# 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.

Scoring models incorporated:

| Name | Function | Notes | Assumptions |
|---|---|---|---|
| Unweighted Naive | number of accessible points / (mean transit time * mean standard deviation in transit time) | Mean transit time to all accessible destinations | Assumes that accessibility is defined by access to all amenities |
| Weighted Naive | popularity weighted accessible points / (mean transit time * mean standard deviation in transit time) | Mean transit time to all accessible destinations | Assumes that accessibility is defined by access to all amenities and that amenity popularity defines significance of an accessible amenity |
| Unweighted Sum | 1 / (nearest amenity transit time + standard deviation in nearest transit time) | Only considers the nearest 1 to 3 amenities of a certain category. Sum is used to prevent skewing of data (difference(1/(0.01*0.01) and 1/(6*6)) »> difference(1/(0.01+0.01) and 1/(6+6))) | Assumes accessibility only defined by access to the nearest amenity type |

***Import libraries***

```r
library(tidyverse)

# For pretty knitting
library(lemon)
knit_print.data.frame <- lemon_print
knit_print.tbl <- lemon_print
knit_print.summary <- lemon_print
```

***Import data***

```r
## Import raw Travel Time Matrix (ttm)

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

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

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 %"
```

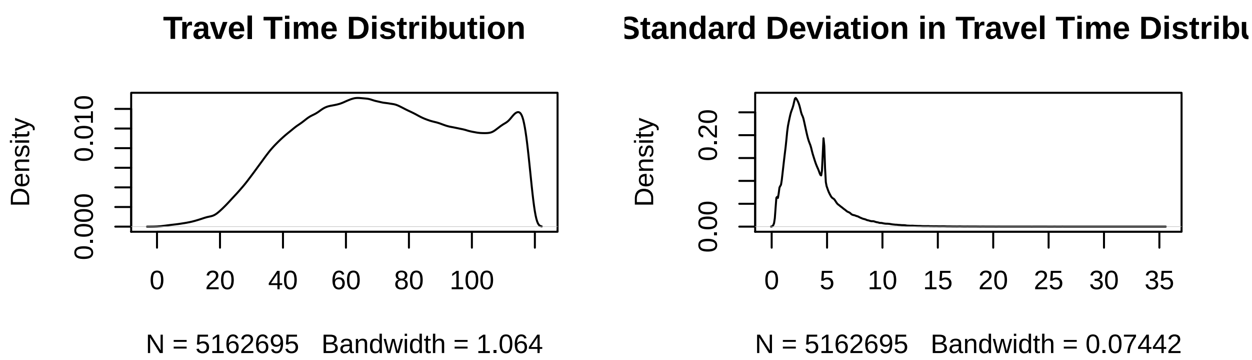
```r
# convert Ids from double to factor
ttm$fromId <- as.factor(ttm$fromId)
ttm$toId <- as.factor(ttm$toId)

summary(ttm[,3:4])
```

```
##  avg_unique_time  sd_unique_time
##  Min.   :  0.00   Min.   : 0.1601
##  1st Qu.: 52.54   1st Qu.: 1.9428
##  Median : 72.18   Median : 2.8868
##  Mean   : 72.79   Mean   : 3.4044
##  3rd Qu.: 94.21   3rd Qu.: 4.3813
##  Max.   :119.00   Max.   :35.3553
```

```r
sample_n(ttm, 5)
```

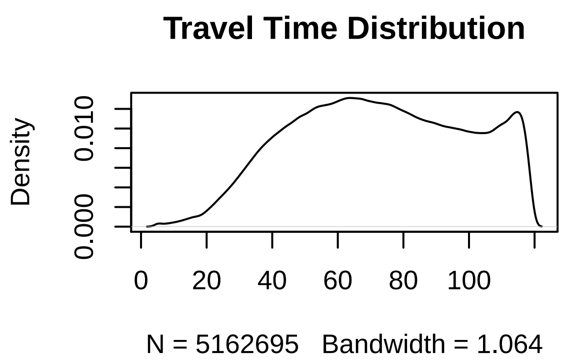
```
##        fromId toId avg_unique_time sd_unique_time
## 1 59153493001 8286        59.58974       1.802308
## 2 59150513009 8216       100.69440       7.037936
## 3 59152214005 9337        69.30769       1.935185
## 4 59153867002 4607        68.17949       6.223351
## 5 59151371002 5954        52.02564       2.241794
```

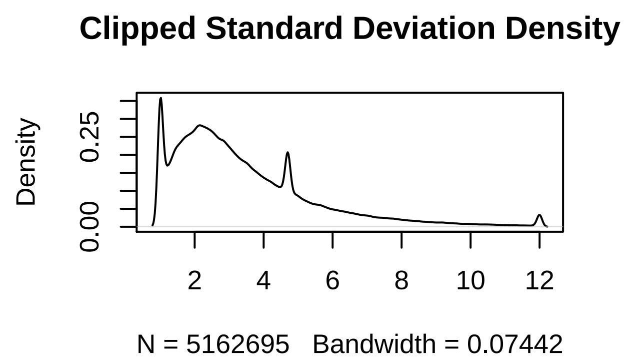
```r
par(mfrow = c(2, 2))

plot(density(ttm[,3]), main = 'Travel Time Distribution')
plot(density(ttm[,4]), main = 'Standard Deviation in Travel Time Distribution')
```

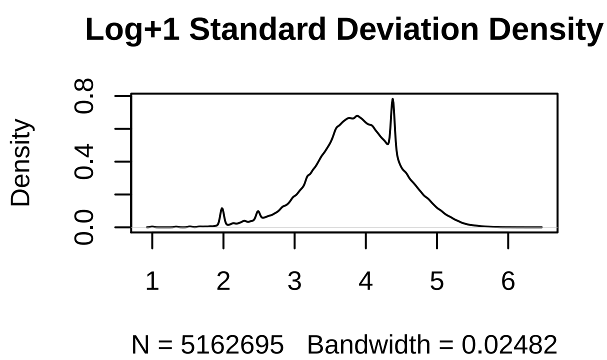



***Mini wrangling to remove extreme values***

```r
par(mfrow = c(2, 2))

# All travel time less than 5 minutes will be set to 5 minutes
ttm$avg_unique_time <- pmax(ttm$avg_unique_time, 5)
plot(density(ttm$avg_unique_time), main = 'Travel Time Distribution')

# Option 1: correct edges but not the skew
# All uncertainties greater than 12 minutes will be set to 12
# All uncertainties less than 1 minute will be set to 1 minute
test_sd <- pmax(pmin(ttm$sd_unique_time, 12), 1)
plot(density(test_sd), main = 'Clipped Standard Deviation Density')

# Option 2: correct the skew in addition to edges
ttm$sd_unique_time <- log(ttm$sd_unique_time) - min(log(ttm$sd_unique_time)) + 1
plot(density(ttm$sd_unique_time), main = 'Log+1 Standard Deviation Density')
```




### Base Functions

```
##############################
## NORMALIZATION FUNCTIONS
##############################

# normalize all numeric columns in a dataframe to a custom range [x,y]
normalize_df <- function(df, x = 0.01, y = 0.99, log = FALSE) {
  num_cols <- which(sapply(df, is.numeric)) # numeric columns
  if (log == TRUE) { df[num_cols] <- log(df[num_cols]) }

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

  cust_norm <- function(vec, min, range, x, y) {
    norm1 <- (vec - min)/range
    norm2 <- norm1*(y - x) + x
    norm2
  }

  if (length(min_vec) > 1) {
    # if there are multiple numeric columns
    normed <- mapply(cust_norm, df[num_cols], min = min_vec, range = range_vec, x = x, y = y)
    df[num_cols] <- normed
```

```r
  } else {
    # if there is 1 numeric column
    normed <- sapply(df[num_cols], norm, min = min_vec, range = range_vec, x = x, y = y)
    df[num_cols] <- as.numeric(normed)
  }
  df
}

# normalize vector to a custom range [x,y]
normalize_vec <- function(vec, x, y, log = FALSE) {
  if (log == TRUE) { vec <- log(vec) }
  norm_v <- (vec - min(vec))/(max(vec) - min(vec))
  custom_norm_v <- norm_v*(y - x) + x
  custom_norm_v
}


##############################
## SCORING FUNCTIONS
##############################

# naive score function : accessible_points / (mean * std)
naive_score <- function(fromIds, mean_time, mean_sd_time, n_accessible, x=0.001, y=0.999, log = FALSE) {

  # normalize the score function with custom parameters
  norm_score <- normalize_vec(n_accessible / (mean_time*mean_sd_time), x = x, y = y, log = log)

  df <- data.frame('fromId' = as.factor(fromIds), 'score' =  norm_score)
  #df <- df[order(df$norm_score, decreasing=TRUE, na.last=FALSE), ] # order doesn't matter
  df
}

# naive score function2 : 1 / (mean + std)
naive_score2 <- function(fromIds, mean_time, mean_sd_time, x=0.001, y=0.999, log = FALSE) {
  # normalize the score function with custom parameters
  norm_score <- normalize_vec( 1/(mean_time*mean_sd_time), x = x, y = y, log = log)
  df <- data.frame('fromId' = as.factor(fromIds), 'score' =  norm_score)
  df
}

# simplest score function using only mean time
simple_score <- function(fromIds, mean_time,x=0.001, y=0.999, log = FALSE) {
  norm_score <- normalize_vec(1/mean_time, x, y, log) # custom score normalization
  df <- data.frame('fromId' = as.factor(fromIds), 'score' =  norm_score)
  df
}
```

---

### Unweighted Accessibility to All Destinations within Contraints

```r
# avg travel time to all accessible destinations
ttm_all_dest <- ttm %>%
```

```r
                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()
                )

summary(ttm_all_dest)[,2:4]
```

**Aggregate on from destinations and compute unweighted average to all accessible destinations.**

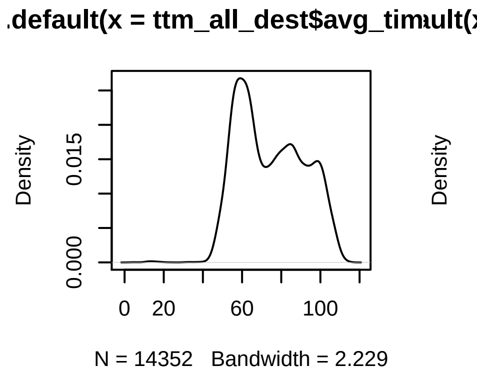
```
##  avg_time_to_allpoi avg_sd_time_to_uniquepoi n_accessible_poi
##  Min.   :  5.00     Min.   :1.980            Min.   :  1.0
##  1st Qu.: 60.18     1st Qu.:3.624            1st Qu.:355.0
##  Median : 72.98     Median :3.874            Median :382.0
##  Mean   : 74.75     Mean   :3.907            Mean   :359.7
##  3rd Qu.: 88.54     3rd Qu.:4.158            3rd Qu.:397.0
##  Max.   :114.03     Max.   :6.073            Max.   :419.0
##
```

```r
sample_n(ttm_all_dest, 5)
```

Table 2: Summary Table

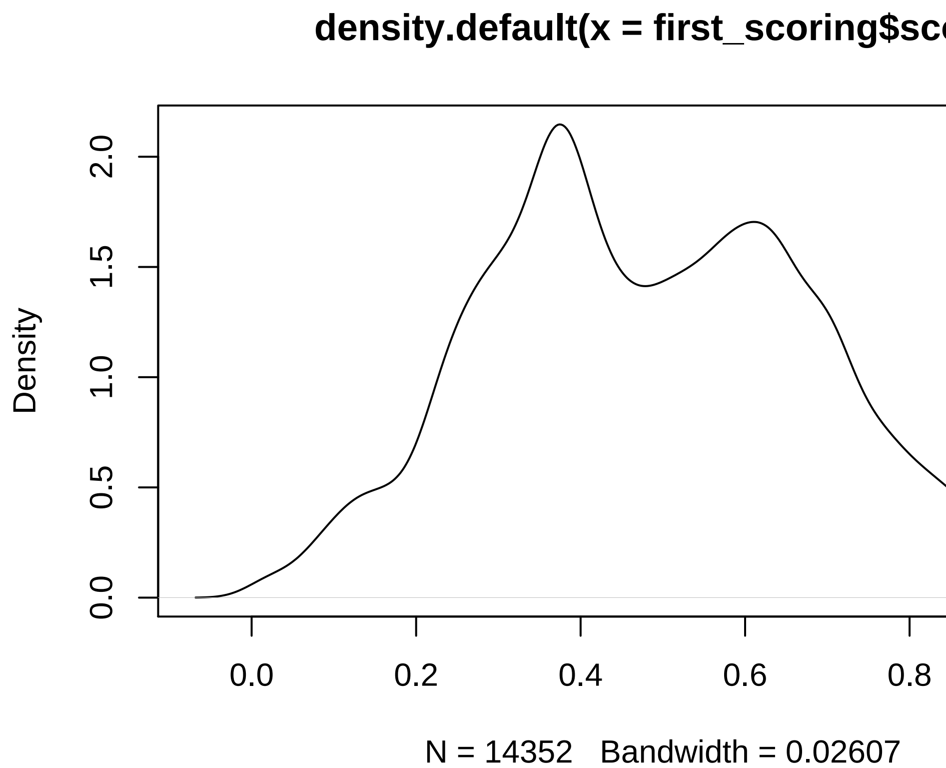
| fromId | avg_time_to_allpoi | avg_sd_time_to_uniquepoi | n_accessible_poi |
|---|---:|---:|---:|
| 59151866002 | 70.00045 | 3.612504 | 402 |
| 59152207001 | 76.04349 | 3.583223 | 408 |
| 59153925002 | 83.35577 | 3.705974 | 388 |
| 59150108001 | 84.04729 | 4.353707 | 368 |
| 59150004012 | 81.63391 | 3.930396 | 366 |

```r
# visualizing distribution of data
par(mfrow = c(2, 3))

plot(density(ttm_all_dest$avg_time_to_allpoi))
plot(density(ttm_all_dest$avg_sd_time_to_uniquepoi))
plot(density(ttm_all_dest$n_accessible_poi))
```




**Visualizing the distribution of aggregated data and aggregated log(data)**

```
## Get and visualize score distributions
# without log since it was applied in the mini wrangling
attach(ttm_all_dest)
first_scoring <- naive_score(fromIds = fromId,
                             mean_time = avg_time_to_allpoi,
                             mean_sd_time = avg_sd_time_to_uniquepoi,
                             n_accessible = n_accessible_poi,
                             x = 0.01, y = 0.99, log = FALSE)

detach(ttm_all_dest)

plot(density(first_scoring$score))
```




**Compute and Visualizing the scores**

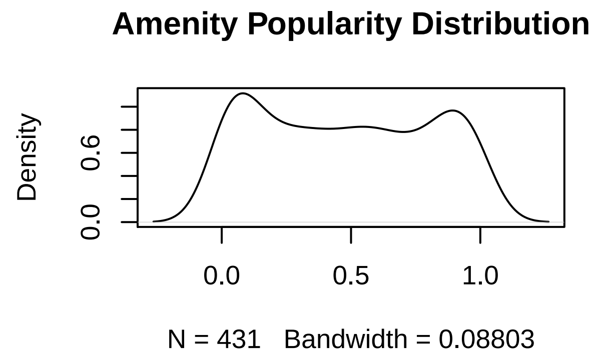
```
summary(first_scoring$score)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0100  0.3392  0.4741  0.4839  0.6334  0.9900
```

---

## Weighted Accessibility to All Destinations within Contraints

```
# 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 which assumes cultural facilities to be more often popular or unpopular, not betw
POI_popularity$wt <- rbeta(length(poi_ids), shape1=0.65, shape2=0.65)
par(mfrow=c(2,2))
plot(density(POI_popularity$wt), main = 'Amenity Popularity Distribution')

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




**Import the scoring weights and join them to the ttm frame**

```r
# avg travel time to all accessible destinations
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) # weighted component
                    )

summary(ttm_all_dest_wts)[,2:4]
```

**Re-Perform the fromId Aggregation to Include the Destination Weights**

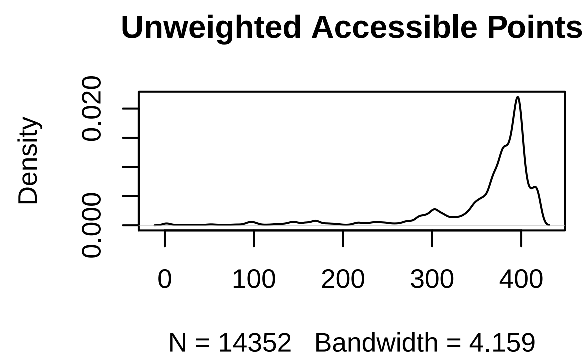
```
##  avg_time_to_allpoi avg_sd_time_to_uniquepoi n_accessible_poi
##  Min.   :  5.00     Min.   :1.980            Min.   :  0.0061
##  1st Qu.: 60.18     1st Qu.:3.624            1st Qu.:171.6774
##  Median : 72.98     Median :3.874            Median :184.2365
##  Mean   : 74.75     Mean   :3.907            Mean   :172.8400
##  3rd Qu.: 88.54     3rd Qu.:4.158            3rd Qu.:191.1963
##  Max.   :114.03     Max.   :6.073            Max.   :200.9137
##
```

```r
sample_n(ttm_all_dest_wts, 5)
```

Table 3: Summary Table

| fromId | avg_time_to_allpoi | avg_sd_time_to_uniquepoi | n_accessible_poi |
|---|---|---|---|
| 59150008006 | 94.41969 | 4.128770 | 167.3363 |
| 59150126008 | 89.20185 | 4.616020 | 175.9330 |
| 59152740005 | 107.21645 | 4.590560 | 138.0705 |
| 59152233008 | 88.88452 | 4.031413 | 192.7101 |
| 59150474004 | 59.40766 | 3.772179 | 191.0783 |

```
# visualizing distribution of data and log(data)
par(mfrow = c(2, 2))

plot(density(ttm_all_dest$n_accessible_poi), main = 'Unweighted Accessible Points')
plot(density(ttm_all_dest_wts$n_accessible_poi), main = 'Weighted Accessible Points')
```



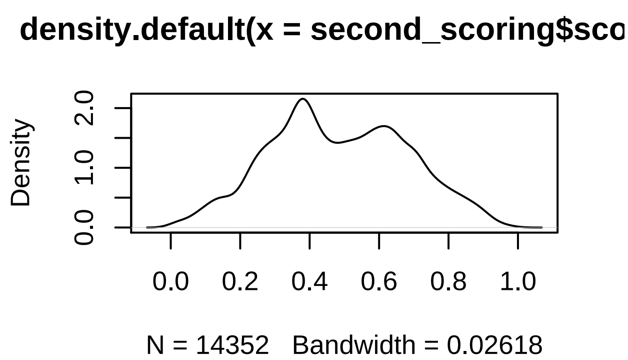

**Visualize and compare the distribution of the n_accessible_poi**

```
## Get and visualize score distributions
# without log since it was applied in the mini wrangling
attach(ttm_all_dest_wts)
second_scoring <- naive_score(fromIds = fromId,
                              mean_time = avg_time_to_allpoi,
                              mean_sd_time = avg_sd_time_to_uniquepoi,
                              n_accessible = n_accessible_poi,
                              x = 0.01, y = 0.99, log = FALSE)
```

```r
detach(ttm_all_dest_wts)

par(mfrow = c(2, 2))

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

**Compute and Visualizing the scores**

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0100  0.3406  0.4758  0.4849  0.6360  0.9900
```




---

## Unweighted Accessibility to nearest 1, 2, and 3 amenities by amenity type

### Due to the diverse functionality of different amenities, we will consider them separate for accessibility computation.

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

sample_n(amenities, 3)
```

**Import amenity types**

```
##     id         lat          lon                                    type
```

```
## 1 8922 49.21419488 -122.9031079 theatre/performance and concert hall
## 2 2217 49.21466644 -122.9229655                               artist
## 3 3140  49.2683622 -123.0692806                              gallery
##                         name            city city_id
## 1 The Bernie Legge Theatre New Westminster 5915029
## 2              Candice James New Westminster 5915029
## 3     Doctor Vigari Gallery       Vancouver 5915022
```

```
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

sample_n(amenities, 3)
```

```
##     id                 type
## 1 3379               artist
## 2 9360 library or archives
## 3 5988 library or archives
```

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

Table 4: Summary Table

| type | count |
|---|---|
| gallery | 99 |
| museum | 92 |
| library or archives | 88 |
| theatre/performance and concert hall | 75 |
| artist | 48 |
| heritage or historic site | 28 |
| miscellaneous | 6 |
| art or cultural centre | 5 |
| festival site | 2 |

```
ttm_amenities <- ttm %>% left_join(amenities, by = c('toId' = 'id'))
sample_n(ttm_amenities, 5)
```

**Join amenity type factor to the ttm**

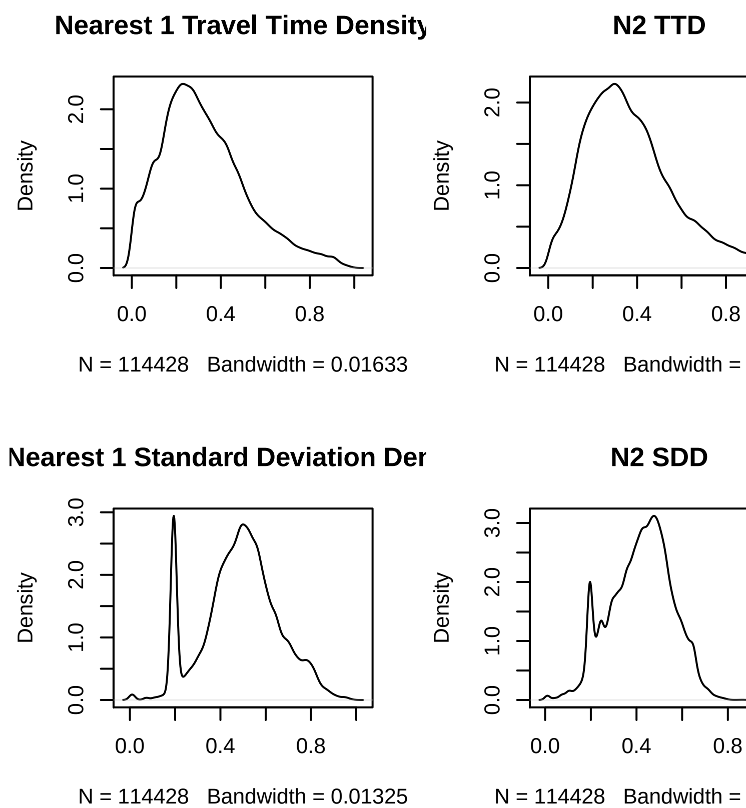
```
##        fromId toId avg_unique_time sd_unique_time                type
## 1 59152188019 9122        69.28205       4.067497 library or archives
## 2 59150963005 8119        37.69231       3.028403             gallery
## 3 59151478001  688        51.53846       3.630389             gallery
## 4 59152152005 5239        60.74359       3.665723              artist
## 5 59150678002 3543        72.12821       3.180711              artist
```

```
ttm_amenities_agg <- ttm_amenities %>%
                     group_by(fromId, type) %>%
                     summarise(nearest1 = min(avg_unique_time),
                               nearest2 = mean(sort(avg_unique_time)[1:2], na.rm = TRUE), # minumum 2
                               nearest3 = mean(sort(avg_unique_time)[1:3], na.rm = TRUE), # minimum 3
                               sd1 = sd_unique_time[which.min(avg_unique_time)],
```

```
                                 sd2 = mean(sort(sd_unique_time)[1:2], na.rm = TRUE),
                                 sd3 = mean(sort(sd_unique_time)[1:3], na.rm = TRUE))

# Normalize
normalized_ttm_amenities_agg <- normalize_df(ttm_amenities_agg)

# Log Normalize
log_normalized_ttm_amenities_agg <- normalize_df(ttm_amenities_agg, log = TRUE)
```

**Aggregate on amenity type to get the nearest destination times a**

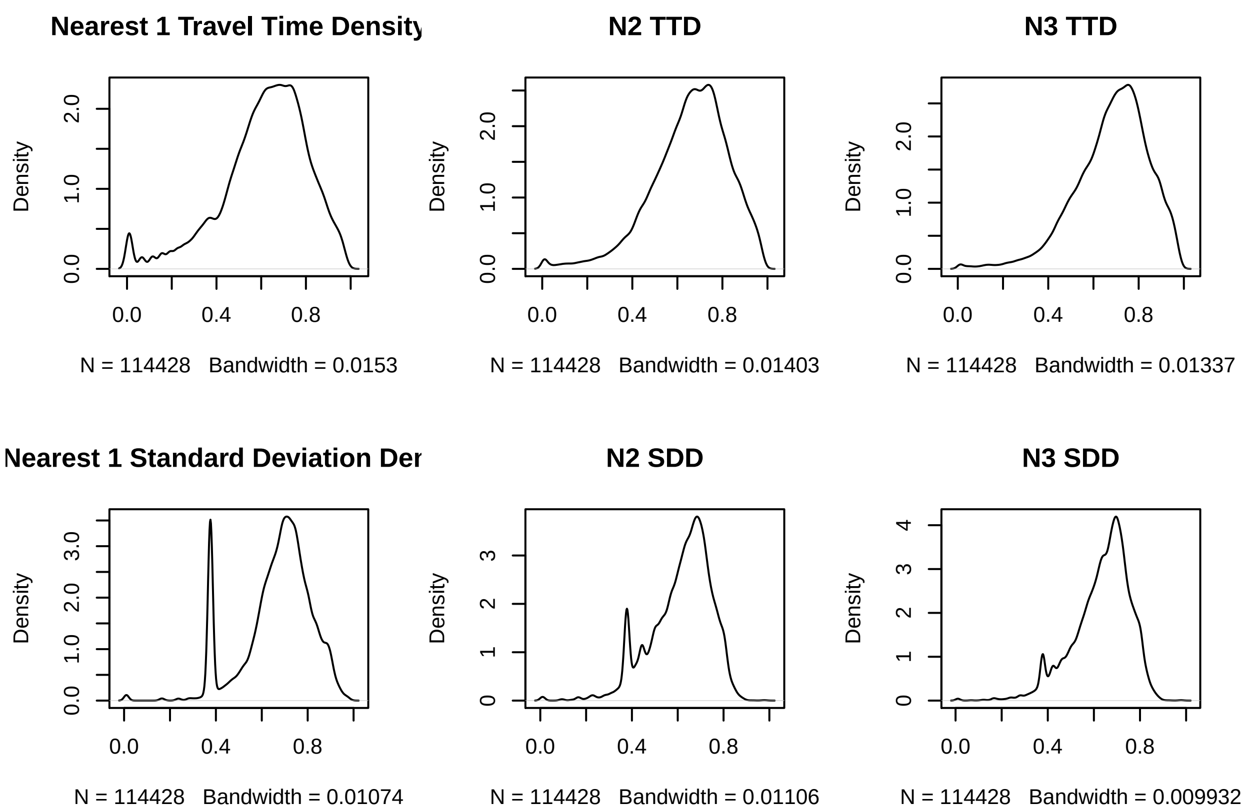
```
par(mfrow = c(2,3))

# Normalized
attach(normalized_ttm_amenities_agg)
plot(density(nearest1), main = 'Nearest 1 Travel Time Density')
plot(density(nearest2), main = 'N2 TTD')
plot(density(nearest3), main = 'N3 TTD')
plot(density(sd1), main = 'Nearest 1 Standard Deviation Density')
plot(density(sd2), main = 'N2 SDD')
plot(density(sd3), main = 'N3 SDD')
```




**Visualize normalized and original data distributions**

```
detach(normalized_ttm_amenities_agg)

# Log Normalize
```

```r
attach(log_normalized_ttm_amenities_agg)
plot(density(nearest1), main = 'Nearest 1 Travel Time Density')
plot(density(nearest2), main = 'N2 TTD')
plot(density(nearest3), main = 'N3 TTD')
plot(density(sd1), main = 'Nearest 1 Standard Deviation Density')
plot(density(sd2), main = 'N2 SDD')
plot(density(sd3), main = 'N3 SDD')
```




```r
detach(log_normalized_ttm_amenities_agg)
```

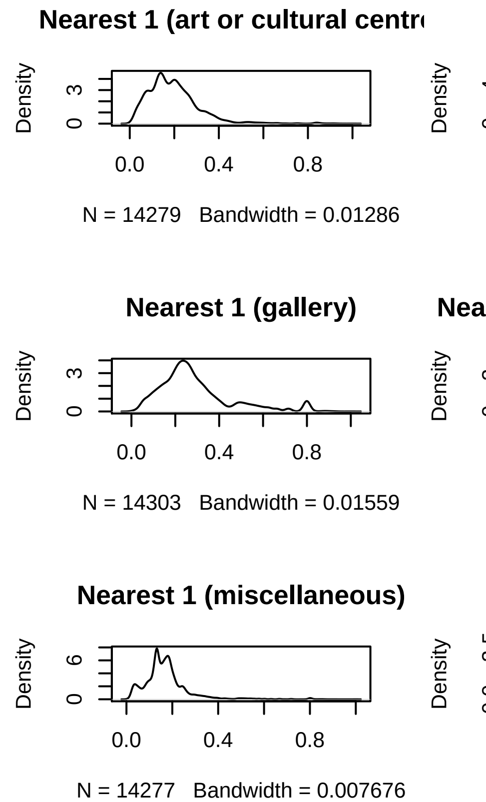
**Compute scores for each amenity type and nearest x condition** ***Since there are 9 amenity types and 3 nearest x conditions, we'll have a total of 27 score sets.***

```r
# desperate function for applying over lists where argument is column name
score_lists <- function(dfs, nearest_destinations = NULL, log = FALSE) {

  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) {
```

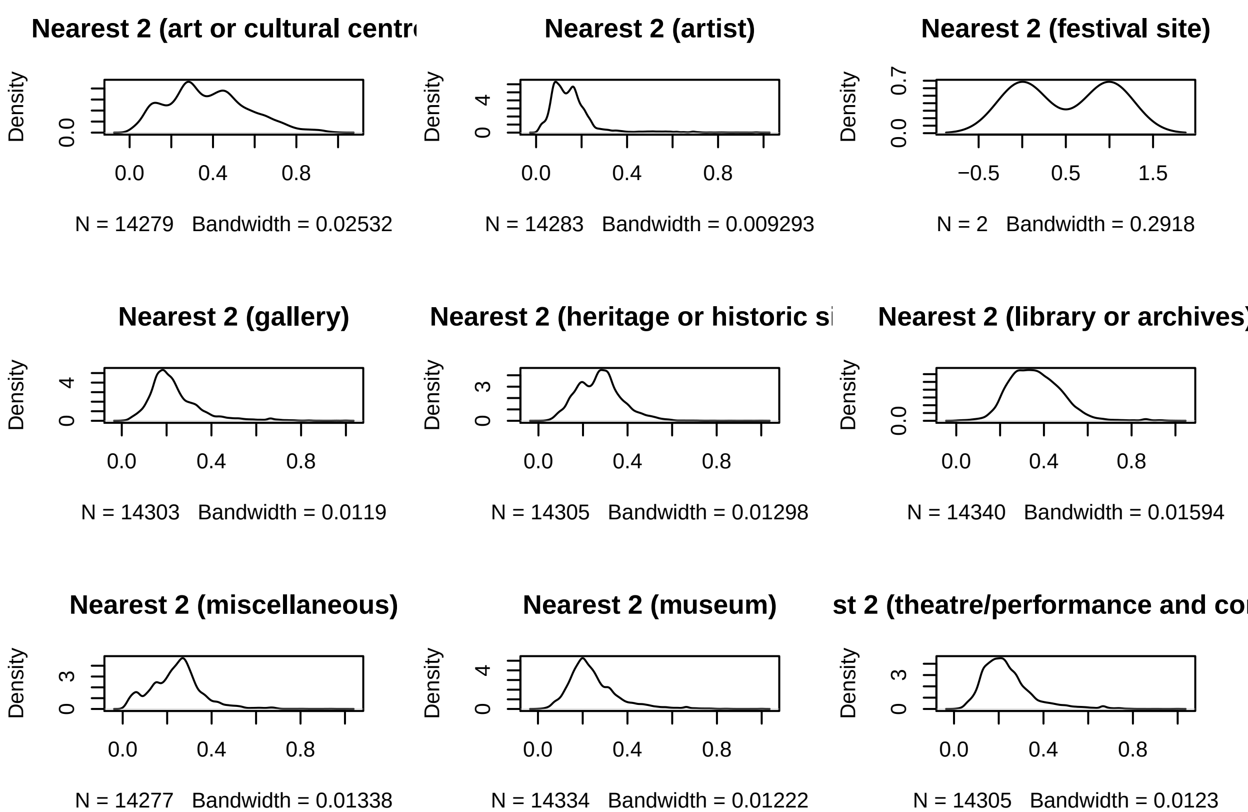
```r
    # iterate over lists because I can't get column arguments to work in apply type fxns
    for (df in 1:length(dfs)) {
      score_df <- naive_score2(dfs[[df]]$fromId, dfs[[df]]$nearest1, dfs[[df]]$sd1, log = log)
      collection[[df]] <- score_df
    }
  } else if (nearest_destinations == 2) {

    # iterate over lists because I can't get column arguments to work in apply type fxns
    for (df in 1:length(dfs)) {
      score_df <- naive_score2(dfs[[df]]$fromId, dfs[[df]]$nearest2, dfs[[df]]$sd2, log = log)
      collection[[df]] <- score_df
    }
  } 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(dfs)) {
      score_df <- naive_score2(dfs[[df]]$fromId, dfs[[df]]$nearest3, dfs[[df]]$sd3, log = log)
      collection[[df]] <- score_df
    }
  }
  collection
}
```

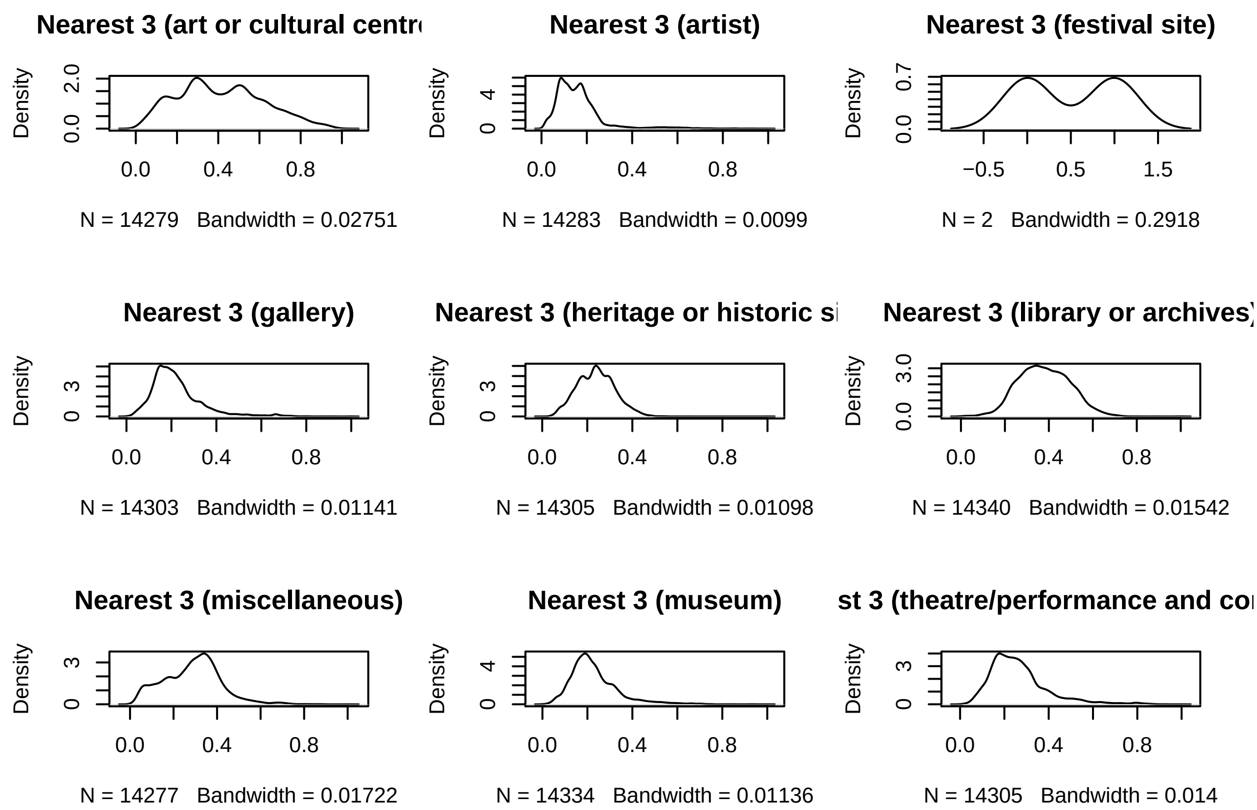
```r
# list of dataframes for each amenity type
attach(normalized_ttm_amenities_agg)
nearest1_set <- split(normalized_ttm_amenities_agg[, c(1,3,6)], type)
nearest2_set <- split(normalized_ttm_amenities_agg[, c(1,4,7)], type)
nearest3_set <- split(normalized_ttm_amenities_agg[, c(1,5,8)], type)
detach(normalized_ttm_amenities_agg)


nearest1_score_set <- score_lists(nearest1_set, nearest_destinations = 1, log = TRUE)
nearest2_score_set <- score_lists(nearest2_set, nearest_destinations = 2, log = TRUE)
nearest3_score_set <- score_lists(nearest3_set, nearest_destinations = 3, log = TRUE)
```

```r

par(mfrow = c(3, 3))

all_scores <- list(nearest1_score_set, nearest2_score_set, nearest3_score_set)

plot_density <- function(df, title) {
  plot(density(df$score), main = title)
}

## Visualize Score Densities

for (n in 1:3) {
  for (i in seq_along(names(nearest1_set))) {
    vec <- all_scores[[n]][[i]]['score']
    plot_density(vec, title = glue::glue('Nearest {n} ({names(nearest1_set)[i]})'))
  }
}
```




**Visualize scores for different amenities and the nearest neighbour factor**






---

### Exporting all Score Sets

```r
# First scoring
write.csv(first_scoring, '../data/score_sets/all_destination_scores.csv')

# Second scoring
write.csv(second_scoring, '../data/score_sets/all_destination__simulated_weighted_scores.csv')

# Nearest 1 scoring
for (i in seq_along(names(nearest1_set))) {
  name <- gsub('/', '_', gsub(' ', '_', names(nearest1_set)[i]))
  write.csv(nearest1_set[i], glue::glue('../data/score_sets/nearest1_{name}_scores.csv'))
}

# Nearest 3 scoring
for (i in seq_along(names(nearest3_set))) {
  name <- gsub('/', '_', gsub(' ', '_', names(nearest3_set)[i]))
  write.csv(nearest3_set[i], glue::glue('../data/score_sets/nearest3_{name}_scores.csv'))
}
```