Score Computation Notebook

Purpose

The purpose of this notebook is to use the raw travel time data to experiment with different methods of aggregation and score modeling.

```r
library(tidyverse)
```

```r
## Import raw TTM

ttm <- read.csv('../data/clean/ttm.csv')

origins <- 15197 # known origins
poi <- 432       # known destinations

# convert Ids from double to factor
ttm$fromId <- as.factor(ttm$fromId)
ttm$toId <- as.factor(ttm$toId)

summary(ttm); head(ttm)
```

```
##          fromId              toId          avg_unique_time  sd_unique_time
##  59150358004:    419   2169   :  14269   Min.   :  0.00   Min.   : 0.1601
##  59150359002:    419   7955   :  14269   1st Qu.: 52.54   1st Qu.: 1.9428
##  59150360006:    419   8982   :  14267   Median : 72.18   Median : 2.8868
##  59150425005:    419   317    :  14266   Mean   : 72.79   Mean   : 3.4044
##  59150425008:    419   8915   :  14263   3rd Qu.: 94.21   3rd Qu.: 4.3813
##  59150425009:    419   268    :  14262   Max.   :119.00   Max.   :35.3553
##  (Other)    :5160181   (Other):5077099
```

```
##        fromId toId avg_unique_time sd_unique_time
## 1 59150004004   10        99.76316       5.364721
## 2 59150004004   15        72.48718       3.401794
## 3 59150004004  157        96.69231       3.001349
## 4 59150004004 1759       106.82051       4.388213
## 5 59150004004 1760        46.58974       2.642944
## 6 59150004004 1822        76.64103       3.990035
```

```r
paste('Percent Origins considered:', round(length(unique(ttm$fromId))/origins*100, 2), '%')
```

```
## [1] "Percent Origins considered: 94.44 %"
```

```r
paste('Percent Destinations considered:', round(length(unique(ttm$toId))/poi*100, 2), '%')
```

```
## [1] "Percent Destinations considered: 99.77 %"
```

Base Functions

```r
# Function for normalizing numeric columns
normalize <- function(df) {
  num_cols <- which(sapply(df, is.numeric)) # numeric columns

  min_vec <- sapply(df[num_cols], min)
  max_vec <- sapply(df[num_cols], max)
  range_vec <- (max_vec - min_vec)/0.99

  norm <- function(vec, min, range) (vec - min*0.99)/range # use 0.99 to avoid zero values

  if (length(min_vec) > 1) {
    normed <- mapply(norm, df[num_cols], min = min_vec, range = range_vec)
    df[num_cols] <- normed
  } else {
    normed <- sapply(df[num_cols], norm, min = min_vec, range = range_vec)
    df[num_cols] <- as.numeric(normed)
  }
  df
}


# Naive score function
# 1 is perfect transit accessibility /// 0 is no transit accessibility
# Higher avg_time = Lower score (inverse)
# Higher sd_time = Lower score (inverse)
# More accessible destinations = Higher score (multiply)
naive_score <- function(fromIds, mean_time, mean_sd_time, n_accessible) {
  score <- n_accessible / (mean_time*mean_sd_time)
  df <- data.frame('fromId' = as.factor(fromIds), 'score' =  score)
  df <- df[order(df$score, decreasing=TRUE, na.last=FALSE), ]
  df <- normalize(df)
  df
}
```

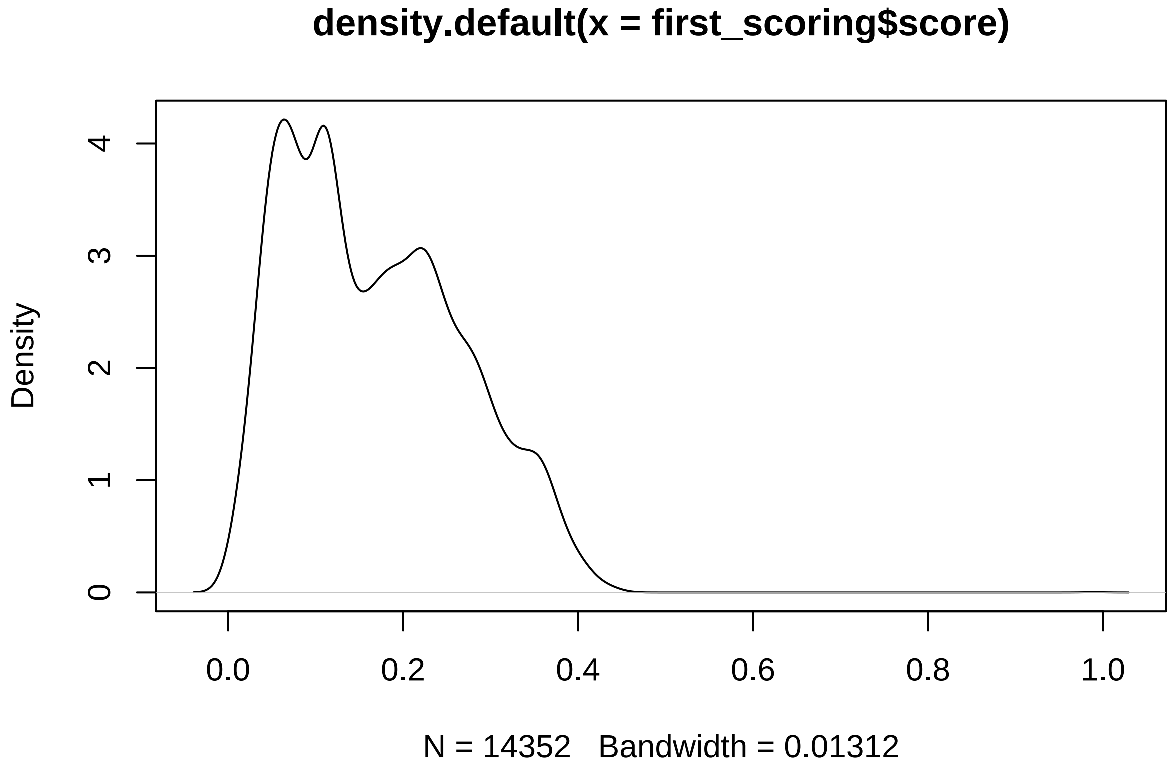
First Scoring

Unweighted accessibility score from a given origin (dissemination block) to all points of interest.

```
# aggregates on each origin computing avg travel time
# to all destinations from each origin
ttm_all_dest <- ttm %>%
                group_by(fromId) %>%
                summarise(
                  avg_time_to_allpoi = mean(avg_unique_time),
                  # sd_time_to_allpoi = sd(avg_unique_time), # unrealistic std
                  avg_sd_time_to_uniquepoi = mean(sd_unique_time),
                  n_accessible_poi = n()
                )

# Fill NAs in standard deviation with 80th percentile value
#upper80_sd <- quantile(ttm_all_dest$sd_time_to_allpoi, 0.8, na.rm=TRUE)
# ttm_all_dest <- ttm_all_dest %>% replace_na(list('sd_time_to_allpoi'=upper80_sd))

head(ttm_all_dest)
```

```
## # A tibble: 6 x 4
##   fromId      avg_time_to_allpoi avg_sd_time_to_uniquepoi n_accessible_poi
##   <fct>                    <dbl>                    <dbl>            <int>
## 1 59150004004               79.0                     3.33              366
## 2 59150004005               81.6                     3.08              366
## 3 59150004006               82.4                     2.98              366
## 4 59150004011               79.9                     3.29              366
## 5 59150004012               81.6                     3.10              366
## 6 59150004013               86.3                     2.60              363
```

```
## Get scores
first_scoring <- naive_score(ttm_all_dest$fromId,
                    ttm_all_dest$avg_time_to_allpoi,
                    ttm_all_dest$avg_sd_time_to_uniquepoi,
                    ttm_all_dest$n_accessible_poi)
plot(density(first_scoring$score))
```

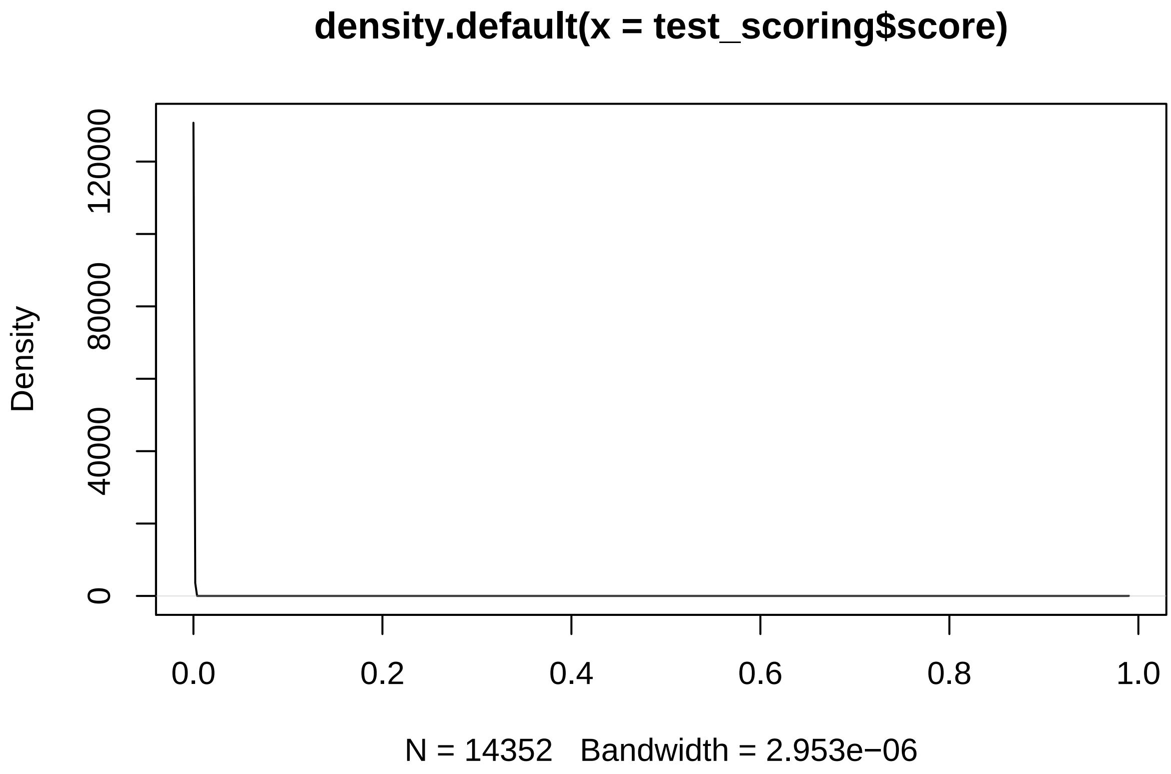


** Notice if we normalize before score computation we get one extremely outlying score**

```
norm_ttm_all_dest <- normalize(ttm_all_dest)
summary(norm_ttm_all_dest)
```

```
##          fromId      avg_time_to_allpoi  avg_sd_time_to_uniquepoi
##  59150004004:    1   Min.   :0.0000582   Min.   :0.0001681
##  59150004005:    1   1st Qu.:0.5198081   1st Qu.:0.0825074
##  59150004006:    1   Median :0.6315712   Median :0.1066498
##  59150004011:    1   Mean   :0.6470381   Mean   :0.1235140
##  59150004012:    1   3rd Qu.:0.7674868   3rd Qu.:0.1473295
##  59150004013:    1   Max.   :0.9900582   Max.   :0.9901681
##  (Other)    :14346
##  n_accessible_poi
##  Min.   :0.0000237
##  1st Qu.:0.8384447
##  Median :0.9023921
##  Mean   :0.8496226
##  3rd Qu.:0.9379184
##  Max.   :0.9900237
##
```

```
## Get scores
test_scoring <- naive_score(norm_ttm_all_dest$fromId,
                             norm_ttm_all_dest$avg_time_to_allpoi,
                             norm_ttm_all_dest$avg_sd_time_to_uniquepoi,
                             norm_ttm_all_dest$n_accessible_poi)
plot(density(test_scoring$score))
```



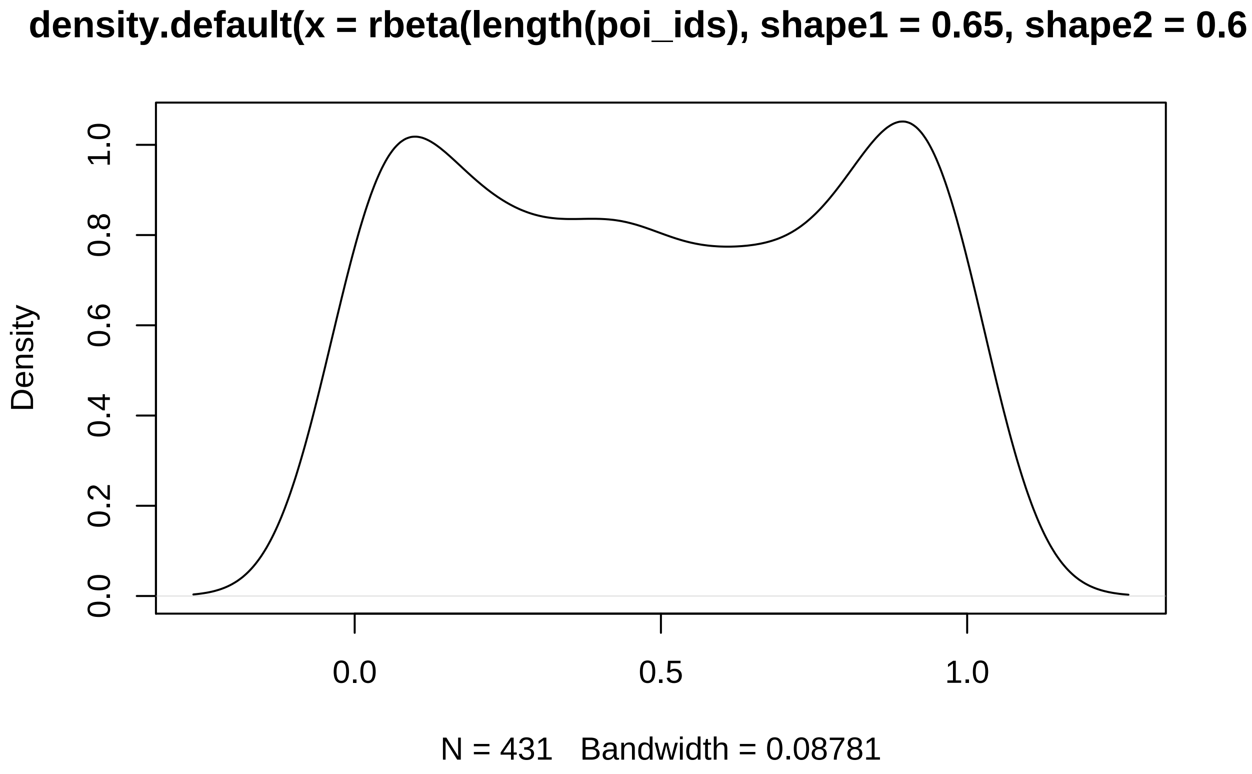
Second Scoring

Weighted accessibility score from a given origin (dissemination block) to all points of interest.

Here we import the scoring weights and join them to the ttm frame.

```
# Import
poi_ids <- unique(ttm$toId)

# Initialize destination popularity dataframe
POI_popularity <- data.frame('id'= poi_ids, stringsAsFactors = TRUE)
```

```
# Generate POI popularity weights
# using rbeta density
# assumes cultural facilities tend to be popular or unpopular, not inbetween
plot(density(rbeta(length(poi_ids), shape1=0.65, shape2=0.65)))
```



```
POI_popularity$wt <- rbeta(length(poi_ids), shape1=0.8, shape2=0.8)

head(POI_popularity)
```

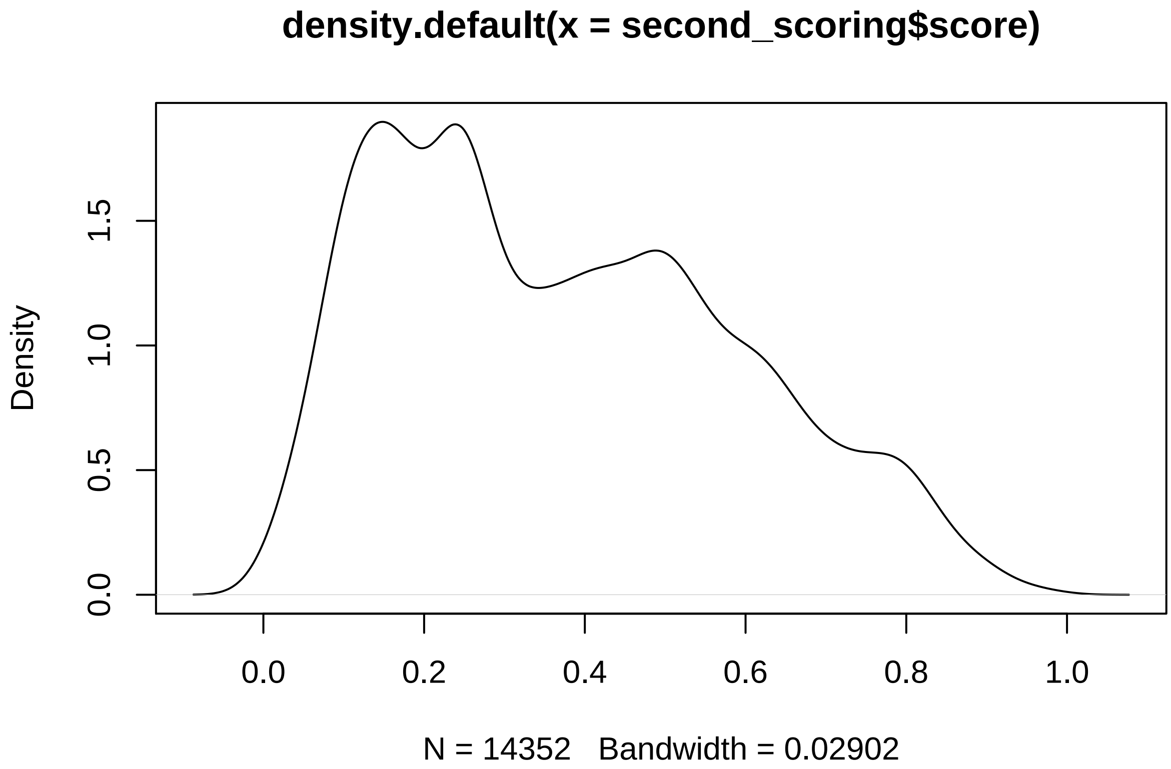
```
##      id         wt
## 1    10 0.63633564
## 2    15 0.60182402
## 3   157 0.82010977
## 4  1759 0.03053257
## 5  1760 0.53505965
## 6  1822 0.10345878
```

Reperform the Second Aggregation to Include the Destination Weights

```
# Join dfs to have a popularity weight column
ttm_weights <- left_join(ttm, POI_popularity, by = c('toId'='id'))


# aggregates on each origin computing avg travel time
# to all destinations from each origin
ttm_all_dest_wts <- ttm_weights %>%
                    group_by(fromId) %>%
                    summarise(
                      avg_time_to_allpoi = mean(avg_unique_time),
                      avg_sd_time_to_uniquepoi = mean(sd_unique_time),
                      n_accessible_poi = sum(wt)
                    )

summary(ttm_all_dest_wts)
```

```
##          fromId      avg_time_to_allpoi avg_sd_time_to_uniquepoi
##  59150004004:    1   Min.   :  0.6667   Min.   : 0.4268         
##  59150004005:    1   1st Qu.: 60.1832   1st Qu.: 2.5180         
##  59150004006:    1   Median : 72.9812   Median : 3.1311         
##  59150004011:    1   Mean   : 74.7523   Mean   : 3.5594         
##  59150004012:    1   3rd Qu.: 88.5449   3rd Qu.: 4.1643         
##  59150004013:    1   Max.   :114.0315   Max.   :25.5696         
##  (Other)    :14346                                              
##  n_accessible_poi  
##  Min.   :  0.05024 
##  1st Qu.:177.89523 
##  Median :192.54241 
##  Mean   :181.29028 
##  3rd Qu.:199.96237 
##  Max.   :212.35359 
## 
```

```
## Get scores
second_scoring <- naive_score(ttm_all_dest_wts$fromId,
                              ttm_all_dest_wts$avg_time_to_allpoi,
                              ttm_all_dest_wts$avg_sd_time_to_uniquepoi,
                              ttm_all_dest_wts$n_accessible_poi)

plot(density(second_scoring$score))
```



Third Scoring

Unweighted, average time of nearest two amenities by amenity (we consider amenity types separate)

Import amenity types.

```r
amenities <- read.csv('../data/clean/vancouver_facilities_2.csv')
head(amenities)
```

```
##    id         lat          lon                        type
## 1  10  49.1763542  -123.112783                      museum
## 2  15   49.261938  -123.151123                      museum
## 3  24   49.278786  -123.098796                      museum
## 4  41  49.2210003 -123.0091848                      artist
## 5  97 49.14709735 -122.6467963 heritage or historic site
## 6 115   49.279222   -123.11624       library or archives
##                                                name      city city_id
## 1                          12 Service Battalion Museum  Richmond 5915015
## 2 15th Field Artillery Regiment Museum And Archives Vancouver 5915022
## 3                               221A Artist Run Centre Vancouver 5915022
## 4                                   7302754 Canada Inc   Burnaby 5915025
## 5             Abc Heritage Preschool And Child Care   Langley 5915001
## 6                                  Accessible Services Vancouver 5915022
```

```r
amenities <- amenities[,c(1,4)] # only need id and type columns
amenities$id <- as.factor(amenities$id)     # convert to factor
amenities$type <- as.factor(amenities$type) # convert to factor
```

```
head(amenities)
```

```
##    id                        type
## 1  10                      museum
## 2  15                      museum
## 3  24                      museum
## 4  41                      artist
## 5  97 heritage or historic site
## 6 115       library or archives
```

```
amenities %>% group_by(type) %>% summarise(count = n())
```

```
## # A tibble: 9 x 2
##   type                               count
## * <fct>                              <int>
## 1 art or cultural centre                 5
## 2 artist                                48
## 3 festival site                          2
## 4 gallery                               99
## 5 heritage or historic site             28
## 6 library or archives                   88
## 7 miscellaneous                          6
## 8 museum                                92
## 9 theatre/performance and concert hall    75
```

Join type factor to ttm.

```
ttm_amenities <- ttm %>% left_join(amenities, by = c('toId' = 'id'))
head(ttm_amenities)
```

```
##        fromId toId avg_unique_time sd_unique_time    type
## 1 59150004004   10        99.76316       5.364721  museum
## 2 59150004004   15        72.48718       3.401794  museum
## 3 59150004004  157        96.69231       3.001349 gallery
## 4 59150004004 1759       106.82051       4.388213  museum
## 5 59150004004 1760        46.58974       2.642944 gallery
## 6 59150004004 1822        76.64103       3.990035  museum
```

Aggregate on type to get nearest destination times.

```
ttm_amenities_agg <- ttm_amenities %>%
                     group_by(fromId, type) %>%
                     summarise(nearest1 = min(avg_unique_time, na.rm = TRUE) + 5, # add 4min to prevent
                               nearest2 = mean(sort(avg_unique_time)[1:2], na.rm = TRUE) + 5,
                               nearest3 = mean(sort(avg_unique_time)[1:3], na.rm = TRUE) + 5,
                               sd1 = log(sd_unique_time[which.min(avg_unique_time)]+ 10), # add 10 min
                               sd2 = log(mean(sort(sd_unique_time)[1:2], na.rm = TRUE)+ 10),
                               sd3 = log(mean(sort(sd_unique_time)[1:3], na.rm = TRUE)+ 10))
```

```
## `summarise()` has grouped output by 'fromId'. You can override using the `.groups` argument.
```

```
head(ttm_amenities_agg)
```

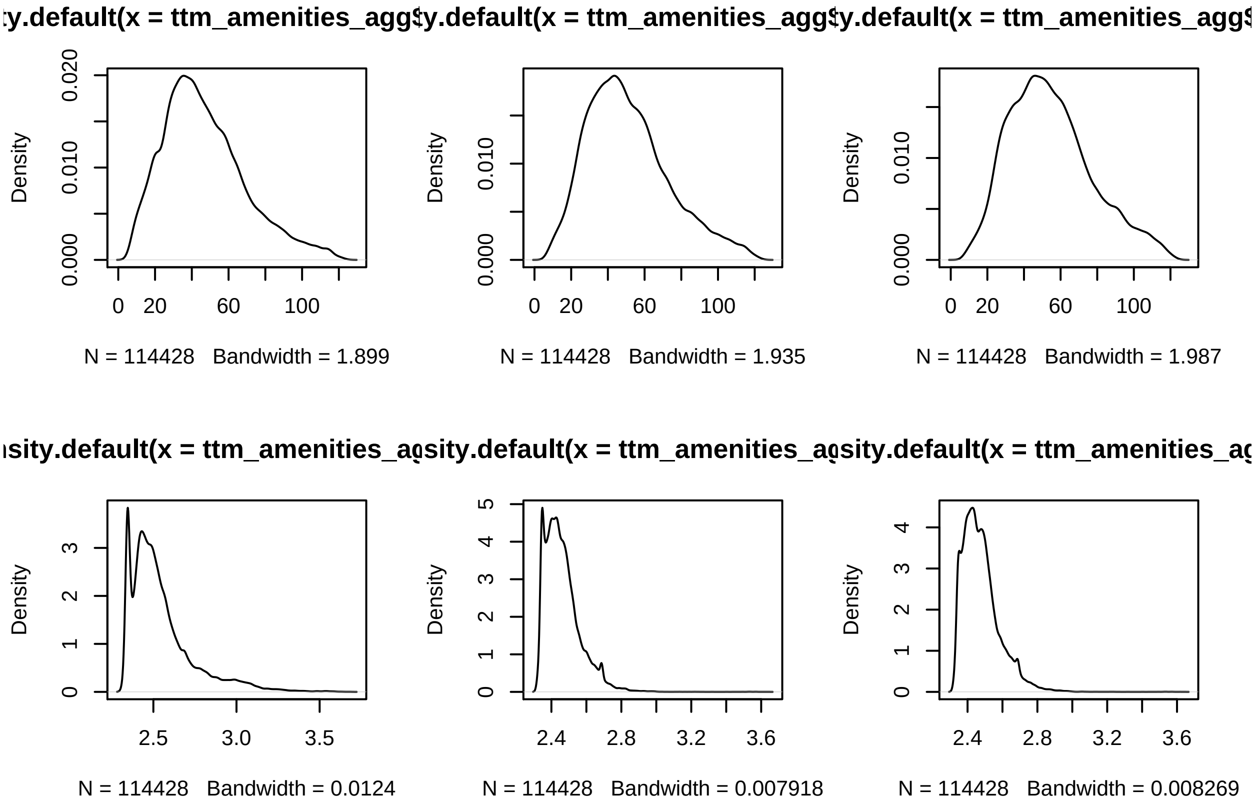
```
## # A tibble: 6 x 8
## # Groups:   fromId [1]
##   fromId      type                      nearest1 nearest2 nearest3   sd1   sd2   sd3
##   <fct>       <fct>                        <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl>
## 1 59150004004 art or cultural cent~     82.4     86.2     87.9  2.56  2.53  2.54
## 2 59150004004 artist                    82.1     84.8     86.2  2.54  2.54  2.54
## 3 59150004004 gallery                   40.8     45.7     47.6  2.48  2.49  2.50
## 4 59150004004 heritage or historic~     61.2     71.4     79.1  2.56  2.51  2.52
## 5 59150004004 library or archives       40.7     47.8     50.3  2.46  2.43  2.46
## 6 59150004004 miscellaneous             41.6     52.1     65.7  2.43  2.49  2.51
```

```r
ttm_amenities_agg_normalized <- normalize(ttm_amenities_agg)
head(ttm_amenities_agg_normalized)
```

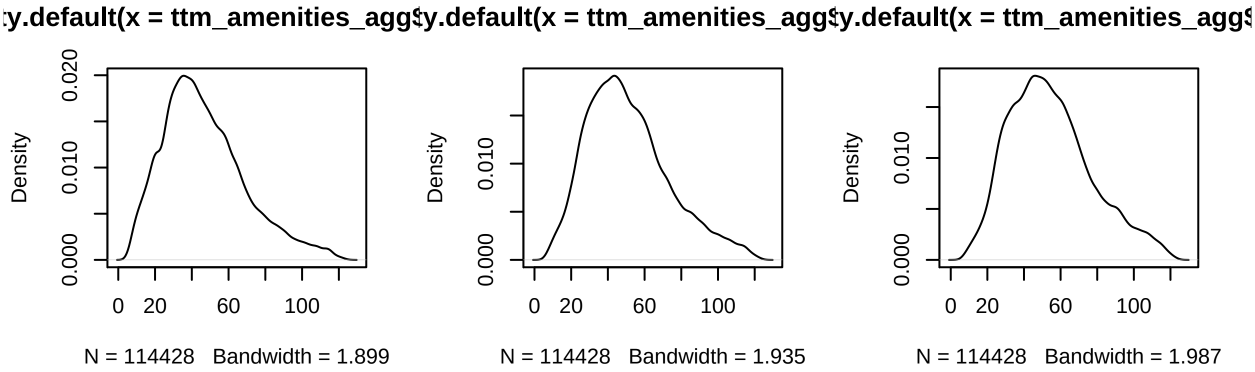
```
## # A tibble: 6 x 8
## # Groups:   fromId [1]
##   fromId      type                  nearest1 nearest2 nearest3   sd1   sd2   sd3
##   <fct>       <fct>                    <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl>
## 1 59150004004 art or cultural cent~    0.644    0.676    0.690 0.190 0.172 0.180
## 2 59150004004 artist                   0.642    0.664    0.676 0.175 0.180 0.181
## 3 59150004004 gallery                  0.298    0.339    0.355 0.131 0.148 0.157
## 4 59150004004 heritage or historic~    0.468    0.553    0.617 0.194 0.158 0.169
## 5 59150004004 library or archives      0.298    0.356    0.377 0.120 0.101 0.120
## 6 59150004004 miscellaneous            0.305    0.392    0.506 0.101 0.147 0.161
```

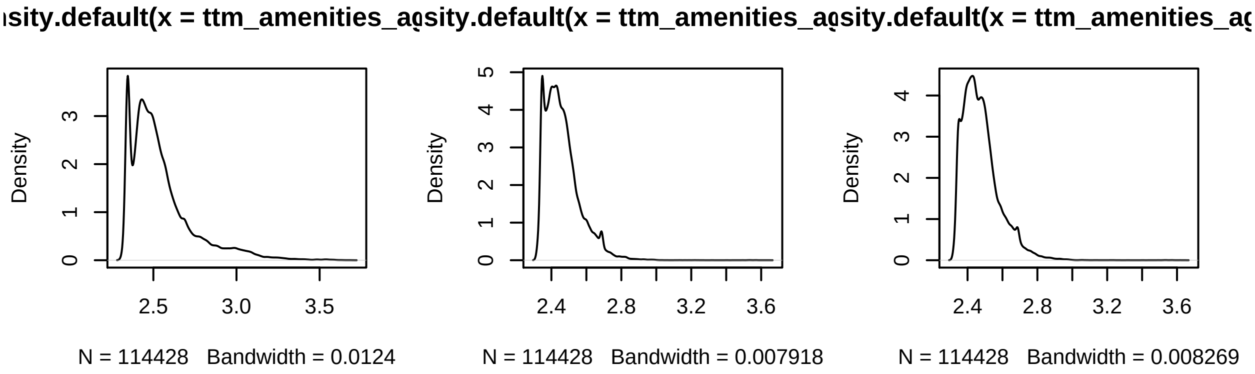
Visualize destination time density from nearest to nearest 1-3 amenities of a given type.

```r
# The SDs are horribly skewed so we'll take the log so our scores aren't as skewed.
par(mfrow = c(2,3))
plot(density(ttm_amenities_agg$nearest1))
plot(density(ttm_amenities_agg$nearest2))
plot(density(ttm_amenities_agg$nearest3))
plot(density(ttm_amenities_agg$sd1))
plot(density(ttm_amenities_agg$sd2))
plot(density(ttm_amenities_agg$sd3))
```



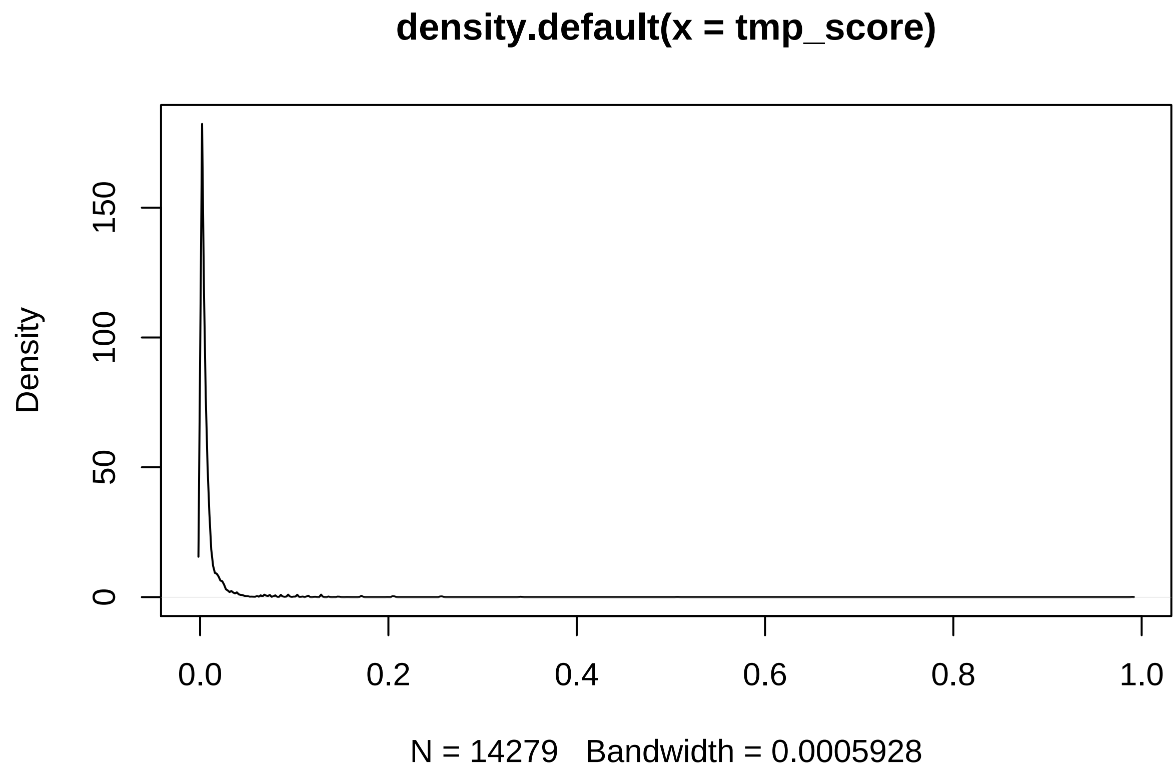
```r
# Much better
par(mfrow = c(2,3))
plot(density(ttm_amenities_agg$nearest1))
plot(density(ttm_amenities_agg$nearest2))
plot(density(ttm_amenities_agg$nearest3))
plot(density(ttm_amenities_agg$sd1))
plot(density(ttm_amenities_agg$sd2))
plot(density(ttm_amenities_agg$sd3))
```





Compute 9x3 sets of scores. Three for the nearest 1, nearest 2, and nearest 3 amenities. Nine for the 9 different types of amenities.

```
nearest1_set <- split(ttm_amenities_agg_normalized[, c(1,3,6)], ttm_amenities_agg$type)
nearest2_set <- split(ttm_amenities_agg_normalized[, c(1,4,7)], ttm_amenities_agg$type)
nearest2_set <- split(ttm_amenities_agg_normalized[, c(1,5,8)], ttm_amenities_agg$type)

score_lists <- function(lst_of_df, nearest_destinations = NULL) {
  library(rlist)
  collection <- NULL

  if (is.null(nearest_destinations)) {
    print('Nearest destinations must be between 1 and 3 inclusive')
    return(NULL)
  } else if (nearest_destinations == 1) {
    # iterate over lists because I can't get column arguments to work in apply type fxns
    for (df in 1:length(lst_of_df)) {
      score <- naive_score(lst_of_df[[df]]$fromId,lst_of_df[[df]]$nearest1,lst_of_df[[df]]$sd1, n_access
      print(head(score))
      collection[[df]] <- score
    }

  } else if (nearest_destinations == 2) {
    # iterate over lists because I can't get column arguments to work in apply type fxns
    for (df in 1:length(lst_of_df)) {
      score <- naive_score(lst_of_df[[df]]$fromId,lst_of_df[[df]]$nearest2,lst_of_df[[df]]$sd2, n_access
      collection[[df]] <- score
    }
  } else if (nearest_destinations == 3) {

    # iterate over lists because I can't get column arguments to work in apply type fxns
    for (df in 1:length(lst_of_df)) {
      score <- naive_score(lst_of_df[[df]]$fromId,lst_of_df[[df]]$nearest3,lst_of_df[[df]]$sd3, n_acces
      collection[[df]] <- score
    }
  }
  collection
}


# TEST THE SCORING
tmp_score <- naive_score(nearest1_set[[1]]$fromId, nearest1_set[[1]]$nearest1, nearest1_set[[1]]$sd1, n

# SEE SCORE DISTRIBUTION --> very bad
plot(density(tmp_score))
```



```
summary(tmp_score)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000048 0.0015399 0.0037032 0.0084326 0.0075202 0.9900048
```

```
summary(nearest1_set[[1]])
```

```
##          fromId          nearest1            sd1
## 59150004004:    1   Min.   :0.008735   Min.   :0.01679
## 59150004005:    1   1st Qu.:0.306524   1st Qu.:0.11844
## 59150004006:    1   Median :0.422569   Median :0.17264
## 59150004011:    1   Mean   :0.451240   Mean   :0.20295
## 59150004012:    1   3rd Qu.:0.567410   3rd Qu.:0.25902
## 59150004013:    1   Max.   :0.990416   Max.   :0.80671
## (Other)    :14273
```

```
#nearest1_score_set <- score_lists(nearest1_set, nearest_destinations = 1)
```