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)
```

1 Introduction

2 Methodology

Usage

- dependancies
- 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 = "UMAP2", y = "UMAP2", shape_val = 1.8330
num_bins_y
```

```
#> [1] 12
```

Generate full hex grid

Generating full hexagonal grid contains main three steps:

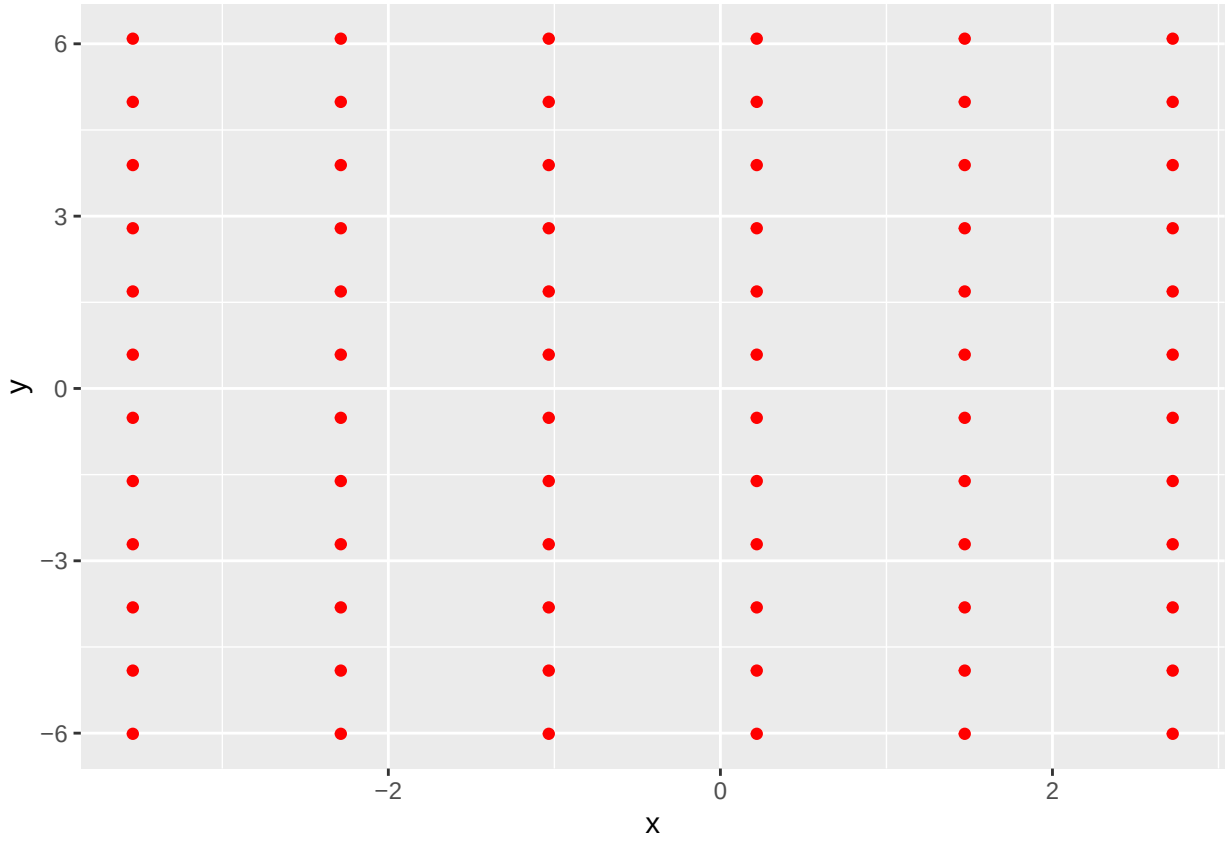
1. Generate all the hexagonal bin centroids

   Steps:

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

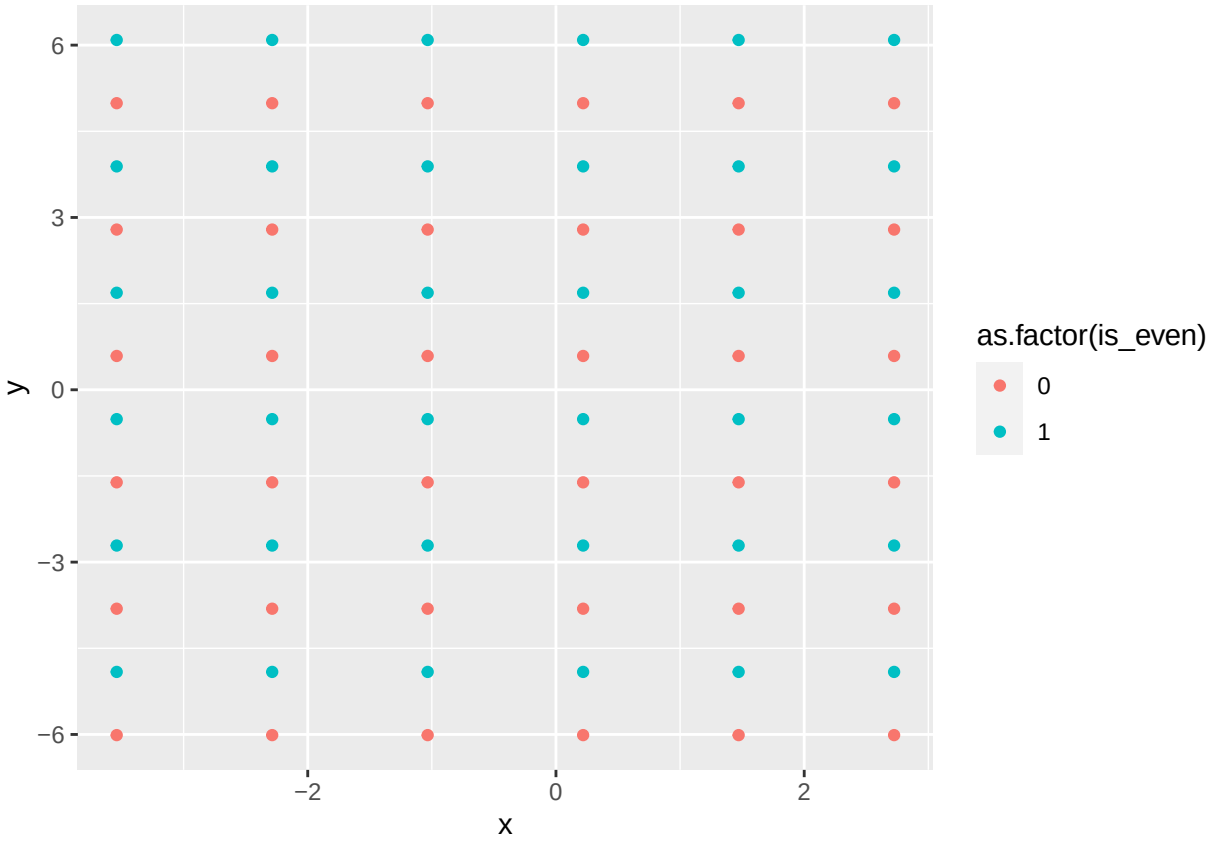
```
cell_area <- 1
cell_diameter <- sqrt(2 * cell_area / sqrt(3))

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")
```



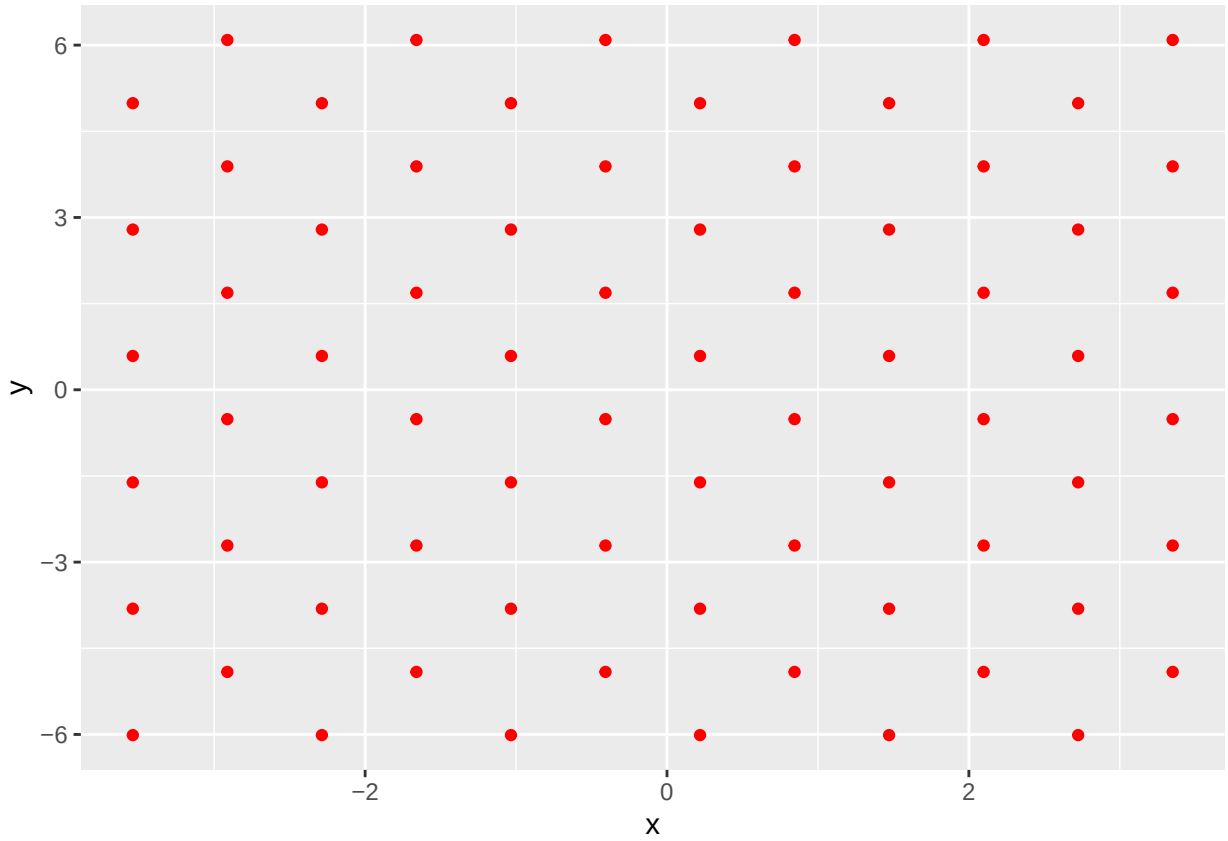
- Second for each x-value, find which y values are in the even row

```
 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")
```



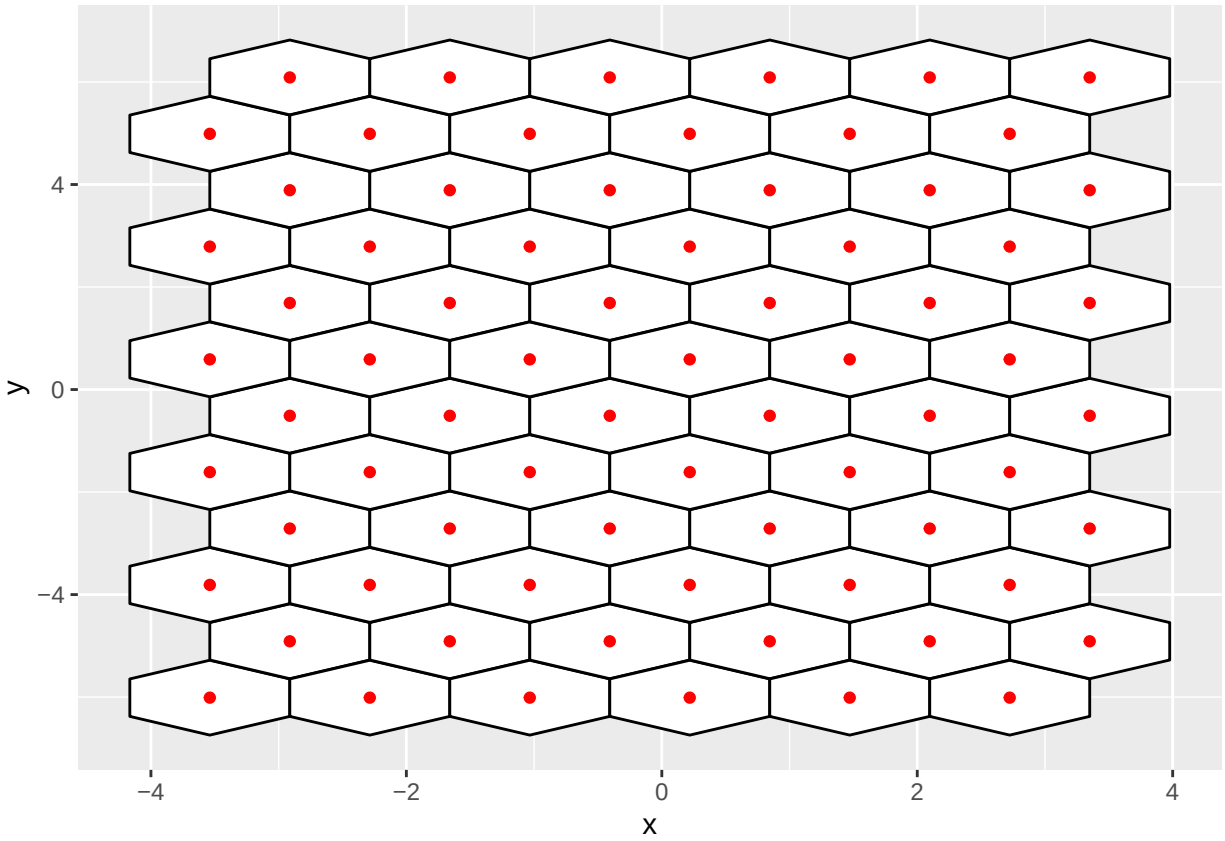

```
all_centroids_df <- generate_full_grid_centroids(nldr_df = s_curve_noise_umap,
                                                 x = "UMAP1", y = "UMAP2",
                                                 num_bins_x = num_bins_x,
                                                 num_bins_y = num_bins_y,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

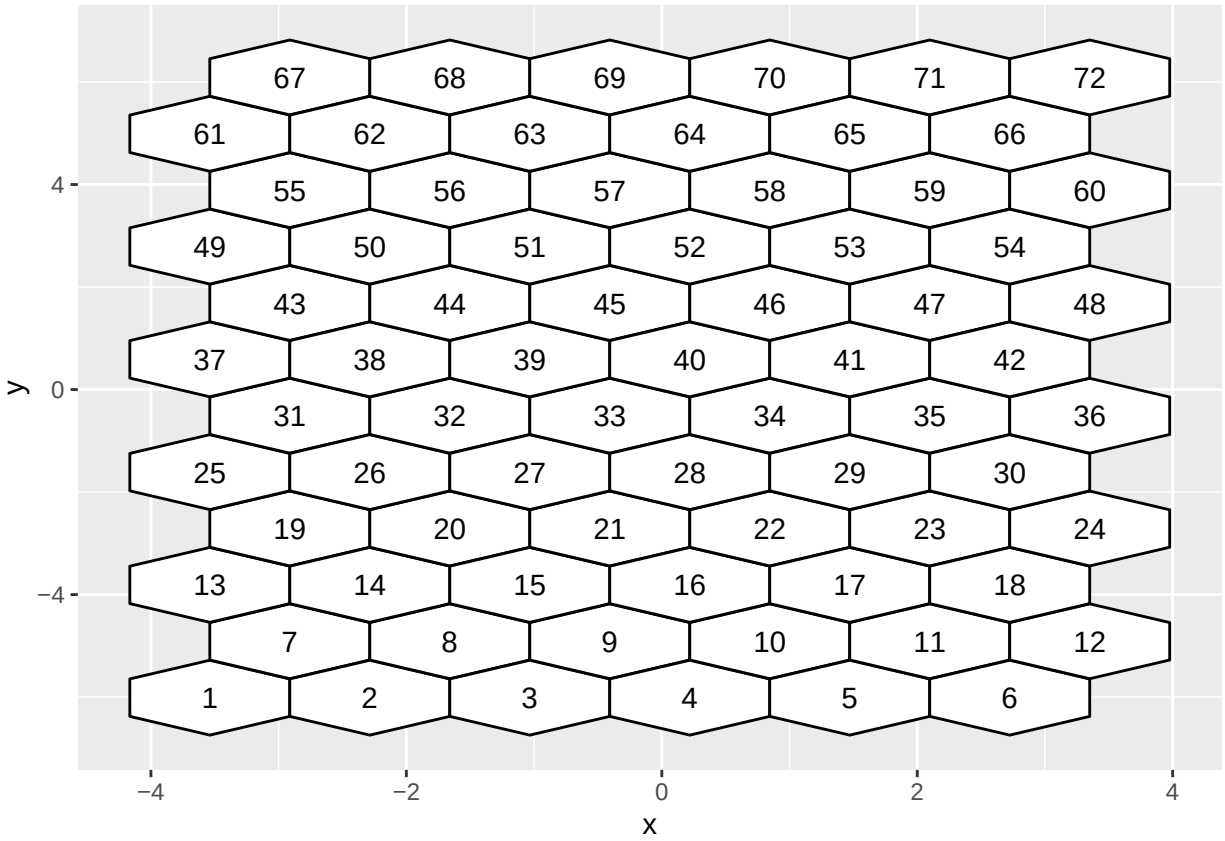


3. Map hexagonal IDs

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

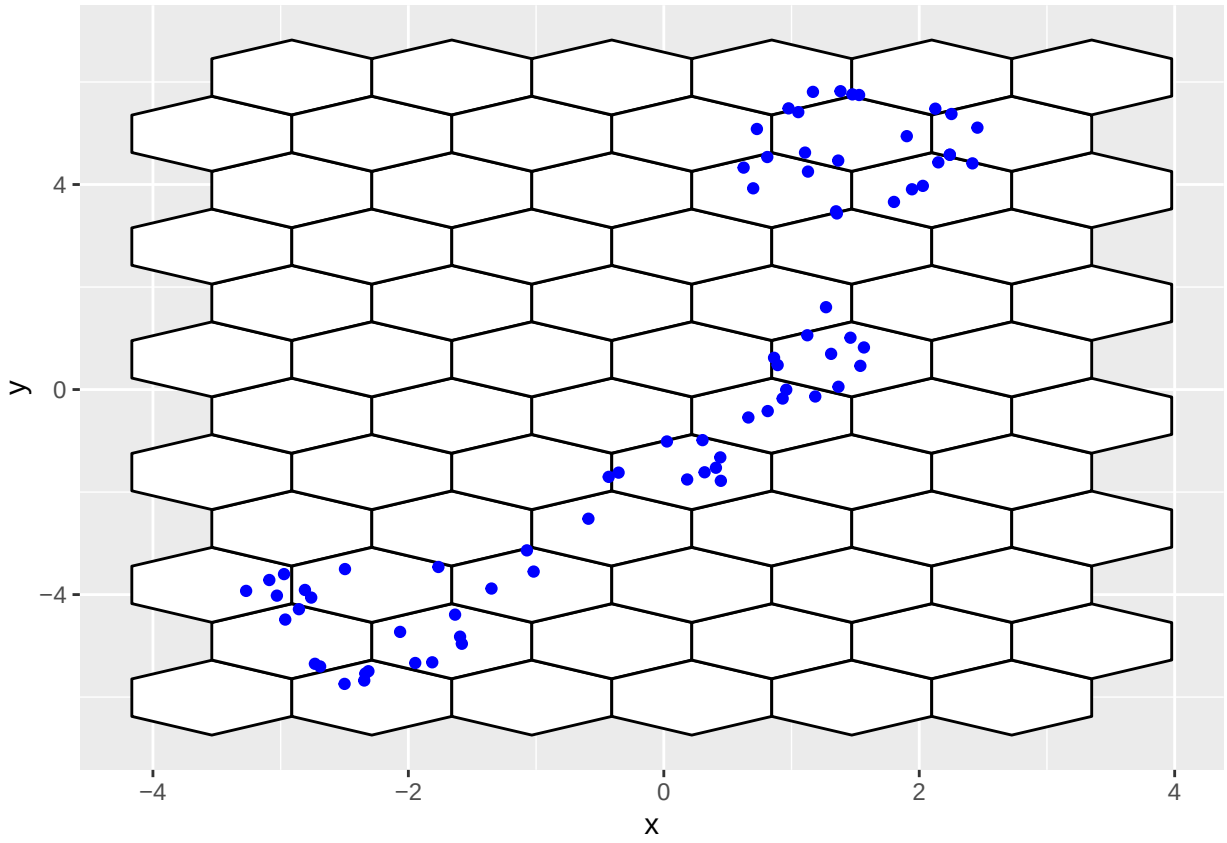
```
df_with_std_counts <- compute_std_counts(nldr_df = s_curve_noise_umap_with_id)
```

6. Extract full grid info

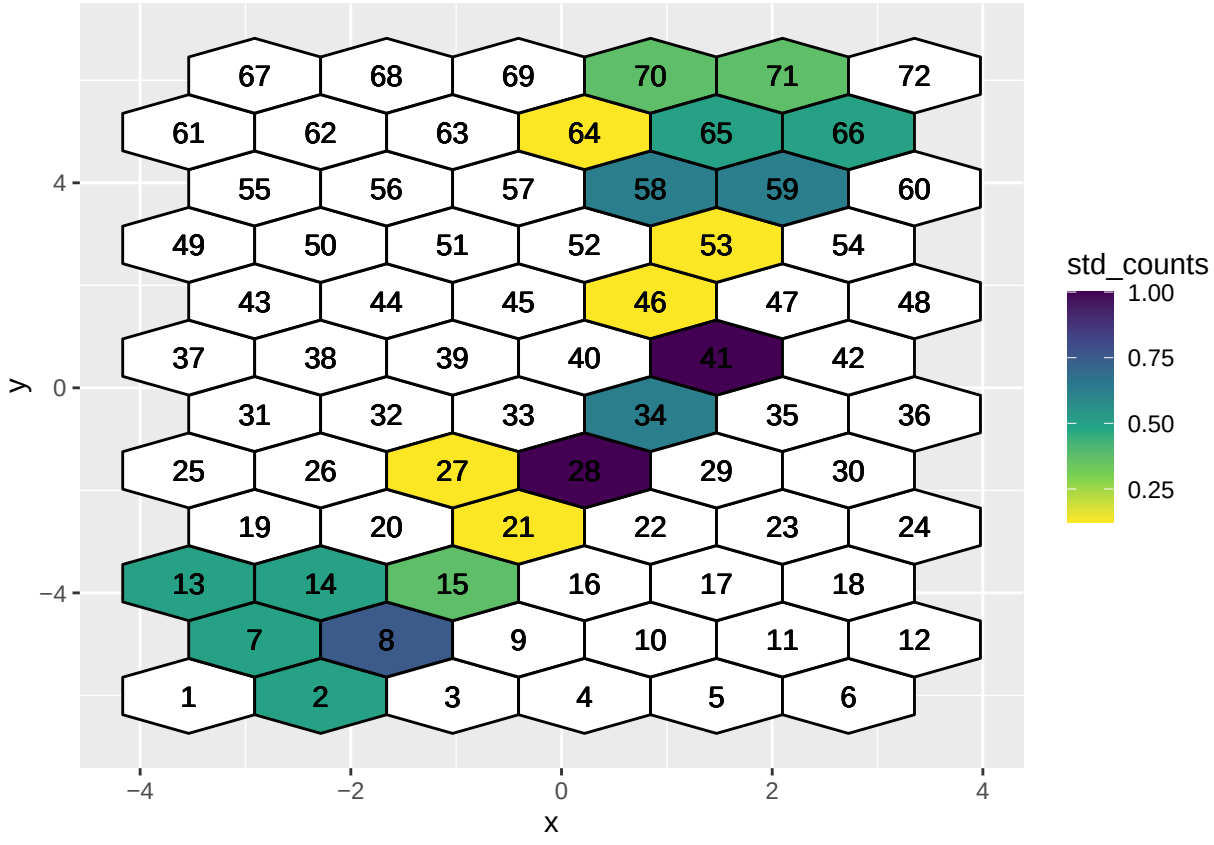
- Assign standardize counts for hex bins
- Join with the hexagonal coordinates

```
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
```

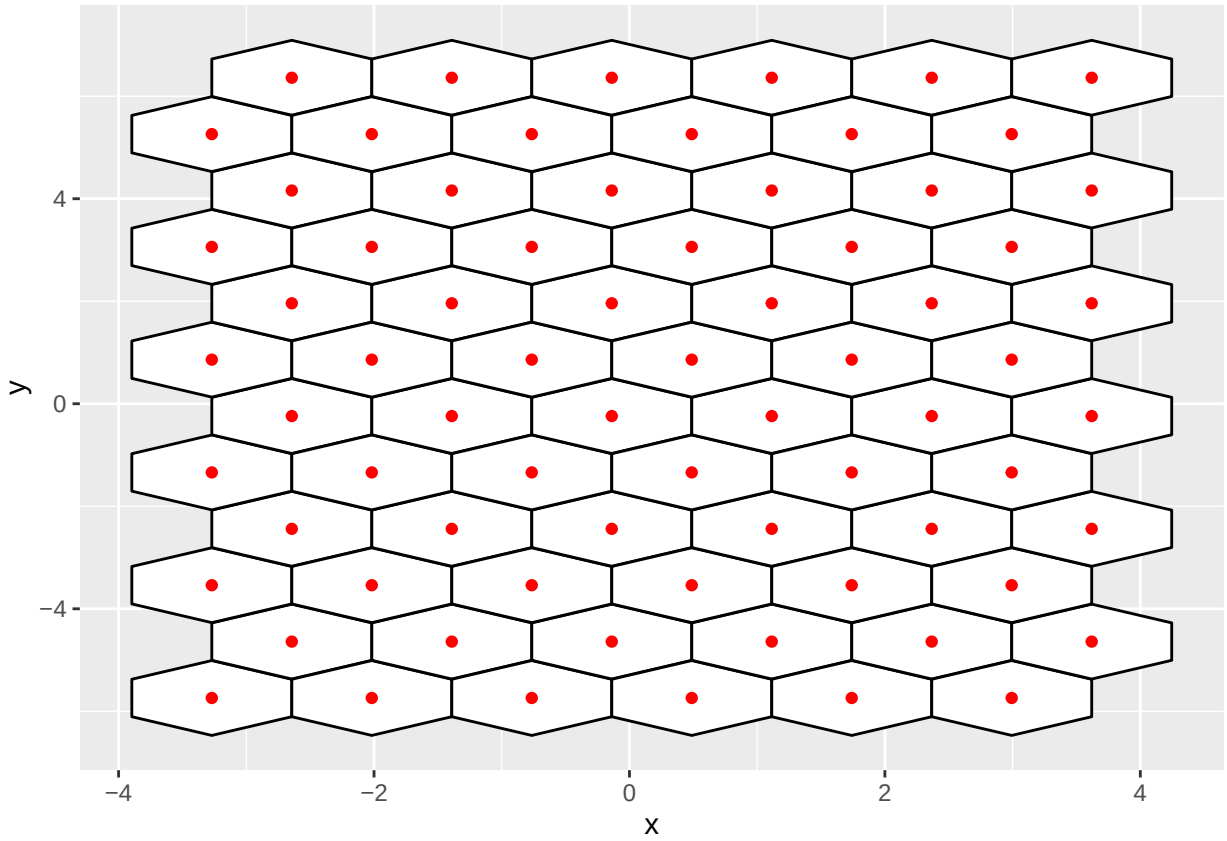
```
tr1_object <- triangulate_bin_centroids(df_bin_centroids, x, y)
tr_from_to_df <- generate_edge_info(triangular_object = tr1_object)
```

Compute parameter defaults

Shift the hexagonal grid origin

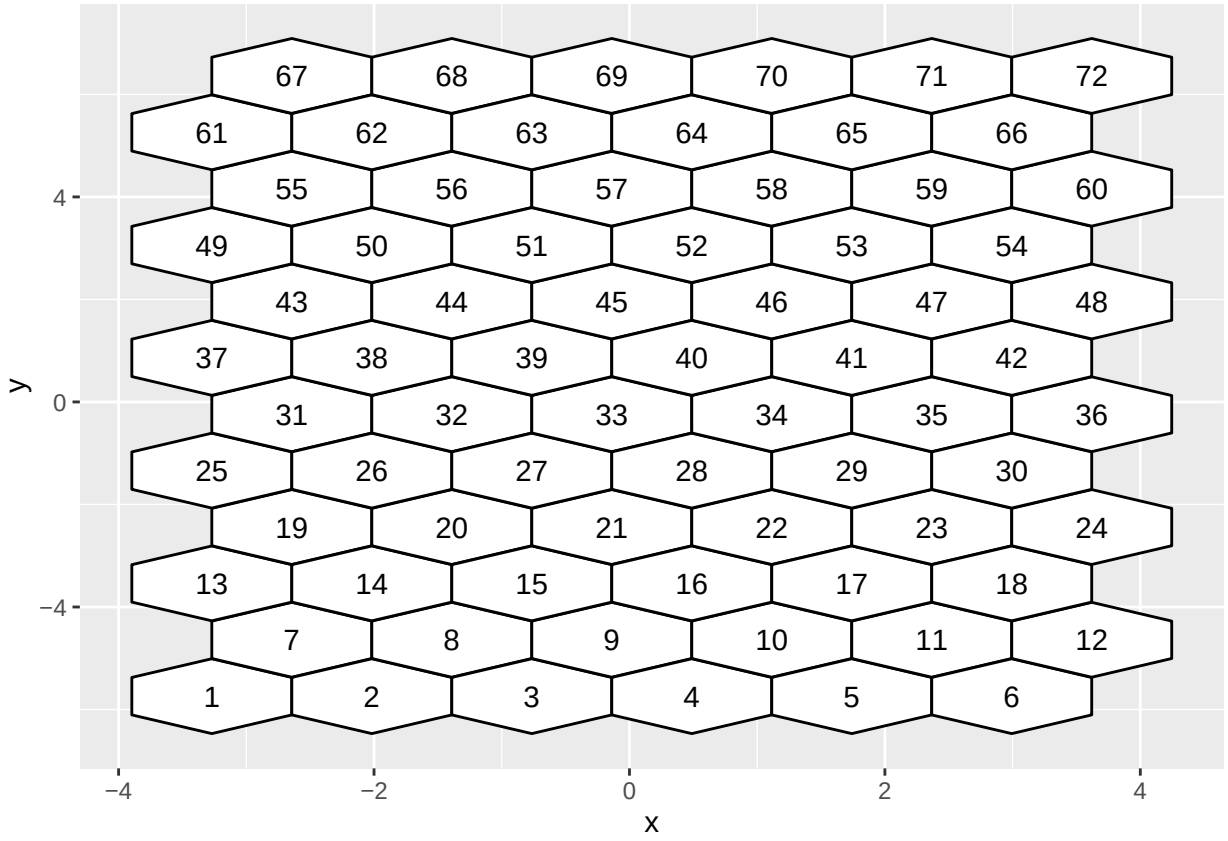
1. Assign shift along the x and y axis (limited the amount should less than the cell_diameter)
2. Generate bounds with shift origin

```
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")
```

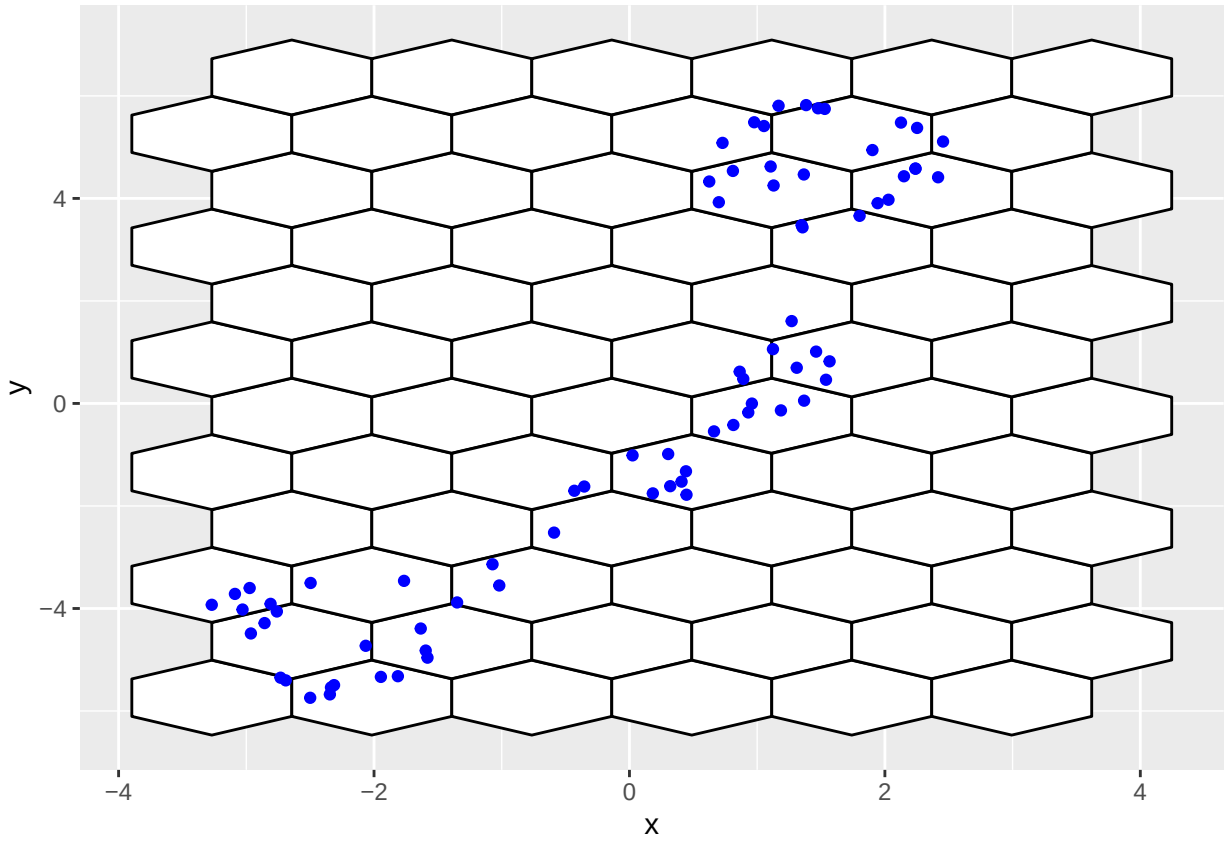



```
full_grid_with_hexbin_id <- map_hexbin_id(all_centroids_df_shift)
```

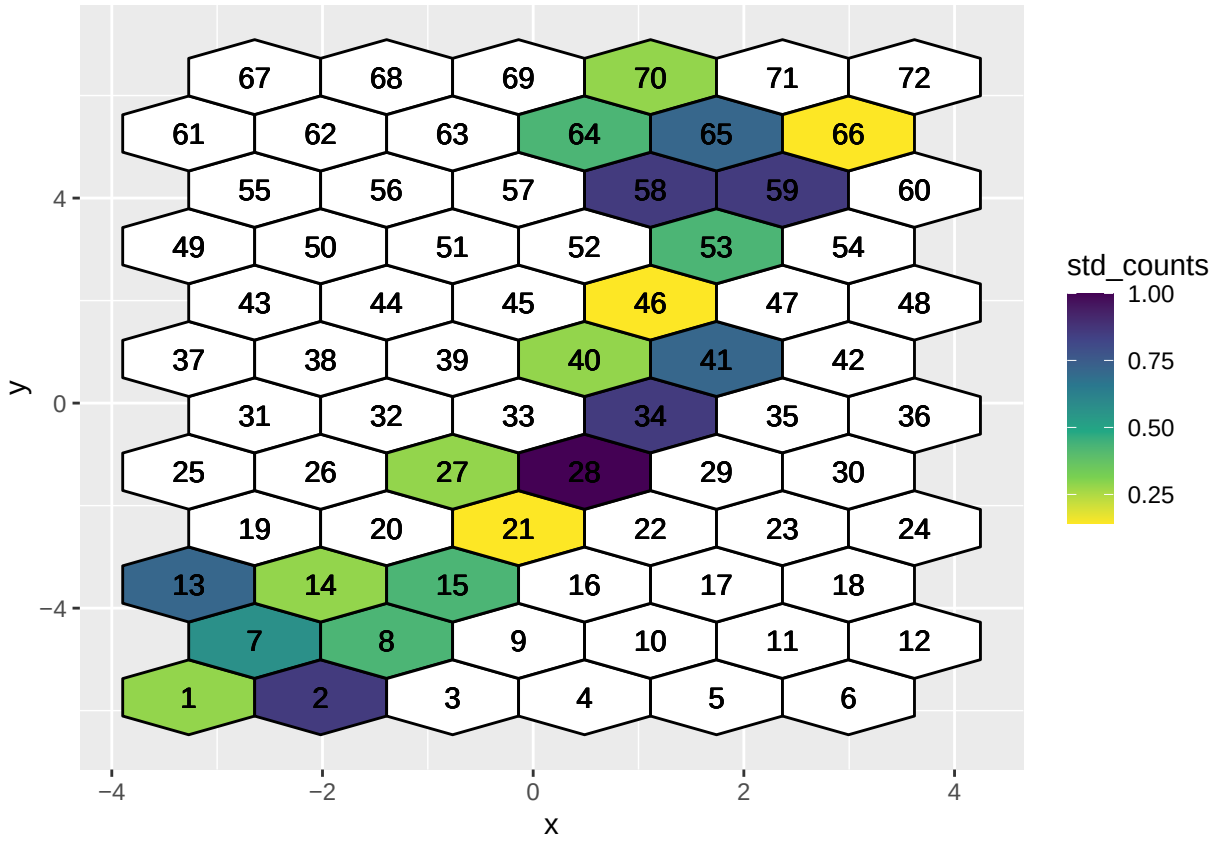
```
ggplot(data = hex_grid, aes(x = x, y = y)) + geom_polygon(fill = "white", color = "black", aes(group = id)) +
  geom_text(data = full_grid_with_hexbin_id, aes(x = c_x, y = c_y, label = hexID))
```




```
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"]]), ] |>
```

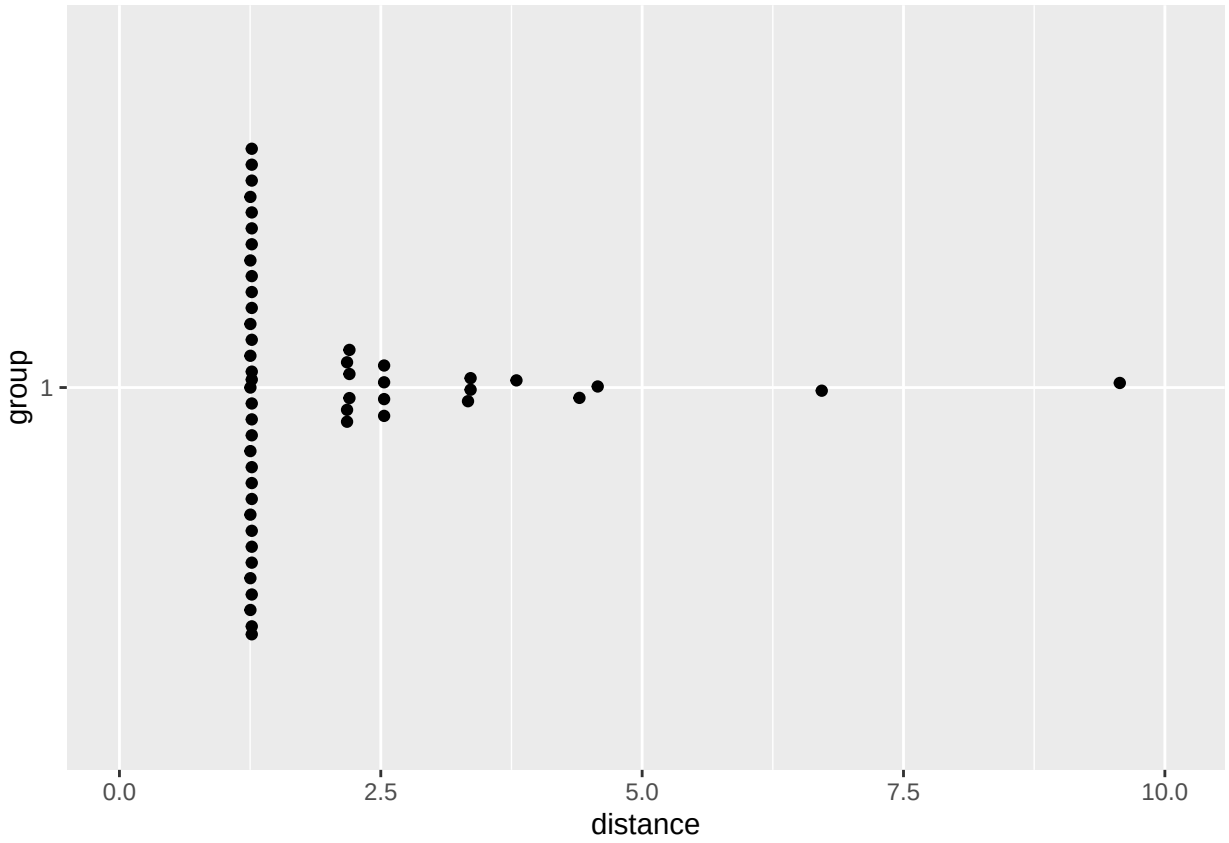
```
  dplyr::select("c_x", "c_y", "hexID", "std_counts") |>
  dplyr::distinct() |>
  dplyr::rename(c("x" = "c_x", "y" = "c_y"))

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

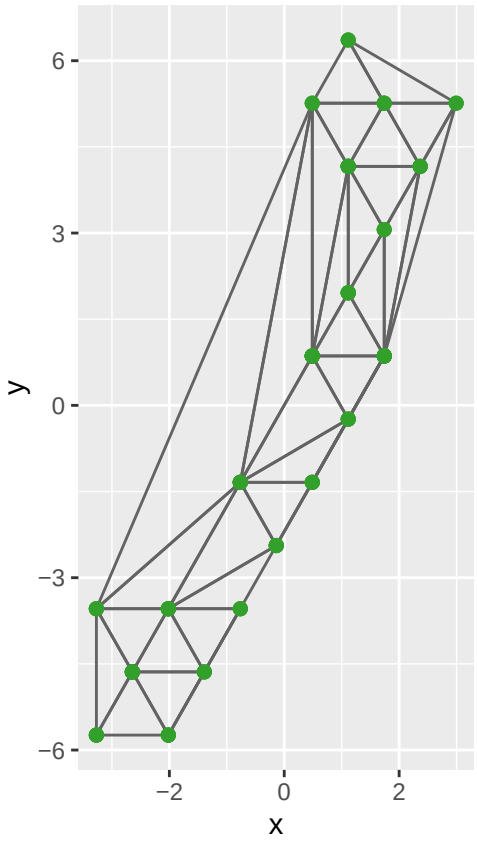
Predict 2D embeddings

Compute residuals

Visualizations

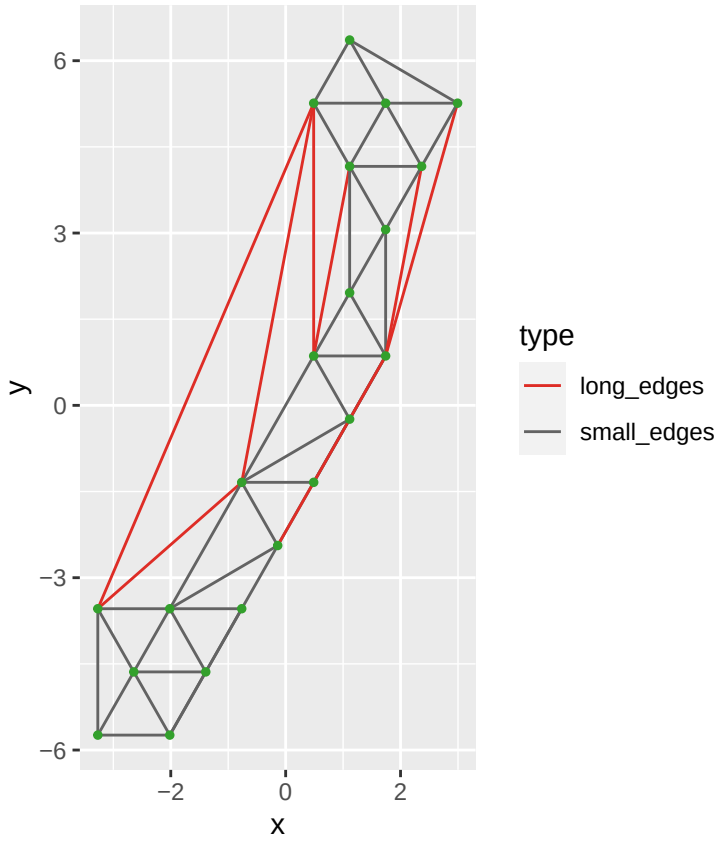
geom_trimesh

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

trimesh
```



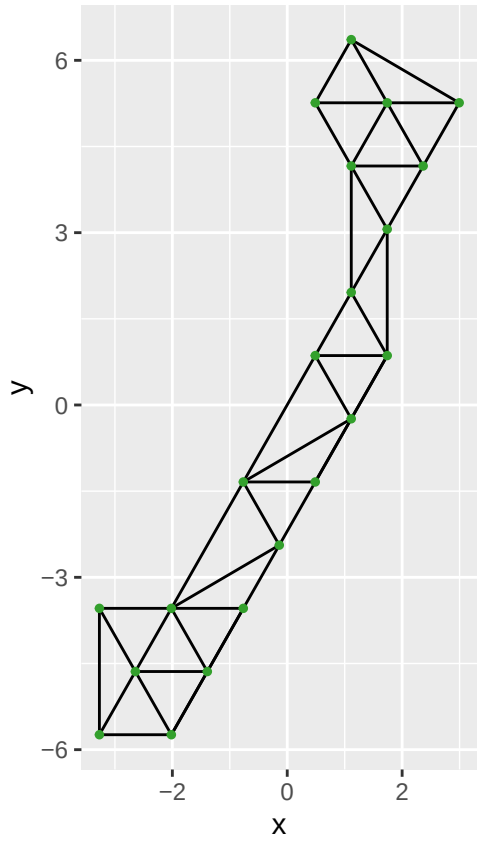
coloured_long_edges

```
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 Examples

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