quollr: An R Package for Visalizing 2D Models in High Dimensional Space

by Jayani P.G. Lakshika, Dianne Cook, Paul Harrison, Michael Lydeamore, and Thiyanga S. Talagala

Abstract An abstract of less than 150 words.

```
#library(quollr)
library(readr)
library(ggplot2)
library(dplyr)
library(ggbeeswarm)
library(Rtsne)
library(umap)
library(phateR)
library(reticulate)
library(rsample)

set.seed(20230531)

use_python("~/miniforge3/envs/pcamp_env/bin/python")
use_condaenv("pcamp_env")

reticulate::source_python(paste0(here::here(), "/scripts/function_scripts/Fit_PacMAP_code.py"))
reticulate::source_python(paste0(here::here(), "/scripts/function_scripts/Fit_TriMAP_code.py"))
```

1 Introduction

2 Methodology

Usage

- dependancies

```
library(tools)
package_dependencies("quollr")
```

- basic example

Compute hexagonal bin configurations

```
num_bins_x <- calculate_effective_x_bins(.data = s_curve_noise_umap, x = "UMAP1", hex_size = NA)
num_bins_x

#> [1] 4

num_bins_y <- calculate_effective_y_bins(.data = s_curve_noise_umap, y = "UMAP2", hex_size = NA)
num_bins_y

#> [1] 8
```

Generate full hex grid

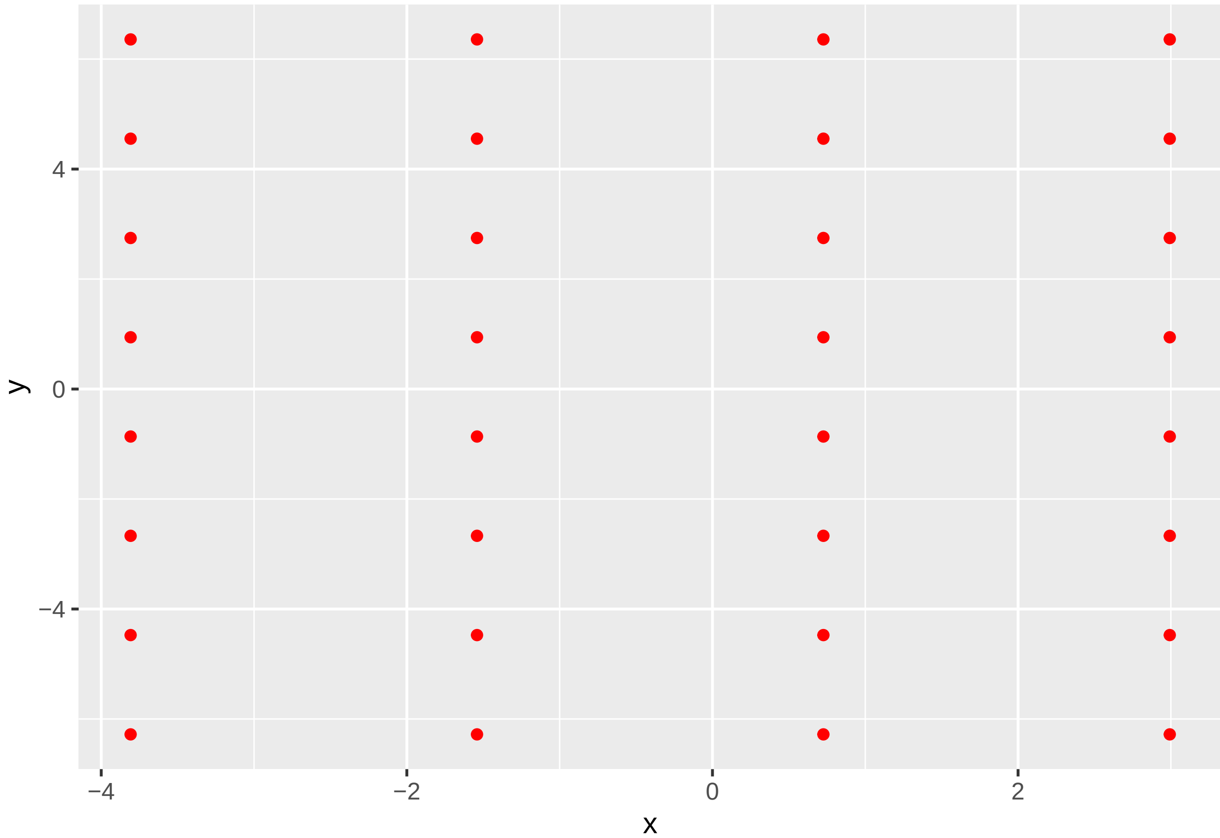
Generating full hexagonal grid contains main three steps:

1. Generate all the hexagonal bin centroids

   Steps:

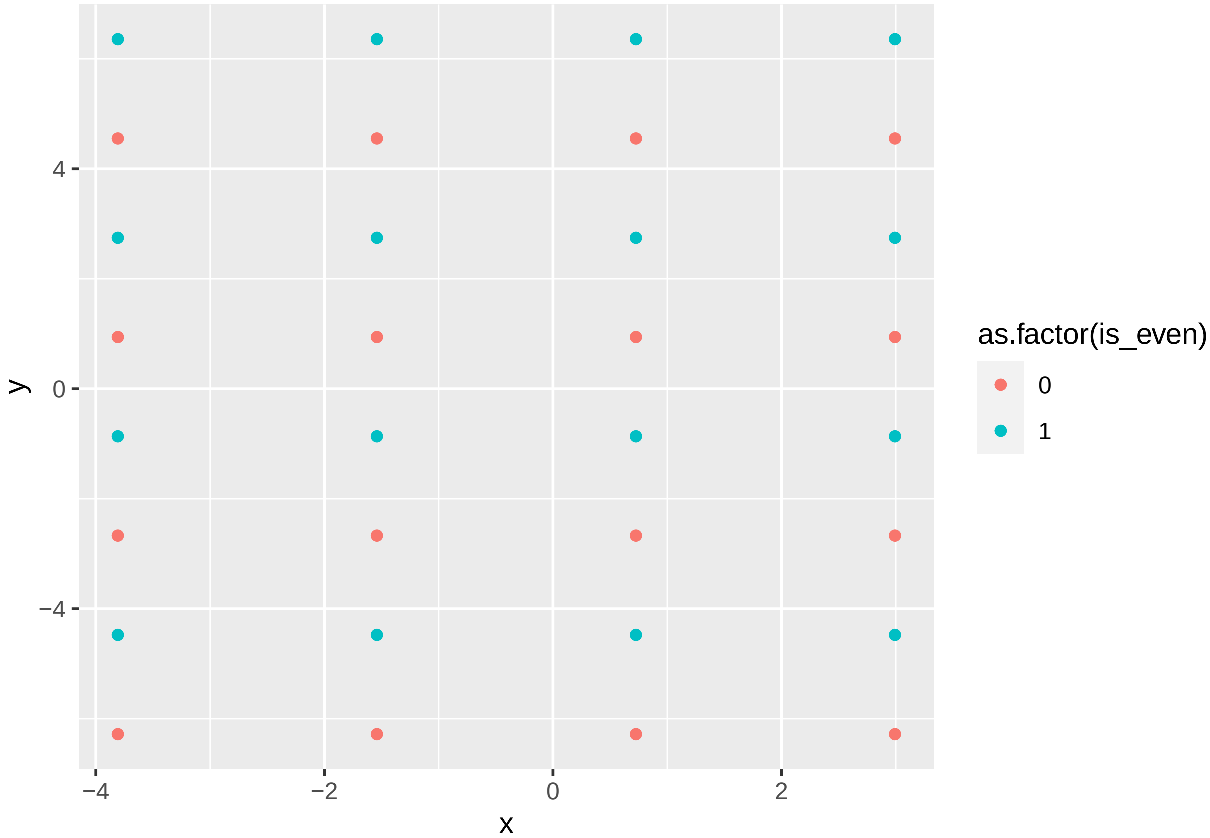
- First compute hex grid bound values along the x and y axis and generate the all the points wthin the hex box

```
cell_area <- 1

hex_size <- sqrt(2 * cell_area / sqrt(3))

buffer_size <- hex_size/2

x_bounds <- seq(min(s_curve_noise_umap[["UMAP1"]]) - buffer_size,
              max(s_curve_noise_umap[["UMAP1"]]) + buffer_size, length.out = num_bins_x)

y_bounds <- seq(min(s_curve_noise_umap[["UMAP2"]]) - buffer_size,
              max(s_curve_noise_umap[["UMAP2"]]) + buffer_size, length.out = num_bins_y)

box_points <- expand.grid(x = x_bounds, y = y_bounds)

ggplot() +
  geom_point(data = box_points, aes(x = x, y = y), color = "red")
```



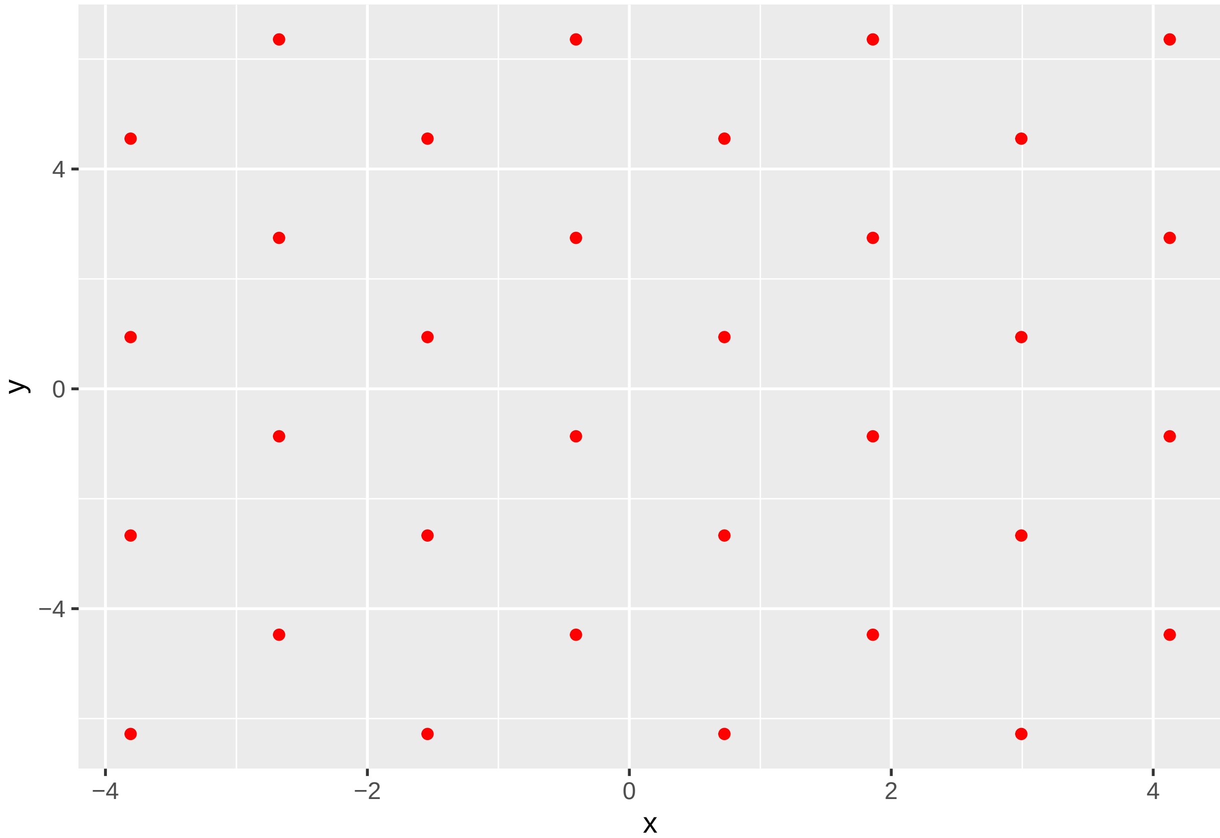
- Second for each x-value, find which y values are in the even row

```
 box_points <- box_points |>
    dplyr::arrange(x) |>
    dplyr::group_by(x) |>
    dplyr::group_modify(~ generate_even_y(.x)) |>
    tibble::as_tibble()

ggplot() +
  geom_point(data = box_points,
             aes(x = x, y = y, colour = as.factor(is_even)))
```



- Then, shift the x values of the even rows

```
## Shift for even values in x-axis
x_shift <- unique(box_points$x)[2] - unique(box_points$x)[1]


box_points$x <- box_points$x + x_shift/2 * ifelse(box_points$is_even == 1, 1, 0)

ggplot() +
  geom_point(data = box_points, aes(x = x, y = y), color = "red")
```



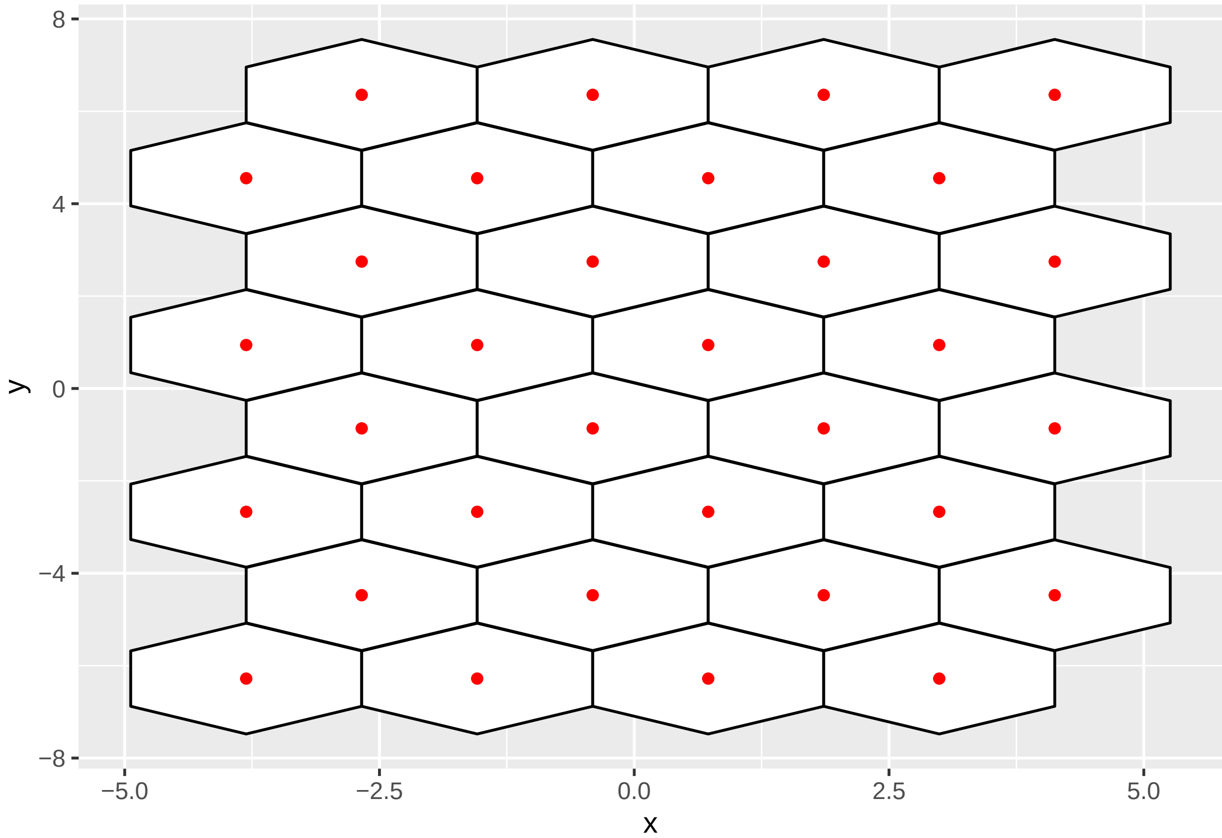

```
all_centroids_df <- generate_full_grid_centroids(nldr_df = s_curve_noise_umap,
                                                 x = "UMAP1", y = "UMAP2",
                                                 num_bins_x = num_bins_x,
                                                 num_bins_y = num_bins_y,
                                                 buffer_size = NA, hex_size = NA)

glimpse(all_centroids_df)

#> Rows: 32
#> Columns: 2
#> $ x <dbl> -3.8076427, -2.6742223, -3.8076427, -2.6742223, -3.8076427, -2.67422~
#> $ y <dbl> -6.2798254, -4.4744481, -2.6690708, -0.8636935, 0.9416838, 2.7470611~
```

2. Generate hexagonal coordinates

Steps: - Compute horizontal width of the hexagon

- Compute vertical width of the hexagon and multiply by a factor for overlapping ($sqrt(3)/2 *$ 1.15)
- Obtain hexagon polygon coordinates
- Obtain the number of hexagons in the full grid
- Generate the coordinates for the hexagons

```
hex_grid <- gen_hex_coordinates(all_centroids_df, hex_size = NA)
glimpse(hex_grid)

#> Rows: 192
#> Columns: 3
#> $ x  <dbl> -2.674222, -2.674222, -3.807643, -4.941063, -4.941063, -3.807643, -~
#> $ y  <dbl> -5.6804828, -6.8791681, -7.4785108, -6.8791681, -5.6804828, -5.0811~
#> $ id <int> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4~

ggplot(data = hex_grid, aes(x = x, y = y)) + geom_polygon(fill = "white", color = "black", aes(group = id)) +
  geom_point(data = all_centroids_df, aes(x = x, y = y), color = "red")
```

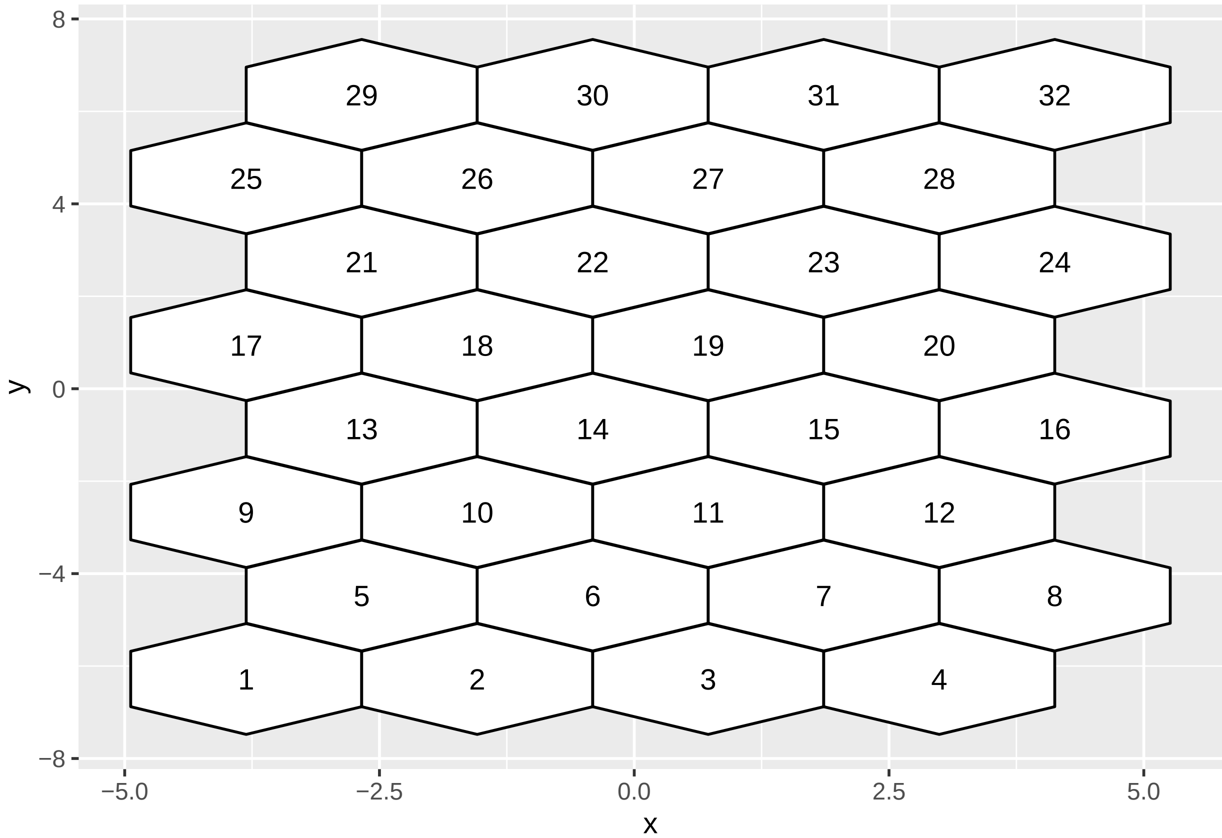


3. Map hexagonal IDs

Steps:

- Filter the data set with specific y value
- Order the x values for a specific y value
- Repeat the process for all unique y values

```
full_grid_with_hexbin_id <- map_hexbin_id(all_centroids_df)

ggplot(data = hex_grid, aes(x = x, y = y)) + geom_polygon(fill = "white", color = "black", aes(group = id)) +
  geom_text(data = full_grid_with_hexbin_id, aes(x = c_x, y = c_y, label = hexID))
```



4. Map polygon IDs

Steps:

- Filter specific hexagon
- Filter specific polygon
- Check the selected hexagonal centroid exists within the polygon
- if so assign that id to centroid, if not check until find the polygon which contains the centroid

```
full_grid_with_polygon_id <- map_polygon_id(full_grid_with_hexbin_id, hex_grid)
```

4. Assign data into hexagons

- Compute distances between nldr coordinates and hex bin centroids
- Find the hexagonal centroid that have the minimum distance

```
s_curve_noise_umap_with_id <- assign_data(s_curve_noise_umap, full_grid_with_hexbin_id)
```

5. Compute standardized counts

- Compute number of data points within each hexagon
- Compute standardise count by dividing the counts by the maximum

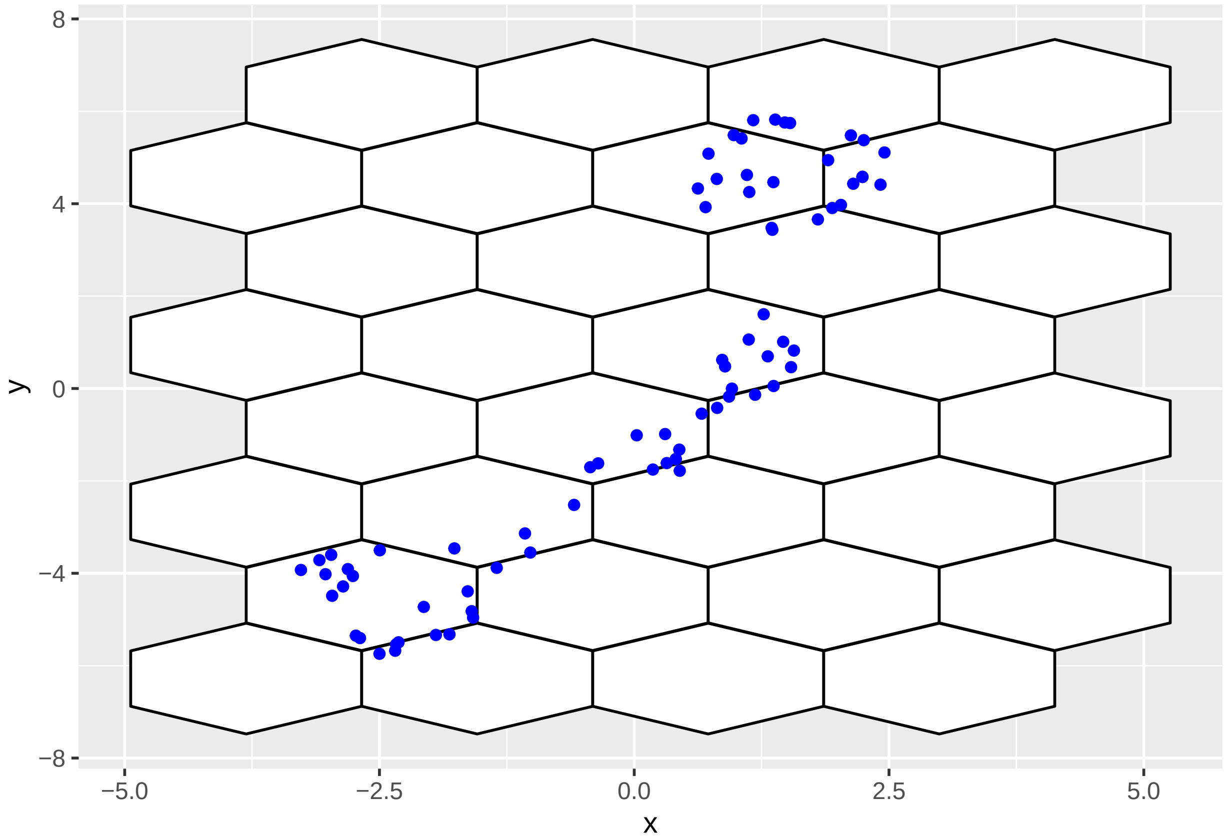
```
df_with_std_counts <- compute_std_counts(nldr_df = s_curve_noise_umap_with_id)
```

6. Extract full grid info

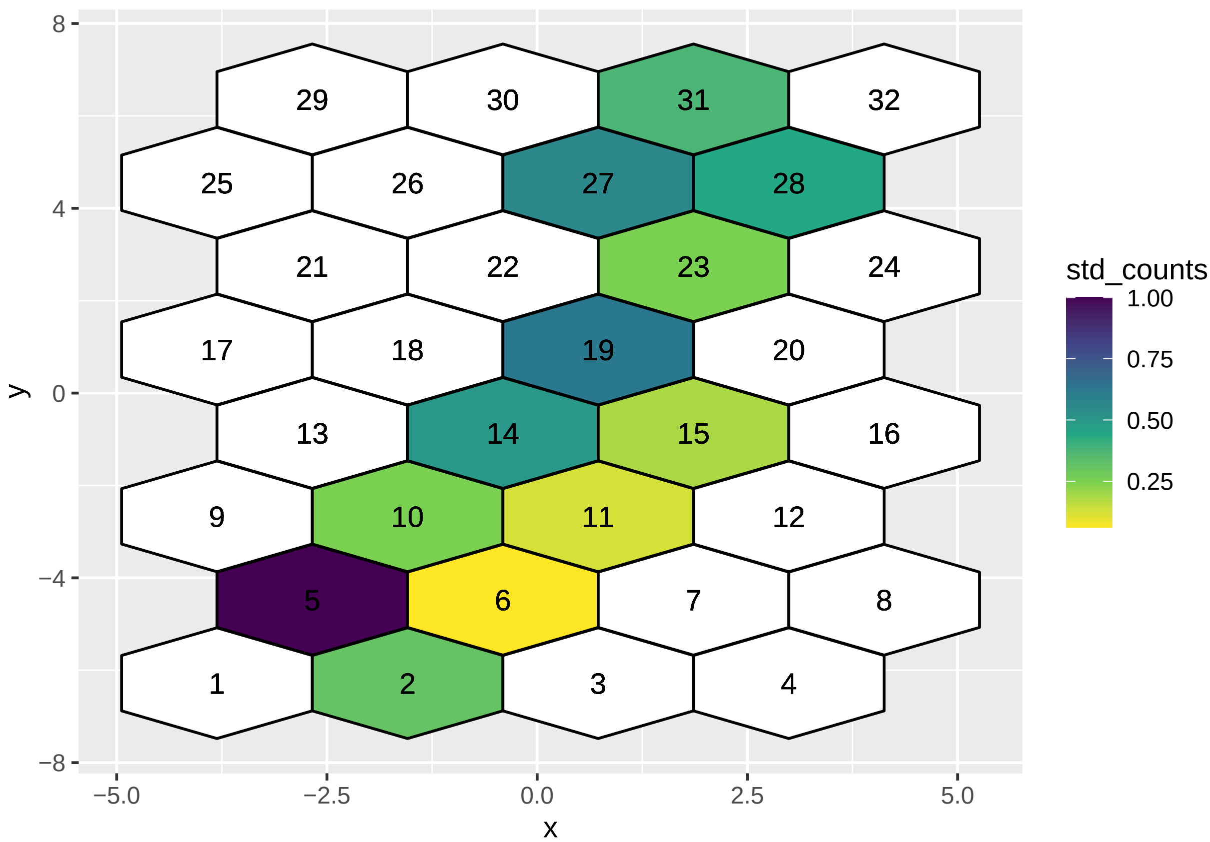
- Assign standardize counts for hex bins
- Join with the hexagonal coordinates

```
hex_full_count_df <- generate_full_grid_info(full_grid_with_polygon_id, df_with_std_counts, hex_grid)
```

```
ggplot(data = hex_grid, aes(x = x, y = y)) + geom_polygon(fill = "white", color = "black", aes(group = id)) +
  geom_point(data = s_curve_noise_umap, aes(x = UMAP1, y = UMAP2), color = "blue")
```




```
ggplot(data = hex_full_count_df, aes(x = x, y = y)) +
  geom_polygon(color = "black", aes(group = polygon_id, fill = std_counts)) +
  geom_text(aes(x = c_x, y = c_y, label = hexID)) +
  scale_fill_viridis_c(direction = -1, na.value = "#ffffff")
```



Buffer size When generating hexagonal bins in R, a buffer is often included to ensure that the data points are evenly distributed within the bins and to prevent edge effects. The buffer helps in two main ways:

1. **Preventing Edge Effects**: Without a buffer, the outermost data points might fall near the boundary of the hexagonal grid, leading to incomplete bins or uneven distribution of data. By adding a buffer, you create a margin around the outer edges of the grid, ensuring that all data points are fully enclosed within the bins.
2. **Ensuring Even Distribution**: The buffer allows for a smoother transition between adjacent bins. This helps in cases where data points are not perfectly aligned with the grid lines, ensuring that each data point is assigned to the nearest bin without bias towards any specific direction.

Overall, including a buffer when generating hexagonal bins helps to produce more accurate and robust binning results, particularly when dealing with real-world data that may have irregular distributions or boundary effects.

Construct the 2D model with different options

Construct the high-D model with different options

```
## To generate a data set with high-D and 2D training data
df_all <- training_data |> dplyr::select(-ID) |>
  dplyr::bind_cols(s_curve_noise_umap_with_id)

## To generate averaged high-D data

df_bin <- avg_highD_data(.data = df_all, column_start_text = "x") ## Need to pass ID column name
```

Generate the triangular mesh

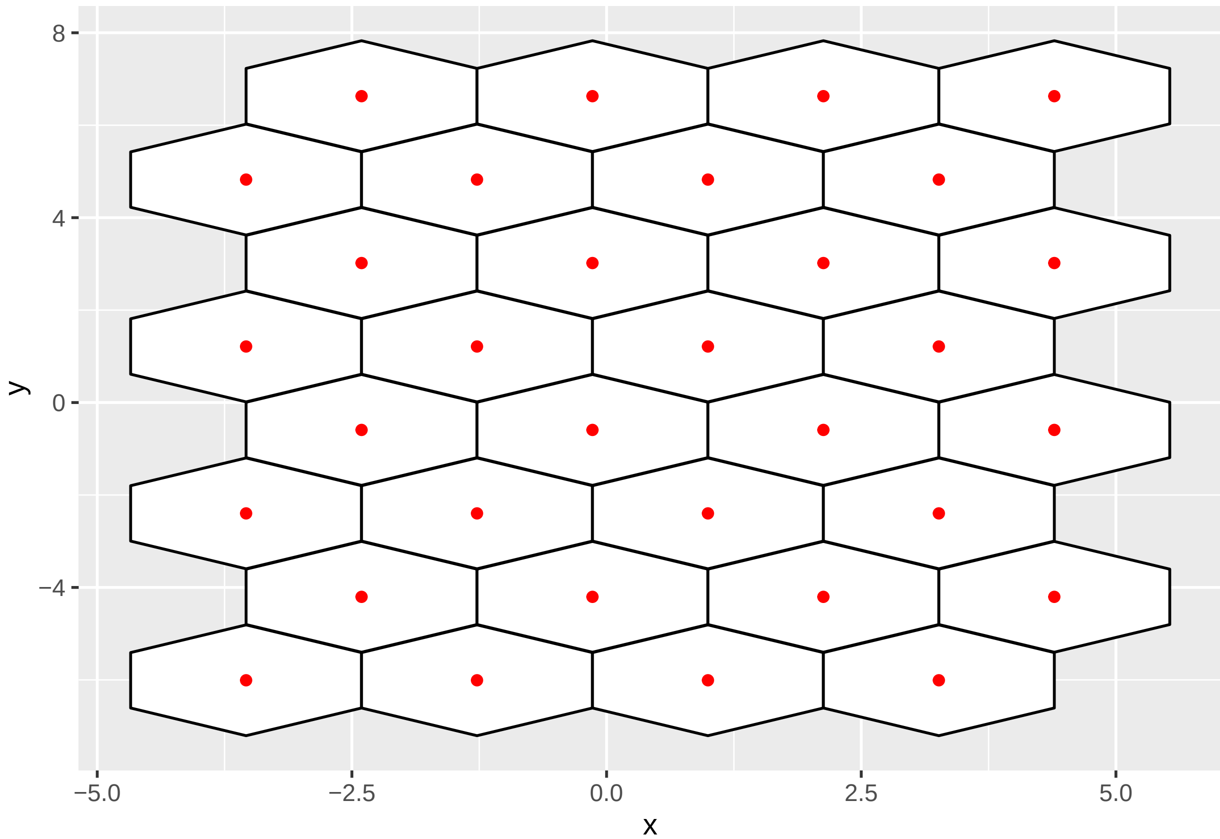
```
df_bin_centroids <- hex_full_count_df[complete.cases(hex_full_count_df[["std_counts"]]), ] |>
  dplyr::select("c_x", "c_y", "hexID", "std_counts") |>
  dplyr::distinct() |>
  dplyr::rename(c("x" = "c_x", "y" = "c_y"))

df_bin_centroids

#>             x          y hexID std_counts
#> 1  -2.6742223 -4.4744481     5     1.0000
#> 2  -1.5408019 -6.2798254     2     0.3125
#> 3  -0.4073814 -4.4744481     6     0.0625
#> 4  -1.5408019 -2.6690708    10     0.2500
#> 5  -0.4073814 -0.8636935    14     0.5000
#> 6   0.7260390 -2.6690708    11     0.1250
#> 7   1.8594594 -0.8636935    15     0.1875
#> 8   0.7260390  0.9416838    19     0.6250
#> 9   1.8594594  2.7470611    23     0.2500
#> 10  0.7260390  4.5524384    27     0.5625
#> 11  1.8594594  6.3578158    31     0.3750
#> 12  2.9928798  4.5524384    28     0.4375

tr1_object <- triangulate_bin_centroids(df_bin_centroids, x, y)
tr_from_to_df <- generate_edge_info(triangular_object = tr1_object)
```

Compute parameter defaults

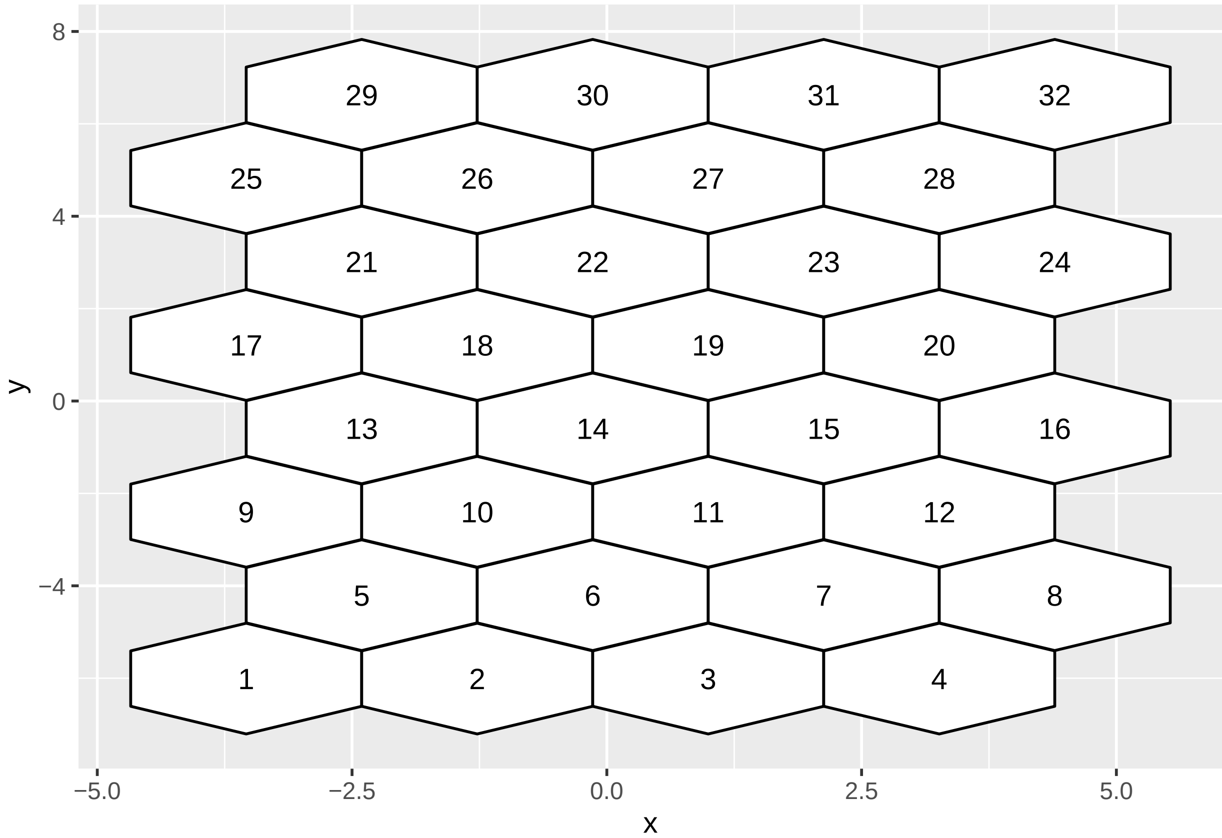
Shift the hexagonal grid origin If shift_x happen to the positive direction of x it should input as a positive value, if not other way If shift_y happen to the positive direction of y it should input as a positive value, if not other way

1. Assign shift along the x and y axis (limited the amount should less than the cell_diameter)
2. Generate bounds with shift origin

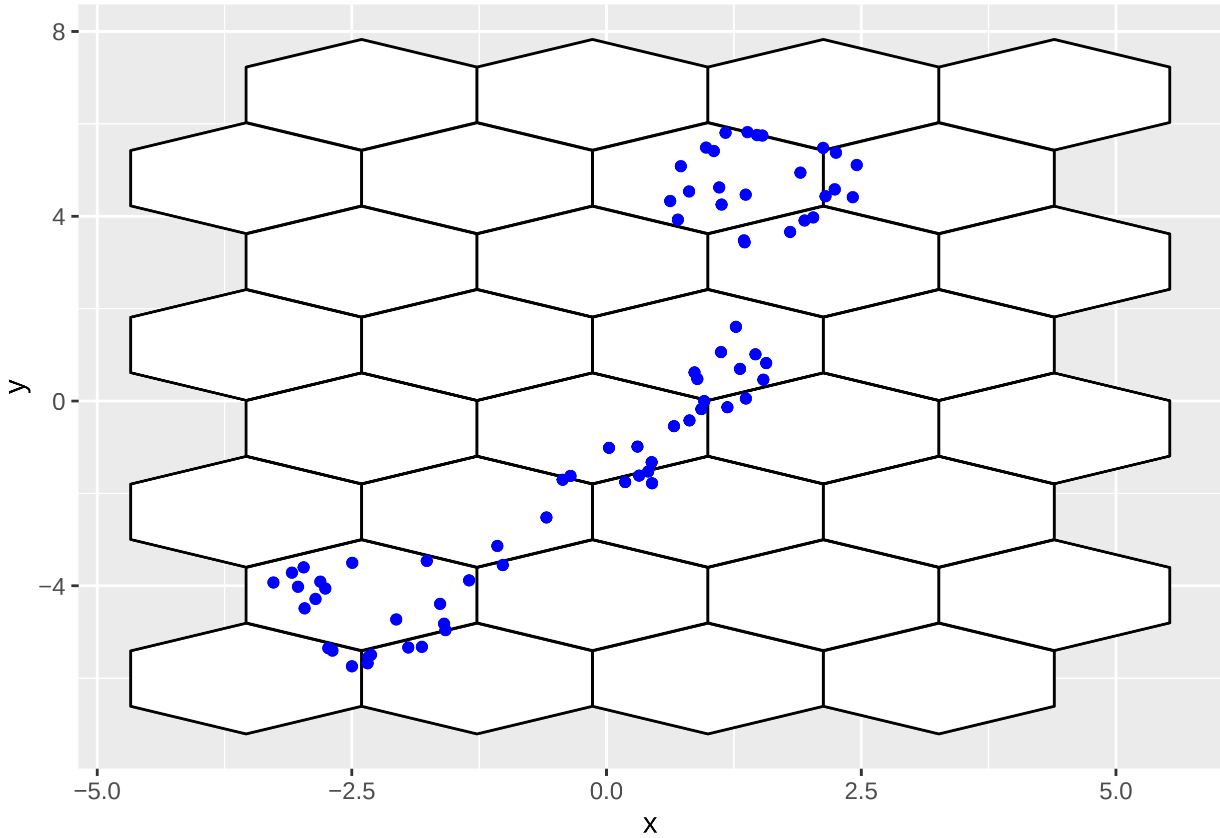
```
all_centroids_df_shift <- extract_coord_of_shifted_hex_grid(nldr_df = s_curve_noise_umap,
                                                    x = "UMAP1", y = "UMAP2",
                                                    num_bins_x = num_bins_x,
                                                    num_bins_y = num_bins_y,
                                                  shift_x = 0.2690002, shift_y = 0.271183,
                                                    buffer_size = NA, hex_size = NA)

glimpse(all_centroids_df_shift)

#> Rows: 32
#> Columns: 2
#> $ x <dbl> -3.5386425, -2.4052221, -3.5386425, -2.4052221, -3.5386425, -2.40522~
#> $ y <dbl> -6.0086424, -4.2032651, -2.3978878, -0.5925105, 1.2128668, 3.0182441~

hex_grid <- gen_hex_coordinates(all_centroids_df_shift)
glimpse(hex_grid)

#> Rows: 192
#> Columns: 3
#> $ x  <dbl> -2.405222, -2.405222, -3.538643, -4.672063, -4.672063, -3.538643, -~
#> $ y  <dbl> -5.409299776, -6.607985117, -7.207327787, -6.607985117, -5.40929977~
#> $ id <int> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4~

ggplot(data = hex_grid, aes(x = x, y = y)) + geom_polygon(fill = "white", color = "black", aes(group = id)) +
  geom_point(data = all_centroids_df_shift, aes(x = x, y = y), color = "red")
```



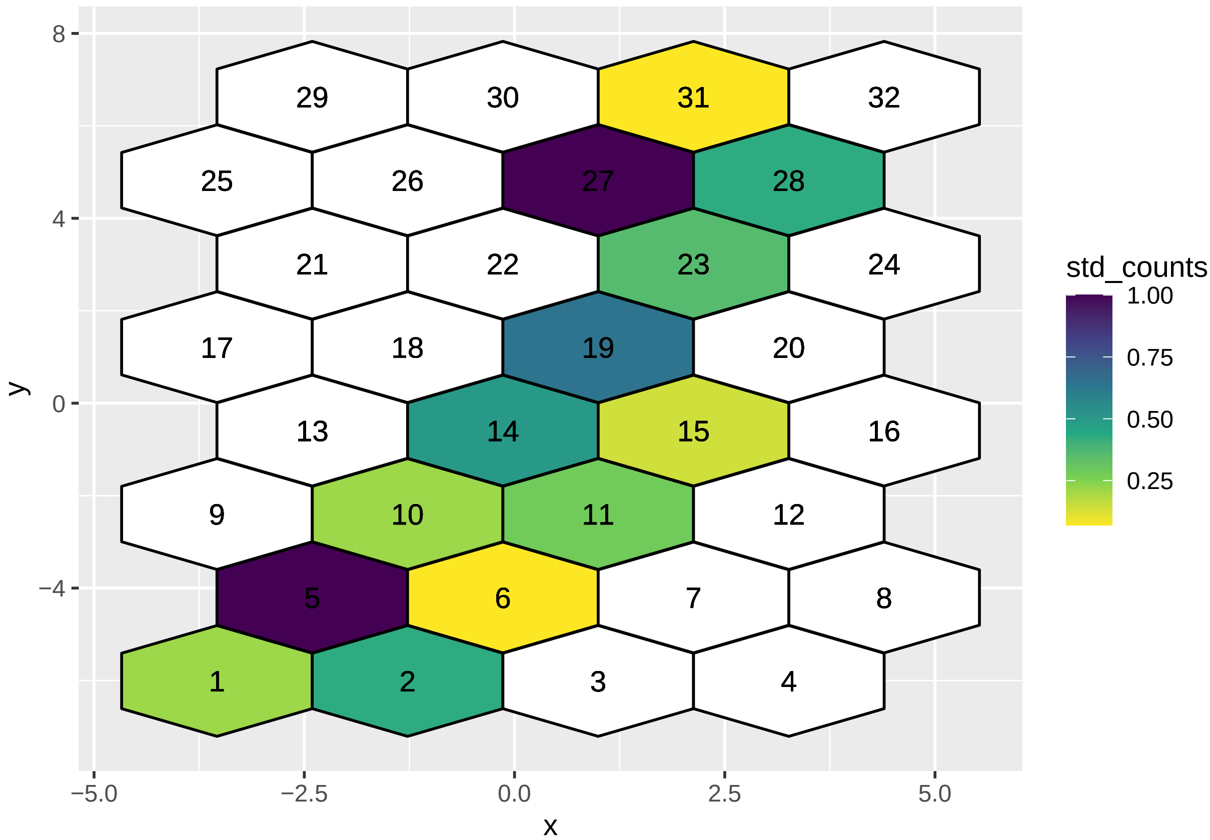

```
full_grid_with_hexbin_id <- map_hexbin_id(all_centroids_df_shift)

ggplot(data = hex_grid, aes(x = x, y = y)) + geom_polygon(fill = "white", color = "black", aes(group = id)) +
  geom_text(data = full_grid_with_hexbin_id, aes(x = c_x, y = c_y, label = hexID))
```




```
full_grid_with_polygon_id <- map_polygon_id(full_grid_with_hexbin_id, hex_grid)

s_curve_noise_umap_with_id <- assign_data(s_curve_noise_umap, full_grid_with_hexbin_id)

df_with_std_counts <- compute_std_counts(nldr_df = s_curve_noise_umap_with_id)

hex_full_count_df <- generate_full_grid_info(full_grid_with_polygon_id, df_with_std_counts, hex_grid)

ggplot(data = hex_grid, aes(x = x, y = y)) + geom_polygon(fill = "white", color = "black", aes(group = id)) +
  geom_point(data = s_curve_noise_umap, aes(x = UMAP1, y = UMAP2), color = "blue")
```




```
ggplot(data = hex_full_count_df, aes(x = x, y = y)) +
  geom_polygon(color = "black", aes(group = polygon_id, fill = std_counts)) +
  geom_text(aes(x = c_x, y = c_y, label = hexID)) +
  scale_fill_viridis_c(direction = -1, na.value = "#ffffff")
```




```
df_bin_centroids <- hex_full_count_df[complete.cases(hex_full_count_df[["std_counts"]]), ] |>
```

```
  dplyr::select("c_x", "c_y", "hexID", "std_counts") |>
  dplyr::distinct() |>
  dplyr::rename(c("x" = "c_x", "y" = "c_y"))

df_bin_centroids

#>             x          y hexID std_counts
#> 1  -3.5386425 -6.0086424     1 0.21428571
#> 2  -2.4052221 -4.2032651     5 1.00000000
#> 3  -1.2718017 -6.0086424     2 0.42857143
#> 4  -0.1383812 -4.2032651     6 0.07142857
#> 5  -1.2718017 -2.3978878    10 0.21428571
#> 6  -0.1383812 -0.5925105    14 0.50000000
#> 7   0.9950392 -2.3978878    11 0.28571429
#> 8   2.1284596 -0.5925105    15 0.14285714
#> 9   0.9950392  1.2128668    19 0.64285714
#> 10  2.1284596  3.0182441    23 0.35714286
#> 11  0.9950392  4.8236214    27 1.00000000
#> 12  2.1284596  6.6289988    31 0.07142857
#> 13  3.2618800  4.8236214    28 0.42857143

tr1_object <- triangulate_bin_centroids(df_bin_centroids, x, y)
tr_from_to_df <- generate_edge_info(triangular_object = tr1_object)

bin_centroids_shift <- ggplot(data = hex_full_count_df, aes(x = c_x, y = c_y)) +
  geom_point(color = "#bdbdbd") +
  geom_point(data = shifted_hex_coord_df, aes(x = c_x, y = c_y), color = "#feb24c") +
  coord_cartesian(xlim = c(-5, 8), ylim = c(-10, 10)) +
  theme_void() +
 theme(legend.position="none", legend.direction="horizontal", plot.title = element_text(size = 7, hjust = 0.5, v
       axis.title.x = element_blank(), axis.title.y = element_blank(),
       axis.text.x = element_blank(), axis.ticks.x = element_blank(),
       axis.text.y = element_blank(), axis.ticks.y = element_blank(),
     panel.grid.major = element_blank(), panel.grid.minor = element_blank(), #change legend key width
       legend.title = element_text(size=8), #change legend title font size
       legend.text = element_text(size=6)) +
  guides(fill = guide_colourbar(title = "Standardized count")) +
 annotate(geom = 'text', label = "a", x = -Inf, y = Inf, hjust = -0.3, vjust = 1, size = 3)

hex_grid_shift <- ggplot(data = shifted_hex_coord_df, aes(x = x, y = y)) +
  geom_polygon(fill = NA, color = "#feb24c", aes(group = polygon_id)) +
  geom_polygon(data = hex_full_count_df, aes(x = x, y = y, group = polygon_id),
               fill = NA, color = "#bdbdbd") +
  coord_cartesian(xlim = c(-5, 8), ylim = c(-10, 10)) +
  theme_void() +
 theme(legend.position="none", legend.direction="horizontal", plot.title = element_text(size = 7, hjust = 0.5, v
       axis.title.x = element_blank(), axis.title.y = element_blank(),
       axis.text.x = element_blank(), axis.ticks.x = element_blank(),
       axis.text.y = element_blank(), axis.ticks.y = element_blank(),
     panel.grid.major = element_blank(), panel.grid.minor = element_blank(), #change legend key width
       legend.title = element_text(size=8), #change legend title font size
       legend.text = element_text(size=6)) +
  guides(fill = guide_colourbar(title = "Standardized count")) +
 annotate(geom = 'text', label = "b", x = -Inf, y = Inf, hjust = -0.3, vjust = 1, size = 3)

## Before shift
before_shift_plot <- ggplot(data = hex_full_count_df, aes(x = x, y = y)) +
  geom_polygon(color = "black", aes(group = polygon_id, fill = std_counts)) +
  geom_text(aes(x = c_x, y = c_y, label = hexID), size = 2) +
  scale_fill_viridis_c(direction = -1, na.value = "#ffffff", option = "C") +
  coord_equal() +
  theme_void() +
 theme(legend.position="bottom", legend.direction="horizontal", plot.title = element_text(size = 7, hjust = 0.5
       axis.title.x = element_blank(), axis.title.y = element_blank(),
```

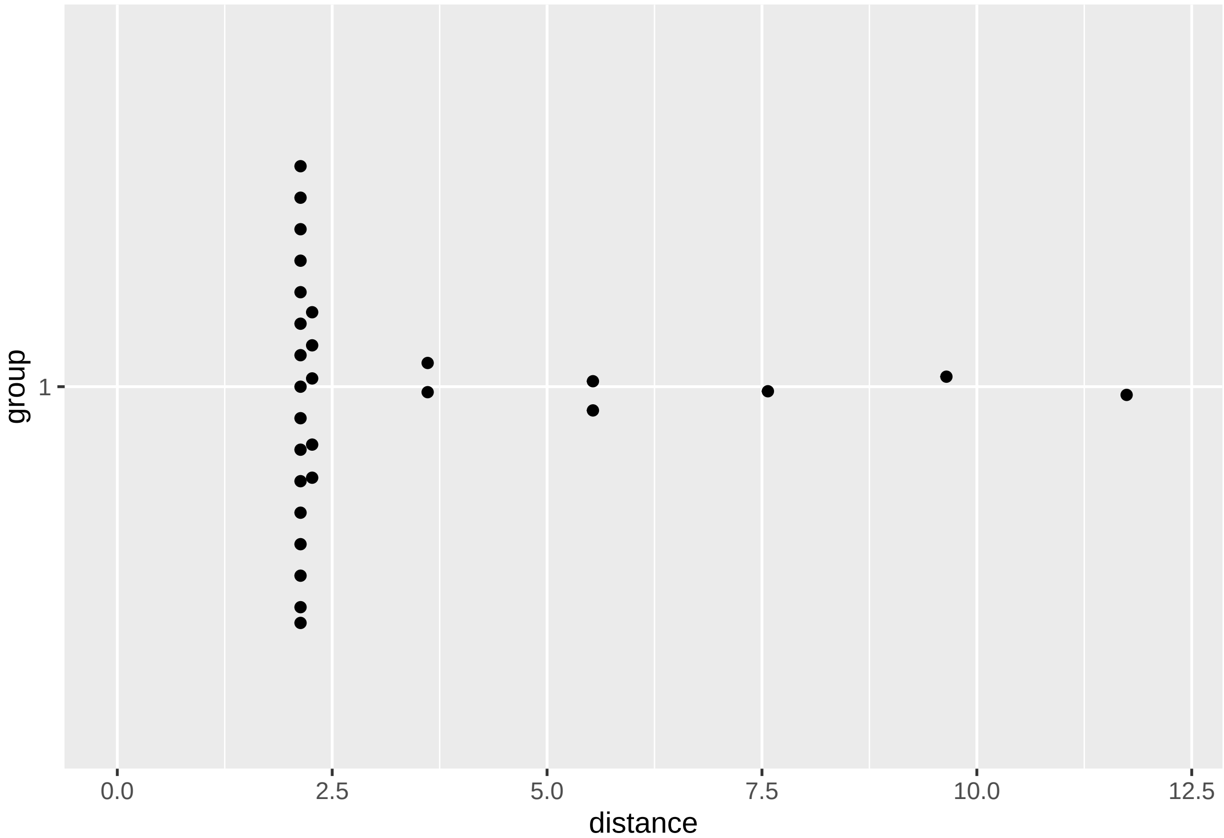
```
        axis.text.x = element_blank(), axis.ticks.x = element_blank(),
        axis.text.y = element_blank(), axis.ticks.y = element_blank(),
      panel.grid.major = element_blank(), panel.grid.minor = element_blank(), #change legend key width
        legend.title = element_text(size=8), #change legend title font size
        legend.text = element_text(size=6)) +
  guides(fill = guide_colourbar(title = "Standardized count")) +
  annotate(geom = 'text', label = "a", x = -Inf, y = Inf, hjust = -0.3, vjust = 1, size = 3)


## After shift
after_shift_plot <- ggplot(data = shifted_hex_coord_df, aes(x = x, y = y)) +
  geom_polygon(color = "black", aes(group = polygon_id, fill = std_counts)) +
  geom_text(aes(x = c_x, y = c_y, label = hexID), size = 2) +
  scale_fill_viridis_c(direction = -1, na.value = "#ffffff", option = "C") +
  coord_equal() +
  theme_void() +
  theme(legend.position="none", legend.direction="horizontal", plot.title = element_text(size = 7, hjust = 0.5, v
        axis.title.x = element_blank(), axis.title.y = element_blank(),
        axis.text.x = element_blank(), axis.ticks.x = element_blank(),
        axis.text.y = element_blank(), axis.ticks.y = element_blank(),
      panel.grid.major = element_blank(), panel.grid.minor = element_blank(), #change legend key width
        legend.title = element_text(size=8), #change legend title font size
        legend.text = element_text(size=6)) +
  guides(fill = guide_colourbar(title = "Standardized count")) +
  annotate(geom = 'text', label = "b", x = -Inf, y = Inf, hjust = -0.3, vjust = 1, size = 3)
```

Benchmark value to remove the low-density hexagons

```
## As an option first quantile considered as a default
benchmark_to_rm_lwd_hex <- quantile(df_bin_centroids$std_counts)[2] + 0.01

## To identify low density hexagons
df_bin_centroids_low <- df_bin_centroids |>
  dplyr::filter(std_counts <= benchmark_to_rm_lwd_hex)

## To identify low-density hexagons needed to remove by investigating neighbouring mean density
identify_rm_bins <- find_low_density_hexagons(df_bin_centroids_all = df_bin_centroids, num_bins_x = num_bins_x,
                     df_bin_centroids_low = df_bin_centroids_low)
```

Benchmark value to remove the long edges

```
## Compute 2D distances
distance <- cal_2d_dist(.data = tr_from_to_df)

## To plot the distribution of distance
plot_dist <- function(distance_df){
  distance_df$group <- "1"
  dist_plot <- ggplot(distance_df, aes(x = group, y = distance)) +
    geom_quasirandom()+
    ylim(0, max(unlist(distance_df$distance))+ 0.5) + coord_flip()
  return(dist_plot)
}

plot_dist(distance)
```




```
benchmark <- find_benchmark_value(.data = distance, distance_col = "distance")
benchmark <- 3
```

Model function

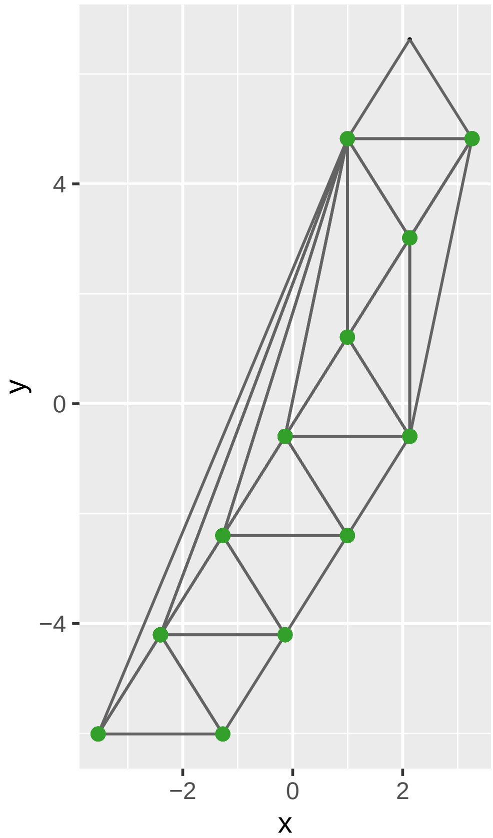
Predict 2D embeddings

Compute residuals

Visualizations

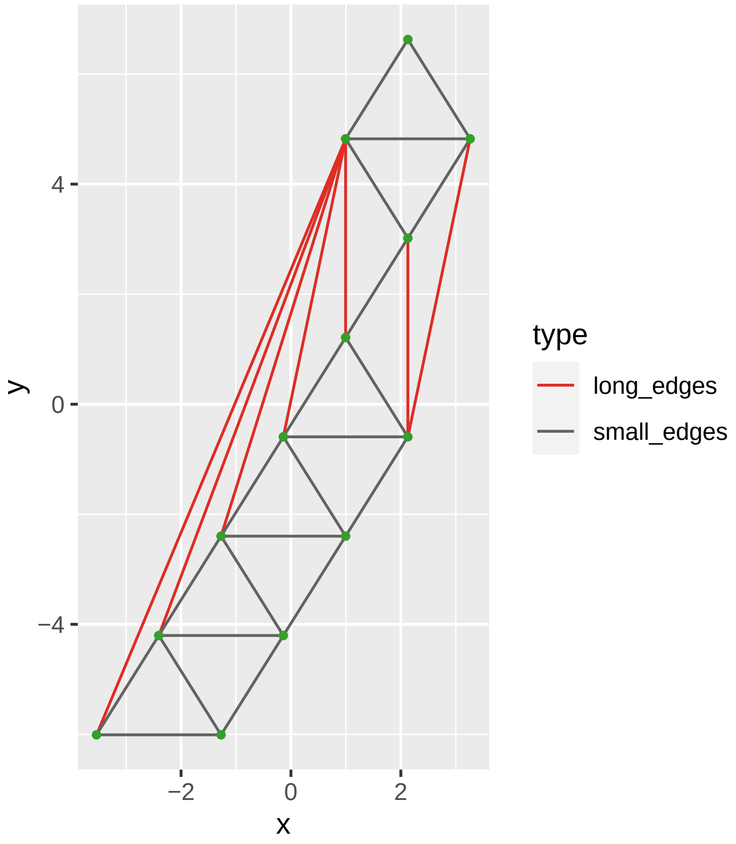
geom_trimesh

```
trimesh <- ggplot(df_bin_centroids, aes(x = x, y = y)) +
  geom_point(size = 0.1) +
  geom_trimesh() +
  coord_equal()

trimesh
```



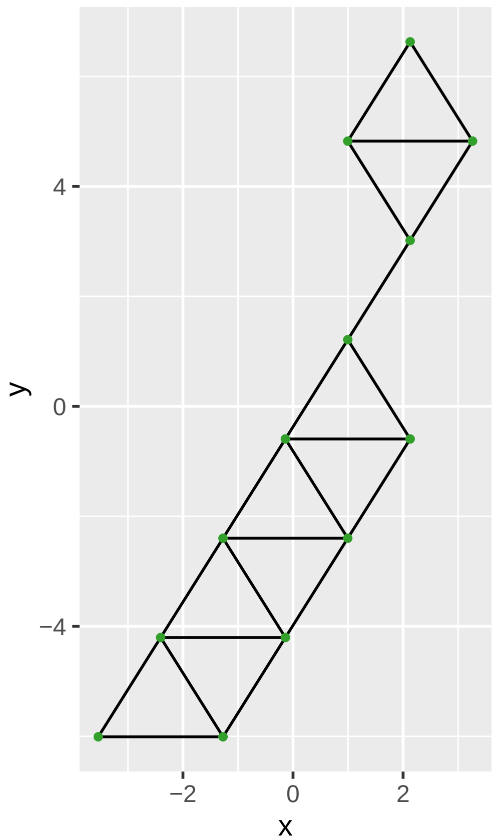
coloured_long_edges

```
trimesh_gr <- colour_long_edges(.data = distance, benchmark_value = benchmark,
                               triangular_object = tr1_object, distance_col = "distance")

trimesh_gr
```



remove long edges

```
trimesh_removed <- remove_long_edges(.data = distance, benchmark_value = benchmark,
                                     triangular_object = tr1_object, distance_col = "distance")
trimesh_removed
```



show_langevitour

```
## To generate a data set with high-D and 2D training data
df_all <- training_data |> dplyr::select(-ID) |>
  dplyr::bind_cols(s_curve_noise_umap_with_id)

## To generate averaged high-D data

df_bin <- avg_highD_data(.data = df_all, column_start_text = "x") ## Need to pass ID column name

tour1 <- show_langevitour(df_all, df_bin, df_bin_centroids, benchmark_value = benchmark,
                          distance = distance, distance_col = "distance")
tour1
```

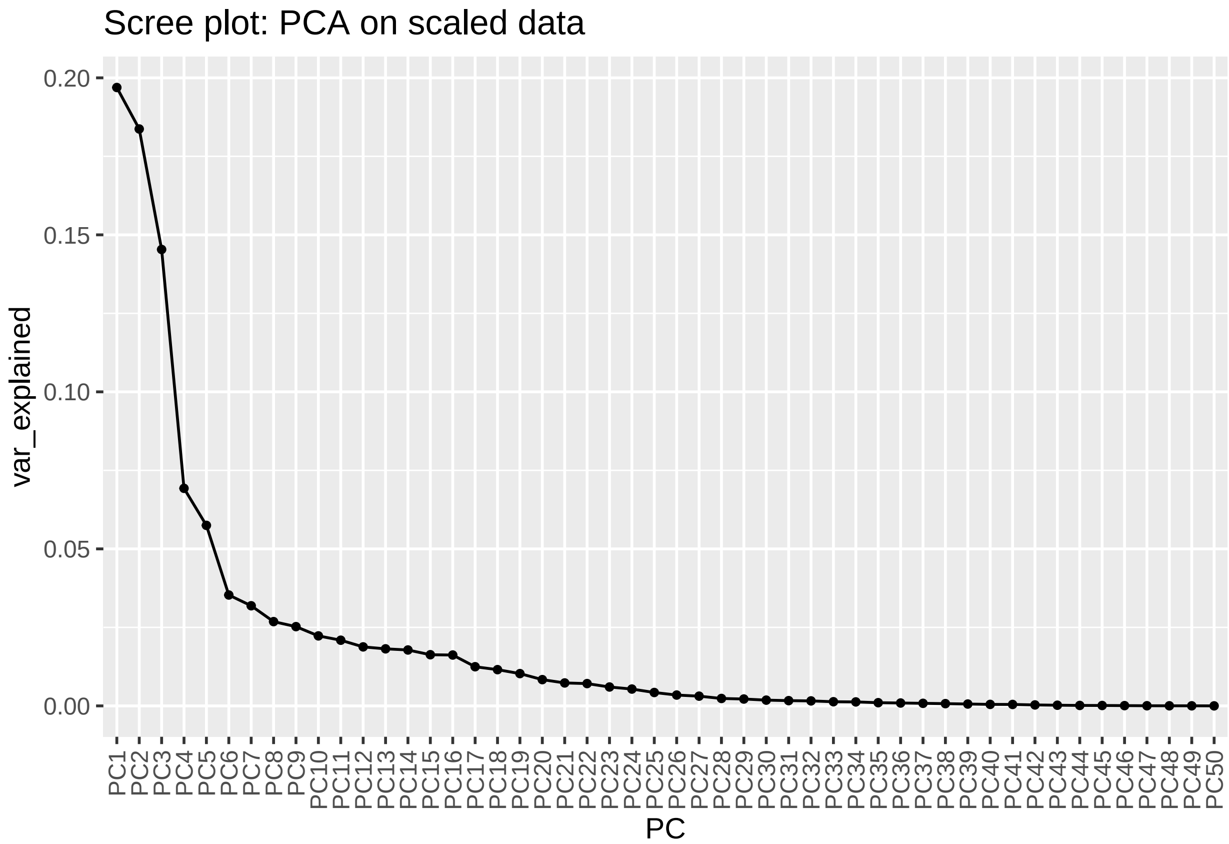
Tests

All functions have tests written and implemented using the **testthat** (Wickham 2011) in R.

3 Application

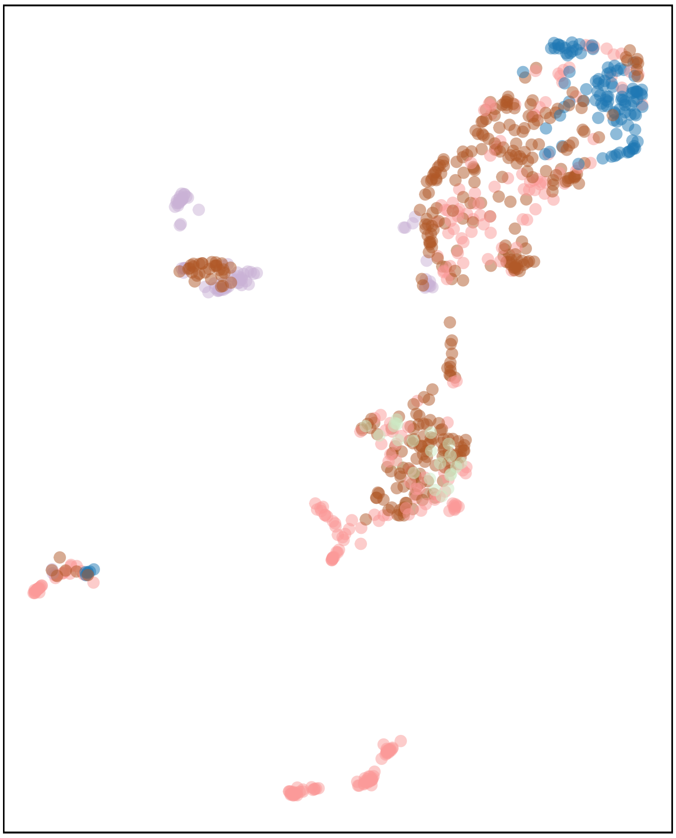
```
medlea_df <- read_csv("data/medlea_dataset.csv")
names(medlea_df)[2:(NCOL(medlea_df) - 1)] <- paste0("x", 1:(NCOL(medlea_df) - 2))

medlea_df <- medlea_df |> ## Since only contains zeros
  select(-x10)

#medlea_df[,2:(NCOL(medlea_df) - 1)] <- scale(medlea_df[,2:(NCOL(medlea_df) - 1)])

calculate_pca <- function(feature_dataset, num_pcs){
  pcaY_cal <- prcomp(feature_dataset, center = TRUE, scale = TRUE)
  PCAresults <- data.frame(pcaY_cal$x[, 1:num_pcs])
  summary_pca <- summary(pcaY_cal)
  var_explained_df <- data.frame(PC= paste0("PC",1:50),
                   var_explained=(pcaY_cal$sdev[1:50])^2/sum((pcaY_cal$sdev[1:50])^2))
 return(list(prcomp_out = pcaY_cal,pca_components = PCAresults, summary = summary_pca, var_explained_pca  = var_
}
features <- medlea_df[,2:(NCOL(medlea_df) - 1)]
pca_ref_calc <- calculate_pca(features, 8)
pca_ref_calc$summary

#> Importance of components:
#>                           PC1    PC2    PC3     PC4     PC5     PC6     PC7
#> Standard deviation     3.1691 3.0609 2.7226 1.87967 1.71219 1.34192 1.27525
#> Proportion of Variance 0.1969 0.1837 0.1453 0.06928 0.05748 0.03531 0.03189
#> Cumulative Proportion  0.1969 0.3806 0.5260 0.59526 0.65274 0.68805 0.71993
#>                            PC8     PC9    PC10    PC11    PC12    PC13   PC14
#> Standard deviation     1.16992 1.13465 1.06628 1.03279 0.97899 0.96264 0.9528
#> Proportion of Variance 0.02684 0.02524 0.02229 0.02091 0.01879 0.01817 0.0178
#> Cumulative Proportion  0.74677 0.77202 0.79431 0.81522 0.83402 0.85219 0.8700
#>                          PC15   PC16    PC17    PC18    PC19    PC20    PC21
#> Standard deviation     0.9116 0.9090 0.79750 0.76725 0.72414 0.65310 0.61052
#> Proportion of Variance 0.0163 0.0162 0.01247 0.01154 0.01028 0.00836 0.00731
#> Cumulative Proportion  0.8863 0.9025 0.91496 0.92650 0.93678 0.94514 0.95245
#>                          PC22    PC23    PC24    PC25    PC26   PC27    PC28
#> Standard deviation     0.6019 0.55399 0.52293 0.46638 0.41959 0.3976 0.34697
#> Proportion of Variance 0.0071 0.00602 0.00536 0.00426 0.00345 0.0031 0.00236
#> Cumulative Proportion  0.9596 0.96557 0.97093 0.97520 0.97865 0.9818 0.98411
#>                           PC29    PC30    PC31    PC32    PC33    PC34    PC35
#> Standard deviation     0.33415 0.30618 0.29237 0.28458 0.26033 0.25420 0.22792
#> Proportion of Variance 0.00219 0.00184 0.00168 0.00159 0.00133 0.00127 0.00102
#> Cumulative Proportion  0.98630 0.98814 0.98982 0.99140 0.99273 0.99400 0.99502
#>                           PC36    PC37    PC38   PC39    PC40    PC41    PC42
#> Standard deviation     0.21644 0.20437 0.19127 0.1744 0.15586 0.15252 0.12519
#> Proportion of Variance 0.00092 0.00082 0.00072 0.0006 0.00048 0.00046 0.00031
#> Cumulative Proportion  0.99594 0.99676 0.99747 0.9981 0.99855 0.99900 0.99931
#>                           PC43    PC44    PC45    PC46    PC47    PC48    PC49
#> Standard deviation     0.10485 0.08598 0.08008 0.06491 0.04841 0.04094 0.03791
#> Proportion of Variance 0.00022 0.00014 0.00013 0.00008 0.00005 0.00003 0.00003
#> Cumulative Proportion  0.99952 0.99967 0.99980 0.99988 0.99992 0.99996 0.99999
#>                           PC50    PC51
#> Standard deviation     0.02347 0.01421
#> Proportion of Variance 0.00001 0.00000
#> Cumulative Proportion  1.00000 1.00000

var_explained_df <- pca_ref_calc$var_explained_pca
data_pca <- pca_ref_calc$pca_components |>
  mutate(ID = 1:NROW(pca_ref_calc$pca_components),
         shape_label = medlea_df$Shape_label)

var_explained_df |>
  ggplot(aes(x = PC,y = var_explained, group = 1))+
  geom_point(size=1)+
```

```
  geom_line()+
  labs(title="Scree plot: PCA on scaled data") +
  scale_x_discrete(limits = paste0(rep("PC", 50), 1:50)) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```

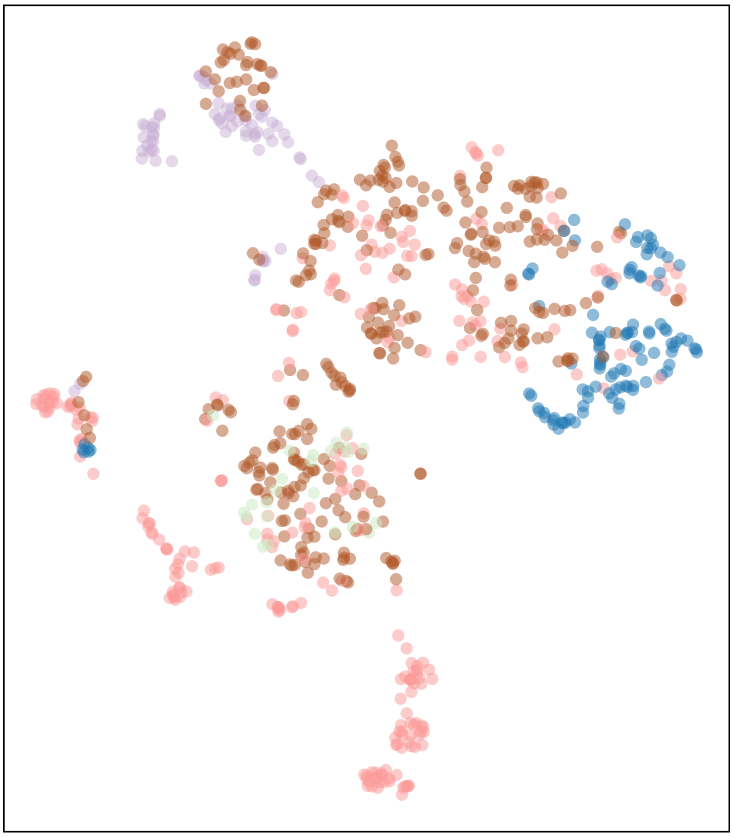



```
data_split <- initial_split(data_pca)
training_data <- training(data_split) |>
  arrange(ID)
test_data <- testing(data_split) |>
  arrange(ID)

UMAP_fit <- umap(training_data |> dplyr::select(-c(ID, shape_label)), n_neighbors = 37, n_components =  2)

UMAP_data <- UMAP_fit$layout |>
  as.data.frame()
names(UMAP_data)[1:(ncol(UMAP_data))] <- paste0(rep("UMAP",(ncol(UMAP_data))), 1:(ncol(UMAP_data)))

UMAP_data <- UMAP_data |>
  mutate(ID = training_data$id)

UMAP_data_with_label <- UMAP_data |>
  mutate(shape_label = training_data$shape_label)


UMAP_data_with_label |>
    ggplot(aes(x = UMAP1,
               y = UMAP2, color = shape_label))+
    geom_point(alpha=0.5) +
    coord_equal() +
   theme(plot.title = element_text(hjust = 0.5, size = 18, face = "bold")) + #ggtitle("(a)") +
  theme_linedraw() +
   theme(legend.position = "none", plot.title = element_text(size = 7, hjust = 0.5, vjust = -0.5),
              axis.title.x = element_blank(), axis.title.y = element_blank(),
              axis.text.x = element_blank(), axis.ticks.x = element_blank(),
              axis.text.y = element_blank(), axis.ticks.y = element_blank(),
```

```
         panel.grid.major = element_blank(), panel.grid.minor = element_blank(), #change legend key width
        legend.title = element_text(size=5), #change legend title font size
        legend.text = element_text(size=4),
         legend.key.height = unit(0.25, 'cm'),
         legend.key.width = unit(0.25, 'cm')) +
 scale_color_manual(values=c("#b15928", "#1f78b4", "#cab2d6", "#ccebc5", "#fb9a99", "#e31a1c", "#6a3d9a", "#ff7
```

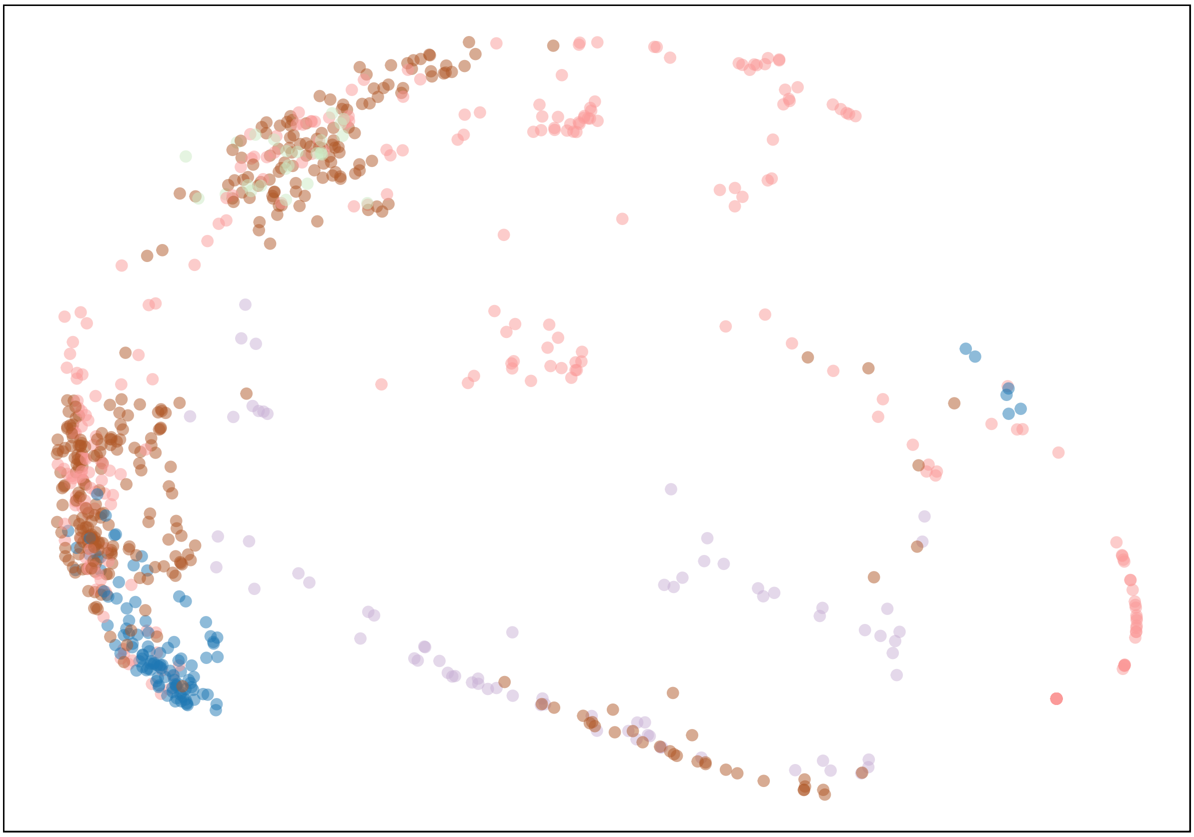


```
tSNE_data <- Fit_tSNE(training_data |> dplyr::select(-c(ID, shape_label)), opt_perplexity = calculate_effective

tSNE_data <- tSNE_data |>
  select(-ID) |>
  mutate(ID = training_data$ID)

tSNE_data_with_label <- tSNE_data |>
  mutate(shape_label = training_data$shape_label)

tSNE_data_with_label |>
    ggplot(aes(x = tSNE1,
               y = tSNE2, color = shape_label))+
    geom_point(alpha=0.5) +
    coord_equal() +
   theme(plot.title = element_text(hjust = 0.5, size = 18, face = "bold")) + #ggtitle("(a)") +
  theme_linedraw() +
   theme(legend.position = "none", plot.title = element_text(size = 7, hjust = 0.5, vjust = -0.5),
              axis.title.x = element_blank(), axis.title.y = element_blank(),
              axis.text.x = element_blank(), axis.ticks.x = element_blank(),
              axis.text.y = element_blank(), axis.ticks.y = element_blank(),
         panel.grid.major = element_blank(), panel.grid.minor = element_blank(), #change legend key width
        legend.title = element_text(size=5), #change legend title font size
        legend.text = element_text(size=4),
         legend.key.height = unit(0.25, 'cm'),
         legend.key.width = unit(0.25, 'cm')) +
 scale_color_manual(values=c("#b15928", "#1f78b4", "#cab2d6", "#ccebc5", "#fb9a99", "#e31a1c", "#6a3d9a", "#ff7
```



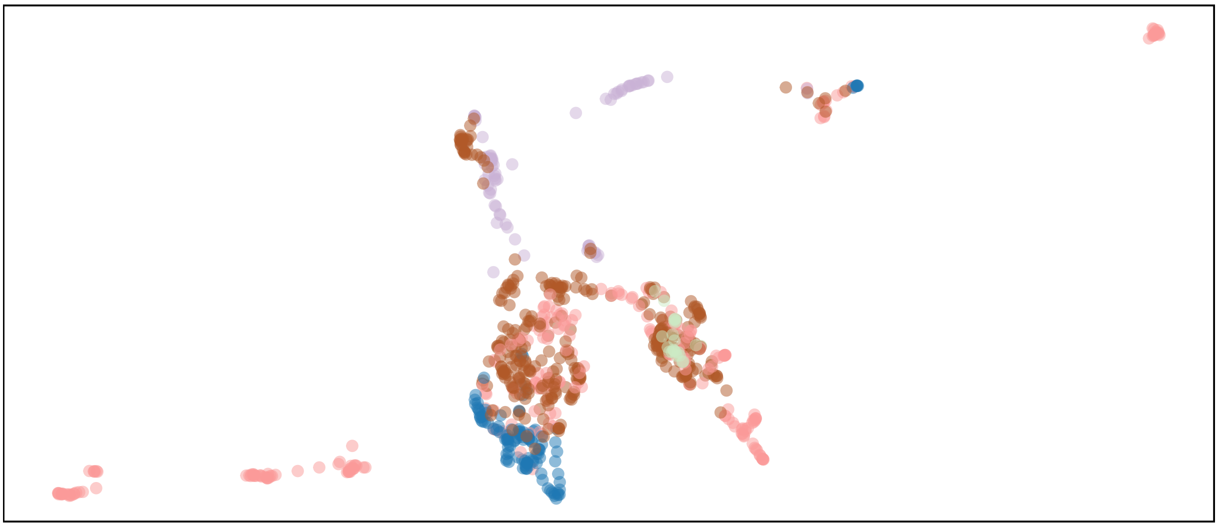
```
PHATE_data <- Fit_PHATE(training_data |> dplyr::select(-c(ID, shape_label)), knn = 5, with_seed = 20240110)

#> Calculating PHATE...
#>   Running PHATE on 824 observations and 8 variables.
#>   Calculating graph and diffusion operator...
#>     Calculating KNN search...
#>     Calculating affinities...
#>   Calculated graph and diffusion operator in 0.01 seconds.
#>   Calculating optimal t...
#>     Automatically selected t = 22
#>   Calculated optimal t in 0.49 seconds.
#>   Calculating diffusion potential...
#>   Calculated diffusion potential in 0.38 seconds.
#>   Calculating metric MDS...
#>   Calculated metric MDS in 7.83 seconds.
#> Calculated PHATE in 8.72 seconds.

PHATE_data <- PHATE_data |>
  select(PHATE1, PHATE2)
PHATE_data <- PHATE_data |>
  mutate(ID = training_data$ID)

PHATE_data_with_label <- PHATE_data |>
  mutate(shape_label = training_data$shape_label)

PHATE_data_with_label |>
    ggplot(aes(x = PHATE1,
               y = PHATE2, color = shape_label))+
    geom_point(alpha=0.5) +
    coord_equal() +
   theme(plot.title = element_text(hjust = 0.5, size = 18, face = "bold")) + #ggtitle("(a)") +
  theme_linedraw() +
   theme(legend.position = "none", plot.title = element_text(size = 7, hjust = 0.5, vjust = -0.5),
              axis.title.x = element_blank(), axis.title.y = element_blank(),
              axis.text.x = element_blank(), axis.ticks.x = element_blank(),
```

```
             axis.text.y = element_blank(), axis.ticks.y = element_blank(),
         panel.grid.major = element_blank(), panel.grid.minor = element_blank(), #change legend key width
        legend.title = element_text(size=5), #change legend title font size
        legend.text = element_text(size=4),
         legend.key.height = unit(0.25, 'cm'),
         legend.key.width = unit(0.25, 'cm')) +
  scale_color_manual(values=c("#b15928", "#1f78b4", "#cab2d6", "#ccebc5", "#fb9a99", "#e31a1c", "#6a3d9a", "#ff7
```

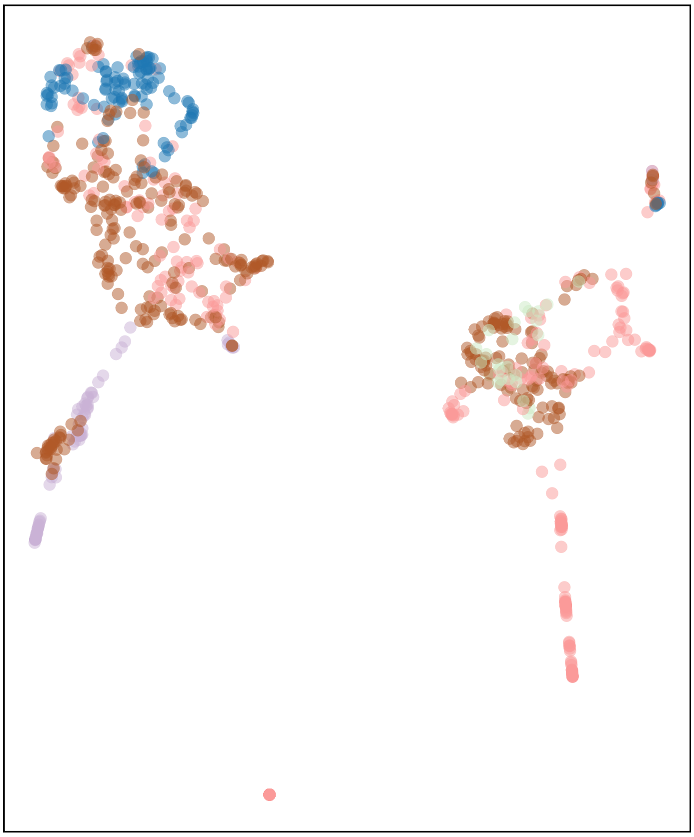


```
tem_dir <- tempdir()

Fit_TriMAP_data(training_data |> dplyr::select(-c(ID, shape_label)), tem_dir)

path <- file.path(tem_dir, "df_2_without_class.csv")
path2 <- file.path(tem_dir, "dataset_3_TriMAP_values.csv")

Fit_TriMAP(as.integer(2), as.integer(5), as.integer(4), as.integer(3), path, path2)

TriMAP_data <- read_csv(path2)
TriMAP_data <- TriMAP_data |>
  mutate(ID = training_data$ID)

TriMAP_data_with_label <- TriMAP_data |>
  mutate(shape_label = training_data$shape_label)

TriMAP_data_with_label |>
    ggplot(aes(x = TriMAP1,
               y = TriMAP2, color = shape_label))+
    geom_point(alpha=0.5) +
    coord_equal() +
   theme(plot.title = element_text(hjust = 0.5, size = 18, face = "bold")) + #ggtitle("(a)") +
  theme_linedraw() +
   theme(legend.position = "none", plot.title = element_text(size = 7, hjust = 0.5, vjust = -0.5),
              axis.title.x = element_blank(), axis.title.y = element_blank(),
              axis.text.x = element_blank(), axis.ticks.x = element_blank(),
              axis.text.y = element_blank(), axis.ticks.y = element_blank(),
```

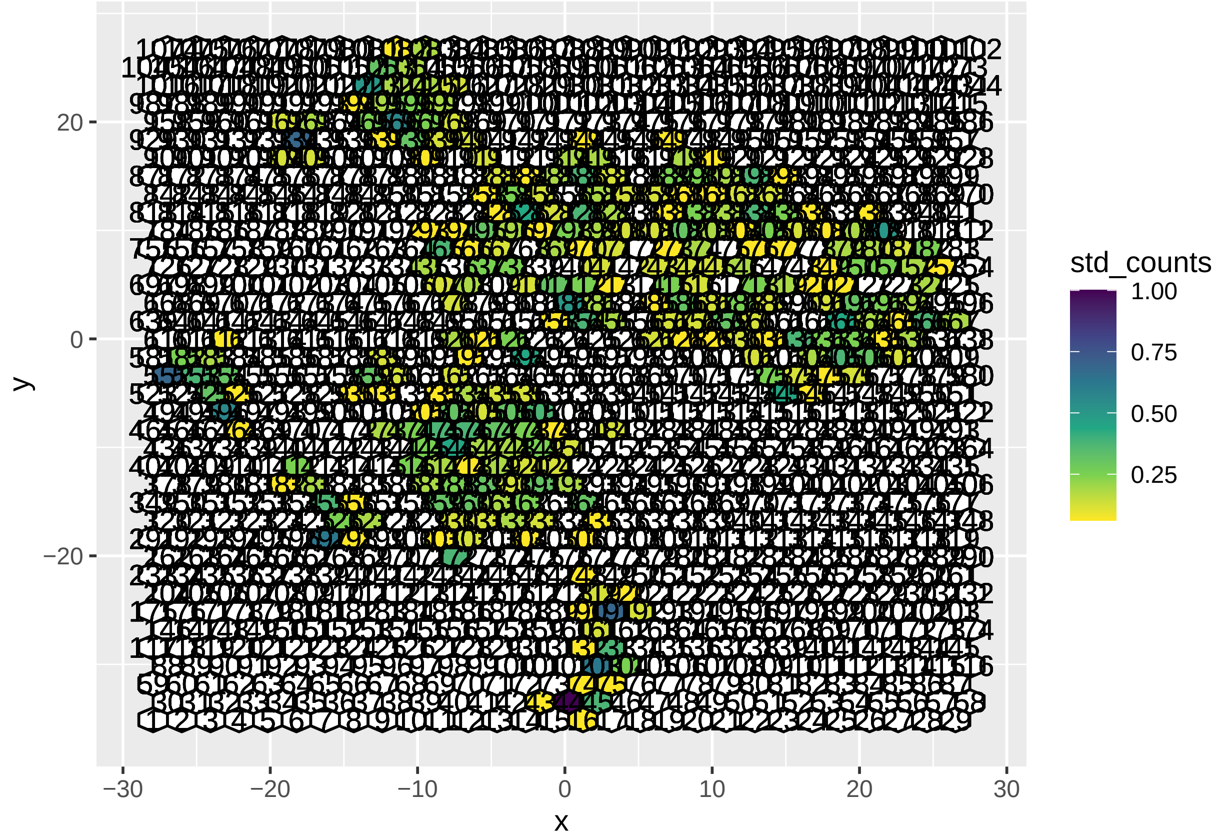
```
        panel.grid.major = element_blank(), panel.grid.minor = element_blank(), #change legend key width
       legend.title = element_text(size=5), #change legend title font size
       legend.text = element_text(size=4),
        legend.key.height = unit(0.25, 'cm'),
        legend.key.width = unit(0.25, 'cm')) +
 scale_color_manual(values=c("#b15928", "#1f78b4", "#cab2d6", "#ccebc5", "#fb9a99", "#e31a1c", "#6a3d9a", "#ff7
```



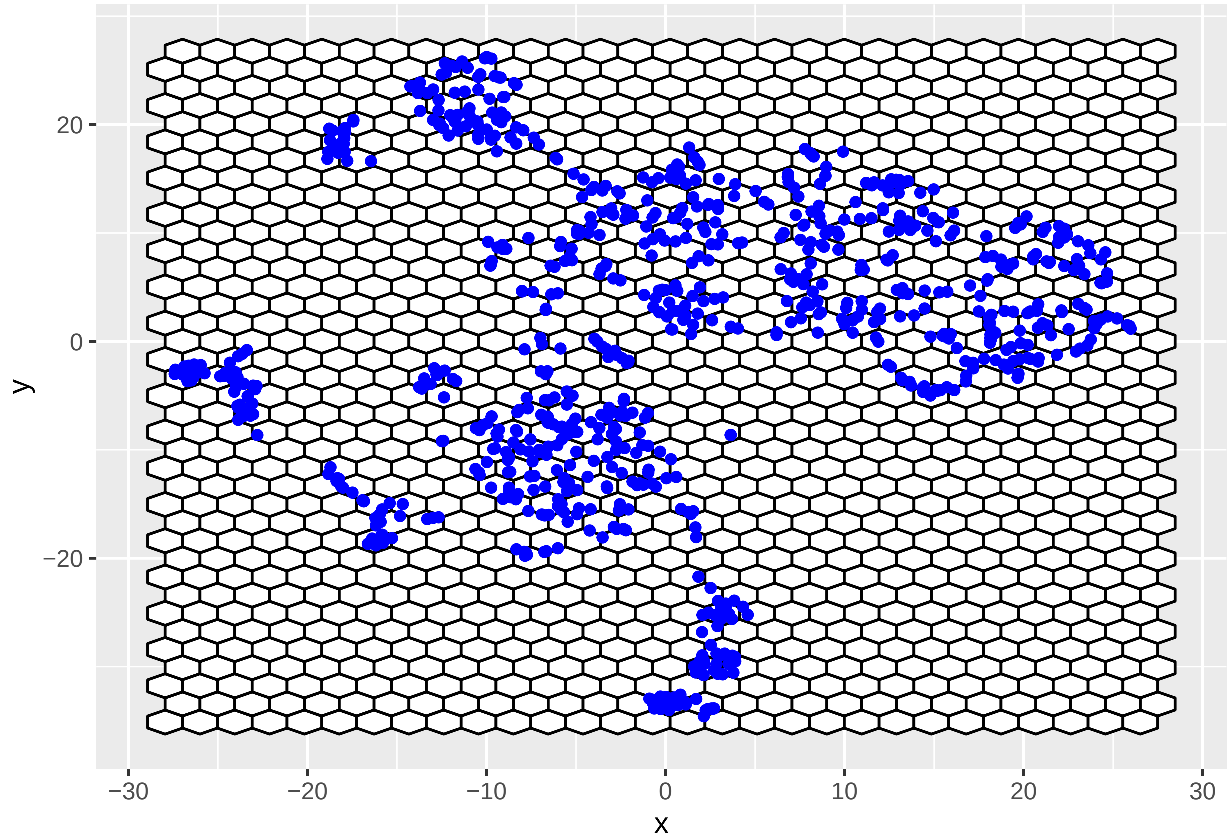
```
tem_dir <- tempdir()

Fit_PacMAP_data(training_data |> dplyr::select(-c(ID, shape_label)), tem_dir)

path <- file.path(tem_dir, "df_2_without_class.csv")
path2 <- file.path(tem_dir, "dataset_3_PaCMAP_values.csv")

Fit_PaCMAP(as.integer(2), as.integer(10), "random", 0.9, as.integer(2), path, path2)

PaCMAP_data <- read_csv(path2)
PaCMAP_data <- PaCMAP_data |>
  mutate(ID = training_data$ID)

PaCMAP_data_with_label <- PaCMAP_data |>
  mutate(shape_label = training_data$shape_label)

PaCMAP_data_with_label |>
    ggplot(aes(x = PaCMAP1,
               y = PaCMAP2, color = shape_label))+
    geom_point(alpha=0.5) +
    coord_equal() +
   theme(plot.title = element_text(hjust = 0.5, size = 18, face = "bold")) + #ggtitle("(a)") +
  theme_linedraw() +
   theme(legend.position = "none", plot.title = element_text(size = 7, hjust = 0.5, vjust = -0.5),
             axis.title.x = element_blank(), axis.title.y = element_blank(),
             axis.text.x = element_blank(), axis.ticks.x = element_blank(),
             axis.text.y = element_blank(), axis.ticks.y = element_blank(),
        panel.grid.major = element_blank(), panel.grid.minor = element_blank(), #change legend key width
       legend.title = element_text(size=5), #change legend title font size
       legend.text = element_text(size=4),
        legend.key.height = unit(0.25, 'cm'),
        legend.key.width = unit(0.25, 'cm')) +
 scale_color_manual(values=c("#b15928", "#1f78b4", "#cab2d6", "#ccebc5", "#fb9a99", "#e31a1c", "#6a3d9a", "#ff7
```



```
num_bins_x <- calculate_effective_x_bins(.data = tSNE_data, x = "tSNE1", hex_size = NA)

num_bins_y <- calculate_effective_y_bins(.data = tSNE_data, y = "tSNE2", hex_size = NA)
num_bins_y

#> [1] 38

all_centroids_df <- generate_full_grid_centroids(nldr_df = tSNE_data,
                                                 x = "tSNE1", y = "tSNE2",
                                                 num_bins_x = num_bins_x,
                                                 num_bins_y = num_bins_y,
                                                 buffer_size = NA, hex_size = NA)


hex_grid <- gen_hex_coordinates(all_centroids_df)

full_grid_with_hexbin_id <- map_hexbin_id(all_centroids_df)

full_grid_with_polygon_id <- map_polygon_id(full_grid_with_hexbin_id, hex_grid)

tSNE_data_with_id <- assign_data(tSNE_data, full_grid_with_hexbin_id)

df_with_std_counts <- compute_std_counts(nldr_df = tSNE_data_with_id)

hex_full_count_df <- generate_full_grid_info(full_grid_with_polygon_id, df_with_std_counts, hex_grid)

ggplot(data = hex_full_count_df, aes(x = x, y = y)) +
  geom_polygon(color = "black", aes(group = polygon_id, fill = std_counts)) +
  geom_text(aes(x = c_x, y = c_y, label = hexID)) +
  scale_fill_viridis_c(direction = -1, na.value = "#ffffff")
```




```
ggplot(data = hex_grid, aes(x = x, y = y)) + geom_polygon(fill = "white", color = "black", aes(group = id)) +
  geom_point(data = tSNE_data, aes(x = tSNE1, y = tSNE2), color = "blue")
```




```
df_bin_centroids <- hex_full_count_df[complete.cases(hex_full_count_df[["std_counts"]]), ] |>
  dplyr::select("c_x", "c_y", "hexID", "std_counts") |>
```

```
  dplyr::distinct() |>
  dplyr::rename(c("x" = "c_x", "y" = "c_y"))

df_bin_centroids
```

```
#>               x            y hexID std_counts
#> 1   -26.9783746  -3.336399929   552     0.6875
#> 2   -25.0331153  -3.336399929   553     0.3750
#> 3   -26.0057449  -1.663988875   582     0.2500
#> 4   -23.0878560  -6.681222036   496     0.5625
#> 5   -24.0604856  -5.008810982   525     0.3125
#> 6   -23.0878560  -3.336399929   554     0.3125
#> 7   -24.0604856  -1.663988875   583     0.1875
#> 8   -23.0878560   0.008422179   612     0.0625
#> 9   -22.1152263  -8.353633090   468     0.0625
#> 10  -22.1152263  -5.008810982   526     0.0625
#> 11  -19.1973374 -13.370866252   382     0.0625
#> 12  -19.1973374  16.732532718   904     0.1250
#> 13  -19.1973374  20.077354826   962     0.1250
#> 14  -17.2520781 -13.370866252   383     0.1875
#> 15  -18.2247077 -11.698455198   412     0.2500
#> 16  -17.2520781  16.732532718   905     0.1250
#> 17  -18.2247077  18.404943772   934     0.6875
#> 18  -17.2520781  20.077354826   963     0.1875
#> 19  -16.2794484 -18.388099413   297     0.6250
#> 20  -15.3068188 -16.715688359   326     0.2500
#> 21  -16.2794484 -15.043277306   355     0.3750
#> 22  -14.3341891 -18.388099413   298     0.0625
#> 23  -13.3615595 -16.715688359   327     0.1875
#> 24  -14.3341891 -15.043277306   356     0.0625
#> 25  -14.3341891  -5.008810982   530     0.0625
#> 26  -13.3615595  -3.336399929   559     0.3125
#> 27  -13.3615595  20.077354826   965     0.2500
#> 28  -14.3341891  21.749765879   994     0.0625
#> 29  -13.3615595  23.422176933  1023     0.5000
#> 30  -12.3889298  -8.353633090   473     0.1875
#> 31  -12.3889298  -5.008810982   531     0.0625
#> 32  -11.4163002  -3.336399929   560     0.1250
#> 33  -12.3889298  -1.663988875   589     0.1250
#> 34  -12.3889298  18.404943772   937     0.0625
#> 35  -11.4163002  20.077354826   966     0.5625
#> 36  -12.3889298  21.749765879   995     0.1875
#> 37  -11.4163002  23.422176933  1024     0.1875
#> 38  -12.3889298  25.094587987  1053     0.3125
#> 39  -11.4163002  26.766999041  1082     0.0625
#> 40   -9.4710409 -13.370866252   387     0.1875
#> 41  -10.4436705 -11.698455198   416     0.2500
#> 42   -9.4710409 -10.026044144   445     0.2500
#> 43  -10.4436705  -8.353633090   474     0.2500
#> 44   -9.4710409  -6.681222036   503     0.0625
#> 45   -9.4710409   6.698066395   735     0.1875
#> 46   -9.4710409  10.042888502   793     0.0625
#> 47   -9.4710409  16.732532718   909     0.0625
#> 48  -10.4436705  18.404943772   938     0.3125
#> 49   -9.4710409  20.077354826   967     0.2500
#> 50  -10.4436705  21.749765879   996     0.2500
#> 51   -9.4710409  23.422176933  1025     0.1875
#> 52  -10.4436705  25.094587987  1054     0.1875
#> 53   -9.4710409  26.766999041  1083     0.1875
#> 54   -7.5257816 -20.060510467   272     0.3750
#> 55   -8.4984112 -18.388099413   301     0.0625
#> 56   -7.5257816 -16.715688359   330     0.1250
#> 57   -8.4984112 -15.043277306   359     0.3125
#> 58   -7.5257816 -13.370866252   388     0.2500
```

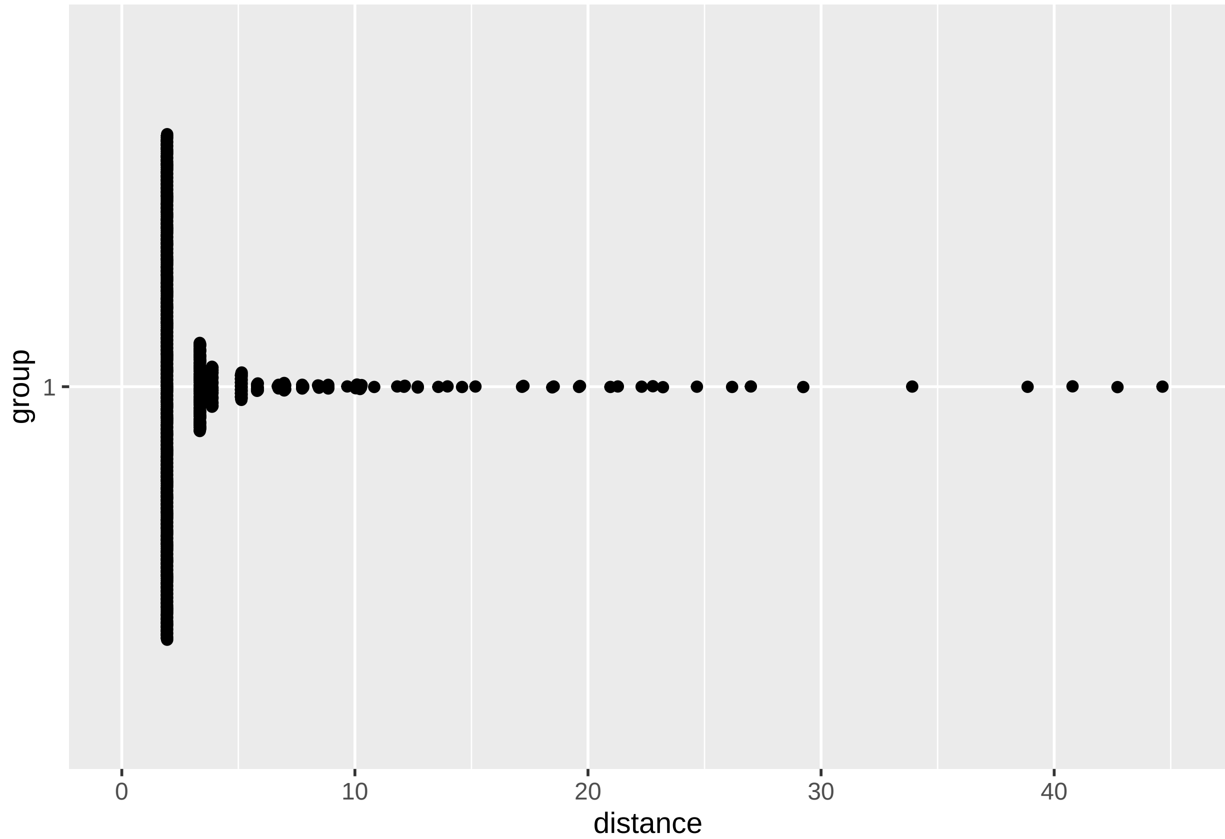
```
#> 59   -8.4984112 -11.698455198   417     0.1875
#> 60   -7.5257816 -10.026044144   446     0.4375
#> 61   -8.4984112  -8.353633090   475     0.3750
#> 62   -7.5257816  -6.681222036   504     0.3125
#> 63   -8.4984112  -5.008810982   533     0.0625
#> 64   -7.5257816  -3.336399929   562     0.1250
#> 65   -7.5257816   0.008422179   620     0.1875
#> 66   -7.5257816   3.353244287   678     0.1250
#> 67   -8.4984112   5.025655341   707     0.1250
#> 68   -8.4984112   8.370477449   765     0.3750
#> 69   -7.5257816  10.042888502   794     0.0625
#> 70   -8.4984112  18.404943772   939     0.1250
#> 71   -7.5257816  20.077354826   968     0.1250
#> 72   -8.4984112  21.749765879   997     0.1875
#> 73   -7.5257816  23.422176933  1026     0.1250
#> 74   -6.5531519 -18.388099413   302     0.1250
#> 75   -5.5805223 -16.715688359   331     0.1250
#> 76   -6.5531519 -15.043277306   360     0.3125
#> 77   -5.5805223 -13.370866252   389     0.3125
#> 78   -6.5531519 -11.698455198   418     0.0625
#> 79   -5.5805223 -10.026044144   447     0.1875
#> 80   -6.5531519  -8.353633090   476     0.3750
#> 81   -5.5805223  -6.681222036   505     0.1250
#> 82   -6.5531519  -5.008810982   534     0.1875
#> 83   -6.5531519  -1.663988875   592     0.0625
#> 84   -5.5805223   0.008422179   621     0.0625
#> 85   -6.5531519   5.025655341   708     0.1875
#> 86   -5.5805223   6.698066395   737     0.2500
#> 87   -6.5531519   8.370477449   766     0.0625
#> 88   -5.5805223  10.042888502   795     0.3125
#> 89   -5.5805223  13.387710610   853     0.0625
#> 90   -5.5805223  16.732532718   911     0.1250
#> 91   -6.5531519  18.404943772   940     0.1250
#> 92   -3.6352630 -16.715688359   332     0.1875
#> 93   -4.6078926 -15.043277306   361     0.1875
#> 94   -3.6352630 -13.370866252   390     0.1250
#> 95   -4.6078926 -11.698455198   419     0.1875
#> 96   -3.6352630 -10.026044144   448     0.1875
#> 97   -4.6078926  -8.353633090   477     0.3125
#> 98   -3.6352630  -6.681222036   506     0.3125
#> 99   -4.6078926  -5.008810982   535     0.1250
#> 100  -3.6352630   0.008422179   622     0.2500
#> 101  -3.6352630   6.698066395   738     0.2500
#> 102  -4.6078926   8.370477449   767     0.1250
#> 103  -3.6352630  10.042888502   796     0.1875
#> 104  -4.6078926  11.715299556   825     0.0625
#> 105  -3.6352630  13.387710610   854     0.2500
#> 106  -4.6078926  15.060121664   883     0.1250
#> 107  -1.6900037 -33.439798898    43     0.0625
#> 108  -2.6626333 -18.388099413   304     0.0625
#> 109  -1.6900037 -16.715688359   333     0.1250
#> 110  -2.6626333 -15.043277306   362     0.2500
#> 111  -1.6900037 -13.370866252   391     0.3125
#> 112  -2.6626333 -11.698455198   420     0.1250
#> 113  -1.6900037 -10.026044144   449     0.2500
#> 114  -2.6626333  -8.353633090   478     0.2500
#> 115  -1.6900037  -6.681222036   507     0.3750
#> 116  -2.6626333  -5.008810982   536     0.1250
#> 117  -2.6626333  -1.663988875   594     0.4375
#> 118  -2.6626333   5.025655341   710     0.1250
#> 119  -1.6900037  10.042888502   797     0.0625
#> 120  -2.6626333  11.715299556   826     0.4375
#> 121  -1.6900037  13.387710610   855     0.1250
#> 122  -2.6626333  15.060121664   884     0.0625
```

```
#> 123   0.2552556 -33.439798898    44     1.0000
#> 124   0.2552556 -13.370866252   392     0.1875
#> 125  -0.7173740 -11.698455198   421     0.1250
#> 126   0.2552556 -10.026044144   450     0.1250
#> 127  -0.7173740  -8.353633090   479     0.0625
#> 128  -0.7173740   1.680833233   653     0.0625
#> 129   0.2552556   3.353244287   682     0.5000
#> 130  -0.7173740   5.025655341   711     0.3125
#> 131  -0.7173740   8.370477449   769     0.1875
#> 132   0.2552556  10.042888502   798     0.2500
#> 133  -0.7173740  11.715299556   827     0.1250
#> 134  -0.7173740  15.060121664   885     0.1875
#> 135   0.2552556  16.732532718   914     0.1875
#> 136   1.2278853 -35.112209952    16     0.0625
#> 137   2.2005149 -33.439798898    45     0.3750
#> 138   1.2278853 -31.767387844    74     0.0625
#> 139   2.2005149 -30.094976790   103     0.6250
#> 140   1.2278853 -28.422565737   132     0.0625
#> 141   2.2005149 -26.750154683   161     0.1250
#> 142   1.2278853 -25.077743629   190     0.0625
#> 143   2.2005149 -23.405332575   219     0.1250
#> 144   1.2278853 -21.732921521   248     0.0625
#> 145   1.2278853 -18.388099413   306     0.0625
#> 146   2.2005149 -16.715688359   335     0.0625
#> 147   1.2278853 -15.043277306   364     0.3125
#> 148   1.2278853   1.680833233   654     0.3750
#> 149   2.2005149   3.353244287   683     0.1875
#> 150   1.2278853   5.025655341   712     0.2500
#> 151   2.2005149   6.698066395   741     0.1250
#> 152   1.2278853   8.370477449   770     0.0625
#> 153   2.2005149  10.042888502   799     0.1875
#> 154   1.2278853  11.715299556   828     0.3750
#> 155   2.2005149  13.387710610   857     0.1875
#> 156   1.2278853  15.060121664   886     0.3750
#> 157   2.2005149  16.732532718   915     0.1875
#> 158   1.2278853  18.404943772   944     0.0625
#> 159   3.1731446 -31.767387844    75     0.0625
#> 160   4.1457742 -30.094976790   104     0.2500
#> 161   3.1731446 -28.422565737   133     0.3750
#> 162   3.1731446 -25.077743629   191     0.6875
#> 163   4.1457742 -23.405332575   220     0.0625
#> 164   3.1731446  -8.353633090   481     0.1250
#> 165   3.1731446   1.680833233   655     0.1875
#> 166   3.1731446   5.025655341   713     0.0625
#> 167   3.1731446   8.370477449   771     0.1250
#> 168   4.1457742  10.042888502   800     0.1250
#> 169   3.1731446  11.715299556   829     0.1875
#> 170   4.1457742  13.387710610   858     0.1250
#> 171   3.1731446  15.060121664   887     0.1250
#> 172   5.1184039 -25.077743629   192     0.1250
#> 173   6.0910335   0.008422179   627     0.1250
#> 174   6.0910335   3.353244287   685     0.0625
#> 175   6.0910335   6.698066395   743     0.1250
#> 176   6.0910335  10.042888502   801     0.1250
#> 177   6.0910335  13.387710610   859     0.1250
#> 178   8.0362928   0.008422179   628     0.0625
#> 179   7.0636632   1.680833233   657     0.1250
#> 180   8.0362928   3.353244287   686     0.3125
#> 181   7.0636632   5.025655341   715     0.2500
#> 182   8.0362928   6.698066395   744     0.1250
#> 183   7.0636632   8.370477449   773     0.0625
#> 184   8.0362928  10.042888502   802     0.3125
#> 185   7.0636632  11.715299556   831     0.0625
#> 186   8.0362928  13.387710610   860     0.0625
```

```
#> 187   7.0636632  15.060121664   889     0.2500
#> 188   8.0362928  16.732532718   918     0.1875
#> 189   7.0636632  18.404943772   947     0.0625
#> 190   9.9815521   0.008422179   629     0.0625
#> 191   9.0089225   1.680833233   658     0.1250
#> 192   9.9815521   3.353244287   687     0.1250
#> 193   9.0089225   5.025655341   716     0.1250
#> 194   9.9815521   6.698066395   745     0.1250
#> 195   9.0089225   8.370477449   774     0.1875
#> 196   9.9815521  10.042888502   803     0.1875
#> 197   9.0089225  11.715299556   832     0.2500
#> 198   9.9815521  13.387710610   861     0.0625
#> 199   9.0089225  15.060121664   890     0.2500
#> 200   9.9815521  16.732532718   919     0.0625
#> 201  11.9268114   0.008422179   630     0.1250
#> 202  10.9541818   1.680833233   659     0.3125
#> 203  11.9268114   3.353244287   688     0.2500
#> 204  11.9268114   6.698066395   746     0.1875
#> 205  11.9268114  10.042888502   804     0.0625
#> 206  10.9541818  11.715299556   833     0.1875
#> 207  11.9268114  13.387710610   862     0.1250
#> 208  10.9541818  15.060121664   891     0.1875
#> 209  13.8720707  -3.336399929   573     0.2500
#> 210  12.8994411  -1.663988875   602     0.1250
#> 211  13.8720707   0.008422179   631     0.0625
#> 212  12.8994411   1.680833233   660     0.1250
#> 213  13.8720707   3.353244287   689     0.1250
#> 214  12.8994411   5.025655341   718     0.2500
#> 215  12.8994411   8.370477449   776     0.0625
#> 216  13.8720707  10.042888502   805     0.2500
#> 217  12.8994411  11.715299556   834     0.3750
#> 218  13.8720707  13.387710610   863     0.1250
#> 219  12.8994411  15.060121664   892     0.3750
#> 220  14.8447004  -5.008810982   545     0.4375
#> 221  15.8173300  -3.336399929   574     0.1250
#> 222  15.8173300   0.008422179   632     0.3750
#> 223  14.8447004   5.025655341   719     0.1875
#> 224  14.8447004   8.370477449   777     0.0625
#> 225  15.8173300  10.042888502   806     0.1250
#> 226  14.8447004  11.715299556   835     0.2500
#> 227  14.8447004  15.060121664   893     0.0625
#> 228  16.7899597  -5.008810982   546     0.0625
#> 229  17.7625893  -3.336399929   575     0.0625
#> 230  16.7899597  -1.663988875   604     0.1875
#> 231  17.7625893   0.008422179   633     0.2500
#> 232  17.7625893   3.353244287   691     0.1250
#> 233  16.7899597   5.025655341   720     0.0625
#> 234  17.7625893   6.698066395   749     0.0625
#> 235  17.7625893  10.042888502   807     0.0625
#> 236  16.7899597  11.715299556   836     0.0625
#> 237  19.7078486  -3.336399929   576     0.1250
#> 238  18.7352190  -1.663988875   605     0.3750
#> 239  19.7078486   0.008422179   634     0.2500
#> 240  18.7352190   1.680833233   663     0.4375
#> 241  19.7078486   3.353244287   692     0.3125
#> 242  18.7352190   5.025655341   721     0.0625
#> 243  19.7078486   6.698066395   750     0.2500
#> 244  18.7352190   8.370477449   779     0.1875
#> 245  19.7078486  10.042888502   808     0.1875
#> 246  20.6804783  -1.663988875   606     0.3125
#> 247  21.6531079   0.008422179   635     0.0625
#> 248  20.6804783   1.680833233   664     0.1875
#> 249  21.6531079   3.353244287   693     0.2500
#> 250  21.6531079   6.698066395   751     0.2500
```

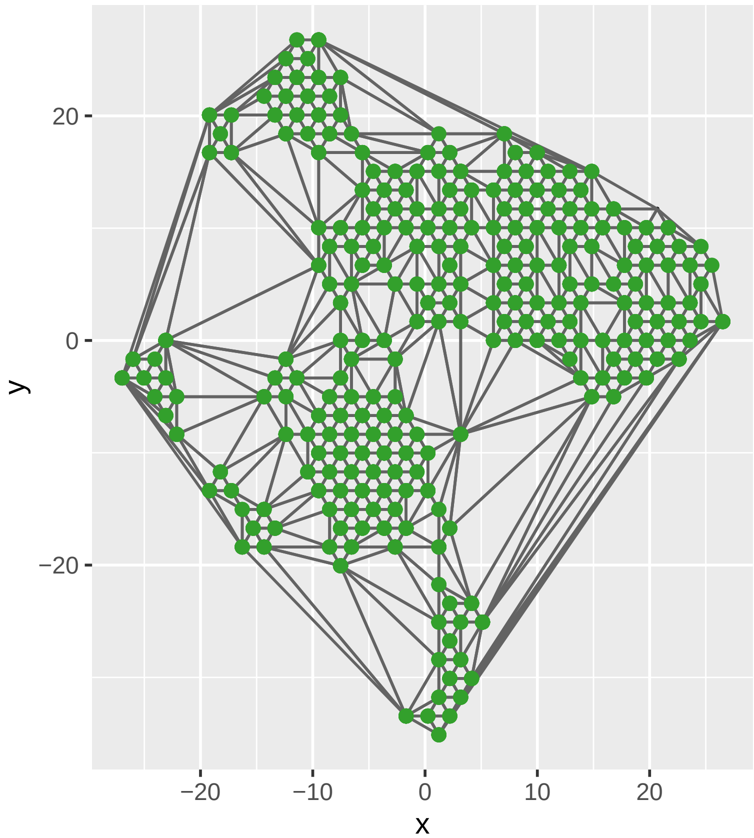
```
#> 251  20.6804783   8.370477449   780     0.1875
#> 252  21.6531079  10.042888502   809     0.5000
#> 253  20.6804783  11.715299556   838     0.0625
#> 254  22.6257376  -1.663988875   607     0.1250
#> 255  23.5983672   0.008422179   636     0.1875
#> 256  22.6257376   1.680833233   665     0.0625
#> 257  23.5983672   3.353244287   694     0.1875
#> 258  23.5983672   6.698066395   752     0.1875
#> 259  22.6257376   8.370477449   781     0.1250
#> 260  24.5709969   1.680833233   666     0.3750
#> 261  24.5709969   5.025655341   724     0.1875
#> 262  25.5436265   6.698066395   753     0.0625
#> 263  24.5709969   8.370477449   782     0.2500
#> 264  26.5162562   1.680833233   667     0.1875
```

```
tr1_object <- triangulate_bin_centroids(df_bin_centroids, x, y)
tr_from_to_df <- generate_edge_info(triangular_object = tr1_object)

## To generate a data set with high-D and 2D training data
df_all <- training_data |> dplyr::select(-c(ID, shape_label)) |>
  dplyr::bind_cols(tSNE_data_with_id)

## To generate averaged high-D data

df_bin <- avg_highD_data(.data = df_all, column_start_text = "PC") ## Need to pass ID column name

## Compute 2D distances
distance <- cal_2d_dist(.data = tr_from_to_df)

plot_dist(distance)
```

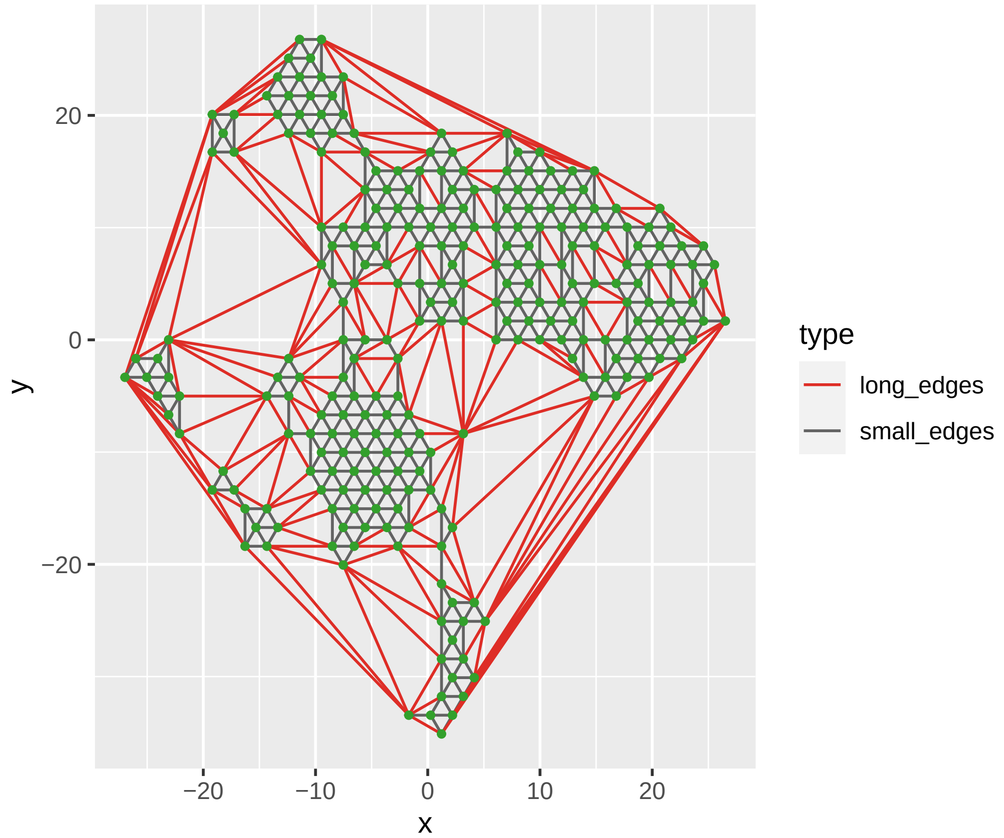



```
benchmark <- find_benchmark_value(.data = distance, distance_col = "distance")

trimesh <- ggplot(df_bin_centroids, aes(x = x, y = y)) +
```

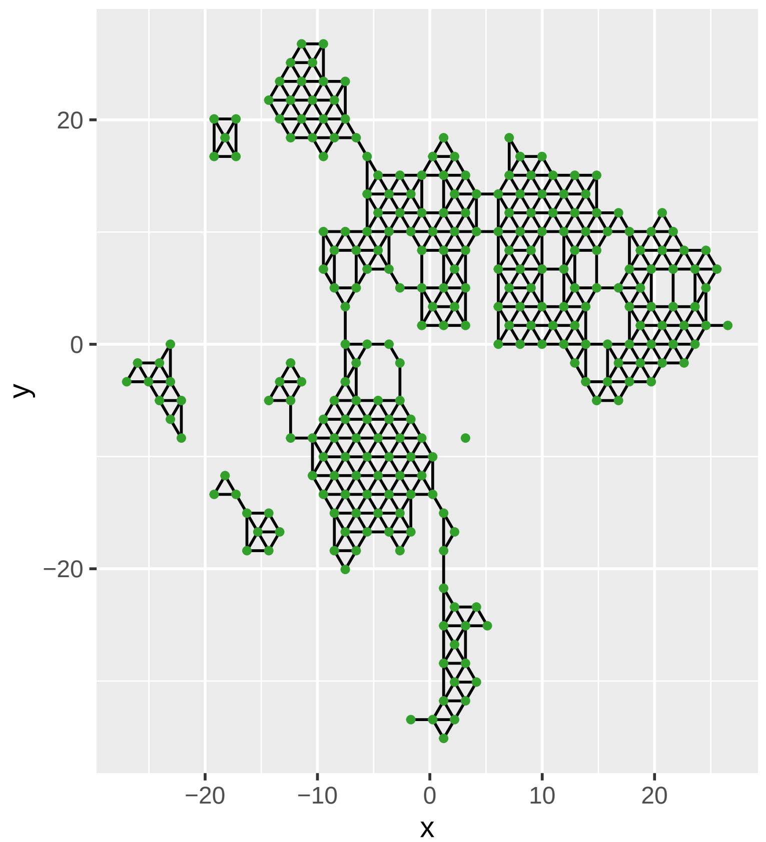
```
  geom_point(size = 0.1) +
  geom_trimesh() +
  coord_equal()

trimesh
```




```
trimesh_gr <- colour_long_edges(.data = distance, benchmark_value = benchmark,
                                triangular_object = tr1_object, distance_col = "distance")

trimesh_gr
```




```
trimesh_removed <- remove_long_edges(.data = distance, benchmark_value = benchmark,
                                     triangular_object = tr1_object, distance_col = "distance")
trimesh_removed
```




```
tour1 <- show_langevitour(df_all, df_bin, df_bin_centroids, benchmark_value = benchmark,
```

```
                    distance = distance, distance_col = "distance", column_start_text = "PC")
tour1
```

4 Conclusion

5 Acknowledgements

This article is created using **knitr** (Xie 2015) and **rmarkdown** (Xie, Allaire, and Grolemund 2018) in R with the `rjtools::rjournal_article` template. The source code for reproducing this paper can be found at: https://github.com/JayaniLakshika/paper-quollr.

References

Wickham, Hadley. 2011. "Testthat: Get Started with Testing." *The R Journal* 3: 5–10. https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf.

Xie, Yihui. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.name/knitr/.

Xie, Yihui, J. J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown.

Jayani P.G. Lakshika
Monash University
Department of Econometrics and Business Statistics, VIC 3800 Australia
https://jayanilakshika.netlify.app/
ORCiD: 0000-0002-6265-6481
jayani.piyadigamage@monash.edu

Dianne Cook
Monash University
Department of Econometrics and Business Statistics, VIC 3800 Australia
http://www.dicook.org/
ORCiD: 0000-0002-3813-7155
dicook@monash.edu

Paul Harrison
Monash University
MGBP, BDInstitute, VIC 3800 Australia
ORCiD: 0000-0002-3980-268X
paul.harrison@monash.edu

Michael Lydeamore
Monash University
Department of Econometrics and Business Statistics, VIC 3800 Australia
ORCiD: 0000-0001-6515-827X
michael.lydeamore@monash.edu

Thiyanga S. Talagala
University of Sri Jayewardenepura
Department of Statistics, Gangodawila, Nugegoda 10100 Sri Lanka
https://thiyanga.netlify.app/
ORCiD: 0000-0002-0656-9789
ttalagala@sjp.ac.lk