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

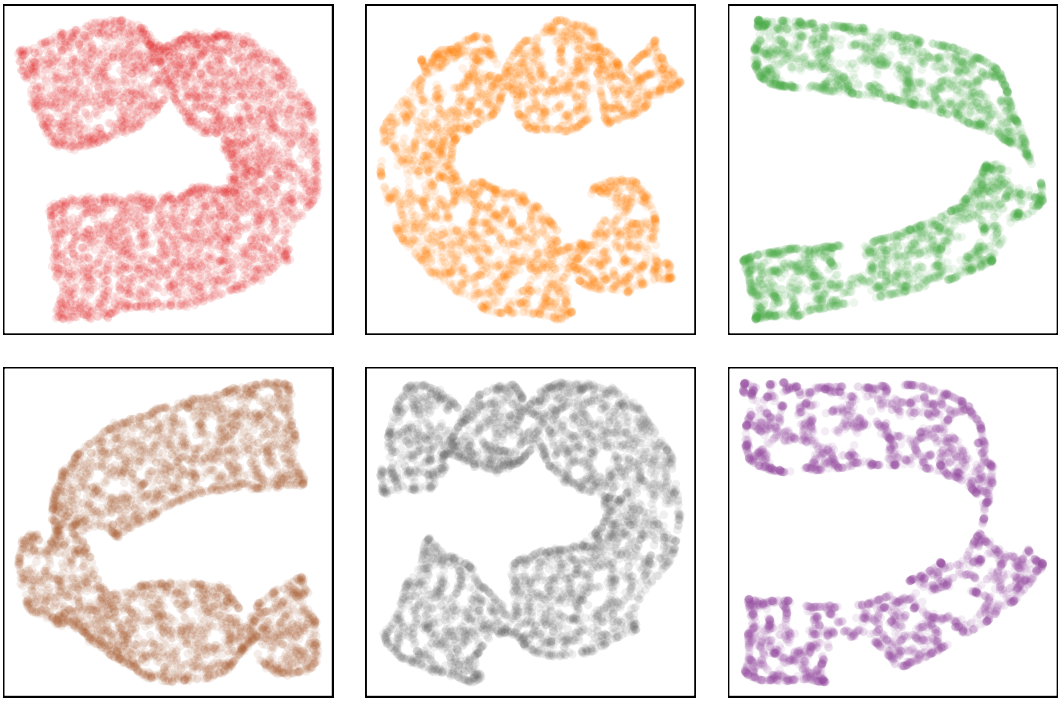
by Jayani P. Gamage, Dianne Cook, Paul Harrison, Michael Lydeamore, and Thiyanga S. Talagala

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 `triangular_3d_data` data from the **cardinalR** package is used to illustrate the algorithm and its application within 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 (CARN) at `https://CRAN.R-project.org/package=quollr`.

Add the image generated with S-curve data



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, and describe a visual heuristic to describe parameter selection. Finally, we give a brief conclusion of the paper and discuss potential opportunities for use of our algorithm.

2 Implementation

The package can be installed from CRAN:

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

The development version can be installed from GitHub:

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

Usage

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

`s_curve_noise_training` is a 3-*D* S-curve data set with additional four noise dimensions which is used to train the model. `s_curve_noise_umap` is the UMAP 2-*D* embedding for `s_curve_noise_training` data set. Each data set contains a unique ID column.

Scaling the data

The algorithm begins by scaling NLDR data to a standard scale using the `gen_scaled_data()` function. This function standardizes the data and provides the original limits of embeddings and the aspect ratio.

```
scaled_nldr_obj_scurve <- gen_scaled_data(data = s_curve_noise_umap)

s_curve_noise_umap_scaled <- scaled_nldr_obj_scurve$scaled_nldr

lim1 <- scaled_nldr_obj_scurve$lim1
lim2 <- scaled_nldr_obj_scurve$lim2
r2 <- diff(lim2)/diff(lim1)
```

The mains steps for the algorithm can be executed by the main function `fit_highd_model()`, or can be run separately for more flexibility. When constructing the 2-*D* model, the user can choose either to fit the 2-*D* model with hexagonal bin centroids or bin means using `is_bin_centroid` argument.

If a user would like to perform steps of the algorithm themselves, additional user input will be needed for the function that perform each step. For example, if the user wishes to use already binning data, the `extract_hexbin_centroids()` function can be used directly.

The number of bins along the x-axis, the ratio of the ranges of the original embedding components, the buffer amount as a proportion of data, and if `is_rm_lwd_hex = TRUE`, benchmark value to remove low density hexagons are parameters that will be determined within `fit_highd_model()`, if they are not provided. They are created as they are needed throughout the following example. The function `fit_highd_model()` provides the fitted model in 2-*D*, and *p*-*D*.

```
fit_highd_model(
  highd_data = s_curve_noise_training,
  nldr_data = s_curve_noise_umap_scaled,
  bin1 = 12,
  r2 = r2,
  q = 0.1,
  is_bin_centroid = TRUE
  )
```

```
#> $df_bin
#> # A tibble: 86 x 8
#>    hb_id       x1    x2    x3        x4        x5        x6        x7
#>    <int>    <dbl> <dbl> <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
#>  1    15  0.968   0.198  1.21  0.000518  0.00161   0.00240  -0.000547
#>  2    16  0.730   0.142  1.67 -0.000689  0.000495  0.00168   0.000794
#>  3    17  0.323   0.158  1.94 -0.00265  -0.00523  -0.00942  -0.00115
#>  4    18 -0.203   0.154  1.98 -0.00269  -0.00107  -0.0382   -0.00167
#>  5    19 -0.628   0.114  1.78 -0.000131 -0.00323   0.0347   -0.00288
#>  6    27  0.987   0.579  1.14 -0.00124  -0.00121   0.0199   -0.00363
#>  7    28  0.847   0.537  1.51  0.000374 -0.00215  -0.00901  -0.000548
#>  8    29  0.469   0.476  1.87  0.00117  -0.000439 -0.0191    0.000251
#>  9    30 -0.00747 0.491  1.99 -0.000570  0.00114  -0.000815  0.000786
#> 10    31 -0.525   0.449  1.84  0.00141  -0.000575 -0.0112    0.000714
#> # i 76 more rows
#>
#> $df_bin_centroids
#> # A tibble: 86 x 6
#>    hexID   c_x      c_y bin_counts std_counts drop_empty
#>    <int> <dbl>    <dbl>      <int>      <dbl> <lgl>
#>  1    15 0.158 0.000959         43     0.518  FALSE
#>  2    16 0.262 0.000959         35     0.422  FALSE
#>  3    17 0.365 0.000959         21     0.253  FALSE
#>  4    18 0.468 0.000959         11     0.133  FALSE
#>  5    19 0.571 0.000959          8     0.0964 FALSE
#>  6    27 0.107 0.0904           10     0.120  FALSE
#>  7    28 0.210 0.0904           56     0.675  FALSE
#>  8    29 0.313 0.0904           43     0.518  FALSE
#>  9    30 0.416 0.0904           54     0.651  FALSE
#> 10    31 0.520 0.0904           42     0.506  FALSE
#> # i 76 more rows
```

Constructing the 2-D model

Constructing the 2-D model mainly contains (i) binning data, (ii) obtaining bin centroids, and (iii) indicating neighbors by line segments connecting centroids.

Computing hexagon grid configurations

The configurations of the hexagonal grid is defined by the number of bins in each direction. To find the number of bins along the y-axis, `calc_bins_y()` is used. This function takes as input the number of bins along the x-axis, the ratio of the ranges of the original embedding components, and the buffer amount as a proportion of the data. Additionally, this function provides the bin width.

```
calc_bins_y(
  bin1 = 12,
  r2 = r2,
  q = 0.1
  )
```

```
#> $bin2
#> [1] 13
#>
#> $a1
#> [1] 0.1032943
#>
#> $a2
#> [1] 0.08945548
```

Binning the data

Points are allocated to the bins they fall into based on the nearest centroid. The main steps of the hexagonal binning algorithm can be executed using the `hex_binning()` function, or they can be run

separately for greater flexibility. The parameters used within `hex_binning()` include the scaled NLDR data, the number of bins along the x-axis, the ratio of the ranges of the original embedding components, and the buffer amount as a proportion of the data. The output is an object of the `hex_bin_obj` class, which contains the number of bins in each direction, the coordinates of the hexagonal grid starting point, the details of bin centroids, the coordinates of bins, embedding components with their corresponding hexagon IDs, hex bins with their corresponding standardized counts, the total number of bins, the number of non-empty bins, and the points within each hexagon.

```
hex_binning(
  data = s_curve_noise_umap_scaled,
  bin1 = 12,
  r2 = r2,
  q = 0.1
  )
```

```
#> $a1
#> [1] 0.1032943
#>
#> $a2
#> [1] 0.08945548
#>
#> $bins
#> [1] 12 13
#>
#> $start_point
#> [1] -0.10000000 -0.08849688
#>
#> $centroids
#> # A tibble: 156 x 3
#>    hexID      c_x     c_y
#>    <int>    <dbl>   <dbl>
#>  1     1 -0.1     -0.0885
#>  2     2  0.00329 -0.0885
#>  3     3  0.107   -0.0885
#>  4     4  0.210   -0.0885
#>  5     5  0.313   -0.0885
#>  6     6  0.416   -0.0885
#>  7     7  0.520   -0.0885
#>  8     8  0.623   -0.0885
#>  9     9  0.726   -0.0885
#> 10    10  0.830   -0.0885
#> # i 146 more rows
#>
#> $hex_poly
#> # A tibble: 936 x 3
#>    hex_poly_id        x       y
#>          <int>    <dbl>   <dbl>
#>  1           1 -0.1     -0.0289
#>  2           1 -0.152   -0.0587
#>  3           1 -0.152   -0.118
#>  4           1 -0.1     -0.148
#>  5           1 -0.0484  -0.118
#>  6           1 -0.0484  -0.0587
#>  7           2  0.00329 -0.0289
#>  8           2 -0.0484  -0.0587
#>  9           2 -0.0484  -0.118
#> 10           2  0.00329 -0.148
#> # i 926 more rows
#>
#> $data_hb_id
#> # A tibble: 3,750 x 4
#>     emb1   emb2    ID hb_id
#>    <dbl>  <dbl> <int> <int>
#>  1 0.270 0.839      1   125
```

```
#>  2 0.788 0.466      2    82
#>  3 0.771 0.319      3    69
#>  4 0.306 0.542      5    88
#>  5 0.549 0.806      6   127
#>  6 0.166 0.259      7    52
#>  7 0.672 0.596      9   104
#>  8 0.735 0.354     10    69
#>  9 0.839 0.467     11    82
#> 10 0.240 0.0903    12    28
#> # i 3,740 more rows
#>
#> $std_cts
#> # A tibble: 86 x 3
#>    hb_id     n std_counts
#>    <int> <int>      <dbl>
#>  1    15    43     0.518
#>  2    16    35     0.422
#>  3    17    21     0.253
#>  4    18    11     0.133
#>  5    19     8     0.0964
#>  6    27    10     0.120
#>  7    28    56     0.675
#>  8    29    43     0.518
#>  9    30    54     0.651
#> 10    31    42     0.506
#> # i 76 more rows
#>
#> $tot_bins
#> [1] 156
#>
#> $non_bins
#> [1] 86
#>
#> $pts_bins
#> # A tibble: 86 x 2
#>    hb_id pts_list
#>    <int> <list>
#>  1    15 <int [43]>
#>  2    16 <int [35]>
#>  3    17 <int [21]>
#>  4    18 <int [11]>
#>  5    19 <int [8]>
#>  6    27 <int [10]>
#>  7    28 <int [56]>
#>  8    29 <int [43]>
#>  9    30 <int [54]>
#> 10    31 <int [42]>
#> # i 76 more rows
#>
#> attr(,"class")
#> [1] "hex_bin_obj"
```

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(
  df_bin_centroids_all = df_bin_centroids,
  bin1 = 12,
```

```
  df_bin_centroids_low = df_bin_centroids_low
  )
```

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.

```
tr1_object <- tri_bin_centroids(
  hex_df = df_bin_centroids,
  x = "c_x",
  y = "c_y"
  )

tr_from_to_df <- gen_edges(
  tri_object = tr1_object
  )
```

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.

```
find_lg_benchmark(
  distance_edges = distance_df,
  distance_col = "distance"
  )
```

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 bin to represent bin centroids. This transformation is performed using the `avg_highd_data()` function, which takes *p*-*D* data and their corresponding hexagonal bin IDs as inputs.

```
umap_data_with_hb_id <- hb_obj$data_hb_id

df_all <- bind_cols(
  s_curve_noise_training |> dplyr::select(-ID),
  umap_data_with_hb_id
  )

df_bin <- avg_highd_data(
  data = df_all
  )
```

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 = s_curve_noise_training,
  model_2d = df_bin_centroids,
  model_highd = df_bin
  )
```

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 = s_curve_noise_training,
  model_2d = df_bin_centroids,
  model_highd = df_bin
  )
```

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.

```
augment(
  highd_data = s_curve_noise_training,
  model_2d = df_bin_centroids,
  model_highd = df_bin
  )
```

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 = all_centroids_df,
    aes(x = c_x, y = c_y)
    ) +
  coord_fixed()
```

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 = df_bin_centroids,
```

```
    aes(x = c_x, y = c_y)
    ) +
  coord_fixed()

vis_lg_mesh(
  distance_edges = distance_df,
  benchmark_value = 0.75,
  tr_coord_df = tr_from_to_df,
  distance_col = "distance"
  )

vis_rmlg_mesh(
  distance_edges = distance_df,
  benchmark_value = 0.75,
  tr_coord_df = tr_from_to_df,
  distance_col = "distance"
  )
```

p-D model visualization

Displaying the *p-D* model overlaid on the data is done using the function `show_langevitour()`. 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.

```
show_langevitour(
  df = df_all,
  df_b = df_bin,
  df_b_with_center_data = df_bin_centroids,
  benchmark_value = 0.75,
  distance = distance_df,
  distance_col = "distance",
  use_default_benchmark_val = FALSE,
  col_start = "x"
  )
```

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.

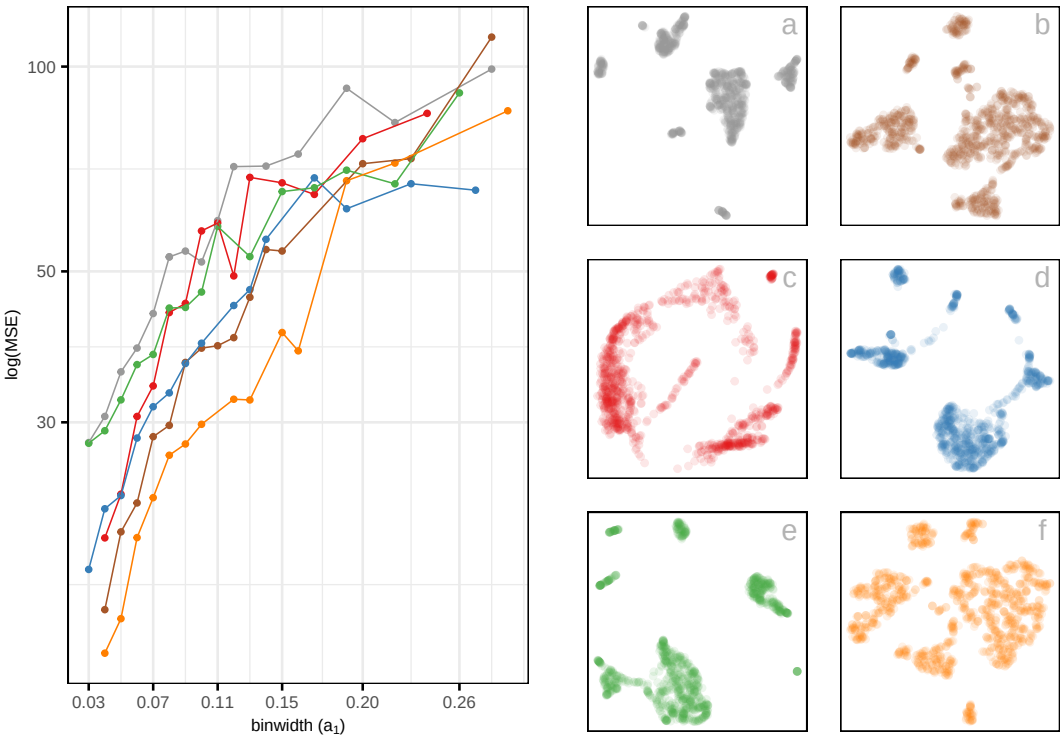
`show_error_link_plots()` helps to see investigate whether the model fits the points everywhere or fits better in some places, or simply mismatches the pattern. The function requires several arguments: points data which contain Non-linear dimension reduction data, high-dimensional data, high-dimensional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

```
show_link_plots(
  point_df = df_exe,
  edge_df = distance_small_df
  )

show_error_link_plots(
  point_df = df_exe,
  edge_df = distance_small_df
  )
```

3 Application

Need to find another application



https://www.nature.com/articles/s41586-018-0590-4#data-availability https://figshare.com/articles/dataset/Robject_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. [p9]

Y. Xie, J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida, 2018. URL https://bookdown.org/yihui/rmarkdown. ISBN 978-1138359338. [p9]

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

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