", "Projecting coordinates")

data_x <- project_coordinates()
data_x <- subset(data_x, select = -c(X,Y))

#Setting the ST-DBSCAN Parameters and running ST-DBSCAN
#Define output file name and path
param <- data.frame(eps=200, eps2=20, minpts=10)
participant_number <- substring(path2, 15, 18)
file_name <- paste("Participant", participant_number, "Imputed STDBSCAN",
    param$eps, param$eps2, paste(param$minpts, ".csv", sep=""))
particip <- paste("Participant", participant_number, sep=" ")
file_path <- paste(out_dir, particip, file_name, sep="\\")

#Run ST-DBSCAN
print(paste("Running ST-DBSCAN with the following parameters: Eps = ", param$eps, ", Eps2 =", param$eps2, ", minpts = ", param$minpts))

#ST-DBSCAN uses projected coordinates
stdb <- stdbscan(x=data_x$Long_proj, y=data_x$Lat_proj,
    time=data_x$cum_time, eps=param$eps, eps2=param$eps2, minpts=param$minpts)

#Append cluster field to participant data
stdb <- as.data.frame(cbind(data_x, stdb_cluster=stdb$cluster))
stdb$transit = (stdb$stdb_cluster == 0)

#Output csv of results
print(sprintf("writing csv of clustered data (STDBSCAN): %s", file_path))
write.csv(stdb, file_path, row.names = FALSE)

#Let's visualize where the clusters are.
The last few lines of the code chunk above produces a figure such as the one below, which shows ST-DBSCAN results. It is a 3-dimensional plot of latitude, longitude and time, and each color represents a different cluster (with one of the colors representing cluster 0 – i.e., transit points).
ADDITIONAL DATA CLEANING, DBSCAN AND TIME VARIABLES

First, we do some data cleaning to make sure that the clusters given by ST-DBSCAN are non-overlapping. For instance, if consecutive points correspond to clusters 3, 3, 3, 4, 4, 3, 3, 4, 4, 3, the clusters need to be separated out, and the code below addresses these issues. (Note that eventually the clusters may be merged.)

Secondly, because of missing data, the number of clusters identified through ST-DBSCAN is generally an underestimate of the total number of destinations that an individual visited during the study period. As an attempt to correct for this bias, an algorithm was developed to supplement ST-DBSCAN and enable for the identification of additional clusters and modify existing ones in the presence of missing data. This missing data algorithm focuses on four distinct scenarios.

In the first scenario, there are two consecutive transit points, called A and B, whose distance from one another is less than eps1 distance units (200 m), but is more than eps2 time units (20 min) apart. Although these points are not placed within a cluster by STDBSCAN due to the value of the eps2 parameter (and possibly the minpts parameter), the supplementary algorithm puts these two points into a cluster if they are less than 12 hours (720 minutes) apart. This is because it is likely that participants went inside a building directly after point A, lost satellite reception, and were inside the building (i.e., at a destination) until reappearing at point B. Twelve hours as the maximum time was selected because larger values might increase the likelihood that an individual left the building between points A and B, but either did not carry or charge the phone.

The remaining three scenarios do not identify new clusters, but either merge existing clusters with transit points, or merge two existing clusters into one. This is needed not for the identification of destinations, but for a more accurate calculation of the temporal variables, such as time spent outside of home or in transit, described below.

In the second scenario, two consecutive points are identified: A is a transit point and B is part of an ST-DBSCAN destination cluster. As in scenario 1, these points are within a distance of eps1 units, but more than eps2 time units apart. An assumption of this
scenario is that the person went inside after point A and did not have good satellite reception until point B. Therefore, if A and B are less than 12 hours apart, transit point A is placed in the cluster which contains B.

The third scenario is similar to the second, except the roles of points A and B are reversed: A is part of an ST-DBSCAN cluster and B is a transit point. If A and B are less than 12 hours apart, transit point B is placed in the cluster which contains A.

In the fourth scenario, two consecutive points A and B are within a distance of eps1, but fall within two separate ST-DBSCAN clusters (1 and 2) because they are more than eps2 time units apart. If points A and B are within 12 hours of one another and the centroids of clusters 1 and 2 are within eps1 (200 m), the algorithm combines the clusters 1 and 2 into a single cluster.
After that, we identify all ST-DBSCAN clusters which correspond to home locations. These are defined as ST-DBSCAN clusters whose centroids are within 200 meters of the geocoded home address. All other destinations are considered to be non-home destinations, and all locations that have the ST-DBSCAN cluster 0 are transit points. Based on that, we can easily calculate the following variables:

1. Time at home destinations
2. Time at non-home destinations
3. Time in transit
4. Missing time, defined as the total of 20160 minutes - (Time at home + Time at other destinations + Time in transit)

Then, we calculate the number of unique destinations for each individual. Individuals may come to some destinations more than once over the course of the study. Such destinations might include their home, work, or treatment providers. To identify unique destinations, we run the DBSCAN clustering algorithm on centroids of ST-DBSCAN clusters. This algorithm uses two parameters, eps (which is the distance between points, which we set to 200 meters as in ST-DBSCAN), and MinPts, which is the minimum number of points needed to form a cluster (which we set to two). Unique destinations are then calculated as the sum of 1) the number of DBSCAN clusters and 2) the number of destinations in cluster 0 (i.e., destinations which only appeared once over the course of the study).

Here’s a plot of the DBSCAN clusters
In addition, we also identify which of the post-processed clusters are home clusters. A home cluster is a cluster whose centroid is within 200 meters of the geocoded home address. We also calculate:

- Time spent at home (we compute the difference between the start and end times for each home cluster and sum across all home clusters)

- Time spent at non-home destinations (we compute the difference between the start and end times for each non-home cluster and sum across all non-home clusters)

- Time in transit (we compute the number of rows in the data where cluster = 0)

- Non-missing time (calculated as the sum of Time at Home, Time at Other Destinations, and Time in Transit)

- Missing time (calculated as 20160 minutes/14 day period - Non-missing Time)

- Activity Space Area (computed as the convex hull around the points). Below is a plot of a sample activity space.
- Distance Traveled (calculated as the distance between all consecutive points that aren’t within the same destination)
- Number of non-missing days (days with 720+ minutes, or 50%+ of data)
- Median Daily Activity Space Area (for non-missing days)

```r
# Set Working Directory
setwd("T:\\")

library(Jmisc)
library(dplyr)
library(fuzzyjoin)

# Make list of paths to data files
data_path_list3 <- list_data_paths("Participant [[:digit:]]{4} Imputed STDBSCAN [[[:digit:]]{3} [[[:digit:]]{2,4} [[[:digit:]]{1,2}.csv",TRUE)

# Directory for exporting dbscan csv
out_dir <- "T:\\Data"

data_x <- read.csv(data_path_list3[1], stringsAsFactors = FALSE, colClasses = c("Date" = "POSIXct"))

# Scenarios 1-4
for (path3 in data_path_list3[2]){
  participant_number <- substr(path3,15,18)
}
dir_name <- dir_name(path3)
cat("\n\n", "PARTICIPANT", participant_number)

# Import data
data_x <- read.csv(path3, stringsAsFactors = FALSE, colClasses = c("Date" = "POSIXct"))

# Record number of unique stdb_clusters before postprocessing
clust_pre <- sort(unique(data_x$stdb_cluster))
num_clust_pre <- length(clust_pre)

#############################################################
##### PRELIMINARY DATA CLEANING ######
#############################################################

# Sometimes ST-DBSCAN places transit points between points that are within a cluster.
# We will make all these transit points be part of the cluster.
# If a cluster is completely within another non-transit cluster, we need to split up the cluster that it is within into two clusters.

data_x$cluster <- data_x$stdb_cluster

# If:
# 1) The point is not in transit cluster, and
# 2) The previous point IS NOT in the same cluster, and
# 3) Some of the earlier points (10+ observations ago) ARE in the same cluster as the original point, then:
# Create new cluster that equals to the original cluster*1000000
for (i in seq(21, nrow(data_x))){
  if (data_x$cluster[i]!=0 &
      data_x$cluster[i-1]!=0 &
      data_x$cluster[i]!=data_x$cluster[i-1] &
      (#data_x$cluster[i]==data_x$cluster[i-2] |
      #data_x$cluster[i]==data_x$cluster[i-3] |
      #data_x$cluster[i]==data_x$cluster[i-4] |
# If:

1) The cluster ID of the point after operation above is the same as the cluster ID before the operation above, and

2) The cluster ID of the point equals the cluster ID of the previous point divided by 1000000 (that is, if previous point was changed by the operation above), then:

Merge the point into the same cluster as the previous point (i.e., assign it the same cluster ID as the previous point)

#That is, if the point's cluster membership wasn't changed, but the previous point's cluster membership WAS changed,

#Change the point's cluster membership

for (i in seq(2, nrow(data_x))) {
  if (data_x$cluster[i] == data_x$stdb_cluster[i] &
      data_x$cluster[i] == data_x$cluster[i-1]/1000000) {
    data_x$cluster[i] = data_x$cluster[i]*1000000
  }
}
Sometimes, after all the cleaning above, there are still some clusters which only contain 1 point.

We will convert these points to 0 (transportation) clusters.

```r
for (i in seq(2, nrow(data_x)-1)){
  if (data_x$cluster[i]!=0 &
      data_x$cluster[i-1]==0 &
      data_x$cluster[i+1]==0){
    data_x$cluster[i]=0
  }
}
```

Removing the cluster variable

```r
data_x$stdb_cluster = data_x$cluster
data_x$cluster <- NULL
```

Let's look at maximum and minimum time by cluster. If the operations above aren't done, then the min and max times can be confusing.

```r
cumtime.max <- as.data.frame(aggregate(data_x$cum_time,list(data_x$stdb_cluster),max))
colnames(cumtime.max) <- c('stdb_cluster','CumTimeMax')
cumtime.max <- cumtime.max[which(cumtime.max$stdb_cluster!=0),]

cumtime.min <- as.data.frame(aggregate(data_x$cum_time,list(data_x$stdb_cluster),min))
colnames(cumtime.min) <- c('stdb_cluster','CumTimeMin')
cumtime.min <- cumtime.min[which(cumtime.min$stdb_cluster!=0),]

cumtime <- cbind(cumtime.min, cumtime.max)[,-3]
```

If time stamp of a point is between minimum and maximum time of a cluster, then we want to assign that point to that cluster.

```r
data_x_cumtime <- fuzzy_left_join(data_x,cumtime, by=c("cum_time"="CumTimeMin", "cum_time"="CumTimeMax"),
                                  match_fun=list(`>=`,'<='))
```

Replace NA cluster ID in stdb_cluster.y with 0
data_x_cumtime$stdb_cluster.y <- replace(data_x_cumtime$stdb_cluster.y, is.na(data_x_cumtime$stdb_cluster.y), 0)

data_x_cumtime$stdb_cluster <- data_x_cumtime$stdb_cluster.y

#Removing unnecessary variables
data_x_cumtime[, c('stdb_cluster.y', 'stdb_cluster.x', 'CumTimeMin', 'CumTimeMax')] <- list(NULL)

data_x <- data_x_cumtime

#Defining transit variable as those where cluster = 0
data_x$transit = (data_x$stdb_cluster == 0)

###############################
#####         SCENARIO 1         ######
###############################
cluster_list <- c()

for (i in seq(2,nrow(data_x))){
  if(between(data_x$time_gap[i],20,720) & data_x$distance[i] < 200 & data_x$stdb_cluster[i]==0 & data_x$stdb_cluster[i-1]==0 & is.na(data_x$ID[i])==FALSE & is.na(data_x$ID[i-1])==FALSE){
    stdb_cluster_id <- max(data_x$stdb_cluster)+1 # create new stdb_cluster number
    data_x$stdb_cluster[i] <- stdb_cluster_id # assign new stdb_cluster number to current obs
    data_x$stdb_cluster[i-1] <- stdb_cluster_id # assign new stdb_cluster number to prev obs
    cluster_list<- c(cluster_list,stdb_cluster_id)
  }
}
}
cat("\n\n", "Scenario 1: Number of stdb_clusters created:", length(cluster_list))
cat("\n\n", "ID(s) of stdb_clusters created:", cluster_list)


df <- data_x
df$NextClus <- lead(df$stdb_cluster)
df$NextDist <- lead(df$distance)
df$NextGap <- lead(df$time_gap)

clustered_obs <- 0
for (i in seq(1,nrow(df)-1)){
  if(between(df$NextGap[i],0,720) & df$NextDist[i] < 200 & df$stdb_cluster[i]==0 & df$NextClus[i]!=0){
    stdb_cluster_id2 <- df$NextClus[i]
    df$stdb_cluster[i] <- stdb_cluster_id2 # add current obs to stdb_cluster of next obs
    cat("\n\n","Scenario 2: Adding observation ID ",df$ID[i]," to stdb_cluster",stdb_cluster_id2)
    clustered_obs <- clustered_obs + 1
  }
}
if (clustered_obs == 0){
  cat("\n\n", "Scenario 2: stdb_clusters unchanged")
}
data_x <- df


df <- data_x
df$LagClus <- lag(df$stdb_cluster)
df$LagGap <- lag(df$time_gap)

clustered_obs <- 0
for (i in seq(2,nrow(df))){
    if(between(df$time_gap[i],0,720) & df$distance[i] < 200 & df$stdb_cluster[i]==0 & df$LagClus[i]!=0){
        stdb_cluster_id3 <- df$LagClus[i]
        df$stdb_cluster[i] <- stdb_cluster_id3
        cat("\n
","Scenario 3: Adding observation ID ",df$ID[i]," to stdb_cluster",stdb_cluster_id3)
        clustered_obs <- clustered_obs + 1
    }
}
if (clustered_obs == 0){
    cat("\n
","Scenario 3: stdb_clusters unchanged")
}
data_x <- df

# SCENARIO 4

#(Code for Scenario 4 in the functions.r file is incorrect, but it is correct here)

df <- data_x

#Calculate centroids of each ST-DBSCAN cluster
centroids <- as.data.frame(aggregate(data_x[,8:9],list(data_x$stdb_cluster),mean))
colnames(centroids) <- c('stdb_cluster','CentroidLongProj','CentroidLatProj')

#Calculate the minimum date in each ST-DBSCAN cluster
centroids_date <- as.data.frame(aggregate(data_x[,1],list(data_x$stdb_cluster), min ))
colnames(centroids_date) <- c('stdb_cluster','CentroidMinDate')

#Merge the minimum date and centroids into a single data set
centroids_fin <- cbind(centroids, centroids_date)

centroids_fin <- centroids_fin[, -4]  # Removing the second stdb_cluster variable

# Removing observations from cluster 0
centroids_fin <- centroids_fin[which(centroids_fin$stdb_cluster != 0),]

# Sorting by time
centroids_fin1 <- centroids_fin[order(centroids_fin$CentroidMinDate),]

# Creating centroids of next cluster
centroids_fin1$NextCentroidLatProj = lead(centroids_fin1$CentroidLatProj)
centroids_fin1$NextCentroidLongProj = lead(centroids_fin1$CentroidLongProj)

# Calculating distance between current cluster centroids and next cluster centroids
centroids_fin1$DistToNextNon0Cluster <- sqrt((centroids_fin1$CentroidLongProj-centroids_fin1$NextCentroidLongProj)**2 +
                              (centroids_fin1$CentroidLatProj-centroids_fin1$NextCentroidLatProj)**2)

# Before merging, create a new data frame with the 0 row
centroids_fin1[nrow(centroids_fin1)+1,] <- NA

# Replace NA cluster ID with 0
centroids_fin1$stdb_cluster<-replace(centroids_fin1$stdb_cluster, is.na(centroids_fin1$stdb_cluster), 0)

# Replace NA distance with 1000000
centroids_fin1$DistToNextNon0Cluster<-replace(centroids_fin1$DistToNextNon0Cluster, is.na(centroids_fin1$DistToNextNon0Cluster), 1000000)

# Let's merge this centroid data frame with the original point level data frame, by cluster ID, and then sort by time
data_x_merge <- merge(data_x, centroids_fin1, by="stdb_cluster")
data_x_merge1 <- data_x_merge[order(data_x_merge$Date),]

data_x_merge1$stdb_cluster_id4 <- data_x_merge1$stdb_cluster

# Let's do the cluster merging
for (i in seq(1, nrow(data_x_merge)-1)){

if (data_x_merge1$stdb_cluster[i] != 0 #If current
cluster is a non-transit cluster
& data_x_merge1$NextClus[i] != 0 #Next cluste
r is a non-transit cluster
& data_x_merge1$stdb_cluster[i] != data_x_merge1$NextClus[i] #Current clu
ster is not equal to next cluster
& data_x_merge1$DistToNextNon0Cluster[i] < 200
& data_x_merge1$time_gap[i+1] <= 720) {
    #Distance between clus
ters is < 200m
    data_x_merge1$stdb_cluster_id4[i] = data_x_merge1$NextClus
    [i]
}

#Now, we are going to take the maximum value of each new cluster (stdb_cluster_id4)
within each old cluster
scenario4 <- as.data.frame(aggregate(data_x_merge1$stdb_cluster_id4,list(data_x_mer
ge1$stdb_cluster),max))
#scenario4$x<-replace(scenario4$x, 3, 3)
#scenario4$x<-replace(scenario4$x, 2, 2)
#Here, each old cluster is called stdb_cluster and each new cluster is called x
colnames(scenario4) <- c('stdb_cluster','x')
#Let's duplicate variable x and call it stdb_cluster_id4fin
scenario4$stdb_cluster_id4fin <- scenario4$x

#For each observation, if stdb_cluster equals to x in the previous observation, var
iable stdb_cluster_id4fin should
#take on its own value from the previous observation. Doing it this way enables the
merging of more than two
#consecutive clusters.
for (i in seq(2,nrow(scenario4))){
    if (scenario4$stdb_cluster[i]==scenario4$x[i-1])
        { #Distance between clusters is < 200m
            scenario4$stdb_cluster_id4fin[i] = scenario4$stdb_cluster_id4fin[i-1]
        }
}

scenario4 <- scenario4[,c(-2)] #Removing variable x
#Merging with point-level data which will now have variable stdb_cluster_id4fin (cluster ID after scenario 4)

data_x_merge2 <- merge(data_x_merge1, scenario4, by="stdb_cluster")
data_x <- data_x_merge2[order(data_x_merge2$Date),]  #Sorting by date

#Subtracting 1 because of transportation cluster

cat("\n\n","Scenario 4: Number of clusters before merging of consecutive clusters:" ,
    length(unique(data_x$stdb_cluster))-1)
cat("\n\n","Scenario 4: Final number of clusters after merging of consecutive clusters: ",
    length(unique(data_x$stdb_cluster_id4fin))-1)
cat("\n\n","Scenario 4: Number of clusters merged: ",
    length(unique(data_x$stdb_cluster))-length(unique(data_x$stdb_cluster_id4fin)))

#Updating transit and stdb_cluster variables

data_x$transit = (data_x$stdb_cluster == 0)
data_x$stdb_cluster <- data_x$stdb_cluster_id4fin
df <- data_x

#Printing Messages

# Final number of unique stdb_clusters after scenarios 1-4
clust_post <- sort(unique(data_x$stdb_cluster))
num_clust_post <- length(clust_post)-1

#Updating the NextClus and LagClus variables

data_x$NextClus = lead(data_x$stdb_cluster)
data_x$LagClus = lag(data_x$stdb_cluster)

######################################
##### IDENTIFYING HOME LOCATIONS #####
######################################

#We are dealing with the participant whose ID equals to the Participant Number. (That is, we are selecting the
#appropriate ID number)

Home <- subset(Projected_Home[which (HomeAll$ID == as.numeric(as.character(participant_number)))] , )
#Participant's Home location

```r
colnames(Home) <- c('ID','Long','Lat')
Home1 <- data.matrix(Home[,2:3])
```

#Calculate distance between projected home location and projected ST-DBSCAN cluster centroids
#Averaging Lat_Proj and Long_Proj by cluster
```r
centroids <- as.data.frame(aggregate(data_x[,9:10],list(data_x$stdb_cluster),mean))
colnames(centroids) <- c('stdb_cluster', 'Long_proj', 'Lat_proj')
centroids$Home_Lat = rep(Home1[2], nrow(centroids))
centroids$Home_Long = rep(Home1[1], nrow(centroids))
```

#Identify each ST-DBSCAN cluster as a home or non-home cluster based on whether its centroid is within 200 m of geocoded home location.
```r
centroids$dist <- sqrt(((centroids$Lat_proj-centroids$Home_Lat)**2)+((centroids$Long_proj-centroids$Home_Long)**2))
```

#Lat/Long Centroid less than or equal to 200 meters from home, should be a home location.
#If it's cluster 0, it's the transit cluster, so we're not interested in it (hence > 0)
```r
HomeClusters <- as.data.frame(centroids[which(centroids$dist < 200 & centroids$stdb_cluster > 0),])
HomeClusters$Cluster <- rep("Home", nrow(HomeClusters))
cat("\n\n", "Participant",participant_number,"had", nrow(HomeClusters), "home clusters.")
```

```r
NonHomeClusters <- as.data.frame(centroids[which(centroids$dist > 200 & centroids$stdb_cluster > 0),])
NonHomeClusters$Cluster <- rep("NonHome", nrow(NonHomeClusters))
cat("\n\n", "Participant",participant_number,"had", nrow(NonHomeClusters), "non-home clusters.")
```

```
###############################################################################
#####  COMPUTING TIME VARIABLES  #######
###############################################################################

#Prepping the data for DBSCAN Analyses below.
DBSCANHome <- rbind(HomeClusters,NonHomeClusters)
#DBSCANHome <- DBSCANHome[c(-1)]
#DBSCANHome <- DBSCANHome[which(DBSCANHome$stdb_cluster>0),]

#Calculate Time Spent at Each Cluster
#Takes the first element of each non-transit cluster
data_x_start <- data_x[!duplicated(data_x$stdb_cluster) & data_x$stdb_cluster > 0,]

#Takes the first element of each reversed non-transit cluster (i.e., the last element of each cluster)
data_x_end <- data_x[rev(!duplicated(rev(data_x$stdb_cluster))) & data_x$stdb_cluster > 0,]

#Prepping data to have start and end times in a single data frame
data_x_start$start_time <- data_x_start$cum_time
data_x_end$end_time <- data_x_end$cum_time

start_end <- as.data.frame(data_x_start[c(1,25)]) #stdb_cluster and start_time
start_end$end_time = data_x_end$end_time

#Total Time variable is the amount of time spent in each cluster (end time - start time + 1 minute)
start_end$TotalTime <- (start_end$end_time - start_end$start_time) + 1
DBSCANHomeTimes<- full_join(start_end,DBSCANHome, by = "stdb_cluster")

#We want to calculate the total time spent at home and out of home.
#The aggregate command creates the table that looks like this:
#Group.1    x
#Home      [Total Time At Home]
#NonHome   [Total Time At Non-Home Destinations]

#So here:
#[1,2] indicates that we're looking at time at home, which is in the 1st row, 2nd column, and
#[2,2] indicates that we're looking at time at non-home destinations, which is in the 2nd row, 2nd column.

#Time at Home
SumHome <- aggregate(DBSCANHomeTimes$TotalTime, by=list(DBSCANHomeTimes$Cluster), sum)[1,2]

cat("\n\n"., "Participant",participant_number,"spent", SumHome, "minutes at home des-
toninations.")

#Time at Non-Home
SumNonHome <- aggregate(DBSCANHomeTimes$TotalTime, by=list(DBSCANHomeTimes$Cluster), sum)[2,2]

cat("\n\n"., "Participant",participant_number,"spent", SumNonHome, "minutes at non-
ome destinations.")

#Time in Transit is simply calculated as the number of rows in the data_x data frame
# where STDBSCAN Cluster = 0.
TransitTime <- as.data.frame(subset(data_x, stdb_cluster == 0))

cat("\n\n"., "Participant",participant_number,"spent", nrow(TransitTime), "minutes i
n transit.")

#Putting all the Time variables into a single data frame
TimeX <- data.frame(matrix(ncol = 7, nrow = 0))
TimeX <- as.data.frame(rbind(c("ID" = participant_number,"HomeMins" = SumHome,"NonH-
omeMins" = SumNonHome,"TransitMins" = nrow(TransitTime), "Perc
entHome"= round(SumHome/sum(SumHome,SumNonHome,nrow(Transit
ime)),4),"PercentNonHome"= round(SumNonHome/sum(SumHome,SumNonHome,nrow(Transi
Time)),4),"PercentTransit"=
round(nrow(TransitTime)/sum(SumHome,SumNonHome,nrow(Transit
ime)),4))))

TimeSheet <- rbind(TimeX, TimeSheet)
TimeSheet <- distinct(TimeSheet, ID, .keep_all=TRUE)
colnames(TimeSheet) <- c("ID","HomeMins","NonHomeMins","TransitMins","Percen
tHome","PercentNonHome","PercentTransit")

#Now it's time to use DBSCAN to compute the number of unique Destinations.
cat(\n\n"., "Running DBSCAN...")
#Computing a Euclidean distance matrix (in meters) between all the ST-DBSCAN cluster centroids

dist_xy <- as.matrix(dist(DBSCANHomeTimes[,5:6], method="euclidean", diag=T))

#Running DBSCAN with parameters eps = 200 and MinPts = 2
db <- fpc::dbscan(dist_xy, eps = 200, MinPts = 2, scale=FALSE, method="dist")

#Add dbscan clusters to a new data frame called dbscan_x
dbscan_x <- cbind(DBSCANHomeTimes, db$cluster)

#Calculate the number of times each cluster appears (this is really only needed for clusters labeled as 0)
dbscan_x$num <- ave(dbscan_x$stdb_cluster, dbscan_x$`db$cluster`, FUN=seq_along)

#Number of Unique Clusters is the # of clusters 1..n + the number of 0 clusters (which are also unique destinations)
#Sometimes there are no 0 clusters, in which case, R yields -Inf as the response, and the number of unique destinations is not calculated.
test <- max(dbscan_x$num[dbscan_x$`db$cluster`==0])
test1 <- ifelse(is.infinite(test),0,test)
NumUniqueDest <- sum(max(dbscan_x$`db$cluster`, na.rm=TRUE), test1, na.rm=TRUE)
cat("\n\n", "Number of unique destinations: ", NumUniqueDest)

#Each cluster given its final DBSCAN cluster number. Cluster 1 is the home cluster.
dbscan_x$findbscanclust <- dbscan_x$`db$cluster` #Taking it from the cluster element in the db list

#If cluster = 0, then cluster ID will be # of 0 clusters + the time (1st, 2nd, 3rd, etc.) it appears in the data
dbscan_x$findbscanclust[dbscan_x$`db$cluster`==0] <- dbscan_x$num[dbscan_x$`db$cluster`==0] +
   max(dbscan_x$`db$cluster`)
We also calculate the total distance traveled (in km.) over the course of the study, as well as activity space area

in sq. km.) The activity space area is calculated using the convex hull command in R.

First, we calculate the total Distance Traveled (in km). Here, we are only calculating the distances between points

which aren't in the same cluster.

dist_traveled <- data_x[which(data_x$stdb_cluster == 0 | data_x$NextClus != data_x$stdb_cluster),]

This is like the dist2_km variable in our ArcGIS/SAS files

total_distance_km = sum(dist_traveled$NextDist, na.rm=TRUE)/1000

cat("\n\n", "Total distance (in km) that participant traveled:",total_distance_km)

Now, let's calculate the activity space area.

We create a dataframe of timestamp, unprojected xy, and projected xy

dataxProj <- as.data.frame(cbind(Date = as.Date(as.numeric(as.POSIXct(data_x$Date)),origin="1990-01-01"), Long =

 data_x$Long,Lat = data_x$Lat,Long_Proj = data_x

$Long_proj,Lat_Proj =

 data_x$Lat_proj))

outpath1 <- paste("S:/RRTC/RRTC 2013-2018/GPS Study - Field Initiated/Data/AccuTracking

Data/ArcMap/Proj",participant_number,".csv",sep="")

write.csv(dataxProj, file_path)

We create a list containing the indices of coordinates that make the convex hull

cchull_indx <- chull(dataxProj[,4:5])

We append the first index to the end of the list (to create a closed loop)

cchull_indx <- c(chull_indx, chull_indx[1])

We extract the coordinates of the convex hull from the matrix of xy coords

cchull_pts <- dataxProj[cchull_indx,4:5]

We plot convex hull as polygon and print area

plot(chull_pts, type="n")

chull_poly <- polygon(chull_pts)

chull_area <- polyarea(chull_pts[,2]/1000,chull_pts[,1]/1000) #converting meters to km#
cat("\n\n", "The activity space area is", chull_area, " sq. km.")

#############################################################
####        MISSING DAYS        ####
#############################################################

# A day is considered missing if more than 50% of the data (i.e., more than 
# 720 minutes) are missing.
# MissingDays
# 86400 = # seconds in a day

# Cluster Lag
data_x$stdb_cluster_lag <- lag(data_x$stdb_cluster)
# Replacing the NA in the first observation with 0
data_x$stdb_cluster_lag <- replace(data_x$stdb_cluster_lag, is.na(data_x$stdb_cluster_lag), 0)

# Cluster Lead
data_x$stdb_cluster_lead <- lead(data_x$stdb_cluster)
# Replacing the NA in the last observation with 0
data_x$stdb_cluster_lead <- replace(data_x$stdb_cluster_lead, is.na(data_x$stdb_cluster_lead), 0)

# Cluster Min and Max Dates
# Let's expand our data frame to include all missing minutes
data_x$timestamp <- as.POSIXct(data_x$Date, format="%m/%d/%y %H:%M:%S")
# [-1, ] removes the zero cluster

MinMaxTimeByCluster <- cbind(MinTimeByCluster, MaxTimeByCluster)

# Let's expand the data frame to include every possible value of cum_time

cat("\n\n", "The activity space area is", chull_area, " sq. km.")
df <- data.frame(seq(data_x$cum_time[1], data_x$cum_time[nrow(data_x)]))
colnames(df) <- "cum_time"
fin <- dplyr::full_join(df, data_x, by=c("cum_time"="cum_time"))
data_x <- fin

# If cum_time in data_x is between min and max time, merge between (use fuzzy join) by cum_time

df <- fuzzy_left_join(data_x, MinMaxTimeByCluster, by=c("cum_time"="MinTimeByCluster",
"cum_time"="MaxTimeByCluster"), match_fun=list(">=", "<="))

df$stdb_cluster <- df$stdb_cluster.x
df$stdb_cluster.x <- NULL
df$stdb_cluster.y <- NULL

# Replacing NA stdb_cluster with some negative value so that it's not missing
df$stdb_cluster <- replace(df$stdb_cluster, is.na(df$stdb_cluster), -5)

# Test is a variable that takes on a value of 1 if Min & Max cumulative time are missing and the observation isn't in
# a transit cluster

df$test <- ifelse(df$stdb_cluster == 0, 0, ifelse(is.na(df$MinTimeByCluster) & is.na(df$MaxTimeByCluster), 1, 0))

nrow_df = nrow(df)

# Removing all variables where test = 1
df <- df[!(df$test == 1),]
data_x <- df

# nonmissing2 <- nrow(data_x)
# missing2 <- 20160 - nrow(data_x)
# transit2 <- nrow(data_x) - (SumHome + SumNonHome)

# Because we did the merging by cum_time, some of the time stamps were missing. We can calculate the
# final time stamp by taking the time stamp of the first observation and adding cum_time*60 minutes to it.
data_x$timestamp.final <- data_x$timestamp[1] + (data_x$cum_time*60)
attributes(data_x$timestamp.final)$tzone <- "America/New_York"
data_x$DayOfMonth <- as.numeric(format(as.Date(data_x$timestamp.final, format="%Y-%m-%d %H:%M:%S", tz="America/New_York"), format="%d"))

df <- transform(data_x, Day=as.numeric(factor(DayOfMonth)))
data_x <- df

######################################
#####      BY DAY VARIABLES      ######
######################################

# create a subset grouped by day for each participant
MissingDays <- data.frame(matrix(ncol = 1, nrow = 0))
ParticipantMissing<- data.frame(matrix(ncol = 1, nrow = 1))
MedianActSpace <- data.frame(matrix(ncol = 1, nrow = 1))
Days <- unique(data_x$Day)
MinsPerDay <- data.frame(matrix(ncol=1, nrow=1))

# We create the median of activity space areas for non-missing days. That is, for each calendar day that is not
# missing, we calculate the activity space area, and then calculate the median of those areas.

# Daily Variables
DayAreas <- matrix(ncol = 1, nrow = 0)
for (i in Days){
  Day <- data_x[data_x$Day == i,]
  DayX <- max(Day$cum_time, na.rm=TRUE) - min(Day$cum_time, na.rm=TRUE) + 1
  MinsPerDay <- rbind(MinsPerDay, DayX)

  # If we have fewer than 720 minutes, it's a missing day!
  if (DayX < 720) {
    Miss = 1
    chull_area1 <- NA # Area is missing
  } else {
    Miss = 0
  }
  # create a list containing the indices of coordinates that make the convex hull
Day1 <- Day[!is.na(Day$Long_proj),]
chull_index <- chull(Day1[,8:9])
# append the first index to the end of the list (to create a closed loop)
chull_index <- c(chull_index, chull_index[1])
# extract the coordinates of the convex hull from the matrix of xy coords
chull_Day <- Day1[chull_index,8:9]
chull_area1 <- polyarea(chull_Day[,2]/1000,chull_Day[,1]/1000) # converting meters to km#
}

# Creating the relevant variables
DayAreas <- rbind(DayAreas,chull_area1)
MedianActSpace <- median(DayAreas, na.rm=TRUE)
MissingDays <- rbind(MissingDays,Miss)

MinsPerDay.df <- as.data.frame(MinsPerDay[-1,])

cat("\n\n", "The median daily activity space area is", MedianActSpace, "sq. km.")
cat("\n\n", "The number of missing days is", (14-length(Days))+sum(MissingDays$X0))
cat("\n\n", "The number of non-missing days is", nrow(MissingDays) - sum(MissingDays$s$X0))

#################################################################
####      OTHER CALCULATIONS      ####
#################################################################

NonMissingMins <- sum(SumHome, SumNonHome, nrow(TransitTime))*1
# NonMissingPts <- nrow(data_x)
# DiffMinsPts <- NonMissingMins - NonMissingPts
MissingMins <- 20160 - NonMissingMins
MinsOutsideHome <- SumNonHome + nrow(TransitTime)
NumDestinations <- nrow(DBSCANHomeTimes)*1
NumNonHomeDest <- nrow(NonHomeClusters)*1

cat("\n\n", "Participant",participant_number," had", NonMissingMins, "non-missing minutes.")
Participant had MissingMins missing minutes.

#Putting everything in a single data frame
OtherX <- data.frame(matrix(ncol = 18, nrow = 0))
OtherX <- as.data.frame(rbind( c("ID" = participant_number, "NonMissingMinutes" = NonMissingMins, "MissingMinutes" = MissingMins, "HomeDestMinutes" = SumHome, "NonHomeDestMinutes" = SumNonHome, "TransitMinutes" = nrow(TransitTime), "OutsideOfHomeMinutes" = MinsOutsideHome, "TotalMinutes" = sum(MissingMins, SumHome, SumNonHome, nrow(TransitTime)), "NonMissingPts" = nrow(data_x[!is.na(data_x$Long_proj)],) ), "TotalDestinations" = NumDestinations, "HomeDestinations" = nrow(HomeClusters), "NonHomeDestinations" = nrow(NonHomeClusters), "UniqueDestinations" = NumUniqueDest, "DistanceTraveledKm" = total_distance_km, "ActivitySpaceAreaSqKm" = chull_area, "TotalDays" = length(Days), "NonMissingDays" = nrow(MissingDays) - sum(MissingDays$X0), "MissingDays" = (14-length(Days))+sum(MissingDays$X0), "MedianDailyActSpace" = MedianActSpace)))

#These two commands are needed when we're running the loop for multiple participants
OtherSheet <- rbind(OtherX, OtherSheet)
OtherSheet <- distinct(OtherSheet, ID,.keep_all=TRUE)
## PARTICIPANT 1002

### Scenario 1: Number of stdb_clusters created: 4

### ID(s) of stdb_clusters created: 68000001 68000002 68000003 68000004

### Scenario 2: Adding observation ID 319 to stdb_cluster 2

### Scenario 2: Adding observation ID 345 to stdb_cluster 3

### Scenario 2: Adding observation ID NA to stdb_cluster 10

### Scenario 2: Adding observation ID 484 to stdb_cluster 11

### Scenario 2: Adding observation ID 3087 to stdb_cluster 13

### Scenario 2: Adding observation ID 3140 to stdb_cluster 16

### Scenario 2: Adding observation ID 3174 to stdb_cluster 17

### Scenario 2: Adding observation ID 3202 to stdb_cluster 18

### Scenario 2: Adding observation ID NA to stdb_cluster 23

### Scenario 2: Adding observation ID 3703 to stdb_cluster 25

### Scenario 2: Adding observation ID 4211 to stdb_cluster 28

### Scenario 2: Adding observation ID 4263 to stdb_cluster 29

### Scenario 2: Adding observation ID NA to stdb_cluster 39

### Scenario 2: Adding observation ID 5644 to stdb_cluster 42
Scenario 2: Adding observation ID NA to stdb_cluster 43

Scenario 2: Adding observation ID NA to stdb_cluster 45

Scenario 2: Adding observation ID 5768 to stdb_cluster 47

Scenario 2: Adding observation ID 5776 to stdb_cluster 48

Scenario 2: Adding observation ID 5797 to stdb_cluster 49

Scenario 2: Adding observation ID 6651 to stdb_cluster 50

Scenario 2: Adding observation ID NA to stdb_cluster 51

Scenario 2: Adding observation ID 6693 to stdb_cluster 52

Scenario 2: Adding observation ID 8348 to stdb_cluster 56

Scenario 2: Adding observation ID 8401 to stdb_cluster 58

Scenario 2: Adding observation ID 8414 to stdb_cluster 59

Scenario 2: Adding observation ID NA to stdb_cluster 60

Scenario 2: Adding observation ID 8970 to stdb_cluster 66

Scenario 2: Adding observation ID 8994 to stdb_cluster 68

Scenario 2: Adding observation ID 9176 to stdb_cluster 75

Scenario 2: Adding observation ID 9729 to stdb_cluster 78

Scenario 2: Adding observation ID 9765 to stdb_cluster 79

Scenario 2: Adding observation ID 10408 to stdb_cluster 84
## Scenario 2: Adding observation ID 10524 to stdb_cluster 88
##
## Scenario 2: Adding observation ID 10590 to stdb_cluster 89
##
## Scenario 3: Adding observation ID 312 to stdb_cluster 1
##
## Scenario 3: Adding observation ID 394 to stdb_cluster 6
##
## Scenario 3: Adding observation ID NA to stdb_cluster 13
##
## Scenario 3: Adding observation ID 3573 to stdb_cluster 19
##
## Scenario 3: Adding observation ID 3696 to stdb_cluster 24
##
## Scenario 3: Adding observation ID 4205 to stdb_cluster 27
##
## Scenario 3: Adding observation ID 4291 to stdb_cluster 29
##
## Scenario 3: Adding observation ID 4401 to stdb_cluster 32
##
## Scenario 3: Adding observation ID 5632 to stdb_cluster 41
##
## Scenario 3: Adding observation ID NA to stdb_cluster 42
##
## Scenario 3: Adding observation ID 5775 to stdb_cluster 47
##
## Scenario 3: Adding observation ID 8408 to stdb_cluster 58
##
## Scenario 3: Adding observation ID NA to stdb_cluster 65
##
## Scenario 3: Adding observation ID 9723 to stdb_cluster 77
##
## Scenario 3: Adding observation ID NA to stdb_cluster 79
##
## Scenario 3: Adding observation ID 9866 to stdb_cluster 80
##
## Scenario 3: Adding observation ID NA to stdb_cluster 68000004
## Scenario 3: Adding observation ID 10413 to stdb_cluster 84

## Scenario 3: Adding observation ID 10485 to stdb_cluster 86

## Scenario 4: Number of clusters before merging of consecutive clusters: 95

## Scenario 4: Final number of clusters after merging of consecutive clusters: 75

## Scenario 4: Number of clusters merged: 20

### Participant 1002 had 12 home clusters.

### Participant 1002 had 63 non-home clusters.

### Participant 1002 spent 13967 minutes at home destinations.

### Participant 1002 spent 3977 minutes at non-home destinations.

### Participant 1002 spent 936 minutes in transit.

### Running DBSCAN...

### Number of unique destinations: 26

### Total distance (in km) that participant traveled: 430.6735

### The activity space area is 109.7901 sq. km.

### The median daily activity space area is 17.8718 sq. km.

### The number of missing days is 0

### The number of non-missing days is 14

### Participant 1002 had 18880 non-missing minutes.
## Participant 1002 had 1280 missing minutes.
write.csv(OtherSheet,file="T:\Data\ParticipantDataMeasuresTEST.csv")