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", cell_area = 1)
num_bins_x

#> [1] 6

shape_val <- calculate_effective_shape_value(.data = s_curve_noise_umap, x = "UMAP1", y = "UMAP2")
shape_val

#> [1] 2.019414

num_bins_y <- calculate_effective_y_bins(.data = s_curve_noise_umap, x = "UMAP1", y = "UMAP2", shape_val = 1.8330
num_bins_y

#> [1] 12
```

Generate full hex grid

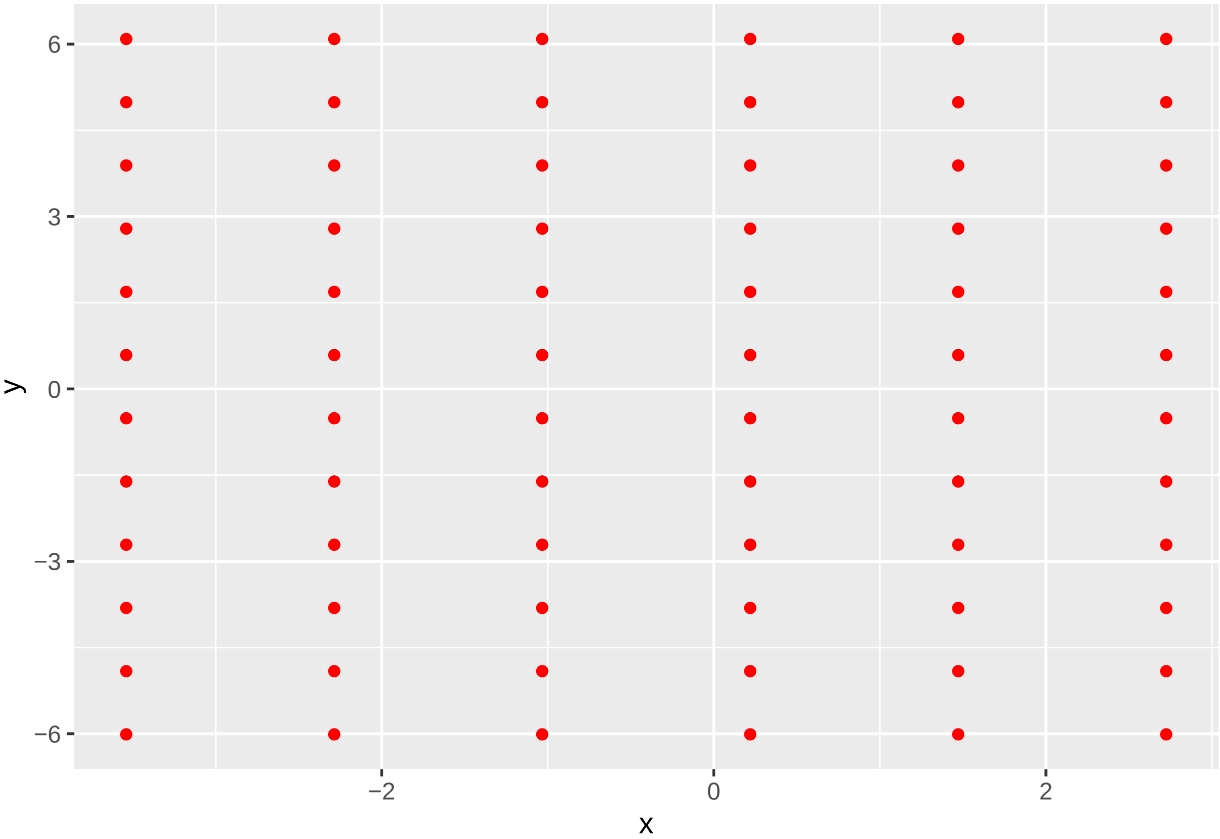
Generating full hexagonal grid contains main three steps:

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

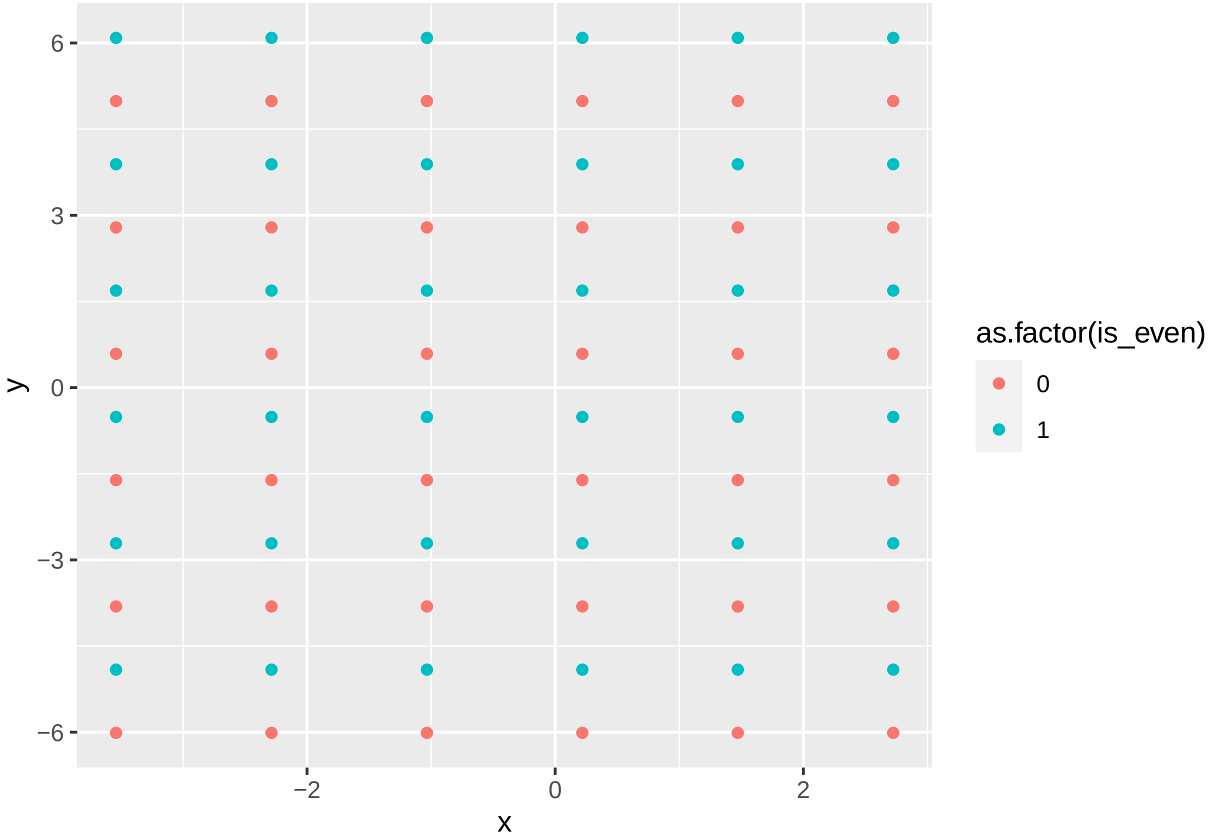
```
cell_area <- 1
cell_diameter <- sqrt(2 * cell_area / sqrt(3))

hex_size <- cell_diameter/2

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

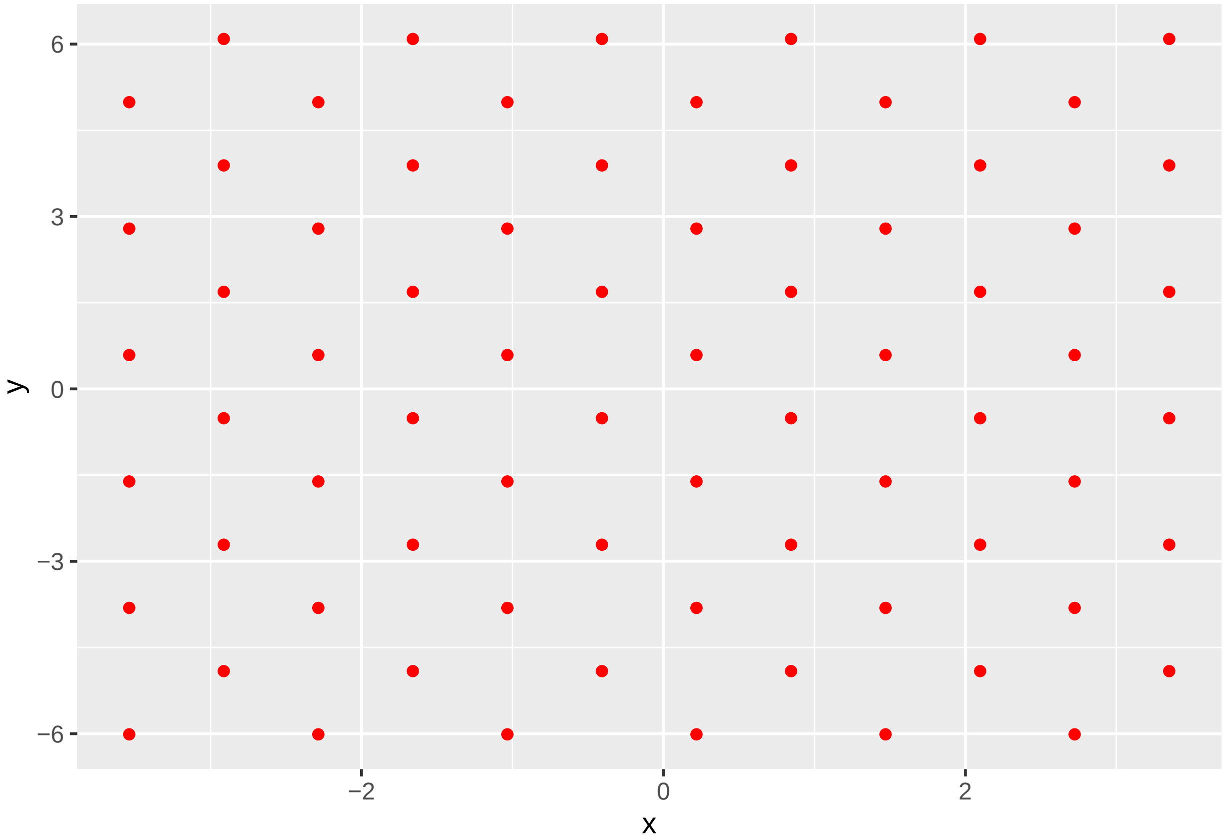
```
box_points <- box_points |>
   dplyr::arrange(x) |>
   dplyr::group_by(x) |>
   dplyr::group_modify(~ generate_even_y(.x)) |>
```

```
    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: 72
#> Columns: 2
#> $ x <dbl> -3.5390002, -2.9126765, -3.5390002, -2.9126765, -3.5390002, -2.91267~
#> $ y <dbl> -6.0111830, -4.9111506, -3.8111182, -2.7110858, -1.6110534, -0.51102~
```

2. Generate hexagonal coordinates

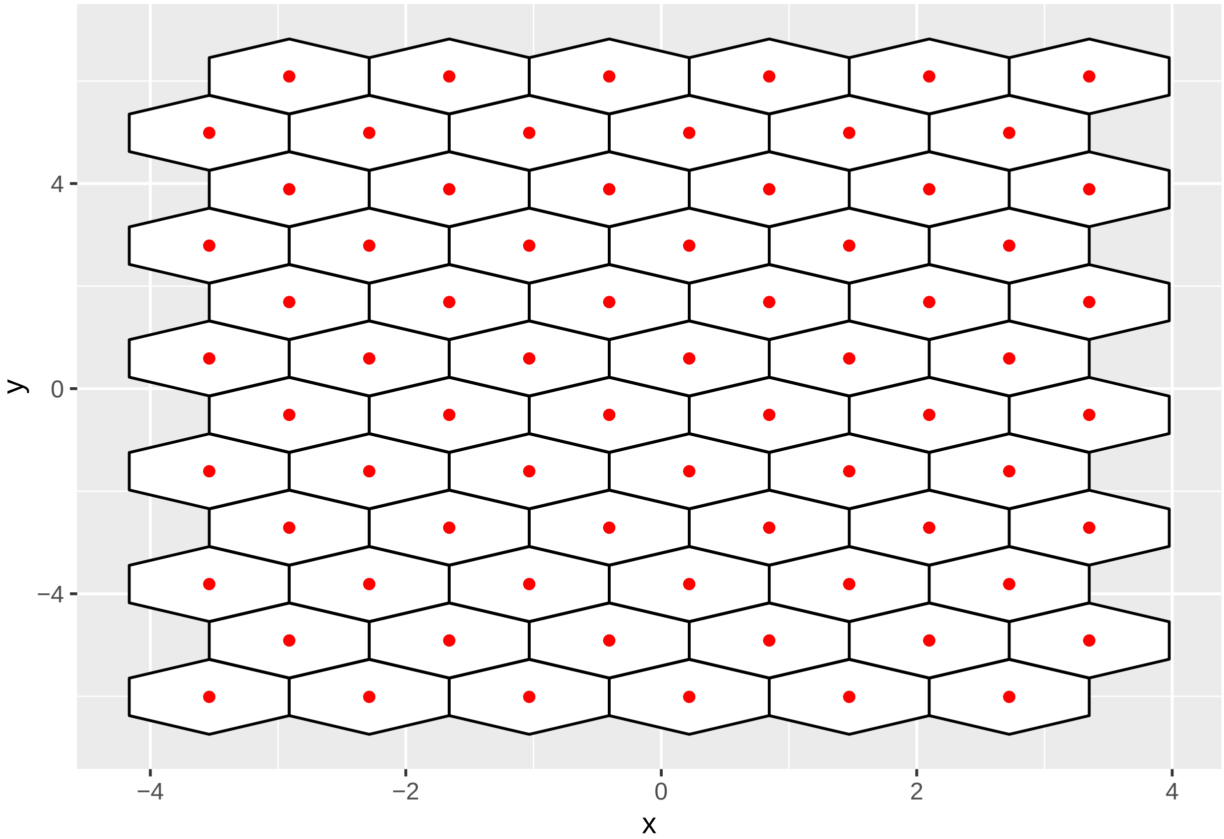
   Steps: - Compute horizontal width of the hexagon

   - 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)
glimpse(hex_grid)
```

```
#> Rows: 432
#> Columns: 3
#> $ x  <dbl> -2.912676, -2.912676, -3.539000, -4.165324, -4.165324, -3.539000, -~
#> $ y  <dbl> -5.645998, -6.376368, -6.741553, -6.376368, -5.645998, -5.280813, -~
#> $ id <int> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 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")
```

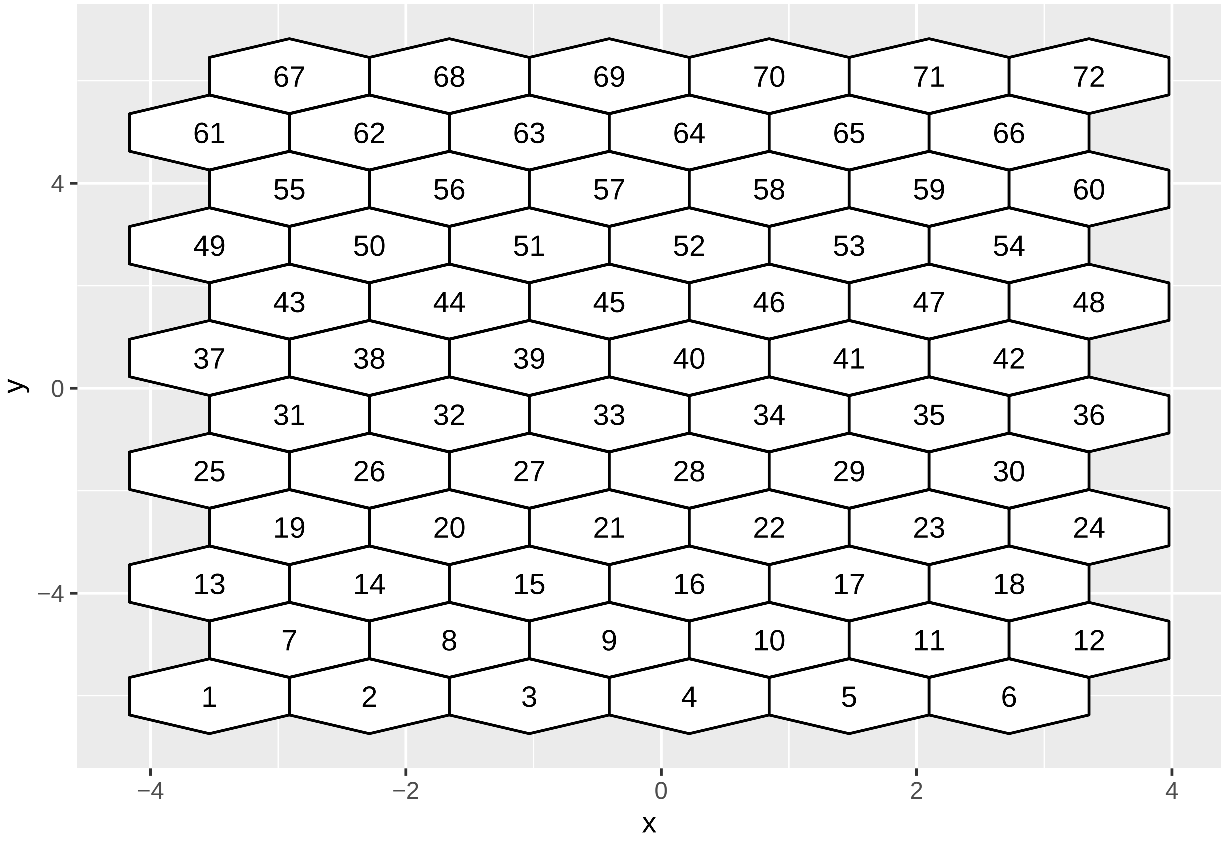


3. Map hexagonal IDs

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

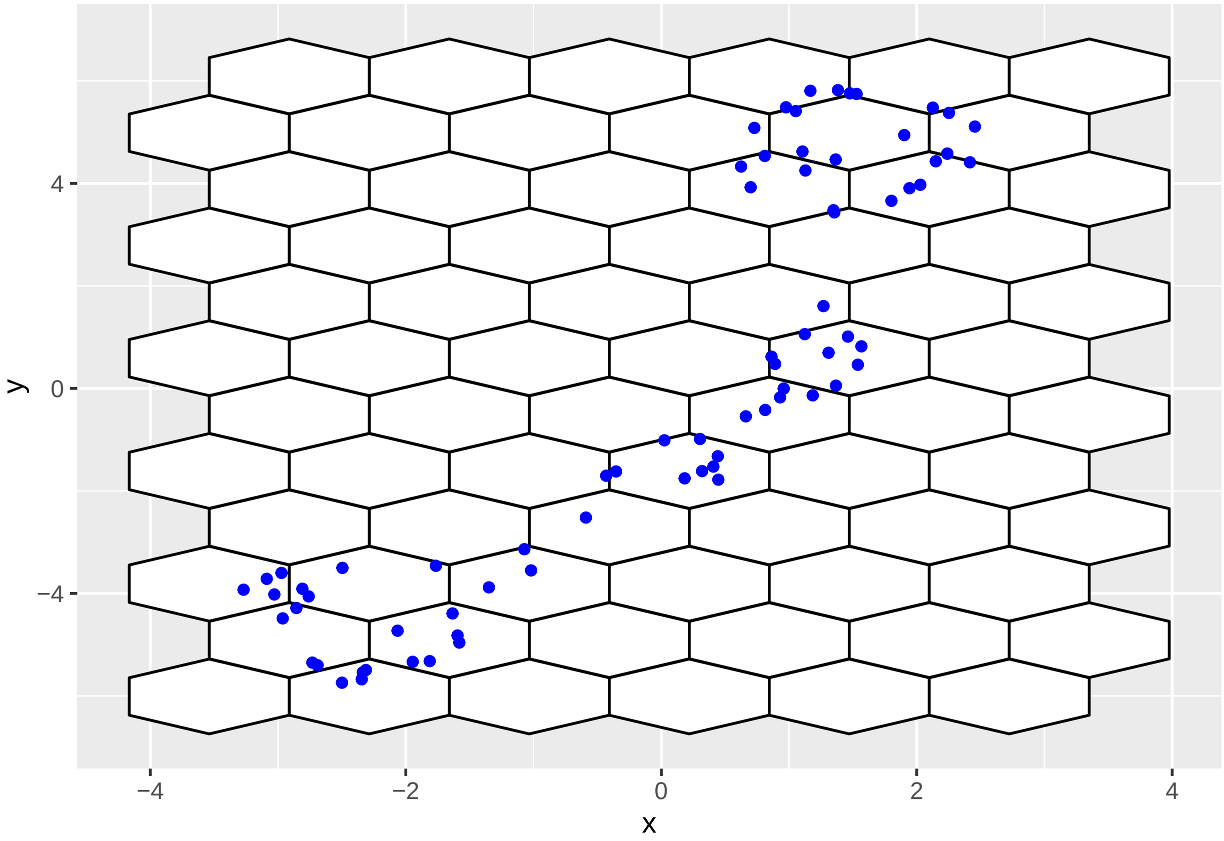
```
df_with_std_counts <- compute_std_counts(nldr_df = s_curve_noise_umap_with_id)
```

6. Extract full grid info

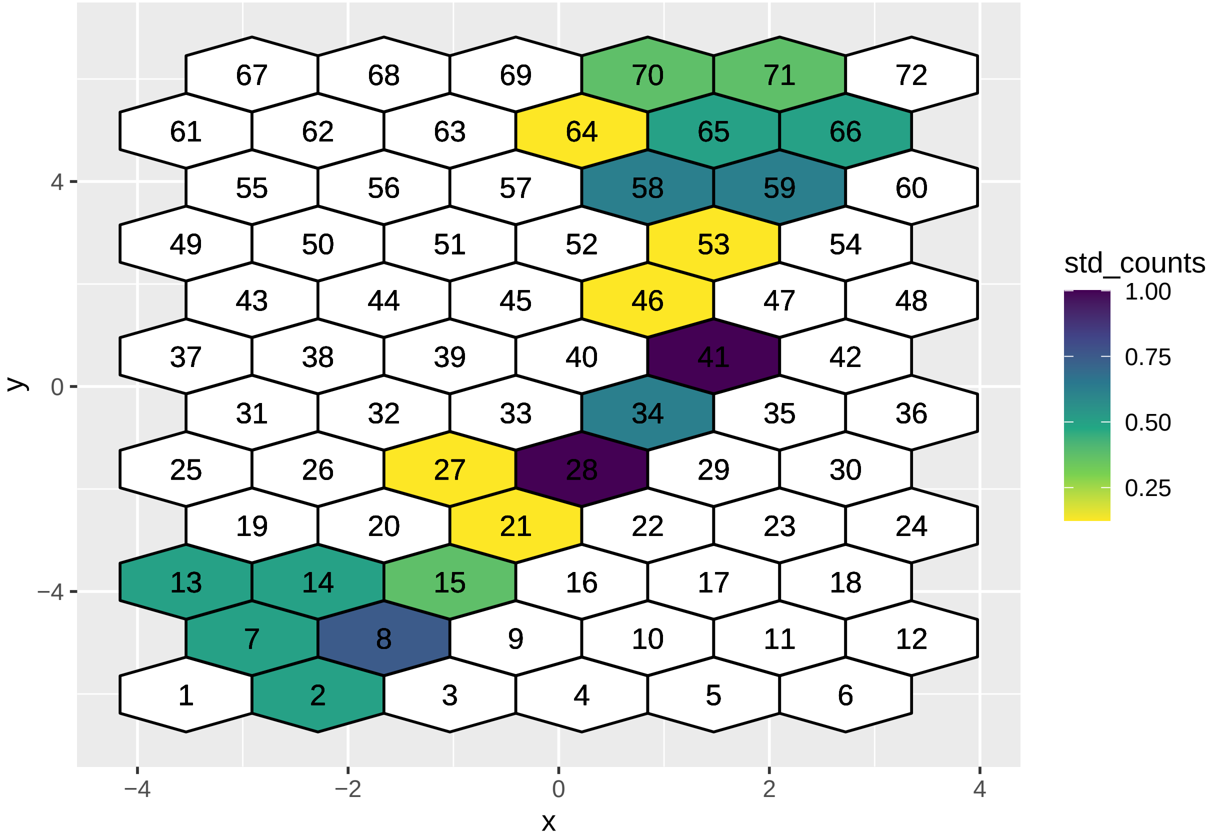
- Assign standardize counts for hex bins
- Join with the hexagonal coordinates

```
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
```

```
#> # A tibble: 20 x 4
#>         x      y hexID std_counts
#>     <dbl>  <dbl> <int>      <dbl>
#>  1 -2.91  -4.91      7      0.5
#>  2 -3.54  -3.81     13      0.5
#>  3 -2.29  -6.01      2      0.5
#>  4 -1.66  -4.91      8      0.75
#>  5 -2.29  -3.81     14      0.5
#>  6 -1.03  -3.81     15      0.375
#>  7 -0.407 -2.71     21      0.125
#>  8 -1.03  -1.61     27      0.125
#>  9  0.219 -1.61     28      1
#> 10  0.845 -0.511    34      0.625
#> 11  0.845  1.69     46      0.125
#> 12  0.845  3.89     58      0.625
#> 13  0.219  4.99     64      0.125
#> 14  0.845  6.09     70      0.375
#> 15  1.47   0.589    41      1
#> 16  1.47   2.79     53      0.125
#> 17  2.10   3.89     59      0.625
#> 18  1.47   4.99     65      0.5
#> 19  2.10   6.09     71      0.375
#> 20  2.72   4.99     66      0.5
```

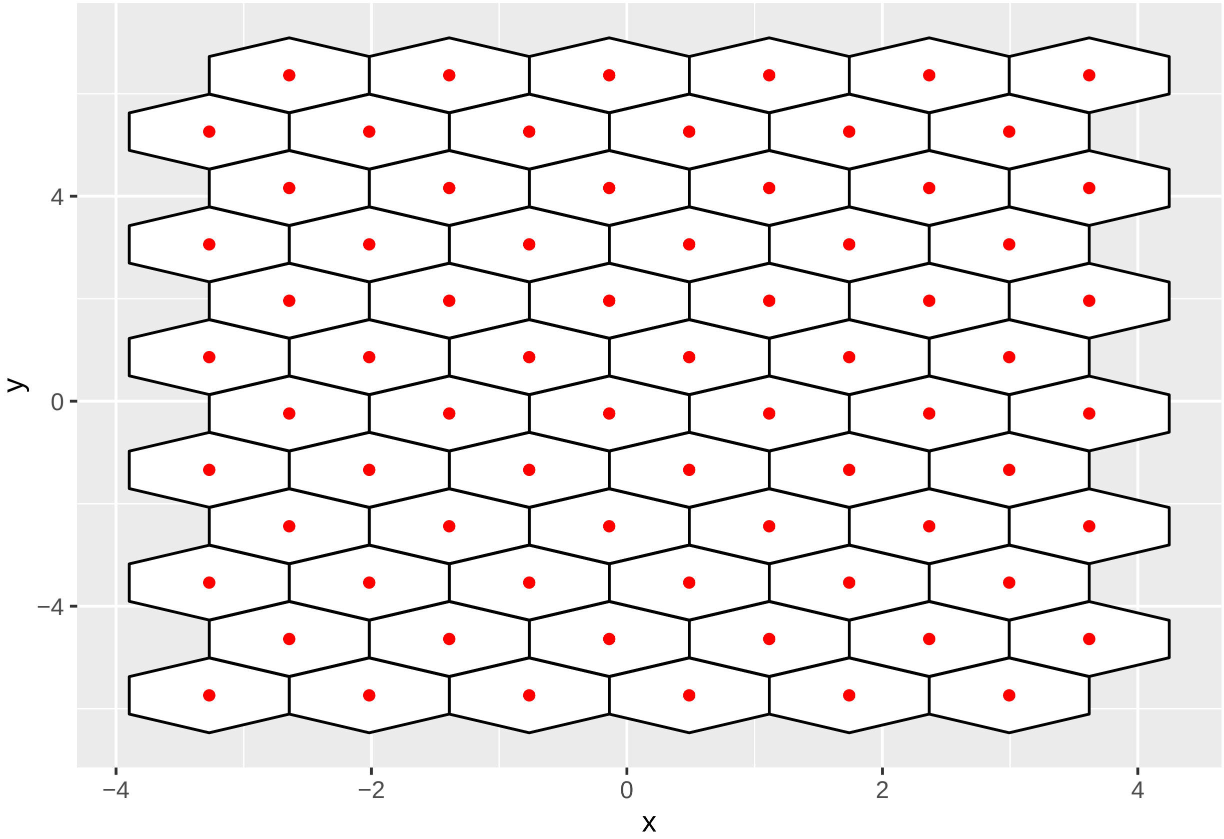
```
tr1_object <- triangulate_bin_centroids(df_bin_centroids, x, y)
tr_from_to_df <- generate_edge_info(triangular_object = tr1_object)
```

Compute parameter defaults

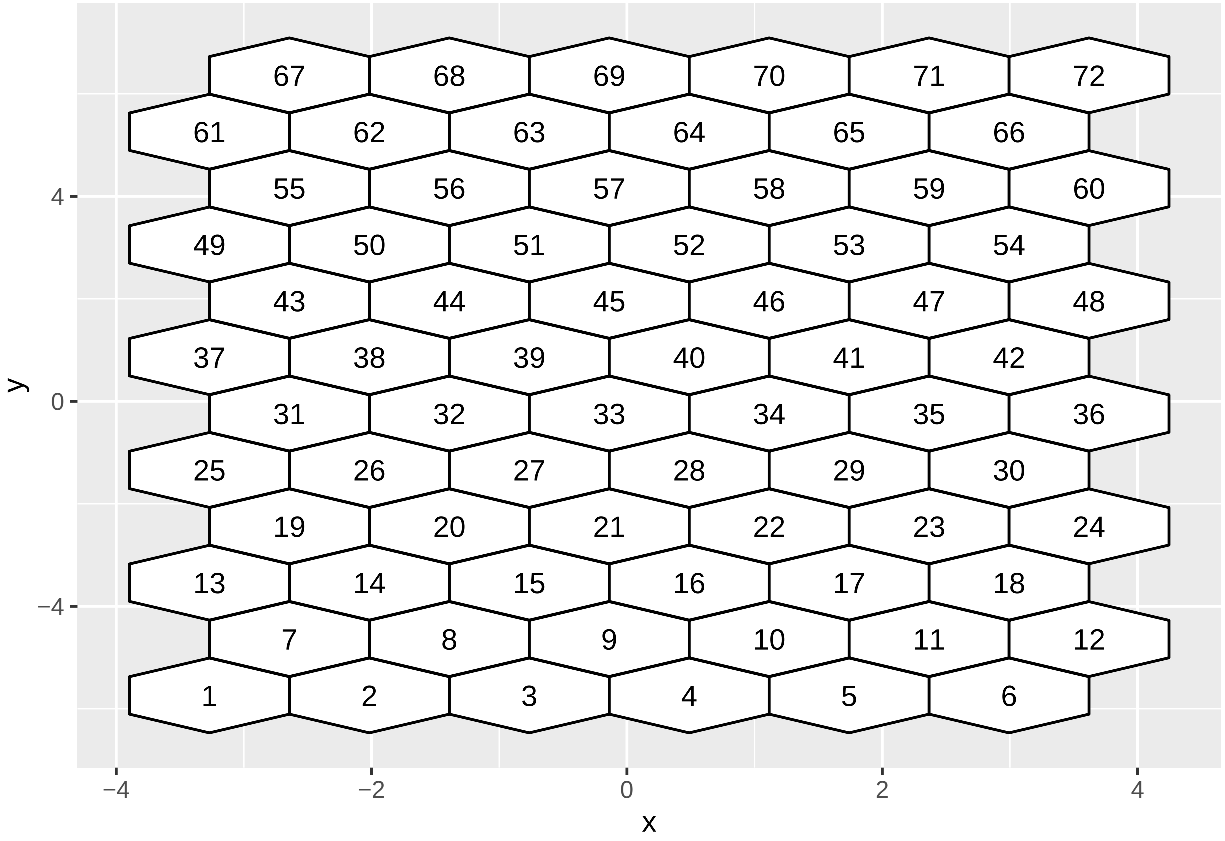
Shift the hexagonal grid origin If shift_x happen to the positive direction of x it should input as a positive value, if not other way If shift_y happen to the positive direction of y it should input as a positive value, if not other way

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

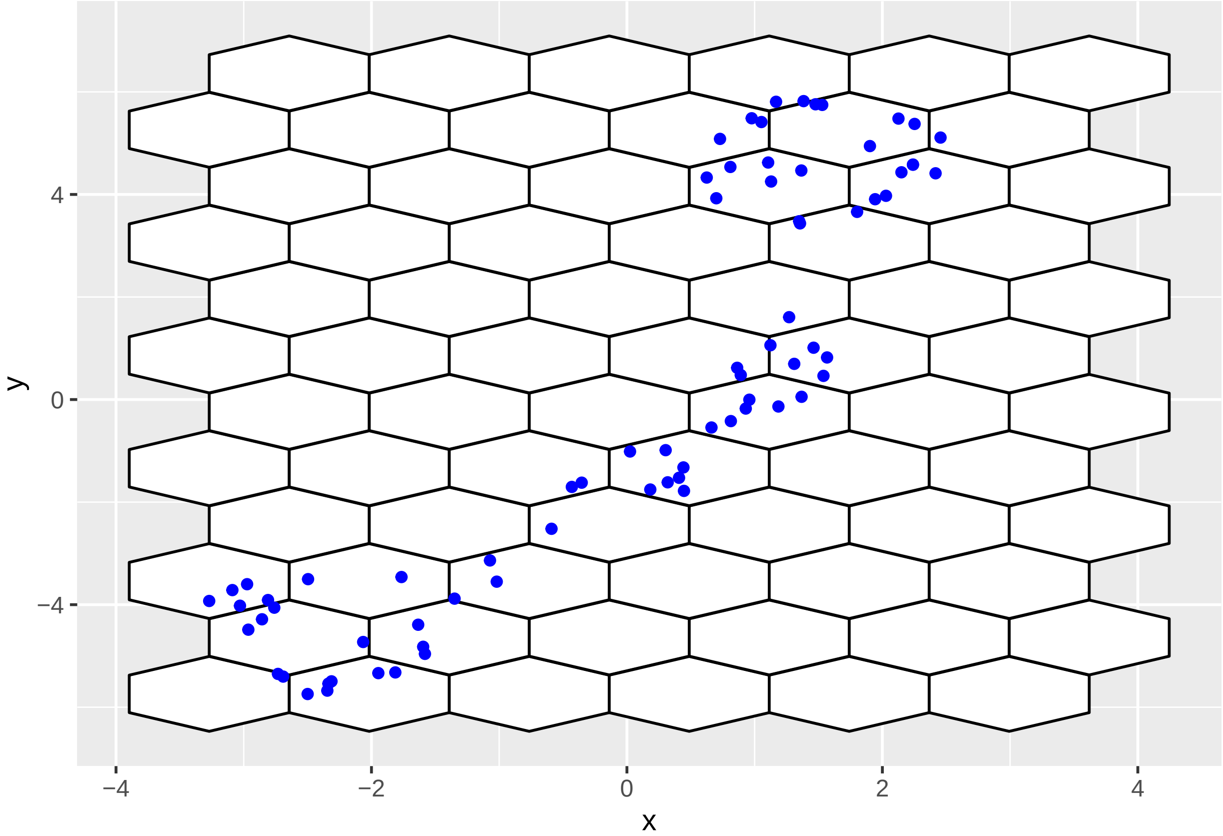
```
all_centroids_df_shift <- extract_coord_of_shifted_hex_grid(nldr_df = s_curve_noise_umap,
                                                  x = "UMAP1", y = "UMAP2",
                                                  num_bins_x = num_bins_x,
                                                  num_bins_y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

glimpse(all_centroids_df_shift)

#> Rows: 72
#> Columns: 2
#> $ x <dbl> -3.2700000, -2.6436763, -3.2700000, -2.6436763, -3.2700000, -2.64367~
#> $ y <dbl> -5.7400000, -4.6399676, -3.5399352, -2.4399028, -1.3398704, -0.23983~

hex_grid <- gen_hex_coordinates(all_centroids_df_shift)
glimpse(hex_grid)

#> Rows: 432
#> Columns: 3
#> $ x  <dbl> -2.643676, -2.643676, -3.270000, -3.896324, -3.896324, -3.270000, -~
#> $ y  <dbl> -5.3748152, -6.1051848, -6.4703696, -6.1051848, -5.3748152, -5.0096~
#> $ 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")
```



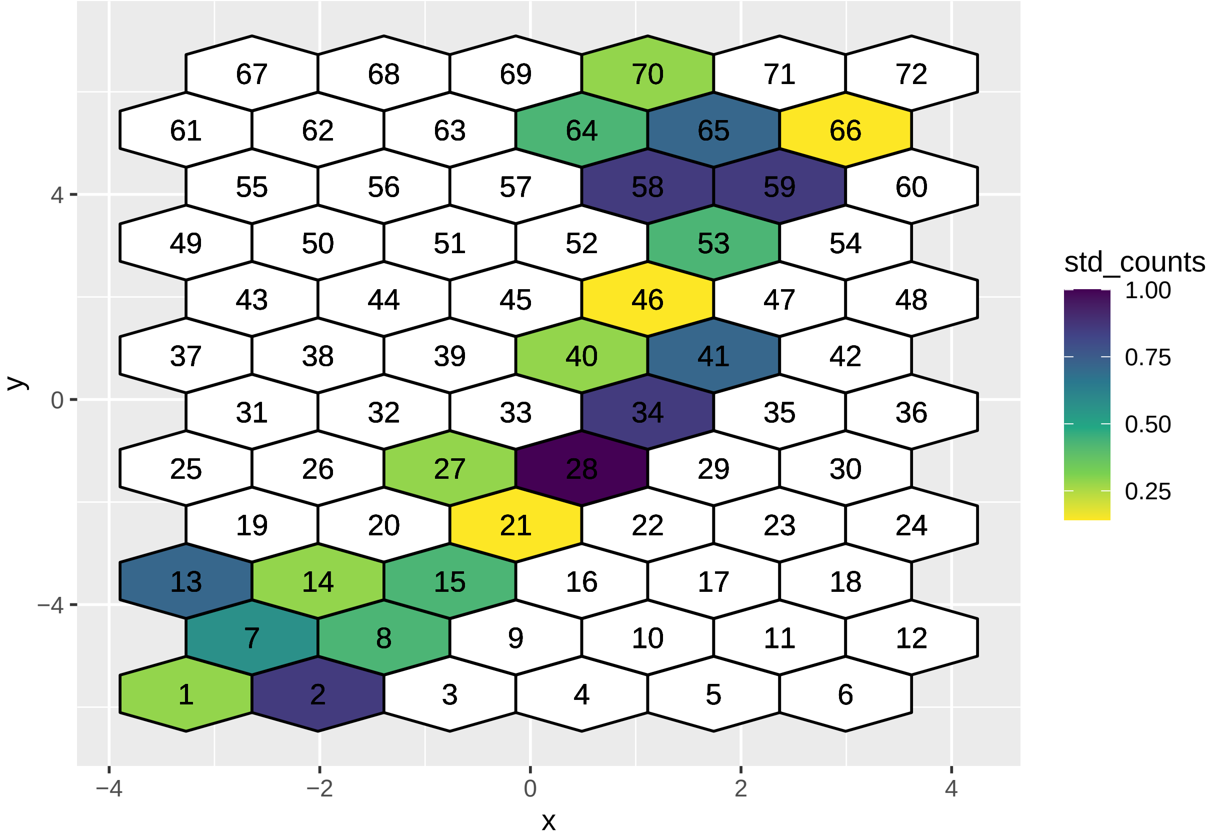
```
full_grid_with_hexbin_id <- map_hexbin_id(all_centroids_df_shift)

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")
```



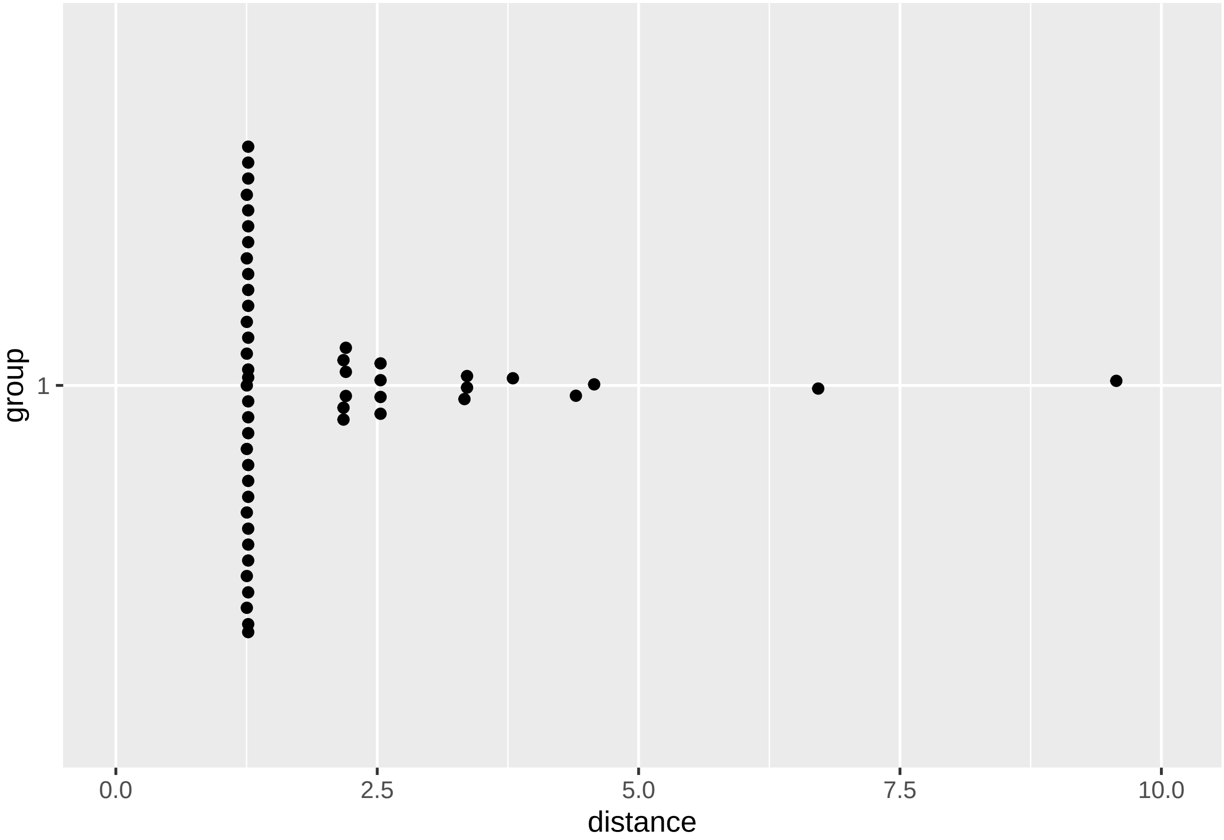
```
df_bin_centroids <- hex_full_count_df[complete.cases(hex_full_count_df[["std_counts"]]), ] |>
```

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

df_bin_centroids

#> # A tibble: 21 x 4
#>         x     y hexID std_counts
#>     <dbl> <dbl> <int>      <dbl>
#>  1 -3.27  -5.74     1      0.286
#>  2 -2.64  -4.64     7      0.571
#>  3 -3.27  -3.54    13      0.714
#>  4 -2.02  -5.74     2      0.857
#>  5 -1.39  -4.64     8      0.429
#>  6 -2.02  -3.54    14      0.286
#>  7 -0.765 -3.54    15      0.429
#>  8 -0.138 -2.44    21      0.143
#>  9 -0.765 -1.34    27      0.286
#> 10  0.488 -1.34    28      1
#> # i 11 more rows

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

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")
```

Model function

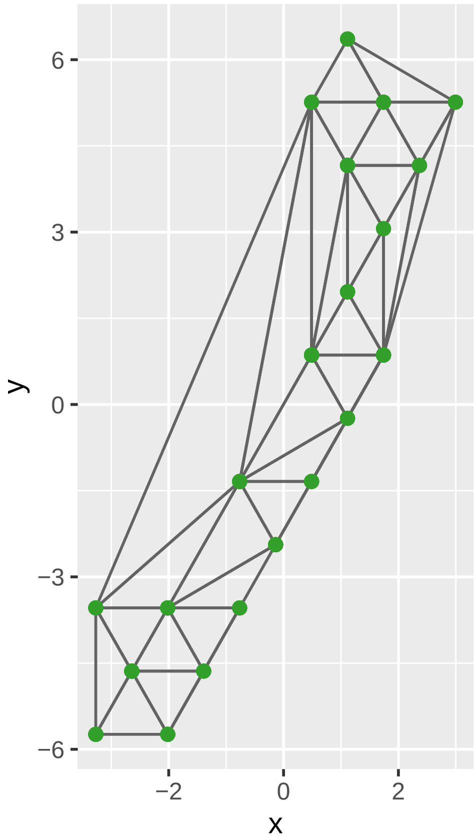
Predict 2D embeddings

Compute residuals

Visualizations

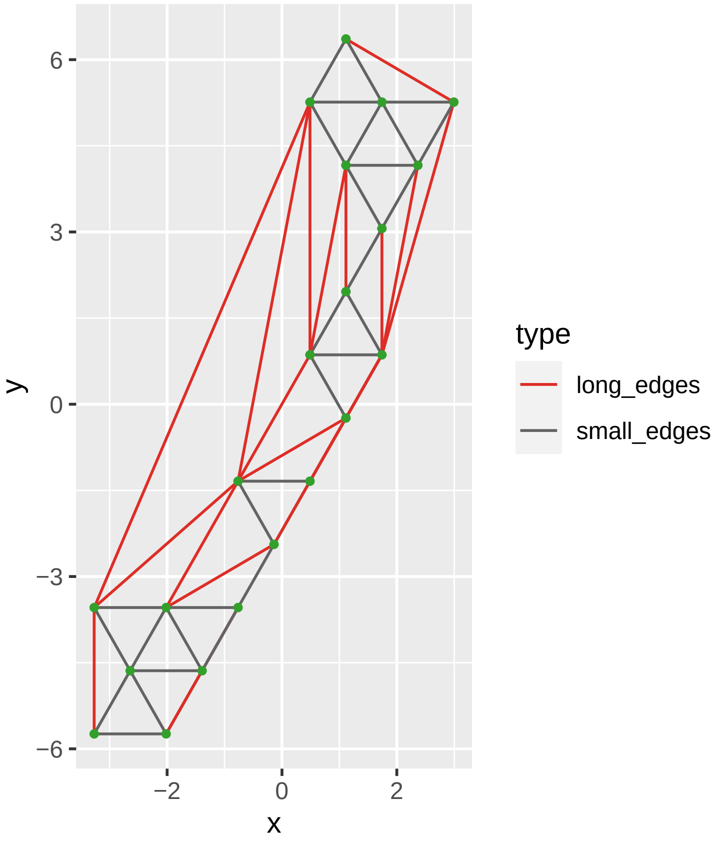
geom_trimesh

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

trimesh
```



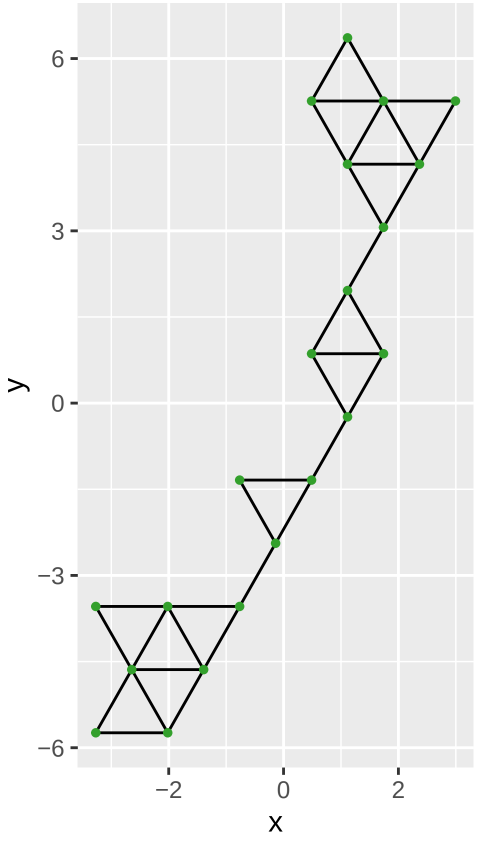
coloured_long_edges

```
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

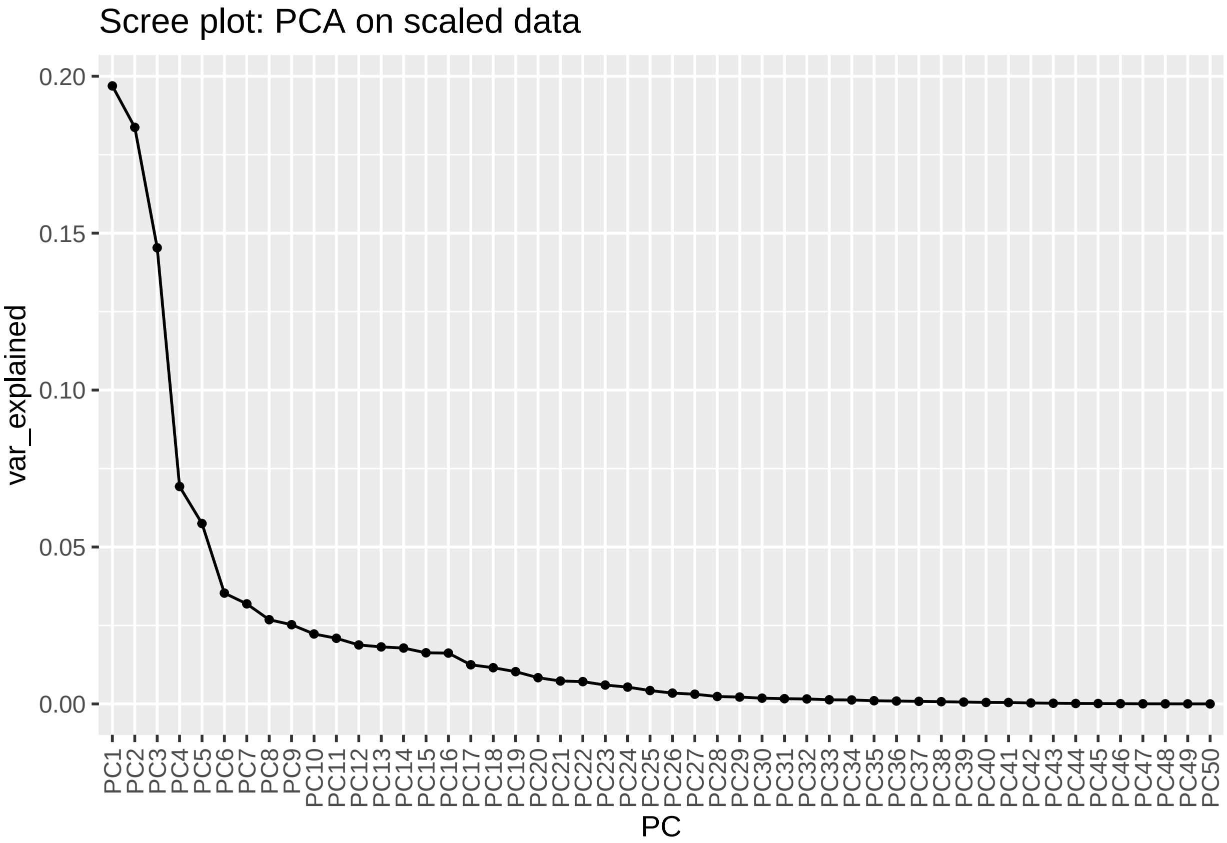
```
tour1 <- show_langevitour(df_all, df_bin, df_bin_centroids, benchmark_value = benchmark,
                          distance = distance, distance_col = "distance")
tour1
```

Tests

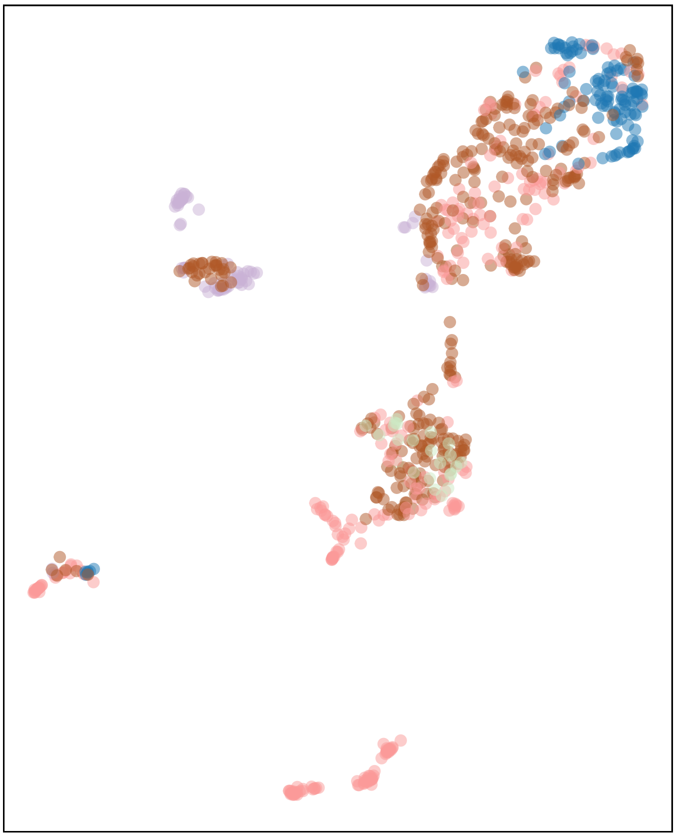
3 Application

```
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 |>
```

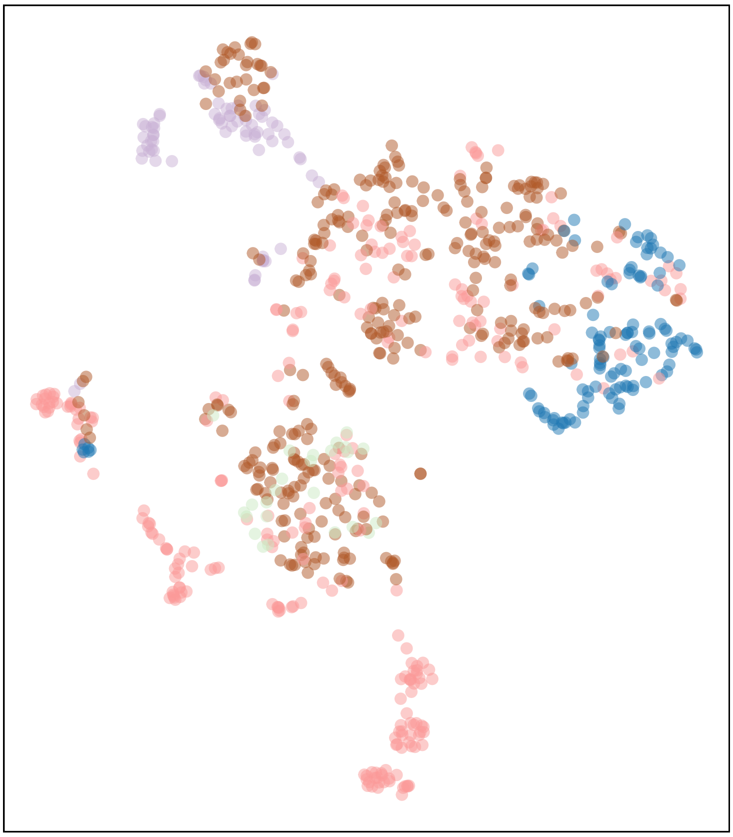
```
  ggplot(aes(x = PC,y = var_explained, group = 1))+
  geom_point(size=1)+
  geom_line()+
  labs(title="Scree plot: PCA on scaled data") +
  scale_x_discrete(limits = paste0(rep("PC", 50), 1:50)) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



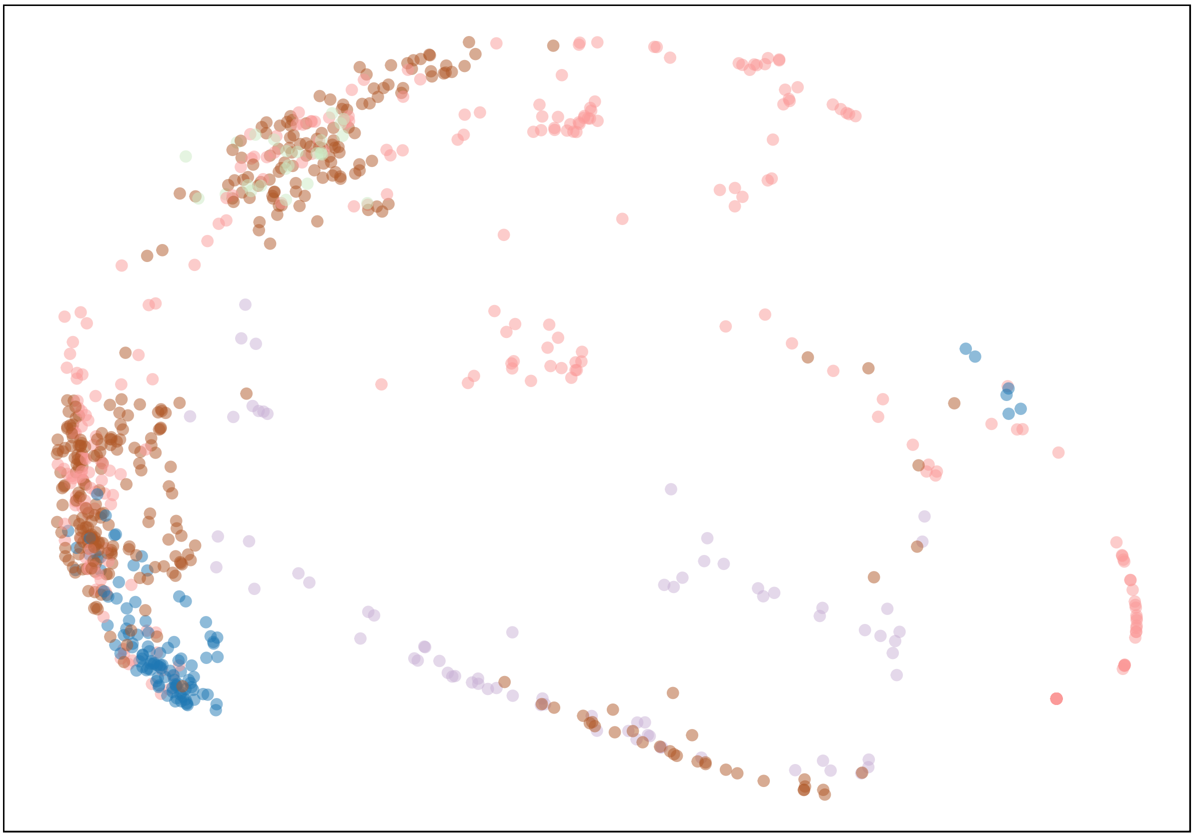
```
data_split <- initial_split(data_pca)
training_data <- training(data_split) |>
  arrange(ID)
test_data <- testing(data_split) |>
  arrange(ID)

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
```



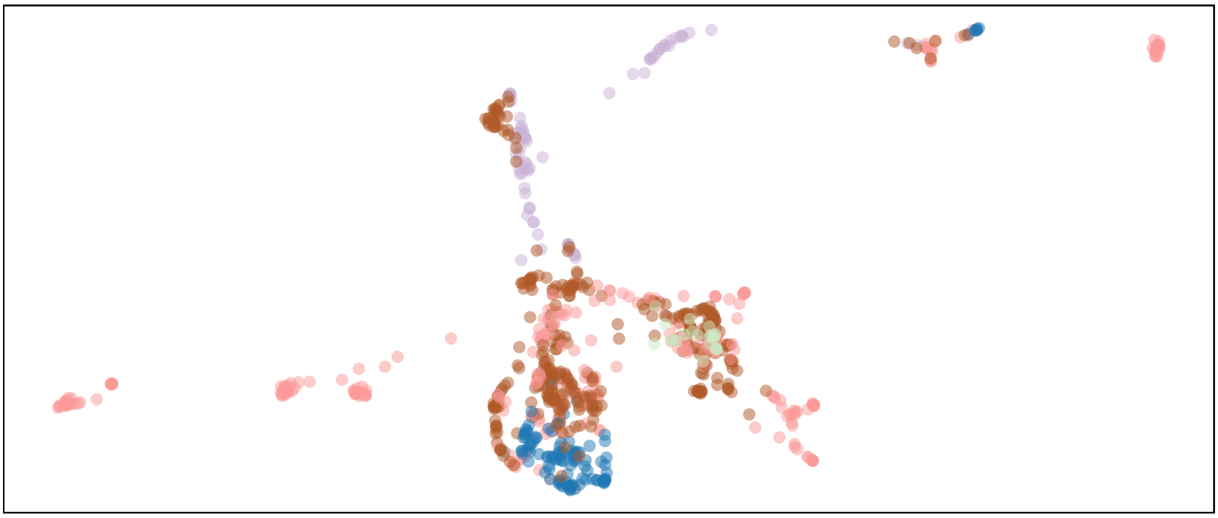
```
tSNE_data <- Fit_tSNE(training_data |> dplyr::select(-c(ID, shape_label)), opt_perplexity = calculate_effective_

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
```



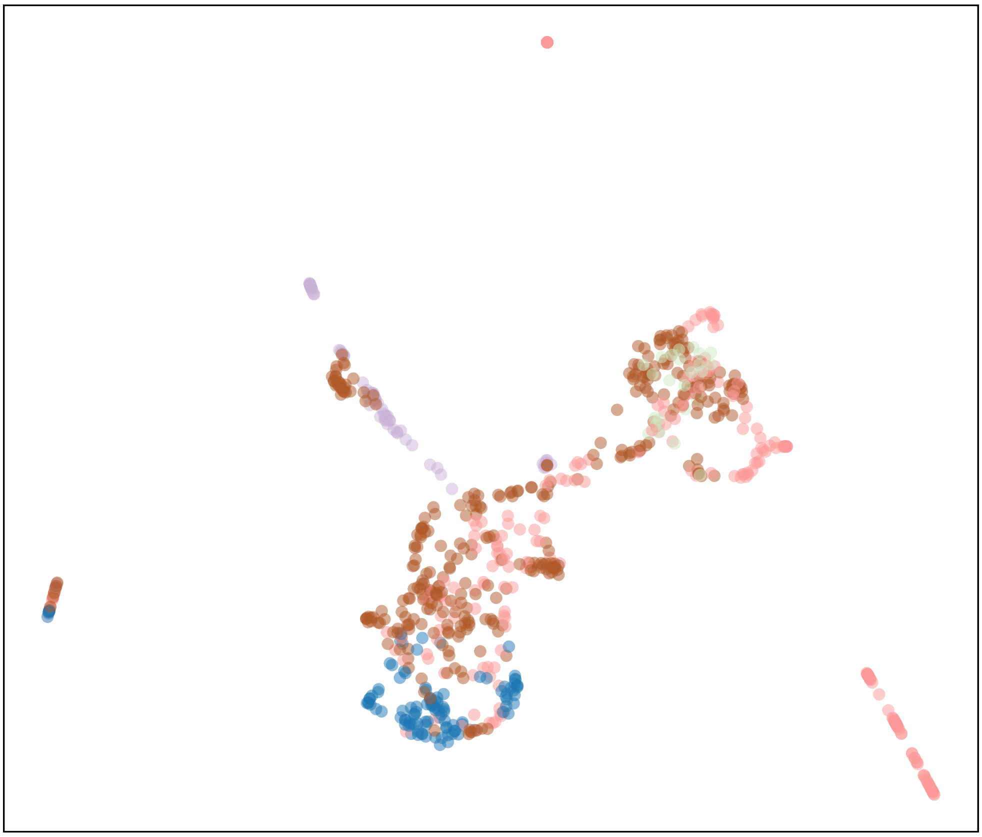
```
PHATE_data <- Fit_PHATE(training_data |> dplyr::select(-c(ID, shape_label)), knn = 5, with_seed = 20240110)

#> 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 0.39 seconds.
#>   Calculating diffusion potential...
#>   Calculated diffusion potential in 0.32 seconds.
#>   Calculating metric MDS...
#>   Calculated metric MDS in 8.83 seconds.
#> Calculated PHATE in 9.55 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
```



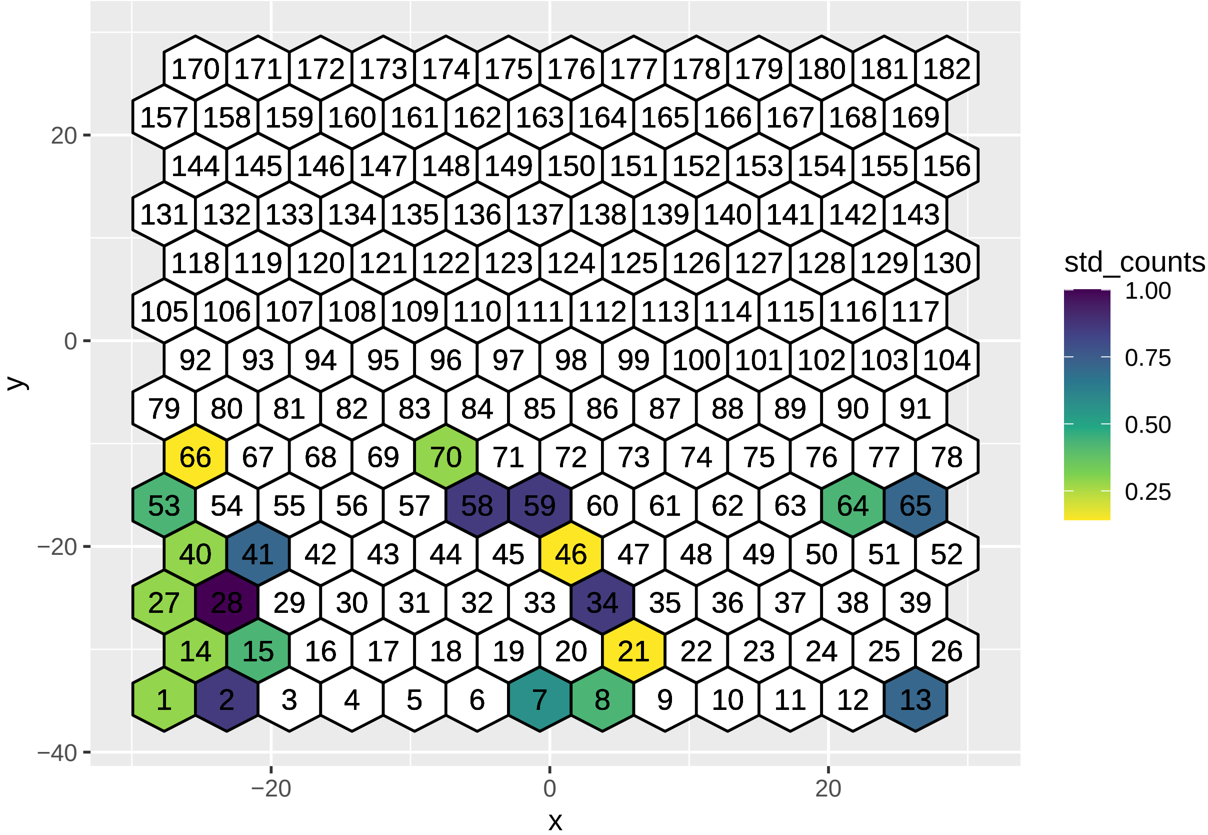
```
num_bins_x <- calculate_effective_x_bins(.data = tSNE_data, x = ”tSNE1”, cell_area = 1)
num_bins_x <- 13

shape_val <- calculate_effective_shape_value(.data = tSNE_data, x = ”tSNE1”, y = ”tSNE2”)
shape_val
```

```
#> [1] 1.13882
```

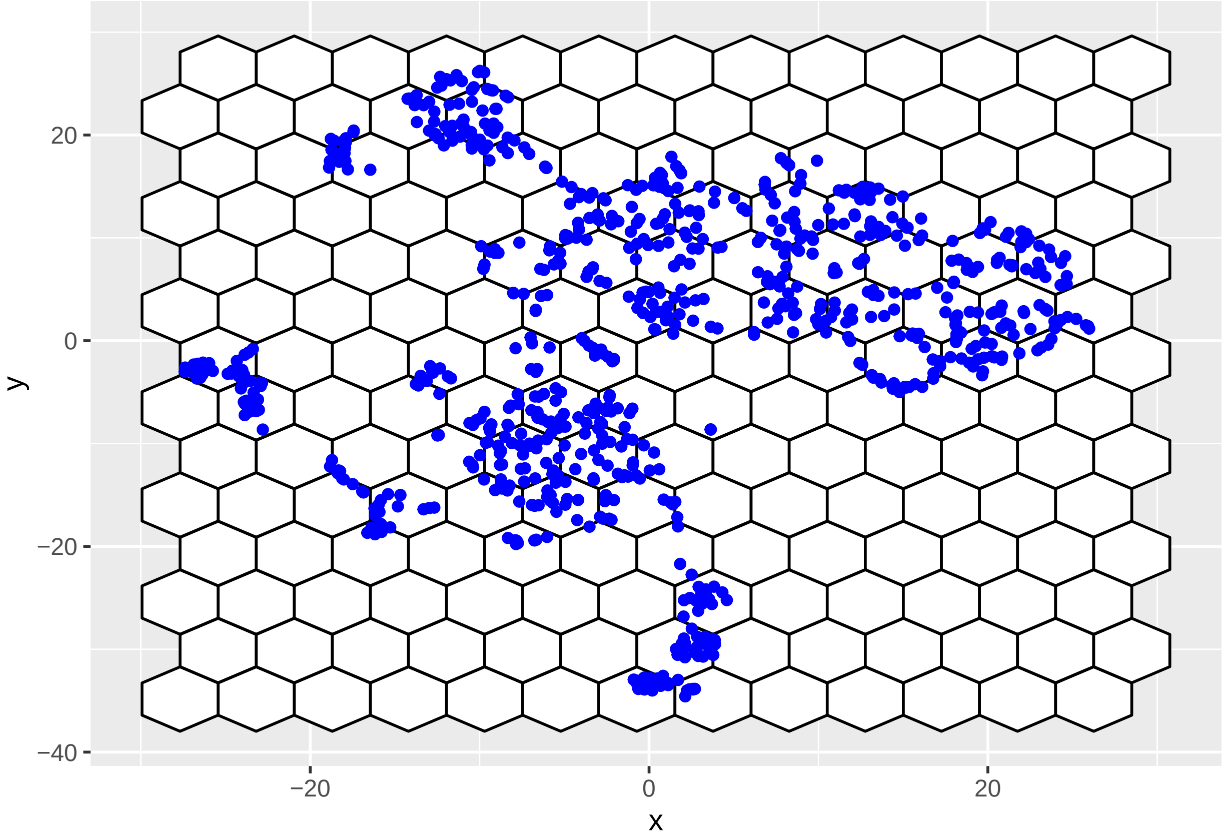
```
num_bins_y <- calculate_effective_y_bins(.data = tSNE_data, x = ”tSNE1”, y = ”tSNE2”, shape_val = 0.8417289, num_
num_bins_y
```

```
#> [1] 14
```

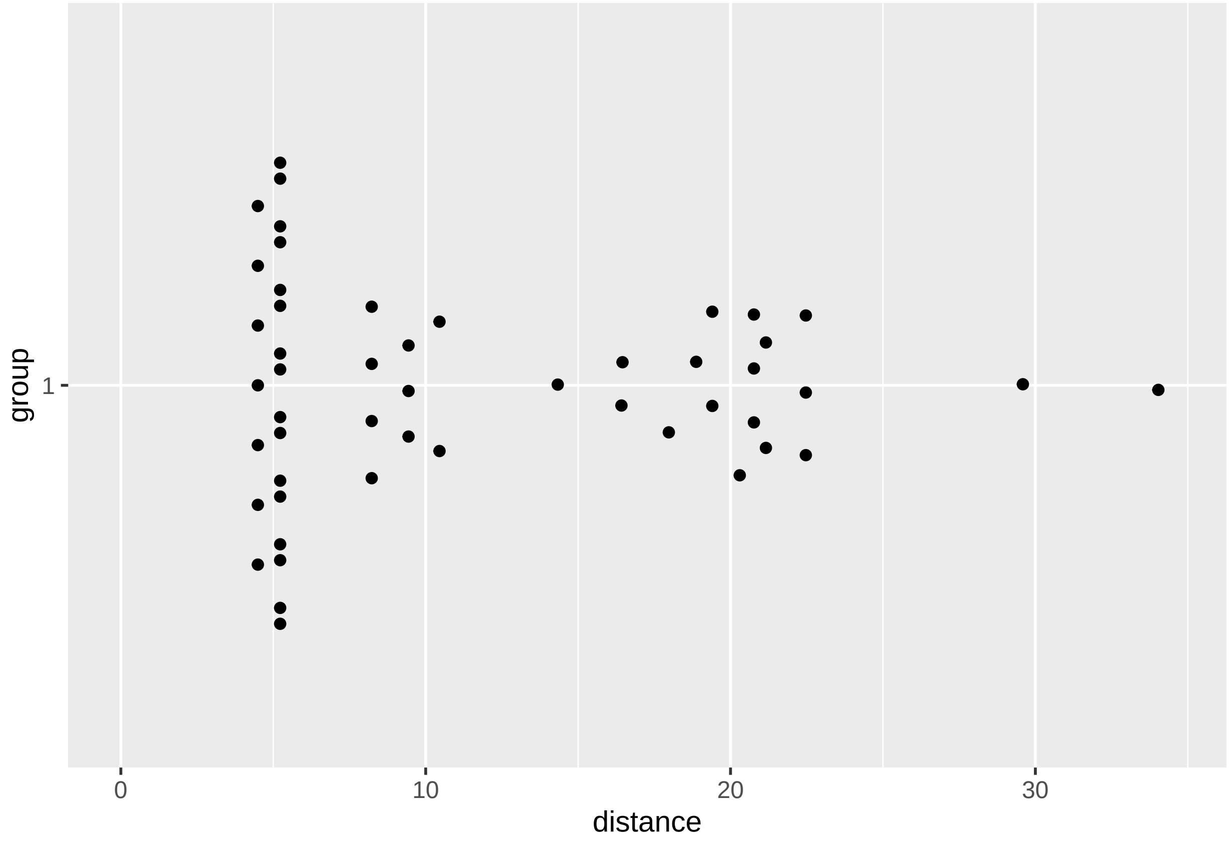
```
all_centroids_df <- generate_full_grid_centroids(nldr_df = tSNE_data,
                                                 x = ”tSNE1”, y = ”tSNE2”,
                                                 num_bins_x = num_bins_x,
                                                 num_bins_y = num_bins_y,
                                                 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 = 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_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”)
```



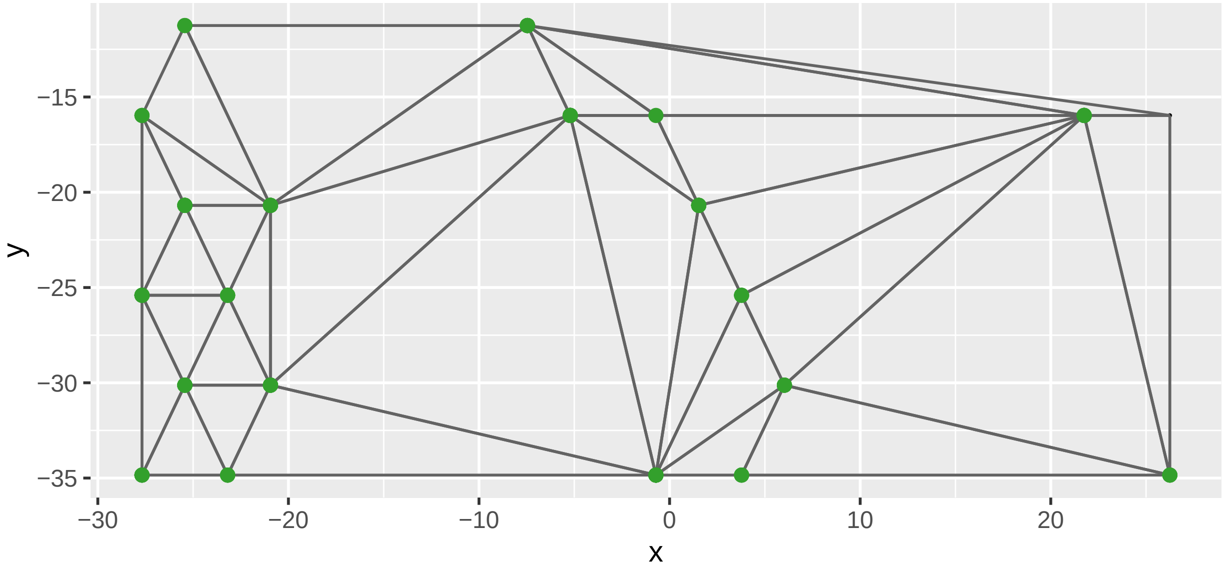
```
ggplot(data = hex_grid, aes(x = x, y = y)) + geom_polygon(fill = "white", color = "black", aes(group = id)) +
  geom_point(data = tSNE_data, aes(x = tSNE1, y = tSNE2), color = "blue")
```



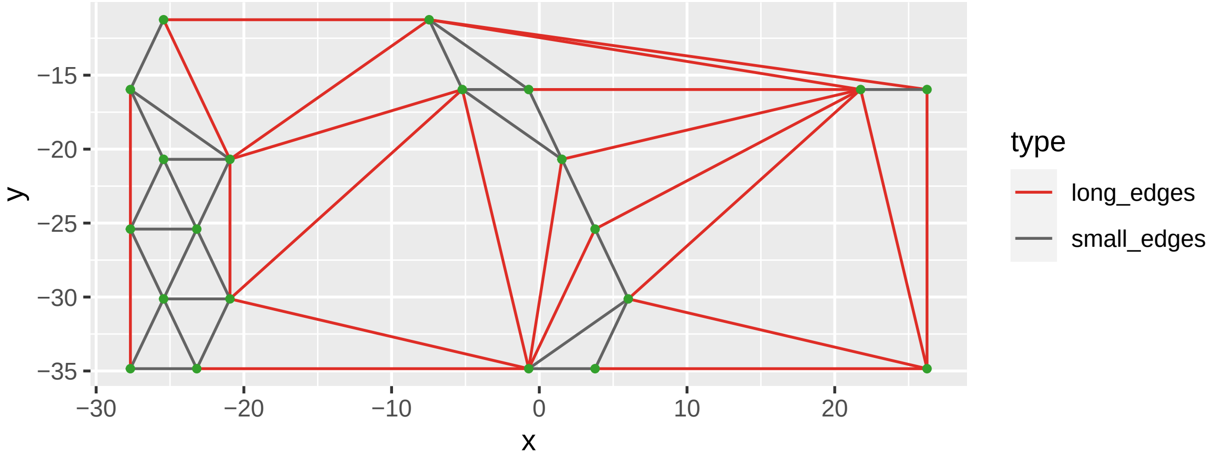
```
df_bin_centroids <- hex_full_count_df[complete.cases(hex_full_count_df[["std_counts"]]), ] |>
  dplyr::select("c_x", "c_y", "hexID", "std_counts") |>
  dplyr::distinct() |>
  dplyr::rename(c("x" = "c_x", "y" = "c_y"))

df_bin_centroids

#> # A tibble: 21 x 4
#>         x     y hexID std_counts
#>     <dbl> <dbl> <int>      <dbl>
#>  1 -27.7 -34.8     1      0.286
#>  2 -25.4 -30.1    14      0.286
#>  3 -27.7 -25.4    27      0.286
#>  4 -25.4 -20.7    40      0.286
#>  5 -27.7 -16.0    53      0.429
#>  6 -25.4 -11.3    66      0.143
#>  7 -23.2 -34.8     2      0.857
#>  8 -20.9 -30.1    15      0.429
#>  9 -23.2 -25.4    28      1
#> 10 -20.9 -20.7    41      0.714
#> # i 11 more rows

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)
```



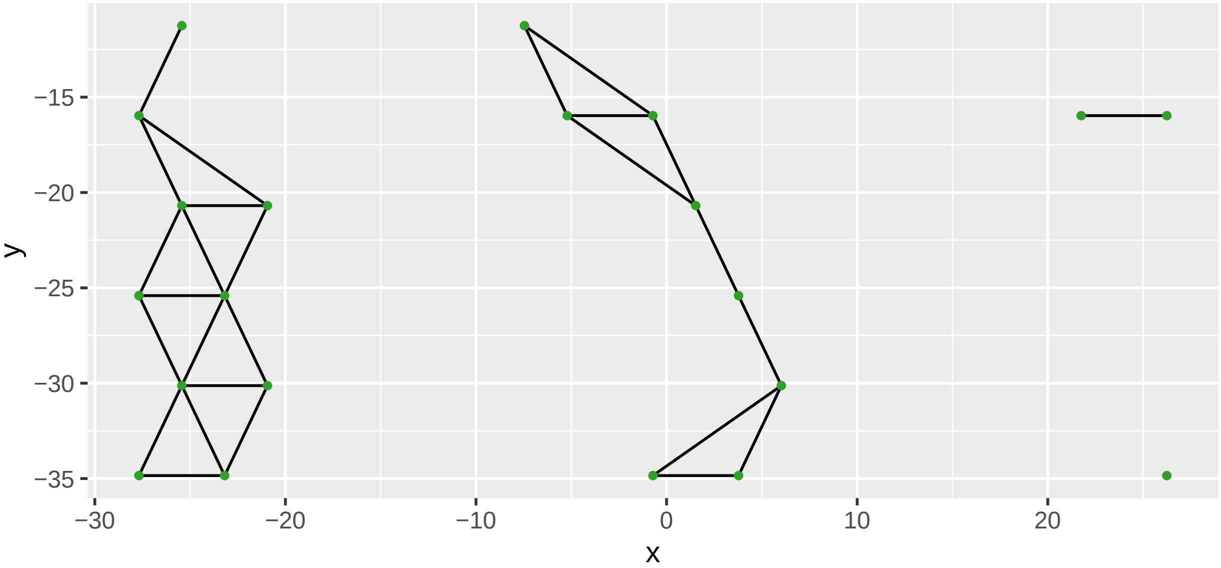
```
benchmark <- find_benchmark_value(.data = distance, distance_col = "distance")

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

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`