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.

Import libraries

```
library(tidyverse)
library(ggplot2)

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

Import Scoring Functions

```
source('Score_Functions.R')

# normalize_vec(vec, x=0.01, y=0.99, log = FALSE)
# normalize_df(df, x = 0.01, y = 0.99, log = FALSE)

# sum score function : SUM [i..n] (1 / (traveltime_i * std_traveltime_i) + ... ))
# sum_score_fxn(df, weight = FALSE, log_normalize_score = TRUE, normalize_df = FALSE, x=1, y=10)
```

```
# function to plot score distributions by type
plot_densities <- function(score_frame1, score_frame2, titl1, titl2) {
  x <- score_frame1 %>%
        ggplot(aes(x = score, color = type)) +
        geom_density() +
        egg::theme_article() +
        theme(aspect.ratio = 0.3) +
        ggtitle(titl1)
  y <- score_frame2 %>%
        ggplot(aes(x = score, color = type)) +
        geom_density() +
        egg::theme_article() +
        theme(aspect.ratio = 0.3)+
        ggtitle(titl2)
  gridExtra::grid.arrange(x, y)
}
```

Import data

```
## Import raw Travel Time Matrix (ttm)
ttm <- read.csv('../../data/clean/ttm.csv')

n_origins <- 15197 # known origins
n_amenities <- 346 # known destinations from considered amenities

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

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

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

```
## [1] "Destinations considered: 124.57 %"
```

```
paste('Rows = ', nrow(ttm))
```

```
## [1] "Rows =  5162695"
```

```
# 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
```

Data Wrangling

Wrangling Notes: - Remove skews and extreme values - Due to the diversity in amenity types (which all serve a unique cultural purpose), we'll consider them independently for accessibility score computations. - Amenities which were interested in studying have already been filtered out in the ttm computation. They are the following: - Museums - Libraries - Galleries - Theatres

Import and join amenity types

```
target_amenities <- c('gallery', 'museum', 'library or archives', 'theatre/performance and concert hall
amenities <- read.csv('../../data/clean/vancouver_facilities_2.csv') %>% filter(type %in% target_amenit

# preview original
sample_n(amenities, 3)
```

```
##     id        lat          lon    type                              name
## 1 1822  49.242077  -123.113759  museum               Bloedel Conservatory
## 2  602 49.2811369 -123.0913998 gallery               Back Gallery Project
## 3 9342  49.278238  -123.121183 gallery Vancouver Contemporary Art Gallery
##        city city_id
## 1 Vancouver 5915022
## 2 Vancouver 5915022
## 3 Vancouver 5915022
```

```r
# clean
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

# preview clean
sample_n(amenities, 3)
```

```
##     id                                type
## 1 3794                 library or archives
## 2 8658                 library or archives
## 3 2335 theatre/performance and concert hall
```

```r
# view summary
amenities %>% group_by(type) %>% summarise(count = n()) %>% arrange(desc(count))
```

Table 1: Summary Table

type	count
gallery	99
museum	92
library or archives	88
theatre/performance and concert hall	75

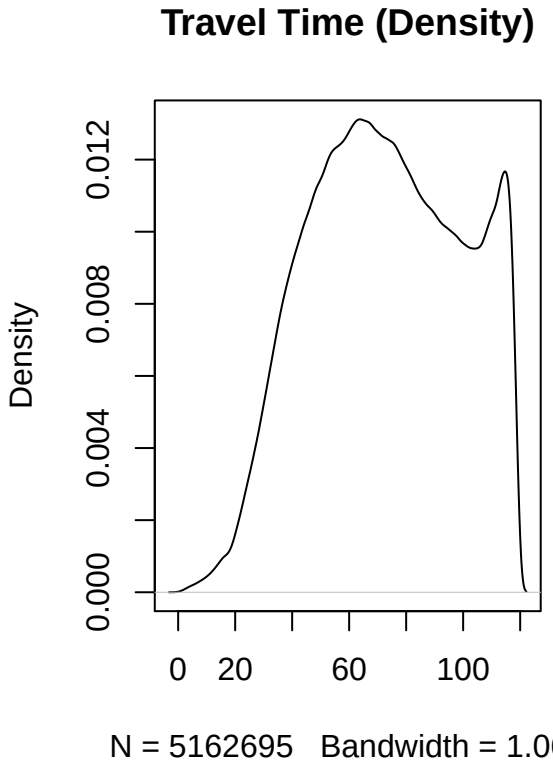
```r
ttm <- ttm %>% left_join(amenities, by = c('toId' = 'id'))

names(ttm)[names(ttm) == 'avg_unique_time'] <- "avg_time"
names(ttm)[names(ttm) == 'sd_unique_time'] <- "sd_time"

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

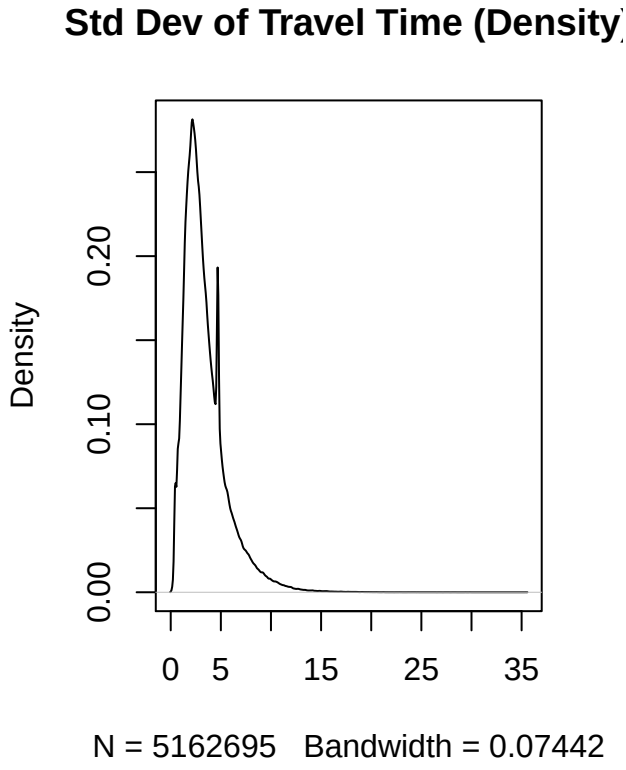
```
##     avg_time         sd_time
##  Min.   :  0.00   Min.   : 0.1601
##  1st Qu.: 52.54   1st Qu.: 1.9428
##  Median : 72.18   Median : 2.8868
##  Mean   : 72.79   Mean   : 3.4044
##  3rd Qu.: 94.21   3rd Qu.: 4.3813
##  Max.   :119.00   Max.   :35.3553
```

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

```
##        fromId toId  avg_time  sd_time                                type
## 1 59150987005 7806  53.58974 3.338121 theatre/performance and concert hall
## 2 59153026021 9316  91.30769 7.874522 theatre/performance and concert hall
## 3 59152213001 3873 104.88570 7.745099                 library or archives
## 4 59150554001 8553  60.71795 1.945964                             museum
## 5 59154012010 7346 108.78380 4.785018                               <NA>
```

```r
par(mfrow = c(1,2))
plot(density(ttm[,3]), main = 'Travel Time (Density)')
plot(density(ttm[,4]), main = 'Std Dev of Travel Time (Density)')
```

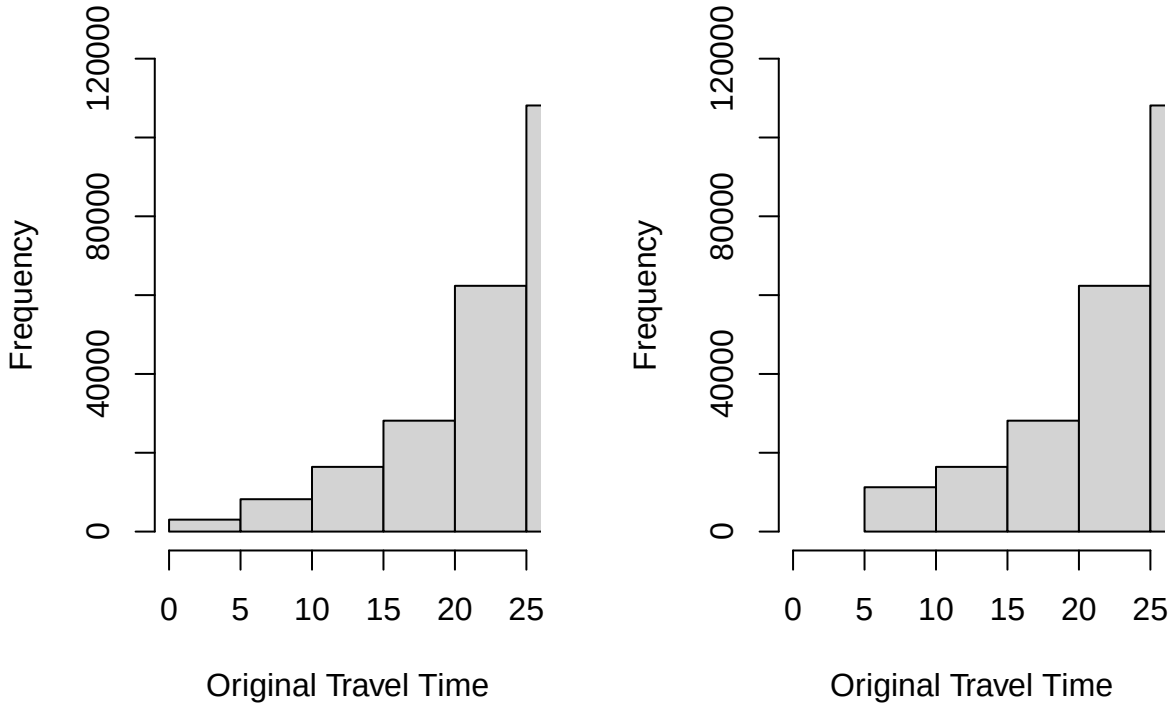




Replace travel times less than 5 minutes to 5 minutes

This is done to prevent infinity values in the scoring. Normalization will be done to prevent zero values but it still creates a largely skewed score if we include travel times that approach zero. 5 minutes is also a realistic time window for any travel time that may take 0 - 5 minutes.

```
par(mfrow = c(1, 2))

hist((ttm$avg_time), xlab = 'Original Travel Time', main = '',
     xlim = c(0, 25), ylim = c(0, 120000))

# set travel times <5 minutes to 5 minutes
min_5min <- pmax(ttm$avg_time, 5)
hist(min_5min, xlab = 'Original Travel Time', main = '',
     xlim = c(0, 25), ylim = c(0, 120000))
```

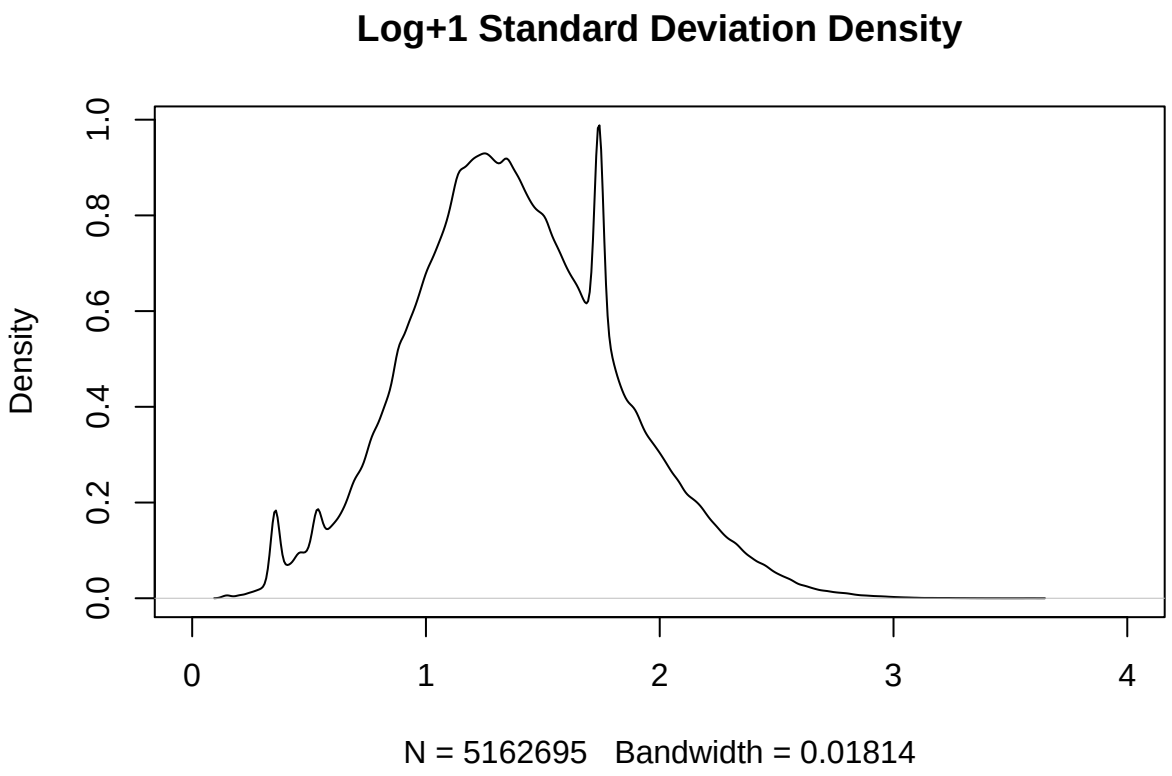



```
ttm$avg_time <- min_5min
```

Correct skew in standard deviation

This will be important to prevent skew amplification in the score computation.

```
# correct the skew in addition to edges close to zero

temp <- log(ttm$sd_time + 1) # +1 just prevents zero values
plot(density(temp), main = 'Log+1 Standard Deviation Density', xlim = c(0,4))
```



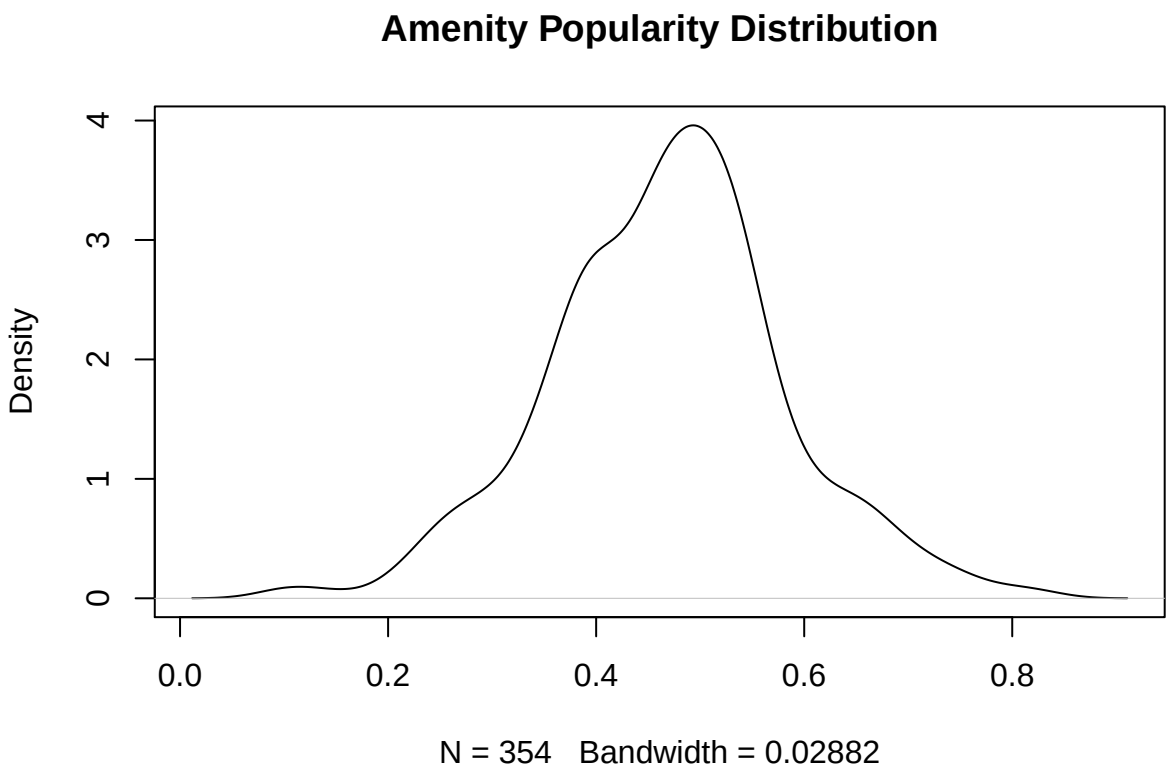

```
# set sd_unique_time to be the Log+1 corrected values
ttm$sd_time <- temp
```

Add Amenity Weights

```
# Import weight
dest_wts <- read.csv('../../data/amenity_score/poi_index.csv')

# clean
dest_wts <- dest_wts[, c(6,7)] # keep weight, id
names(dest_wts) <- c('weight', 'id')
dest_wts$id <-  as.factor(dest_wts$id)
head(dest_wts)
```

```
##      weight   id
## 1 0.4715253 2292
## 2 0.6862229 2692
## 3 0.6713151 4546
## 4 0.5692166 6749
## 5 0.8031763 7153
## 6 0.5694037 7806
```

```
# see weight distribution
plot(density(dest_wts$weight), main = 'Amenity Popularity Distribution')
```




```
# join column
ttm_wts <- left_join(ttm, dest_wts, by = c('toId'='id'))

# If any weights are undefined replace with 1
ttm_wts$weight[is.na(ttm_wts$weight)] <- 1

head(ttm_wts)
```

```
##        fromId toId  avg_time  sd_time    type    weight
## 1 59150004004   10  99.76316 1.850770  museum 0.4829706
## 2 59150004004   15  72.48718 1.482012  museum 0.2669118
## 3 59150004004  157  96.69231 1.386632 gallery 0.5354125
## 4 59150004004 1759 106.82051 1.684214  museum 0.3589059
## 5 59150004004 1760  46.58974 1.292792 gallery 0.4060835
## 6 59150004004 1822  76.64103 1.607443  museum 0.6582380
```

Sum Scoring Method

```
# scores with [1 - 100] df normalization


na.omit(ttm_wts)->ttm_wts


ttm_scores <- sum_score_fxn(ttm_wts,
                            weight = FALSE,
                            log_normalize_score = TRUE,
                            normalize_df = TRUE, x = 1, y = 100)
```

```
ttm_wtd_scores <- sum_score_fxn(ttm_wts,
                                weight = TRUE,
                                log_normalize_score = TRUE,
                                normalize_df = TRUE, x = 1, y = 100)
```

Sum Scoring Method 2 with mean plus sd

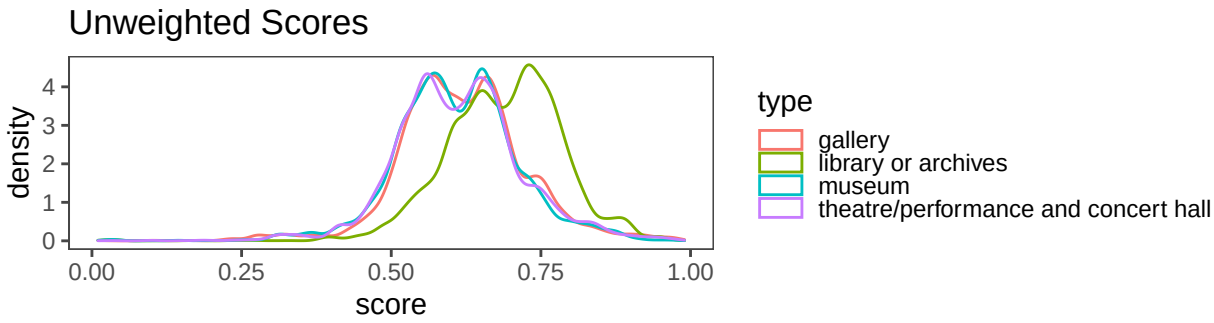
```
ttm_scores_2 <- sum_score_fxn_2(ttm_wts,
                                weight = FALSE,
                                log_normalize_score = TRUE,
                                normalize_df = TRUE, x = 1, y = 100)
```

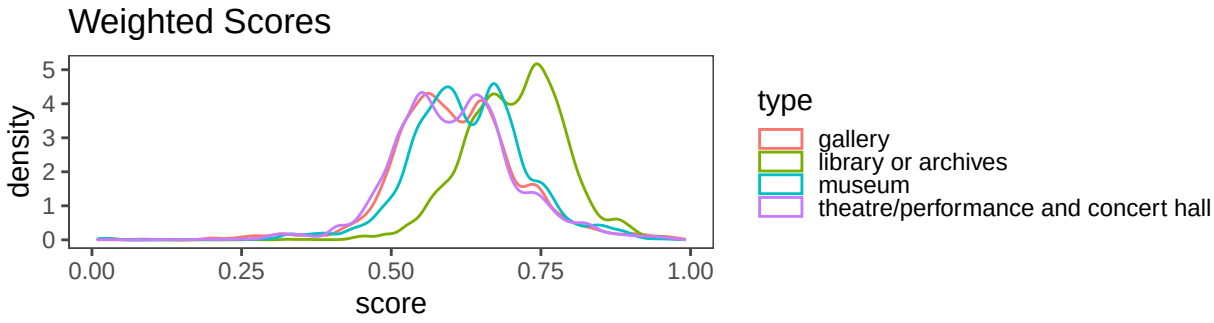
```
## `summarise()` has grouped output by 'fromId'. You can override using the `.groups` argument.
```

```
ttm_wtd_scores_2 <- sum_score_fxn_2(ttm_wts,
                                weight = TRUE,
                                log_normalize_score = TRUE,
                                normalize_df = TRUE, x = 1, y = 100)
```

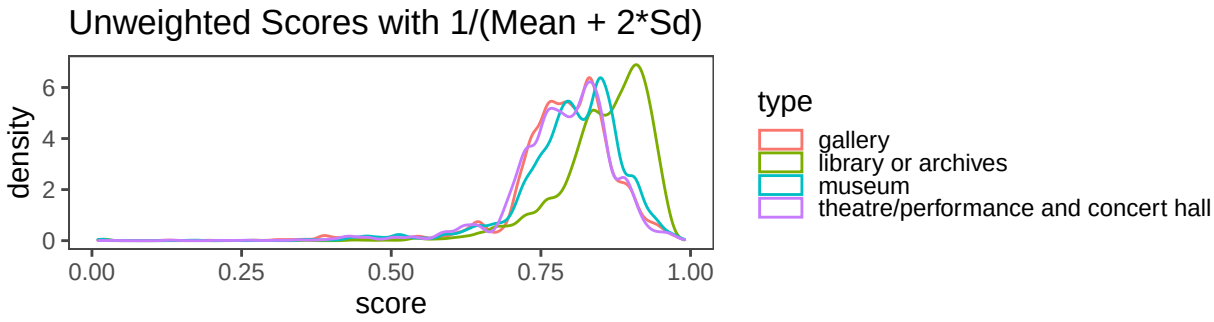
```
## `summarise()` has grouped output by 'fromId'. You can override using the `.groups` argument.
```

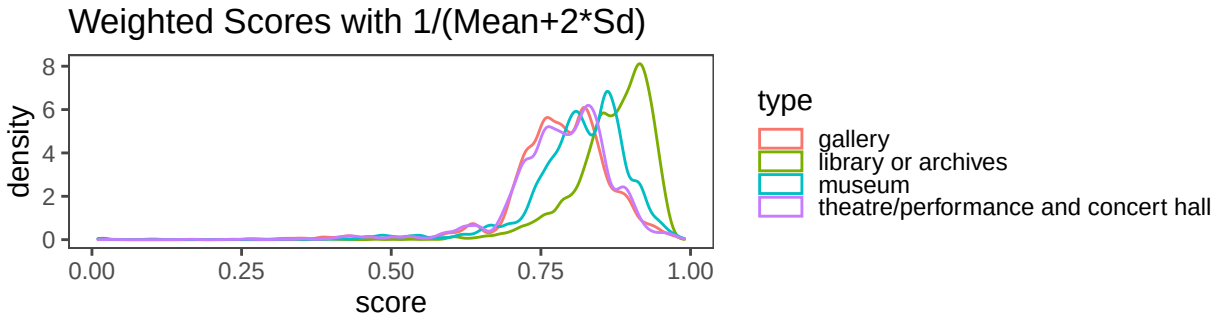
```
par(mfrow=c(1,4))
plot_densities(ttm_scores, ttm_wtd_scores, 'Unweighted Scores', 'Weighted Scores')
```






```
plot_densities(ttm_scores_2, ttm_wtd_scores_2, 'Unweighted Scores with 1/(Mean + 2*Sd)', 'Weighted Score
```





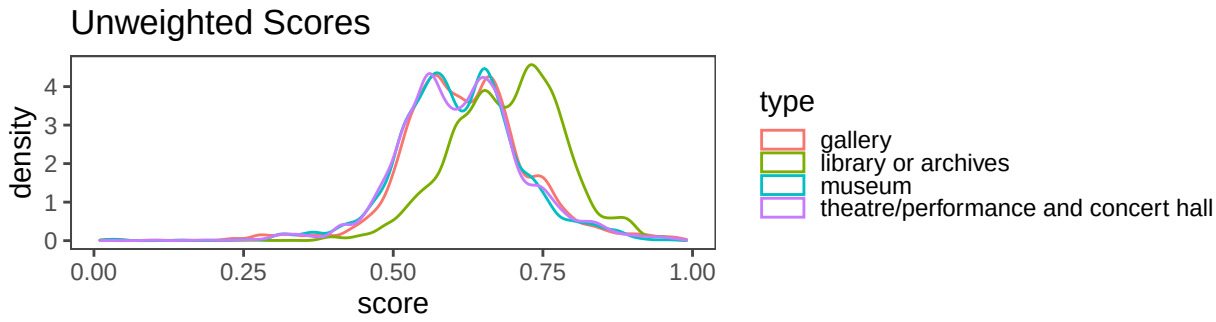

```
# scores with [0.01 - 0.99] df normalization

ttm_scores2 <- sum_score_fxn(ttm_wts,
                             weight = FALSE,
                             log_normalize_score = TRUE,
                             normalize_df = TRUE, x = 0.01, y = 0.99)
```

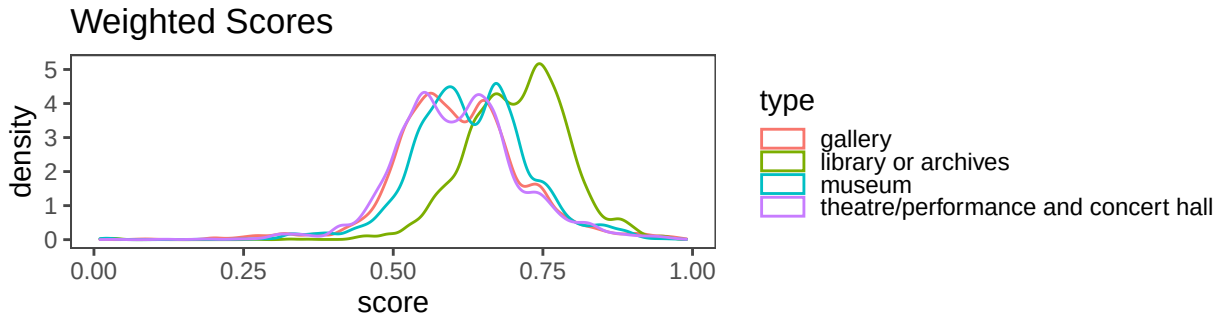
```
## `summarise()` has grouped output by 'fromId'. You can override using the `.groups` argument.
```

```
ttm_wtd_scores2 <- sum_score_fxn(ttm_wts,
                             weight = TRUE,
                             log_normalize_score = TRUE,
                             normalize_df = TRUE, x = 0.01, y = 0.99)
```

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

```
plot_densities(ttm_scores2, ttm_wtd_scores2, 'Unweighted Scores', 'Weighted Scores')
```





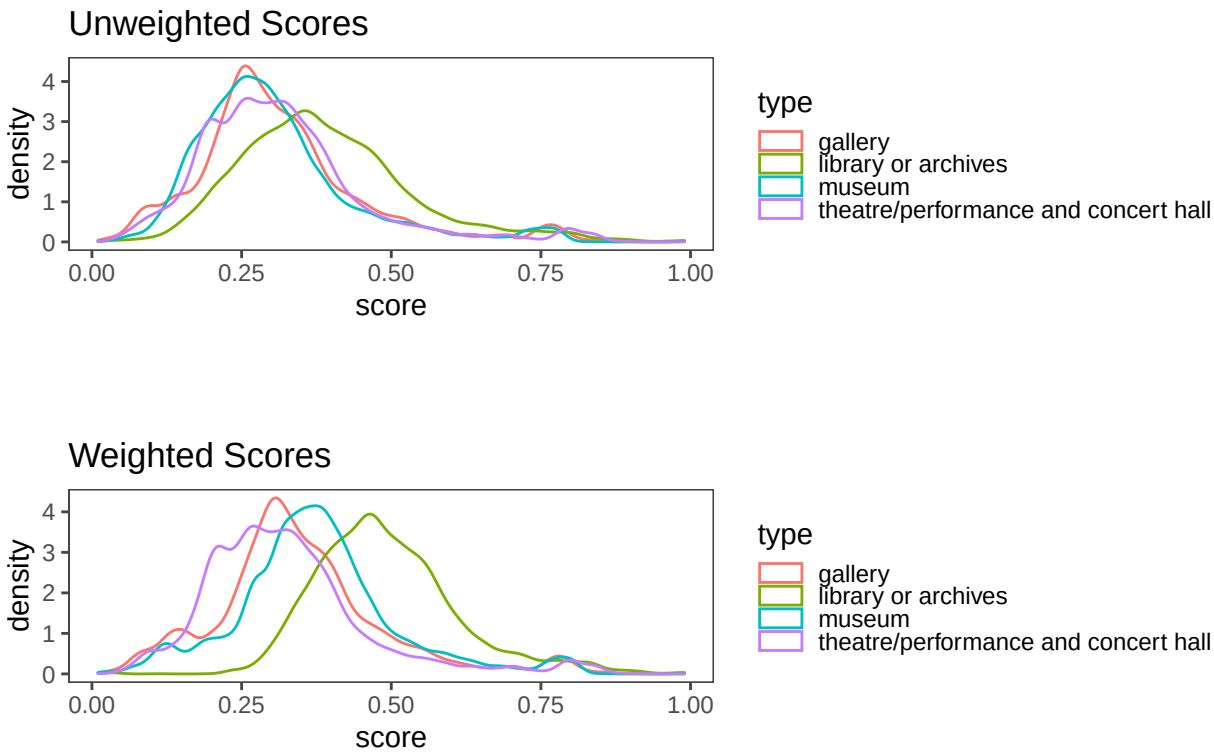
Sum Scoring for the Nearest 1, 2, or 3 Amenities

Note that for nearest 1, the sum is the value itself.

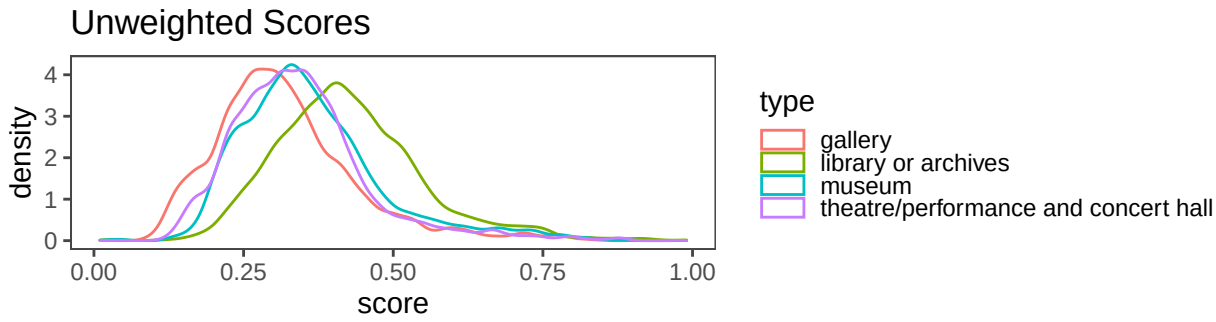
```
# Keep only the nearest 1, 2, or 3 travel times for each dissemination block

nearest_1_ttm <- ttm_wts %>%
                  group_by(fromId, type) %>%
                  summarise(avg_time = min(avg_time),
                            sd_time = sd_time[which.min(avg_time)],
                            weight = weight[which.min(avg_time)])

nearest_2_ttm <- ttm_wts %>%
                  group_by(fromId, type) %>%
                  summarise(avg_time = na.omit(sort(avg_time)[1:2]),
                            sd_time = sd_time[which(na.omit(avg_time == sort(avg_time)[1:2]))],
                            weight = weight[which(na.omit(avg_time == sort(avg_time)[1:2]))])

nearest_3_ttm <- ttm_wts %>%
                  group_by(fromId, type) %>%
                  summarise(avg_time = na.omit(sort(avg_time)[1:3]),
                            sd_time = sd_time[which(na.omit(avg_time == sort(avg_time)[1:3]))],
                            weight = weight[which(na.omit(avg_time == sort(avg_time)[1:3]))])


# scores by nearest amenities

n1_ttm_score <- sum_score_fxn(nearest_1_ttm, weight = FALSE, log_normalize_score = TRUE, normalize_df =
n1_wt_ttm_score <- sum_score_fxn(nearest_1_ttm, weight = TRUE, log_normalize_score = TRUE, normalize_df
```

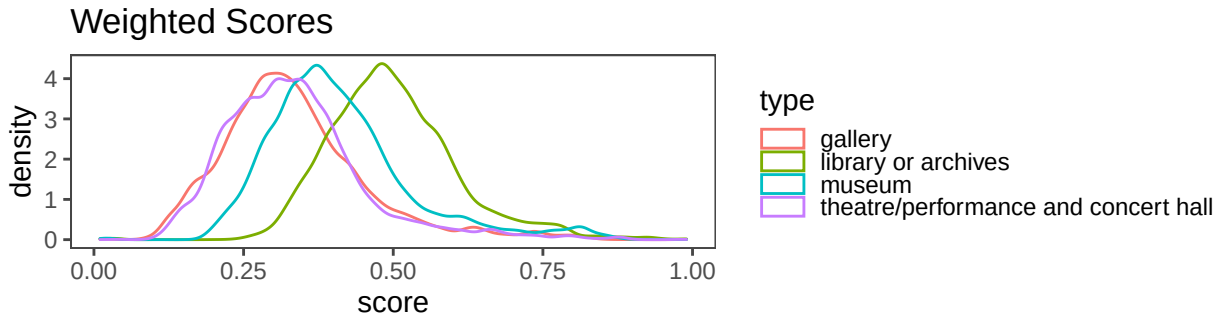
```
n2_ttm_score <- sum_score_fxn(nearest_2_ttm, weight = FALSE, log_normalize_score = TRUE, normalize_df =
n2_wt_ttm_score <- sum_score_fxn(nearest_2_ttm, weight = TRUE, log_normalize_score = TRUE, normalize_df

n3_ttm_score <- sum_score_fxn(nearest_3_ttm, weight = FALSE, log_normalize_score = TRUE, normalize_df =
n3_wt_ttm_score <- sum_score_fxn(nearest_3_ttm, weight = TRUE, log_normalize_score = TRUE, normalize_df
```

```
plot_densities(n1_ttm_score, n1_wt_ttm_score, 'Unweighted Scores', 'Weighted Scores')
```

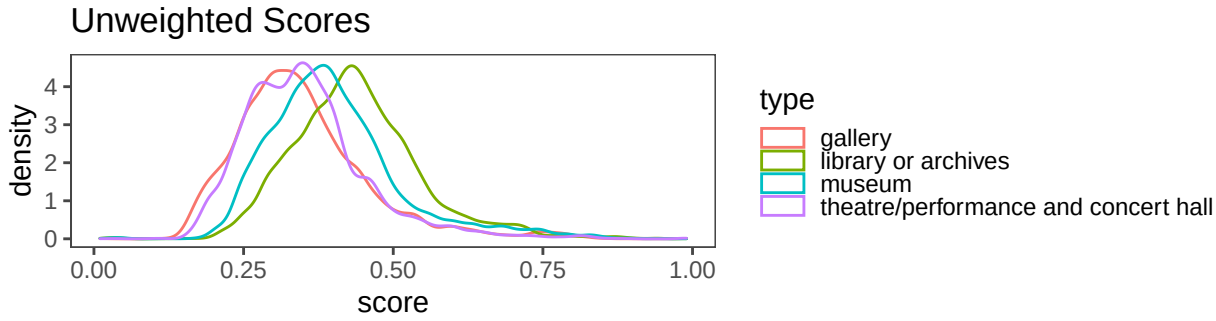


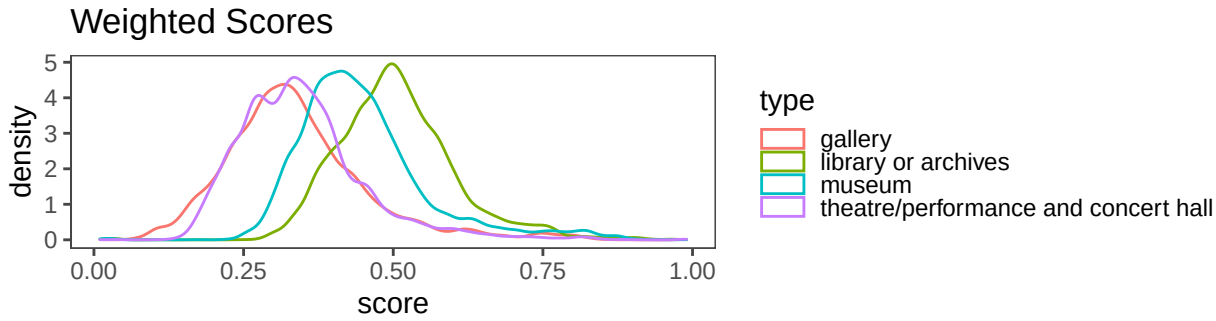

```
plot_densities(n2_ttm_score, n2_wt_ttm_score, 'Unweighted Scores', 'Weighted Scores')
```






```
plot_densities(n3_ttm_score, n3_wt_ttm_score, 'Unweighted Scores', 'Weighted Scores')
```





using score function of 1/(mean+2sd)

```
# scores by nearest amenities
```

```
n1_ttm_score_2 <- sum_score_fxn_2(nearest_1_ttm, weight = FALSE, log_normalize_score = TRUE, normalize_
```

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

```
n1_wt_ttm_score_2 <- sum_score_fxn_2(nearest_1_ttm, weight = TRUE, log_normalize_score = TRUE, normaliz
```

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

```
n2_ttm_score_2 <- sum_score_fxn_2(nearest_2_ttm, weight = FALSE, log_normalize_score = TRUE, normalize_
```

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

```
n2_wt_ttm_score_2 <- sum_score_fxn_2(nearest_2_ttm, weight = TRUE, log_normalize_score = TRUE, normaliz
```

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

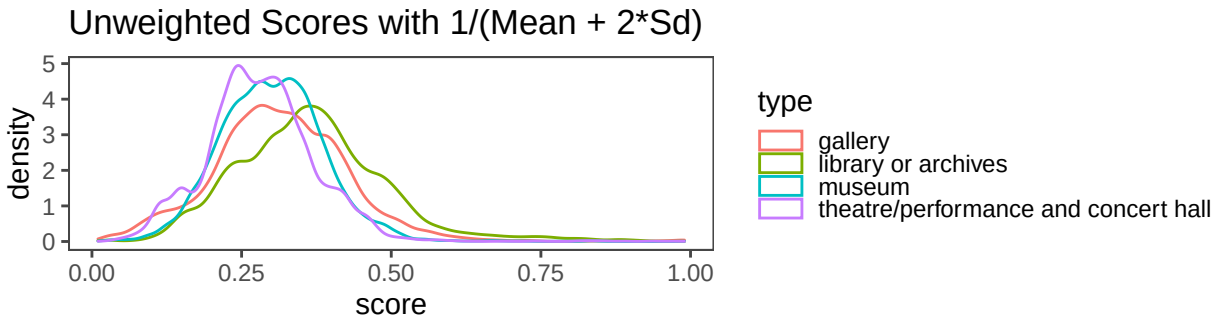
```
n3_ttm_score_2 <- sum_score_fxn_2(nearest_3_ttm, weight = FALSE, log_normalize_score = TRUE, normalize_
```

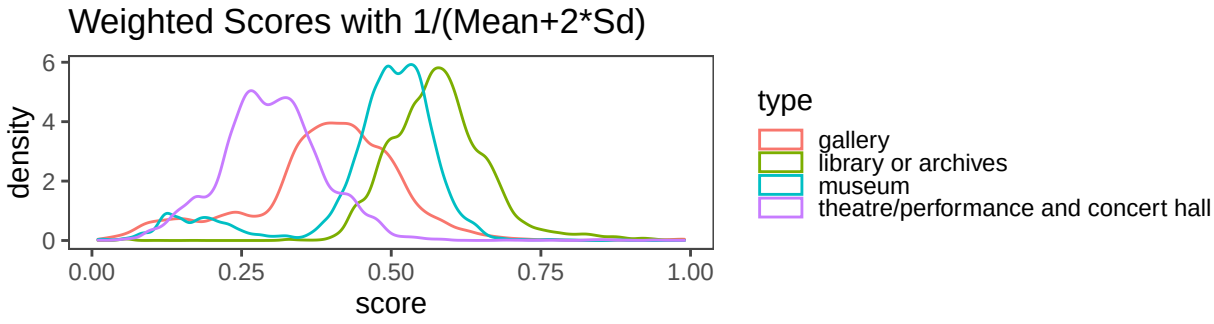
```
## `summarise()` has grouped output by 'fromId'. You can override using the `.groups` argument.
```

```
n3_wt_ttm_score_2 <- sum_score_fxn(nearest_3_ttm, weight = TRUE, log_normalize_score = TRUE, normalize_
```

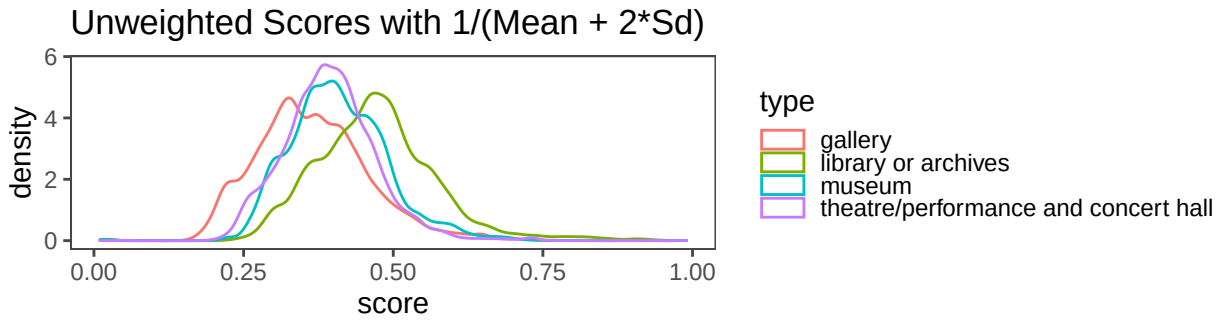
```
## `summarise()` has grouped output by 'fromId'. You can override using the `.groups` argument.
```

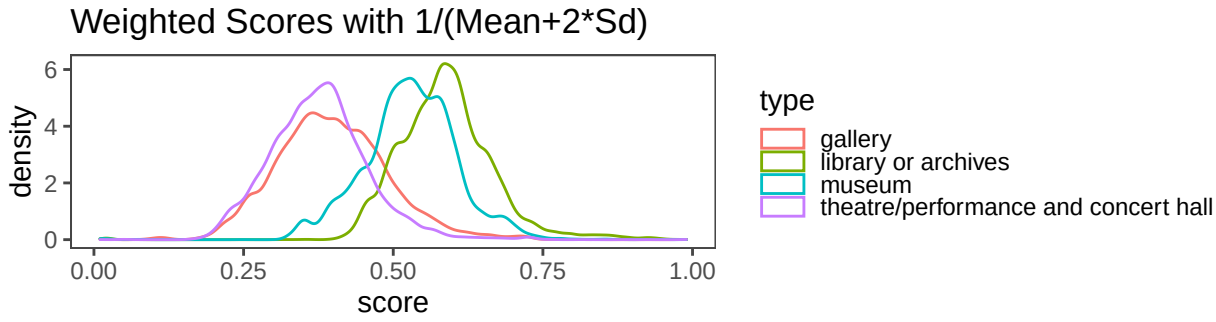
```
plot_densities(n1_ttm_score_2, n1_wt_ttm_score_2, 'Unweighted Scores with 1/(Mean + 2*Sd)', 'Weighted S
```



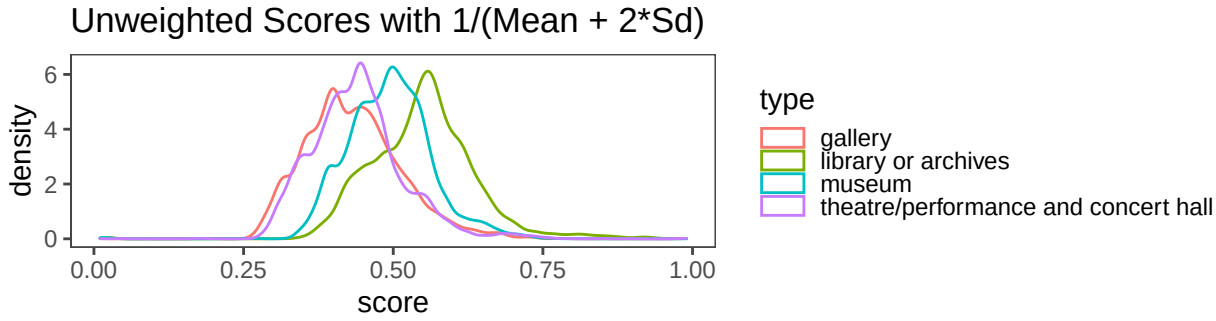


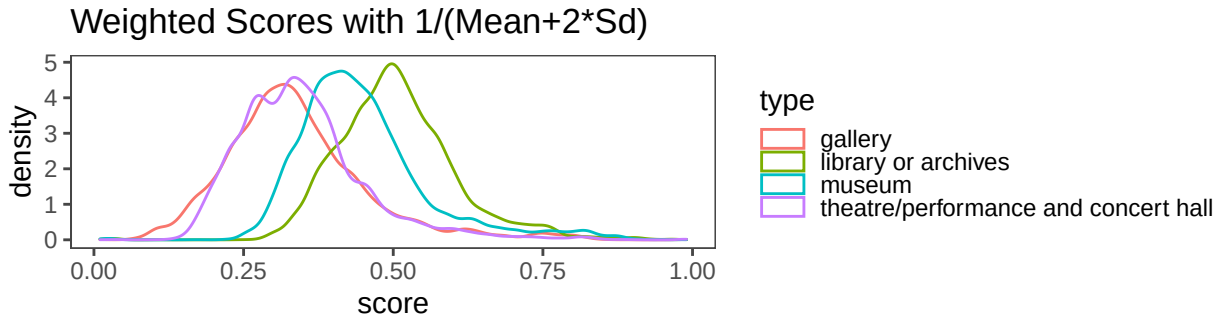

```
plot_densities(n2_ttm_score_2, n2_wt_ttm_score_2, 'Unweighted Scores with 1/(Mean + 2*Sd)', 'Weighted S
```






```
plot_densities(n3_ttm_score_2, n3_wt_ttm_score_2, 'Unweighted Scores with 1/(Mean + 2*Sd)', 'Weighted S
```





Exporting all Score Sets

```
## Add weight column for each score frame
```

```
ttm_scores$weight <- as.factor('no')
ttm_wtd_scores$weight <- as.factor('yes')

n1_ttm_score$weight <- as.factor('no')
n1_wt_ttm_score$weight <- as.factor('yes')

n2_ttm_score$weight <- as.factor('no')
n2_wt_ttm_score$weight <- as.factor('yes')

n3_ttm_score$weight <- as.factor('no')
n3_wt_ttm_score$weight <- as.factor('yes')

## Add nearest_n column for each score frame

ttm_scores$nearest_n <- as.factor('all')
ttm_wtd_scores$nearest_n <- as.factor('all')

n1_ttm_score$nearest_n <- as.factor('1')
n1_wt_ttm_score$nearest_n <- as.factor('1')

n2_ttm_score$nearest_n <- as.factor('2')
n2_wt_ttm_score$nearest_n <- as.factor('2')

n3_ttm_score$nearest_n <- as.factor('3')
n3_wt_ttm_score$nearest_n <- as.factor('3')

## Combine into a long dataframe
all_scores <- list(ttm_scores, ttm_wtd_scores,
                   n1_ttm_score, n1_wt_ttm_score,
                   n2_ttm_score, n2_wt_ttm_score,
                   n3_ttm_score, n3_wt_ttm_score)

long_scores <- data.table::rbindlist(all_scores) %>% arrange(fromId)

## Re-Order columns
long_scores <- long_scores[, c(1, 2, 4, 5, 3)]

## Export
write.csv(long_scores, '../../data/score_sets/long_scores.csv', row.names = FALSE)
```

Old Notes ~ Ignore or reuse later

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

Name	Function	Notes	Assumptions
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
Joseph Unweighted Sum	sum(1 / (normalized_transit_time_i * normalized_standard_time_i))	Sums the transit times as proportional to the mean, then normalizes the scores.	