

# quollr: An R Package for Visualizing 2-D Models from Non-linear Dimension Reductions in High Dimensional Space

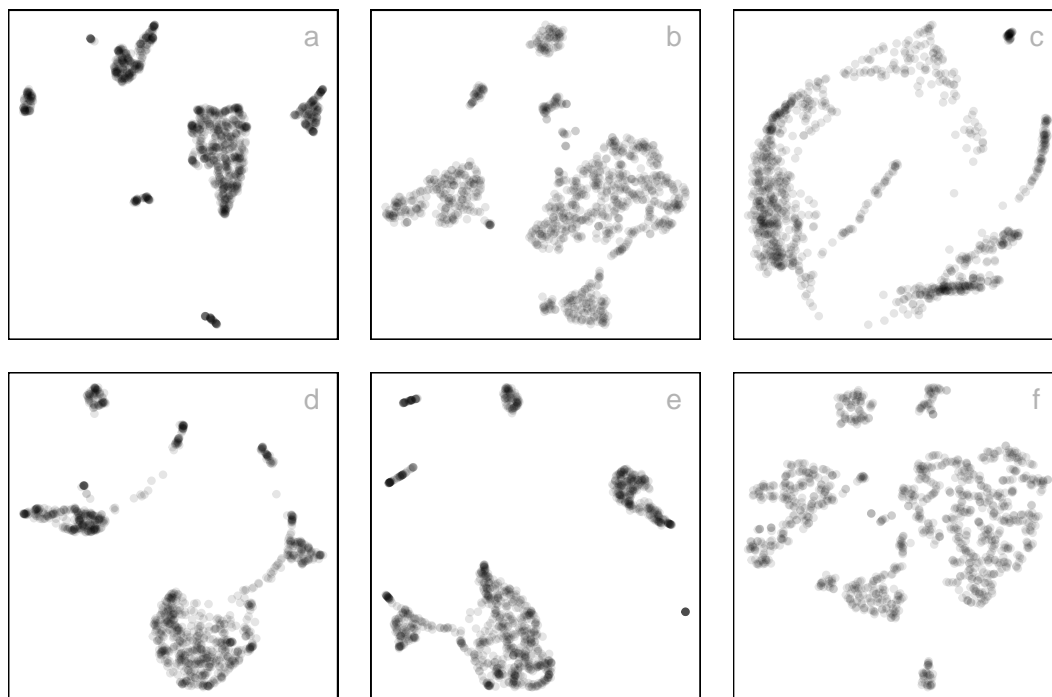
by Jayani P. Gamage, Dianne Cook, Paul Harrison, Michael Lydeamore, and Thiyanaga S. Talagala

**Abstract** Non-linear dimension reduction (NLDR) methods provide a low-dimensional representation of high-dimensional data (p-D) by applying a non-linear transformation. However, the complexity of the transformations and data structures can create wildly different representations depending on the method and (hyper)parameter choices. It is difficult to determine whether any of these representations are accurate, which one is the best, or whether they have missed important structures. The R package **quollr** has been developed as a new visual tool to determine which method and which (hyper)parameter choices provide the most accurate representation of high-dimensional data. The scurve data from the package is used to illustrate the algorithm. scRNA-seq data for Limb muscles of mice are used to discuss the usability of the package.

## 1 Introduction

This paper presents the R package, **quollr** which introduce a new visual tool in determining which Non-linear dimension reduction (NLDR) technique and which (hyper)parameter choice gives most accurate representation of high-dimensional data. The methodology of the algorithm is explained in *cite the methodology paper*. Furthermore, the **quollr** package enables users to perform hexagonal binning (Carr et al., 2023), resulting in the generation of regular hexagons. The software is available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=quollr>.

Add the image generated with the application



The paper is organized as follows. In next section, introduces the implementation of **quollr** package on CRAN, including demonstration of the package's key functions and visualization capabilities. We illustrate the algorithm's functionality to study about clustering data structure in **Application** section. Finally, we give a brief conclusion of the paper and discuss potential opportunities for use of our algorithm.

## 2 Implementation

### Installation

The package can be installed from CRAN:

```
install.packages("quollr")
```

and the development version can be installed from GitHub:

```
devtools::install_github("JayaniLakshika/quollr")
```

### Web site

More documentation of the package can be found at the web site <https://jayanilakshika.github.io/quollr/>.

### Package dependencies

Understanding the dependencies of the quollr package is essential for smooth operation and error prevention. The following dependencies refer to the other R packages that quollr relies on to execute its functions effectively.

```
#> $quollr
#> [1] "dplyr"      "ggplot2"    "grid"       "interp"     "langevitour"
#> [6] "proxy"     "rlang"      "rsample"    "stats"      "tibble"
#> [11] "tidyselect"
```

### Data sets

Add a table of data sets that quollr have

The following demonstration of the package's functionality assumes quollr has been loaded. We also want to load the built-in data sets `scurve` and `scurve_umap`.

`scurve` is a 7- $D$  simulated dataset. It is constructed by simulating 5000 observations from  $\theta \sim U(-3\pi/2, 3\pi/2)$ ,  $X_1 = \sin(\theta)$ ,  $X_2 \sim U(0, 2)$  (adding thickness to the S),  $X_3 = \text{sign}(\theta) \times (\cos(\theta) - 1)$ . The remaining variables  $X_4, X_5, X_6, X_7$  are all uniform error, with small variance. `scurve_umap` is the UMAP 2- $D$  embedding for `scurve` data with `n_neighbors` is 46 and `min_dist` is 0.9. Each data set contains a unique ID column that maps `scurve` and `scurve_umap`.

### Main function

The main steps for the algorithm can be executed by the main function `fit_highd_model()`, or can be run separately for more flexibility.

This function requires several parameters: the high-dimensional data (`highd_data`), the embedding data (`nldr_data`), the number of bins along the x-axis (`bin1`), the buffer amount as a proportion of data (`q`), and benchmark value to extract high density hexagons (`benchmark_highdens`). The function returns an object that includes the hexagonal object, the fitted model in both 2- $D$ , and  $p$ - $D$ , and triangular mesh.

```
fit_highd_model(
  highd_data = scurve,
  nldr_data = scurve_umap,
  bin1 = 15,
  q = 0.1,
  benchmark_highdens = 5)

#> $hb_obj
#> $a1
#> [1] 0.08326136
#>
```

```

#> $a2
#> [1] 0.07210645
#>
#> $bins
#> [1] 15 16
#>
#> $start_point
#> [1] -0.10000000 -0.08923607
#>
#> $centroids
#> # A tibble: 240 x 3
#>   hexID      c_x      c_y
#>   <int>    <dbl>    <dbl>
#> 1     1      -0.1    -0.0892
#> 2     2     -0.0167 -0.0892
#> 3     3     0.0665 -0.0892
#> 4     4     0.150  -0.0892
#> 5     5     0.233  -0.0892
#> 6     6     0.316  -0.0892
#> 7     7     0.400  -0.0892
#> 8     8     0.483  -0.0892
#> 9     9     0.566  -0.0892
#> 10    10     0.649  -0.0892
#> # i 230 more rows
#>
#> $hex_poly
#> # A tibble: 1,440 x 3
#>   hex_poly_id      x      y
#>   <int>    <dbl>    <dbl>
#> 1         1     -0.1    -0.0412
#> 2         1    -0.142   -0.0652
#> 3         1    -0.142   -0.113
#> 4         1     -0.1    -0.137
#> 5         1   -0.0584 -0.113
#> 6         1   -0.0584 -0.0652
#> 7         2    -0.0167 -0.0412
#> 8         2   -0.0584 -0.0652
#> 9         2   -0.0584 -0.113
#> 10        2    -0.0167 -0.137
#> # i 1,430 more rows
#>
#> $data_hb_id
#> # A tibble: 5,000 x 4
#>   emb1 emb2   ID hexID
#>   <dbl> <dbl> <int> <int>
#> 1 0.707 0.839     1   205
#> 2 0.231 0.401     2   109
#> 3 0.232 0.215     3    65
#> 4 0.790 0.564     4   146
#> 5 0.761 0.551     5   146
#> 6 0.445 0.721     6   172
#> 7 0.900 0.137     7    58
#> 8 0.247 0.392     8   110
#> 9 0.325 0.542     9   141
#> 10 0.278 0.231    10    80
#> # i 4,990 more rows
#>
#> $std_cts
#> # A tibble: 142 x 3
#>   hexID bin_counts std_counts
#>   <int>    <int>    <dbl>
#> 1     21        12     0.16
#> 2     22        17     0.227
#> 3     23        22     0.293

```

```

#> 4      24      10      0.133
#> 5      25       7      0.0933
#> 6      26      12      0.16
#> 7      27       1      0.0133
#> 8      36      42      0.56
#> 9      37      42      0.56
#> 10     38      44      0.587
#> # i 132 more rows
#>
#> $tot_bins
#> [1] 240
#>
#> $non_bins
#> [1] 142
#>
#> $pts_bins
#> # A tibble: 142 x 2
#>   hexID pts_list
#>   <int> <list>
#> 1     21 <int [12]>
#> 2     22 <int [17]>
#> 3     23 <int [22]>
#> 4     24 <int [10]>
#> 5     25 <int [7]>
#> 6     26 <int [12]>
#> 7     27 <int [1]>
#> 8     36 <int [42]>
#> 9     37 <int [42]>
#> 10    38 <int [44]>
#> # i 132 more rows
#>
#> attr(,"class")
#> [1] "hex_bin_obj"
#>
#> $model_highd
#> # A tibble: 130 x 8
#>   hexID      x1      x2      x3      x4      x5      x6      x7
#>   <int>   <dbl> <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#> 1     21 -0.992  1.91 1.11 -0.000427 0.000624 0.00749 0.00105
#> 2     22 -0.906  1.93 1.41 -0.0000183 0.00331 -0.0204 -0.000363
#> 3     23 -0.680  1.93 1.72 -0.000810 -0.00259 -0.00449 0.00153
#> 4     24 -0.272  1.93 1.96  0.00251  0.00668 -0.0460  0.00128
#> 5     25  0.0760  1.93 2.00  0.00876  0.00447  0.00851 -0.00195
#> 6     26  0.461  1.93 1.89 -0.00478  0.00492  0.00835  0.00172
#> 7     36 -0.985  1.75 0.853 -0.00202  0.000397 0.00331  0.000338
#> 8     37 -0.980  1.66 1.17 -0.000374 -0.00154  0.0165  0.000126
#> 9     38 -0.821  1.64 1.56 -0.000459  0.000538 -0.0123  0.000780
#> 10    39 -0.484  1.68 1.87  0.00313  0.00241  0.00823 -0.00117
#> # i 120 more rows
#>
#> $model_2d
#> # A tibble: 130 x 5
#>   hexID  c_x  c_y bin_counts std_counts
#>   <int> <dbl> <dbl>   <dbl>   <dbl>
#> 1     21 0.358 -0.0171      12     0.16
#> 2     22 0.441 -0.0171      17     0.227
#> 3     23 0.524 -0.0171      22     0.293
#> 4     24 0.608 -0.0171      10     0.133
#> 5     25 0.691 -0.0171       7     0.0933
#> 6     26 0.774 -0.0171      12     0.16
#> 7     36 0.316  0.0550      42     0.56
#> 8     37 0.400  0.0550      42     0.56
#> 9     38 0.483  0.0550      44     0.587
#> 10    39 0.566  0.0550      39     0.52

```

```
#> # i 120 more rows
#>
#> $trimesh_data
#> # A tibble: 322 x 8
#>   from   to x_from y_from x_to   y_to from_count to_count
#>   <int> <int> <dbl>   <dbl> <dbl> <dbl>       <dbl>   <dbl>
#> 1    16    26 0.275   0.127 0.233 0.199         65      67
#> 2    25    26 0.150   0.199 0.233 0.199         39      67
#> 3    25    37 0.150   0.199 0.191 0.271         39      62
#> 4    36    49 0.108   0.271 0.150 0.343         62      34
#> 5    36    37 0.108   0.271 0.191 0.271         62      62
#> 6     1     7 0.358  -0.0171 0.316 0.0550         12      42
#> 7    48    49 0.0665   0.343 0.150 0.343         45      34
#> 8    37    38 0.191   0.271 0.275 0.271         62      68
#> 9    26    27 0.233   0.199 0.316 0.199         67      56
#> 10   27    38 0.316   0.199 0.275 0.271         56      68
#> # i 312 more rows
```

## Constructing the 2-D Model

Constructing the 2-D model primarily involves (i) scaling the NLDR data, (ii) binning the data, (iii) obtaining bin centroids, (iv) connecting centroids with line segments to indicate neighbors, and (v) Remove low-density hexagons.

## Scaling the Data

The algorithm starts by scaling the NLDR data to a standard range using the `gen_scaled_data()` function. This function standardizes the data so that the first embedding ranges from 0 to 1, while the second embedding scales from 0 to the maximum value of the second embedding. The output includes the scaled NLDR data along with the original limits of the embeddings.

```
scurve_umap_obj <- gen_scaled_data(nldr_data = scurve_umap)
```

```
scurve_umap_obj
```

```
#> $scaled_nldr
#> # A tibble: 5,000 x 3
#>   emb1 emb2   ID
#>   <dbl> <dbl> <int>
#> 1 0.707 0.839     1
#> 2 0.231 0.401     2
#> 3 0.232 0.215     3
#> 4 0.790 0.564     4
#> 5 0.761 0.551     5
#> 6 0.445 0.721     6
#> 7 0.900 0.137     7
#> 8 0.247 0.392     8
#> 9 0.325 0.542     9
#> 10 0.278 0.231    10
#> # i 4,990 more rows
#>
#> $lim1
#> [1] -14.42166 13.32655
#>
#> $lim2
#> [1] -12.43687 12.32455
```

## Computing hexagon grid configurations

The configurations of a hexagonal grid are determined by the number of bins and the bin width in each direction. The function `calc_bins_y()` is used for this purpose. This function accepts an object containing scaled NLDR data in the first and second columns, along with numeric vectors that

represent the limits of the original NLDR data, the number of bins along the x-axis (bin1), and the buffer amount as a proportion.

```
bin_configs <- calc_bins_y(
  nldr_obj = scurve_umap_obj,
  bin1 = 15,
  q = 0.1)
```

```
bin_configs
```

```
#> $bin2
#> [1] 16
#>
#> $a1
#> [1] 0.08326136
#>
#> $a2
#> [1] 0.07210645
```

### Binning the data

Points are allocated to bins based on the nearest centroid. The hexagonal binning algorithm can be executed using the `hex_binning()` function, or its components can be run separately for added flexibility. The parameters used within `hex_binning()` include an object containing scaled NLDR data in the first and second columns, along with numeric vectors that represent the limits of the original NLDR data (`nldr_obj`), the number of bins along the x-axis (`bin1`), and the buffer amount as a proportion of the data (`q`). The output is an object of the `hex_bin_obj` class, which contains the bin widths in each direction (`a1`, `a2`), the number of bins in each direction (`bins`), the coordinates of the hexagonal grid starting point (`start_point`), the details of bin centroids (`centroids`), the coordinates of bins (`hex_poly`), NLDR components with their corresponding hexagon IDs (`data_hb_id`), hex bins with their corresponding standardized counts (`std_cts`), the total number of bins (`tot_bins`), the number of non-empty bins (`non_bins`), and the points within each hexagon (`pts_bins`).

```
hb_obj <- hex_binning(
  nldr_obj = scurve_umap_obj,
  bin1 = 15,
  q = 0.1)
```

```
all_centroids_df <- gen_centroids(
  nldr_obj = scurve_umap_obj,
  bin1 = 15,
  q = 0.1
)
```

```
all_centroids_df
```

```
#> # A tibble: 240 x 3
#>   hexID    c_x    c_y
#>   <int>  <dbl>  <dbl>
#> 1     1 -0.1    -0.0892
#> 2     2 -0.0167 -0.0892
#> 3     3  0.0665 -0.0892
#> 4     4  0.150  -0.0892
#> 5     5  0.233  -0.0892
#> 6     6  0.316  -0.0892
#> 7     7  0.400  -0.0892
#> 8     8  0.483  -0.0892
#> 9     9  0.566  -0.0892
#> 10    10  0.649  -0.0892
#> # i 230 more rows
```

```
all_hex_coord <- gen_hex_coord(
```

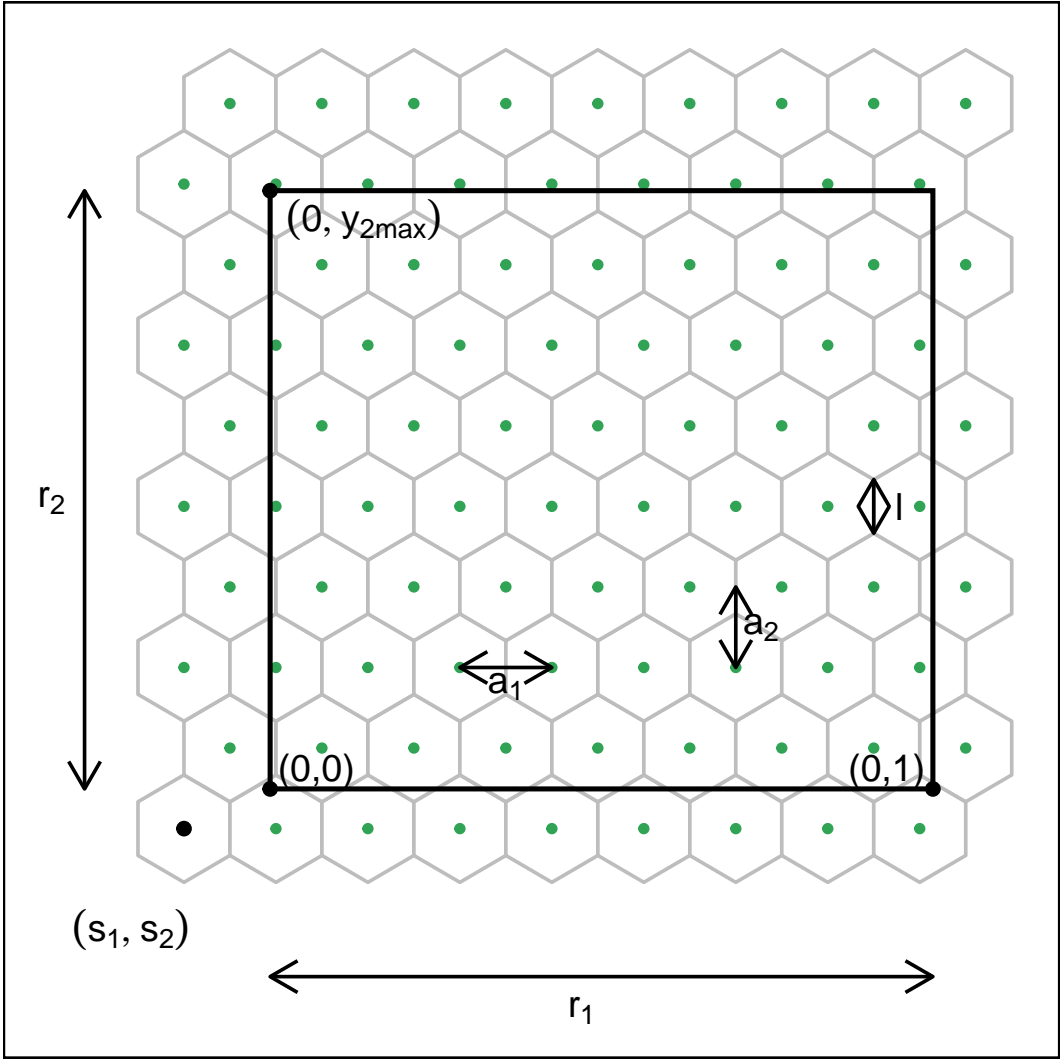


Figure 1: The components of the hexagon grid illustrating notation.

```

    centroids_data = all_centroids_df,
    a1 = bin_configs$a1
  )

all_hex_coord

#> # A tibble: 1,440 x 3
#>   hex_poly_id      x      y
#>   <int>    <dbl>  <dbl>
#> 1         1 -0.1   -0.0412
#> 2         1 -0.142 -0.0652
#> 3         1 -0.142 -0.113
#> 4         1 -0.1   -0.137
#> 5         1 -0.0584 -0.113
#> 6         1 -0.0584 -0.0652
#> 7         2 -0.0167 -0.0412
#> 8         2 -0.0584 -0.0652
#> 9         2 -0.0584 -0.113
#> 10        2 -0.0167 -0.137
#> # i 1,430 more rows

umap_hex_id <- assign_data(
  nldr_obj = scurve_umap_obj,
  centroids_data = all_centroids_df
)

umap_hex_id

#> # A tibble: 5,000 x 4
#>   emb1 emb2   ID hexID
#>   <dbl> <dbl> <int> <int>
#> 1 0.707 0.839     1   205
#> 2 0.231 0.401     2   109
#> 3 0.232 0.215     3    65
#> 4 0.790 0.564     4   146
#> 5 0.761 0.551     5   146
#> 6 0.445 0.721     6   172
#> 7 0.900 0.137     7    58
#> 8 0.247 0.392     8   110
#> 9 0.325 0.542     9   141
#> 10 0.278 0.231    10    80
#> # i 4,990 more rows

std_df <- compute_std_counts(
  scaled_nldr_hexid = umap_hex_id
)

std_df

#> # A tibble: 142 x 3
#>   hexID bin_counts std_counts
#>   <int>    <int>    <dbl>
#> 1     21        12     0.16
#> 2     22        17     0.227
#> 3     23        22     0.293
#> 4     24        10     0.133
#> 5     25         7     0.0933
#> 6     26        12     0.16
#> 7     27         1     0.0133
#> 8     36        42     0.56
#> 9     37        42     0.56
#> 10    38        44     0.587
#> # i 132 more rows

```



```
pts_df <- find_pts(
  scaled_nldr_hexid = umap_hex_id
)
```

```
pts_df
```

```
#> # A tibble: 142 x 2
#>   hexID pts_list
#>   <int> <list>
#> 1     21 <int [12]>
#> 2     22 <int [17]>
#> 3     23 <int [22]>
#> 4     24 <int [10]>
#> 5     25 <int [7]>
#> 6     26 <int [12]>
#> 7     27 <int [1]>
#> 8     36 <int [42]>
#> 9     37 <int [42]>
#> 10    38 <int [44]>
#> # i 132 more rows
```

### Obtaining bin centroids

```
df_bin_centroids <- extract_hexbin_centroids(
  centroids_data = all_centroids_df,
  counts_data = hb_obj$std_cts
)
```

```
df_bin_centroids
```

```
#> # A tibble: 240 x 5
#>   hexID    c_x    c_y bin_counts std_counts
#>   <int> <dbl> <dbl>      <dbl>      <dbl>
#> 1     1 -0.1  -0.0892         0         0
#> 2     2 -0.0167 -0.0892         0         0
#> 3     3  0.0665 -0.0892         0         0
#> 4     4  0.150  -0.0892         0         0
#> 5     5  0.233  -0.0892         0         0
#> 6     6  0.316  -0.0892         0         0
#> 7     7  0.400  -0.0892         0         0
#> 8     8  0.483  -0.0892         0         0
#> 9     9  0.566  -0.0892         0         0
#> 10    10  0.649  -0.0892         0         0
#> # i 230 more rows
```

### Indicating neighbors by line segments connecting centroids

To indicate neighbors, the `tri_bin_centroids()` function is used to triangulate bin centroids. Following this, `gen_edges()` function computes the line segments that connect neighboring bins by providing the triangulated data. This results the coordinates that generate the connecting lines.

```
tr_object <- tri_bin_centroids(
  centroids_data = df_bin_centroids
)
```

```
trimesh <- gen_edges(tri_object = tr_object, a1 = hb_obj$a1)
trimesh
```

```
#> # A tibble: 653 x 8
#>   from to x_from y_from x_to y_to from_count to_count
#>   <int> <int> <dbl> <dbl> <dbl> <dbl>      <dbl>      <dbl>
#> 1     1     2 -0.1  -0.0892 -0.0167 -0.0892         0         0
```

```
#> 2 16 17 -0.0584 -0.0171 0.0249 -0.0171 0 0
#> 3 16 32 -0.0584 -0.0171 -0.0167 0.0550 0 0
#> 4 3 17 0.0665 -0.0892 0.0249 -0.0171 0 0
#> 5 17 18 0.0249 -0.0171 0.108 -0.0171 0 0
#> 6 17 33 0.0249 -0.0171 0.0665 0.0550 0 0
#> 7 31 46 -0.1 0.0550 -0.0584 0.127 0 0
#> 8 32 47 -0.0167 0.0550 0.0249 0.127 0 0
#> 9 32 33 -0.0167 0.0550 0.0665 0.0550 0 0
#> 10 4 18 0.150 -0.0892 0.108 -0.0171 0 0
#> # i 643 more rows
```

In some cases, distant centroids may be connected, resulting in long line segments that can affect the smoothness of the 2-*D* representation. To address this issue, the `find_lg_benchmark()` function is used. This function computes a threshold based on the distances of line segments, determining when long edges should be removed.

### Remove low-density hexagons

In certain scenarios, hexagonal bins may contain a few number of points. To ensure comprehensive coverage of NLDR data, it is important to select hexagonal bins with a suitable number of data points. The `find_low_dens_hex()` function identifies hexagons with low point densities, considering the densities of their neighboring bins as well. Users can initially identify low-density hexagons and then use this function to evaluate how removing them might affect the model fit by examining their neighbors.

```
find_low_dens_hex(
  centroids_data = df_bin_centroids,
  bin1 = 15,
  benchmark_mean_dens = 0.05
)

#> [1] 1 2 3 4 5 11 12 13 14 15 16 17 18 19 30 31 32 33 46
#> [20] 47 61 75 76 90 105 120 135 150 151 166 181 182 196 197 198 211 212 213
#> [39] 214 215 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
```

### Lifting the model into high dimensions

The final step involves lifting the fitted 2-*D* model into *p*-*D* by computing the *p*-*D* mean of data points within each hexagonal bin to represent bin centroids. This transformation is performed using the `avg_highd_data()` function, which takes *p*-*D* data (`highd_data`) and embedding data with their corresponding hexagonal bin IDs as inputs (`scaled_nldr_hexid`).

```
df_bin <- avg_highd_data(
  highd_data = scurve,
  scaled_nldr_hexid = hb_obj$data_hb_id
)

df_bin

#> # A tibble: 142 x 8
#>   hexID    x1    x2    x3    x4    x5    x6    x7
#>   <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 21 -0.992 1.91 1.11 -0.000427 0.000624 0.00749 0.00105
#> 2 22 -0.906 1.93 1.41 -0.0000183 0.00331 -0.0204 -0.000363
#> 3 23 -0.680 1.93 1.72 -0.000810 -0.00259 -0.00449 0.00153
#> 4 24 -0.272 1.93 1.96 0.00251 0.00668 -0.0460 0.00128
#> 5 25 0.0760 1.93 2.00 0.00876 0.00447 0.00851 -0.00195
#> 6 26 0.461 1.93 1.89 -0.00478 0.00492 0.00835 0.00172
#> 7 27 0.719 1.99 1.70 0.0109 -0.00349 -0.0297 -0.00223
#> 8 36 -0.985 1.75 0.853 -0.00202 0.000397 0.00331 0.000338
#> 9 37 -0.980 1.66 1.17 -0.000374 -0.00154 0.0165 0.000126
#> 10 38 -0.821 1.64 1.56 -0.000459 0.000538 -0.0123 0.000780
#> # i 132 more rows
```

## Prediction

The `predict_emb()` function is used to predict 2-*D* embedding for a new *p-D* data point using the fitted model. This function is useful to predict 2-*D* embedding irrespective of the NLDR technique.

In the prediction process, first, the nearest *p-D* model point is identified for a given new *p-D* data point by computing *p-D* Euclidean distance. Then, the corresponding 2-*D* bin centroid mapping for the identified *p-D* model point is determined. Finally, the coordinates of the identified 2-*D* bin centroid is used as the predicted NLDR embedding for the new *p-D* data point.

```
predict_emb(
  highd_data = scurve,
  model_2d = df_bin_centroids,
  model_highd = df_bin
)
```

```
#> # A tibble: 5,000 x 4
#>   pred_emb_1 pred_emb_2 ID pred_hb_id
#>   <dbl>      <dbl> <int>    <int>
#> 1      0.691      0.848     1      205
#> 2      0.191      0.416     2      109
#> 3      0.316      0.199     3       66
#> 4      0.774      0.560     4      146
#> 5      0.774      0.560     5      146
#> 6      0.441      0.704     6      172
#> 7      0.941      0.127     7       58
#> 8      0.275      0.416     8      110
#> 9      0.358      0.560     9      141
#> 10     0.316      0.343    10       96
#> # i 4,990 more rows
```

## Compute residuals and Mean Square Error (MSE)

As a Goodness of fit statistics for the model, `glance()` is used to compute residuals and MSE. These metrics are used to assess how well the fitted model will capture the underlying structure of the *p-D* data.

```
glance(
  highd_data = scurve,
  model_2d = df_bin_centroids,
  model_highd = df_bin
)
```

```
#> # A tibble: 1 x 2
#>   Error    MSE
#>   <dbl> <dbl>
#> 1 1481.  0.0325
```

Furthermore, `augment()` accepts 2-*D* and *p-D* model points, and the *p-D* data and adds information about each observation in the data set. Most commonly, this includes predicted values, residuals, row wise total error, absolute error for the fitted values, and row wise total absolute error.

Users can pass data to `augment()` via either the `training_data` argument or the `newdata` argument. If data is passed to the `training_data` argument, it must be exactly the data that was used to fit the model. Alternatively, datasets can be passed to `newdata` to augment data that was not used during model fitting. This requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no corresponding column will be included in the output.

The augmented dataset is always returned as a `tibble::tibble` with the same number of rows as the passed dataset.

```
model_error <- augment(
  highd_data = scurve,
  model_2d = df_bin_centroids,
```

```

    model_highd = df_bin
  )

model_error

#> # A tibble: 5,000 x 32
#>       ID      x1      x2      x3      x4      x5      x6      x7 pred_hb_id
#>   <int>   <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <int>
#> 1     1 -0.120  1.64  -1.99    0.0104  0.0125  0.0923 -0.00128    205
#> 2     2 -0.0492 1.51   0.00121 -0.0177  0.00726 -0.0362 -0.00535    109
#> 3     3 -0.774  1.30   0.367  -0.00173  0.0156 -0.0962  0.00335     66
#> 4     4 -0.606  0.246 -1.80   -0.00897 -0.0187 -0.0716  0.00126    146
#> 5     5 -0.478  0.0177 -1.88    0.00848  0.00533  0.0998  0.000677    146
#> 6     6  0.818  0.927 -1.58   -0.00318 -0.00980  0.0989  0.00696    172
#> 7     7  0.910  1.40   1.42    0.00699 -0.0182 -0.0710  0.00966     58
#> 8     8 -0.0691 1.59   0.00239  0.0127 -0.0130  0.0396 -0.000185    110
#> 9     9  0.859  1.59  -0.488  -0.0119  0.00421 -0.00440 -0.00595    141
#> 10    10 -0.727  1.62   0.314   0.00251  0.0177 -0.0755 -0.00369     96
#> # i 4,990 more rows
#> # i 23 more variables: model_high_d_x1 <dbl>, model_high_d_x2 <dbl>,
#> #   model_high_d_x3 <dbl>, model_high_d_x4 <dbl>, model_high_d_x5 <dbl>,
#> #   model_high_d_x6 <dbl>, model_high_d_x7 <dbl>, error_square_x1 <dbl>,
#> #   error_square_x2 <dbl>, error_square_x3 <dbl>, error_square_x4 <dbl>,
#> #   error_square_x5 <dbl>, error_square_x6 <dbl>, error_square_x7 <dbl>,
#> #   row_wise_total_error <dbl>, abs_error_x1 <dbl>, abs_error_x2 <dbl>, ...

```

## Visualizations

The package provides five basic visualizations which includes one to visualize the full hexagonal grid in 2-D, three visualizations related to the 2-D model (static visualizations), and one related to the  $p$ -D model (dynamic visualization). Each visualization can be generated using its respective function, as described in this section.

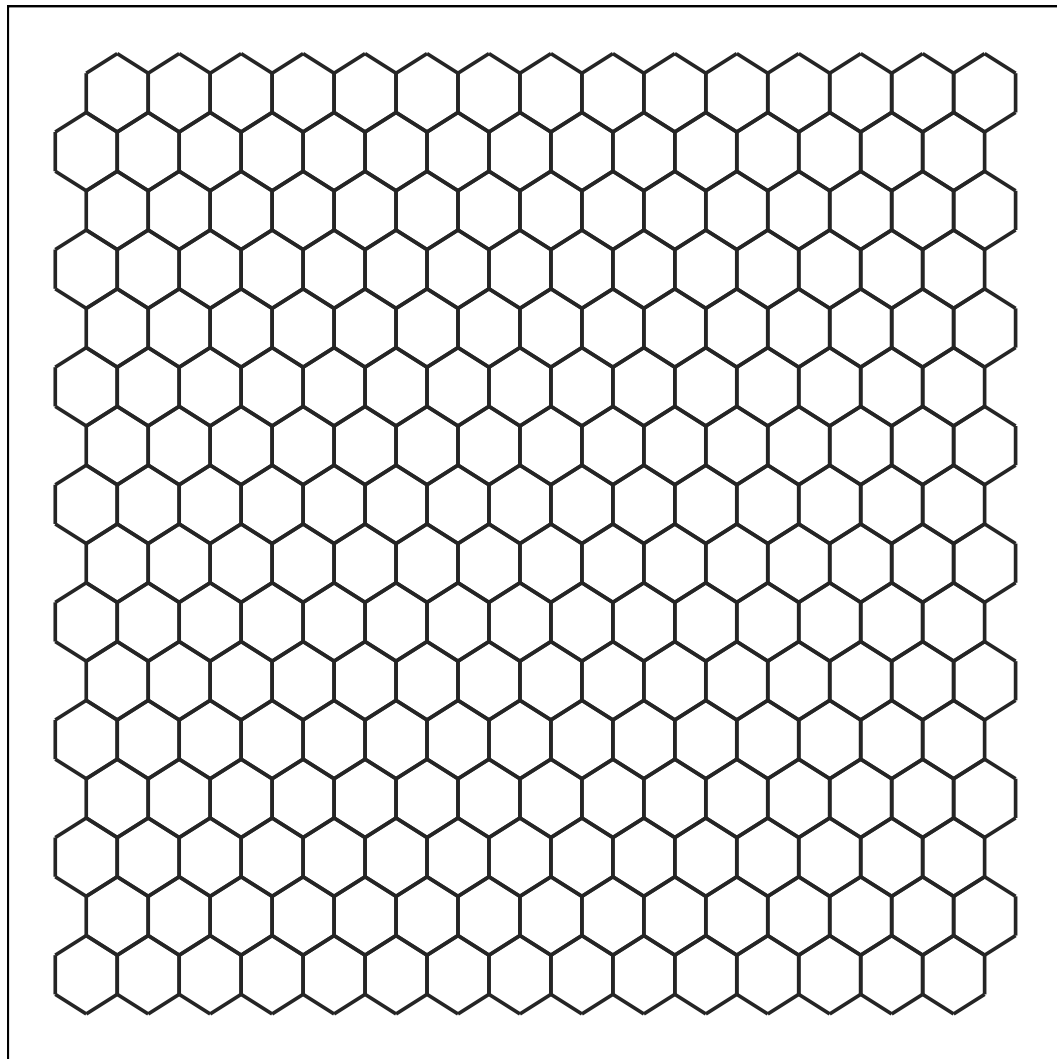
### 2-D model visualization

The `geom_hexgrid()` function is used to plot the hexagonal grid from the provided centroid data set.

```

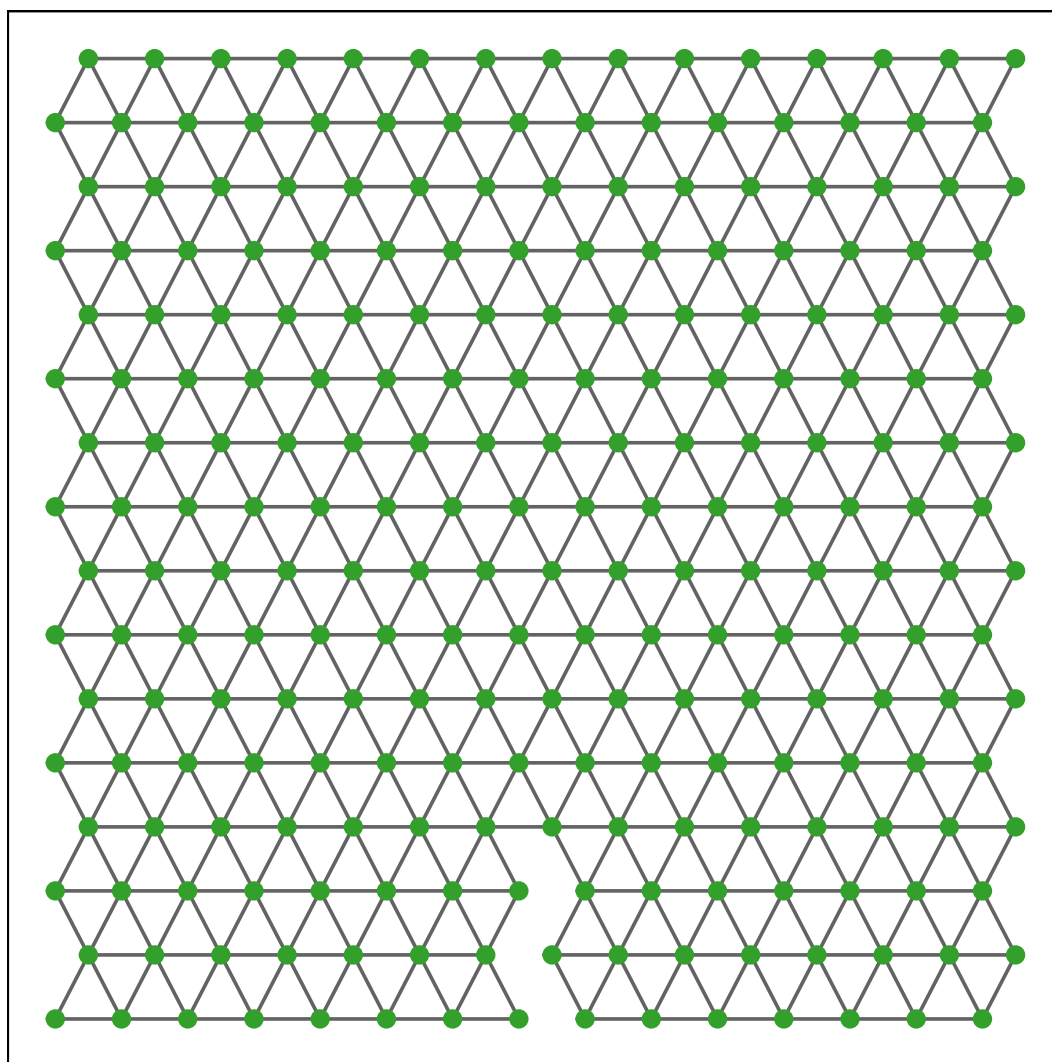
ggplot() +
  geom_hexgrid(
    data = hb_obj$centroids,
    aes(x = c_x, y = c_y)
  )

```



To visualize the 2- $D$  model, mainly three functions are used. As shown in Figure ??a, `geom_trimesh()` to visualize the triangular mesh by adding a new layer to `ggplot()`. After identifying benchmark value to remove long edge, `vis_lg_mesh()` is used to visualize the triangular mesh by coloring the small and long edges. As shown in Figure ??b, the small and long edges are colored by black and red respectively. Following this, `vis_rmlg_mesh()` is used to visualize smoothed 2- $D$  model which is the 2- $D$  model after removing the long edges (see Figure ??c). In `vis_lg_mesh()` and `vis_rmlg_mesh()`, `benchmark_value` argument controls the edge removal in 2- $D$ . Using small value of `benchmark_value`, will produce a triangular mesh with missing data structure; for example `benchmark_value = 0.3` shows two clusters rather than continuous structure, while `benchmark_value = 0.8` creates long edges and mislead the data structure in  $p$ - $D$  space.

```
ggplot() +
  geom_trimesh(
    data = hb_obj$centroids,
    aes(x = c_x, y = c_y)
  )
```



### *p*-D model visualization

Displaying the *p*-D model overlaid on the data is done using the function `show_langevi_tour()`. This visualization is helpful for visually evaluating how well the model fits the data. The function requires several arguments: data along with their corresponding hexagonal bin ID, 2-D and *p*-D model points, the threshold for removing long edges, and the distance data set.

```
df_bin_centroids <- df_bin_centroids |>
  dplyr::filter(bin_counts > 10)
```

```
df_exe <- comb_data_model(
  highd_data = scurve,
  model_highd = df_bin,
  model_2d = df_bin_centroids
)
```

```
df_exe
```

```
#> # A tibble: 5,122 x 8
#>       x1    x2    x3      x4      x5      x6      x7 type
#>   <dbl> <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl> <chr>
#> 1 -0.992  1.91  1.11 -0.000427  0.000624  0.00749  0.00105 model
#> 2 -0.906  1.93  1.41 -0.0000183  0.00331 -0.0204 -0.000363 model
#> 3 -0.680  1.93  1.72 -0.000810 -0.00259 -0.00449  0.00153 model
#> 4  0.461  1.93  1.89 -0.00478  0.00492  0.00835  0.00172 model
#> 5 -0.985  1.75  0.853 -0.00202  0.000397  0.00331  0.000338 model
```

```
#> 6 -0.980 1.66 1.17 -0.000374 -0.00154 0.0165 0.000126 model
#> 7 -0.821 1.64 1.56 -0.000459 0.000538 -0.0123 0.000780 model
#> 8 -0.484 1.68 1.87 0.00313 0.00241 0.00823 -0.00117 model
#> 9 -0.0991 1.70 1.99 0.00103 0.00150 0.00877 -0.000193 model
#> 10 0.295 1.74 1.95 -0.00165 0.000459 0.00330 0.000257 model
#> # i 5,112 more rows
```

```
trimesh <- trimesh |>
  dplyr::filter(from_count > 10,
                to_count > 10)

trimesh <- update_trimesh_index(trimesh)
```

```
show_langevitour(
  point_data = df_exe,
  edge_data = trimesh
)
```

## Link plots

There are mainly two interactive link plots can be generated. `show_link_plots()` helps to examine the fit. The function requires several arguments: points data which contain Non-linear dimension reduction data, high-dimensional data, and high-dimensional model data, and edge data where the from and to links of the edges.

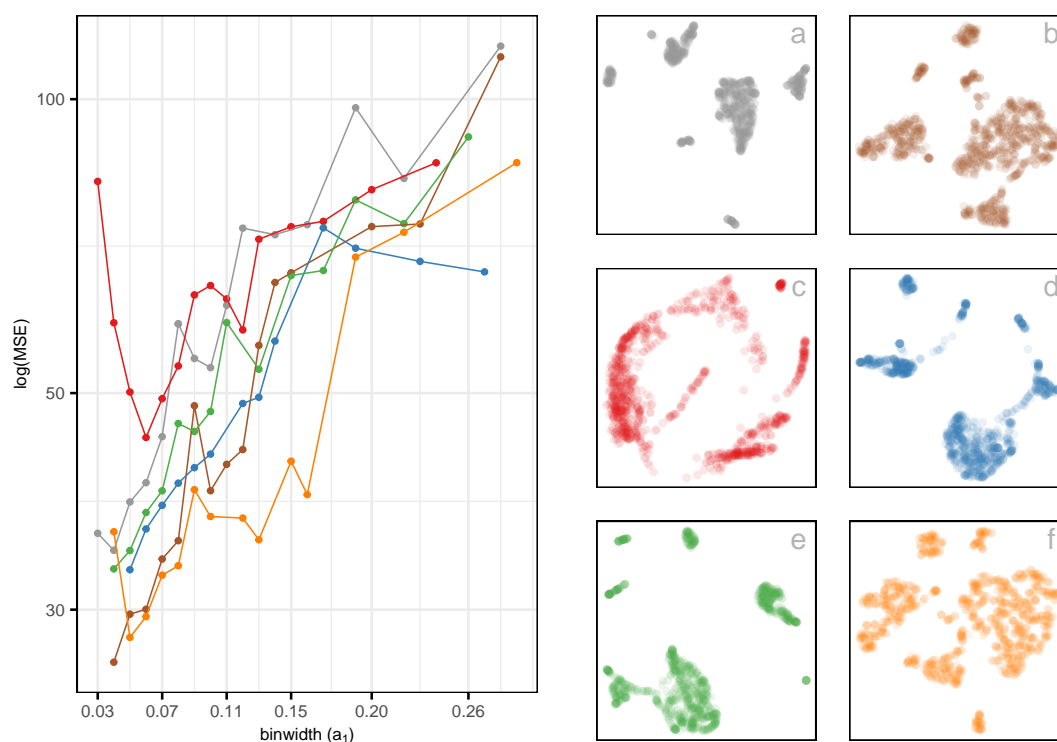
`show_error_link_plots()` helps to see investigate whether the model fits the points everywhere or fits better in some places, or simply mismatches the pattern. The function requires several arguments: points data which contain Non-linear dimension reduction data, high-dimensional data, high-dimensional model data, and model error, and edge data where the from and to links of the edges.

```
df_exe <- comb_all_data_model(
  highd_data = scurve,
  nldr_data = scurve_umap,
  model_highd = df_bin,
  model_2d = df_bin_centroids
)

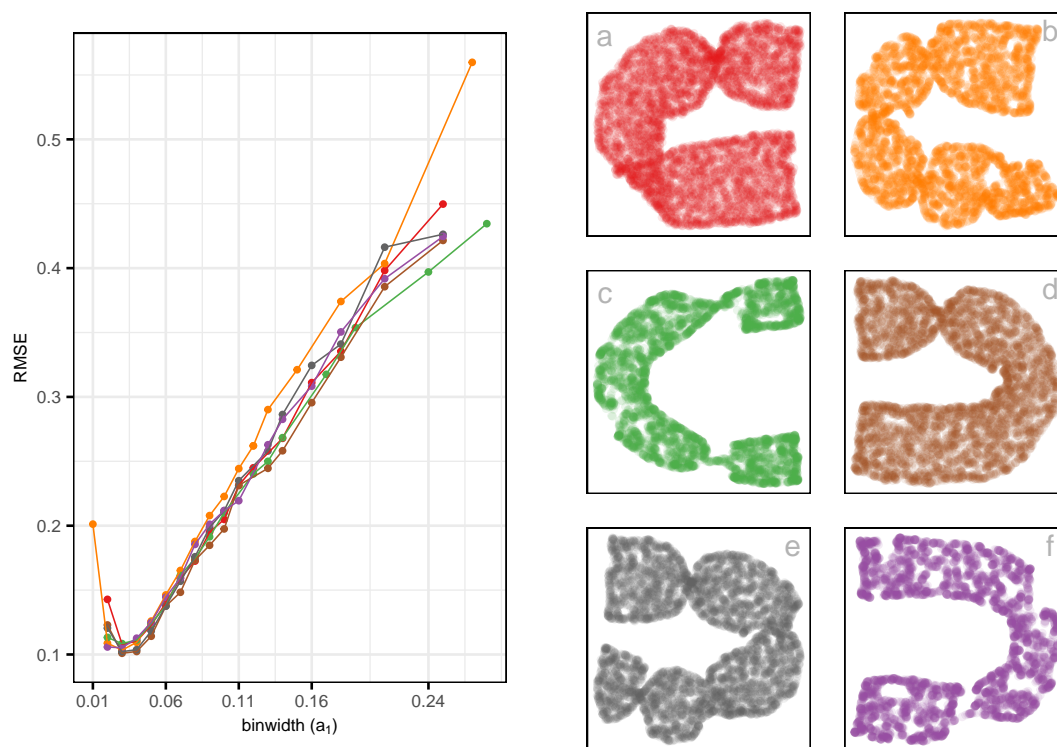
show_link_plots(
  point_data = df_exe,
  edge_data = trimesh
)

df_exe <- comb_all_data_model_error(
  highd_data = scurve,
  nldr_data = scurve_umap,
  model_highd = df_bin,
  model_2d = df_bin_centroids,
  error_data = model_error
)

show_error_link_plots(
  point_data = df_exe,
  edge_data = trimesh
)
```



### Find the most appropriate fit



## 3 Application

Need to find another application

<https://www.nature.com/articles/s41586-018-0590-4#data-availability>  
[https://figshare.com/articles/dataset/Robobject\\_files\\_for\\_tissues\\_processed\\_by\\_Seurat/5821263/1](https://figshare.com/articles/dataset/Robobject_files_for_tissues_processed_by_Seurat/5821263/1)



## 4 Discussion

This paper presents the R package `quollr` to develop a way to take the fitted model, as represented by the positions of points in 2- $D$ , and turn it into a high-dimensional wireframe to overlay on the data, viewing it with a tour.

The paper includes a clustering example to illustrate how `quollr` is useful to assess which NLDR technique and which (hyper)parameter choice gives the most accurate representation. In addition, how to select parameters for hexagonal binning and fitting model are explained.

Possible future improvements would be...

This new tool provides an effective start point for automatically creating regular hexagons and help to evaluate which NLDR technique and which hyperparameter choice gives the most accurate representation of  $p$ - $D$  data.

## 5 Acknowledgements

This article is created using `knitr` (Xie, 2015) and `rmarkdown` (Xie et al., 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>.

## Bibliography

- D. Carr, ported by Nicholas Lewin-Koh, M. Maechler, and contains copies of lattice functions written by Deepayan Sarkar. *hexbin: Hexagonal Binning Routines*, 2023. URL <https://CRAN.R-project.org/package=hexbin>. R package version 1.28.3. [p1]
- Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL <https://yihui.name/knitr/>. ISBN 978-1498716963. [p17]
- Y. Xie, J. Allaire, and G. Golemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida, 2018. URL <https://bookdown.org/yihui/rmarkdown>. ISBN 978-1138359338. [p17]

Jayani P. Gamage  
Monash University  
Department of Econometrics and Business Statistics, VIC 3800 Australia  
<https://jayanilakshika.netlify.app/>  
ORCID: 0000-0002-6265-6481  
[jayani.piyadigamage@monash.edu](mailto: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](mailto:dicook@monash.edu)

Paul Harrison  
Monash University  
MGBP, BDInstitute, VIC 3800 Australia  
ORCID: 0000-0002-3980-268X  
[paul.harrison@monash.edu](mailto: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](mailto: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