Using *sabinaHSBM* for link prediction and network reconstruction with “marginal_all” method

This guide shows a simple use case based on the “marginal_all” method in *sabinaHSBM* to detect both missing (false negatives) and/or spurious (false positives) links in an incomplete and/or error-prone network.

We use the artificially created dataset `dat2` included in the package to show key functionalities, including data preparation, link prediction, and network reconstruction.

Requirements and installation

There are **two ways** to use the package:

Option A: Docker

Non-native UNIX users or those who prefer to avoid manual dependency installation can use the provided **ready-to-use Docker image** which includes R, Python, all dependencies, and *sabinaHSBM* pre-installed (see Supporting Information S1 for Docker container setup).

Once the container is running, simply load the package in your R session with:

```
library(sabinaHSBM)
```

Option B: Native UNIX installation

Native UNIX users can run *sabinaHSBM* locally **if** their system includes:

- R (version $\geq$ 4.0.4) with all required R packages (listed below)
- Python $\geq$ 3.12.3 with the `graph-tool` library (version $\geq$ 2.45)

```
install.packages("remotes")
# If the package is not installed, install sabinaHSBM from GitHub
 if (!requireNamespace("sabinaHSBM", quietly = TRUE)) {
   remotes::install_github("anonbuild/sabinaHSBM")
 }

# Load sabinaHSBM
library(sabinaHSBM)
```

Install and load required R packages if not already available:

```
list.of.packages <- c(
  "dplyr",
  "reshape2",
  "reticulate",
  "stringr",
  "tidyr",
  "ROCR"
)
```

```r
new.packages <-
    list.of.packages[!(list.of.packages %in% installed.packages()[, "Package"])]
if (length(new.packages) > 0) {
  install.packages(new.packages, dependencies = TRUE)
}

for (package in list.of.packages) {
  library(package, character.only = TRUE)
}
```

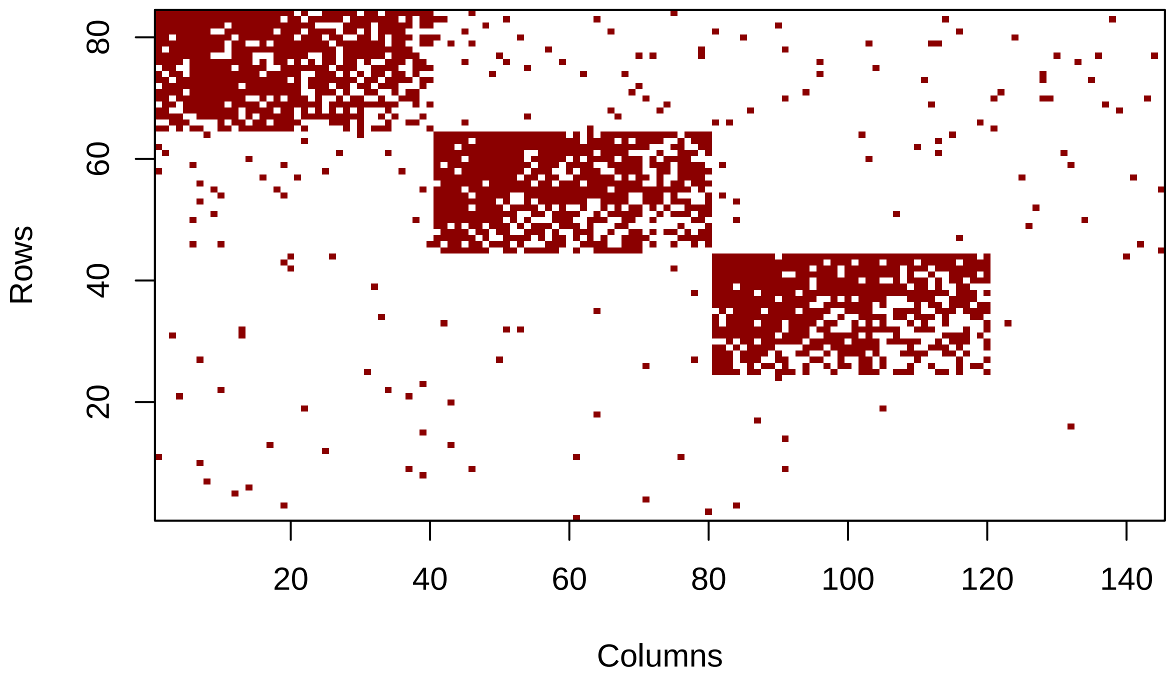
```r
# Record starting time (optional)
start_time <- Sys.time()
```

Load example data

The dataset `dat2` is a binary matrix representing a hypothetical species interactions network. Columns and rows correspond to two different types of nodes (e.g., hosts and parasites), and links (values of `1`, in red) represent interactions between them while `0` (in white) represent lack of an observed interaction. These links are distributed in a pronounced structured pattern with some random noise, to **focus the example on identifying spurious links**.

```r
# Load the dataset
data(dat2, package = "sabinaHSBM")
dat <- dat2
```

```r
# Plot the original matrix
plot_interaction_matrix(adj_mat = dat, order_mat = FALSE)
```

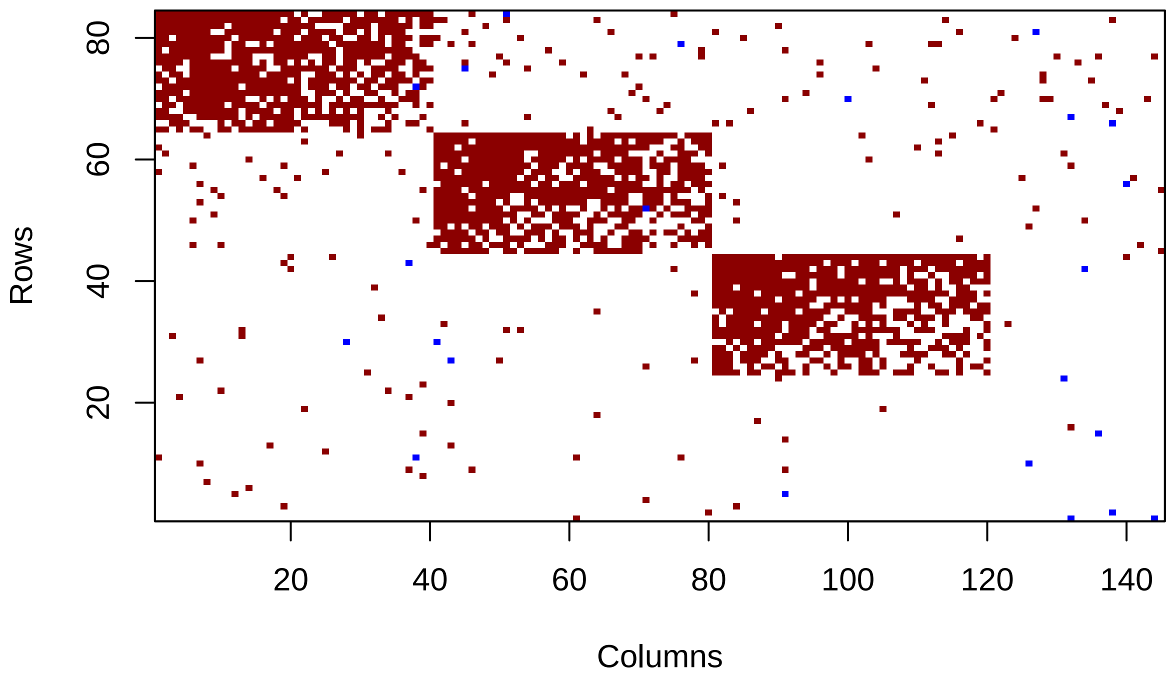


To simulate spurious links and address the `marginal_all` method, we randomly add a small number of false positives (1) to the matrix. This controlled modification allows us to demonstrate how the method can work with both spurious links (observed but potentially erroneous interactions, false positives) and missing links (unobserved but likely interactions, false negatives). In particular, we add a proportion of spurious randomly across all network, and a smaller proportion of spurious for nodes with low degree.

```r
# Add spurious links artificially
set.seed(123)
n_spurious <- ceiling(sum(dat) * 0.01)
n_spurious_rare <- floor(n_spurious/4)
# Find some random spurious
zero_indices <- which(dat == 0)
sampled_indices <- sample(zero_indices, n_spurious)
spurious_links <- arrayInd(sampled_indices, dim(dat))
# Find some spurious targetting nodes with low degree
low_rows <- which(rowSums(dat) == min(rowSums(dat)))
low_cols <- which(colSums(dat) == min(colSums(dat)))
low_links <- as.matrix(expand.grid(low_rows, low_cols))
spurious_rare <- low_links[sample(nrow(low_links),
                                  min(n_spurious_rare, nrow(low_links))), ]
indices_rare <- apply(spurious_rare, 1,
                      function(x) ((x[2] - 1) * nrow(dat) + x[1]))
# Add spurious to data
spurious_links <- rbind(spurious_links, spurious_rare)
sampled_indices <- c(sampled_indices, indices_rare)
total_spurious <- nrow(spurious_links)
dat[sampled_indices] <- 1
```

We now use the `plot_interaction_matrix()` function to visualize the added spurious in blue.

```r
# Plot the matrix with spurious
dat_spur <- dat
dat_spur[sampled_indices] <- 2
plot_interaction_matrix(dat_spur,
                        col = c("white", "red4", "blue1"),
                        order_mat = FALSE)
```



Prepare input data for HSBM

The `hsbm.input` function cleans the dataset and creates cross-validation folds by temporarily holding out subsets of observed links. In our example, it randomly splits observed links into three similar-sized subsets (because here we use 3-fold cross-validation), holding out one subset per fold to test the model's ability to recover removed links.

We set "max_held_per_fold" to the number of added spurious (23) to keep the data more balanced, otherwise, number of removed links would surpass the number of added spurious by a large margin hindering inference. To detect spurious it is better to have the dataset as complete as possible so overall network structure can be more effectively detected.

```r
# Prepare input data
n_folds <- 3        # Number of folds for cross-validation
myInput <- hsbm.input(
  dat,              # Binary matrix of observed links
  n_folds = n_folds,
  max_held_per_fold = total_spurious # Maximum number of removed links per fold
)
```

```
# Summarizes network characteristics
summary(myInput)
```

```
##   n_rows n_cols n_links rows_single_link cols_single_link possible_links
## 1     84    145    1916               11               13          12180
```

Predict link probabilities with "marginal_all"

The `hsbm.predict` function applies the HSBM to predict marginal posterior probabilities of all links—both observed (`1s`) and unobserved (`0s`)—so you can identify spurious and missing links in one step. The function works directly with the processed input created by `hsbm.input`. Computation can be intensive, but parallelization across cores is supported.

```
# Generate HSBM predictions
myPred <- hsbm.predict(
  myInput,              # Input data processed by hsbm.input()
  iter = 10000,         # Number of iterations
  wait = 10000,          # Number of iterations for MCMC equilibration
  rnd_seed = 123,       # Sets seed in python environment for reproducibility
  method = "marginal_all", # Method for link prediction
  save_blocks = TRUE,   # Save group assignments
  save_pickle = FALSE, # Save results as pickle files,
  save_plots = FALSE,   # Save hierarchical edge bundling plots
  n_cores = 3
)
```

Predicted link probabilities and group assignments are stored for each fold.

Below, we extract the link probabilities (p) for fold 1. For example, a high probability of an undocumented (reconstructed) link would suggest that this interaction is likely real but was missed due to sampling limitations. Conversely, it may assign a low p to a recorded interaction, indicating it might be spurious.

```
# View probabilities for fold 1
probabilities_fold1 <- myPred$probs[[1]]
print(probabilities_fold1[sample(1:nrow(probabilities_fold1), 10), ])
```

```
##      v1  v2          p v1_names v2_names     edge_type
## 1347 11  92 1.00000000    row12     col9    documented
## 6623 83 126 0.00310031    row84    col43 reconstructed
## 791   6 114 1.00000000     row7    col31    documented
## 1987 16 120 0.06580658    row17    col37 reconstructed
## 5001 46 153 0.00200020    row47    col70 reconstructed
## 1341 11  85 1.00000000    row12     col2    documented
## 4903 45 136 0.00010001    row46    col53 reconstructed
## 5342 51 187 0.11101110    row52   col104 reconstructed
## 5407 52 202 0.09290929    row53   col119 reconstructed
## 5854 59 203 1.00000000    row60   col120    documented
```

The group assignments provide the hierarchical clustering structure of nodes for each fold. Let's extract the group assignments for each fold:

```
# Check groups for each fold
print(lapply(myPred$groups,
             function(x){
                 g_cols <- grep("^G", names(x))
                 return(apply(x[g_cols], 2, function(y) length(unique(y))))
             }))
```

```
## [[1]]
## G1 G2 G3 G4
##  7  2  1  1
##
## [[2]]
## G1 G2 G3 G4 G5
##  6  4  2  1  1
##
## [[3]]
## G1 G2 G3 G4 G5
##  7  4  2  1  1
```

Hierarchical group assignments provide insight into how nodes are organized across multiple levels. At the first level (G1) nodes are divided into specific groups, reflecting fine-scale patterns. Moving to higher levels (G2, G3, G4) these groups (or communities) are progressively aggregated, revealing broader patterns and relationships or communities.

Network Reconstruction

The `hsbm.reconstructed` function generates a reconstructed binary interaction matrix by combining predictions from all folds. The predicted matrix is transformed to binary values using a user-specified threshold which in this case we set to the value 0.5.

```
# Network reconstruction
myReconst <- hsbm.reconstructed(
  myPred,                 # Predictions processed by hsbm.predict
  rm_documented=FALSE,    # Remove documented links during validation
  na_treatment = "ignore_na", # Handle NA values in predictions
  threshold = 0.5,            # Binarization threshold
  consistency_matrix = "average_thresholded" # Combine fold predictions
)
```

The `myReconst` object includes the final averaged probability matrix, the final reconstructed binary matrix, and evaluation metrics.

Let's explore the model overall performance and network reconstruction details.

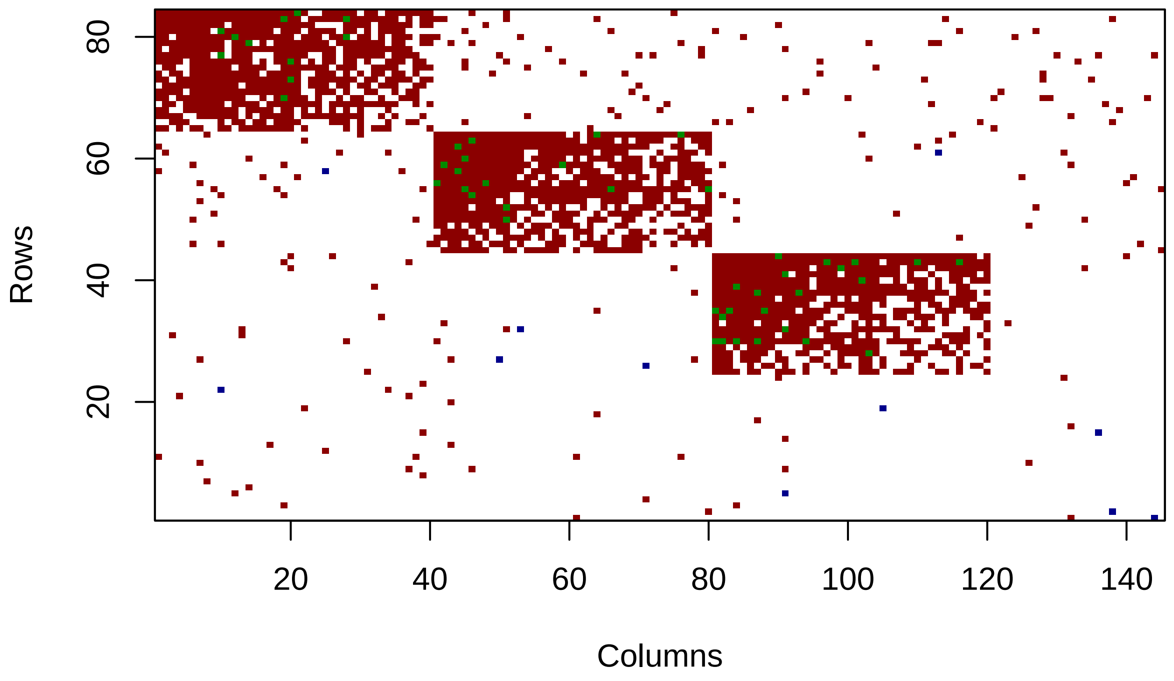
```
# View the reconstruction summary and evaluation metrics0
summary(myReconst)
```

```
## $`Reconstructed Network Metrics`
##   obs_links unobs_links pred_links kept_links spurious_links missing_links
## 1      1916       10264       1954       1905             11            49
##
## $`Evaluation Metrics`
##    mean_RLF  mean_auc mean_aucpr mean_yPRC mean_prec mean_sens mean_spec
## 1 0.1014493 0.9953646   1.028874 0.1573071 0.9711651 0.9775574 0.9945765
##    mean_ACC    mean_ERR  mean_tss
## 1 0.9918993 0.008100712 0.9721339
```

The summary above provides the number of spurious and missing links, as well as key evaluation metrics, such as the recovery link fraction (RLF). These results highlight the predicted interactions, showcasing the method's ability to detect spurious and missing links effectively. The HSBM was able to identify 17% of the added spurious as erroneously recorded links, and predicted 49 missing links. Also it simultaneously recovered an average of 10% of held-out links across the three folds (mean_RLF).

We can now visualize the reconstructed matrix highlighting the predicted missing (in green) and spurious links (in blue).

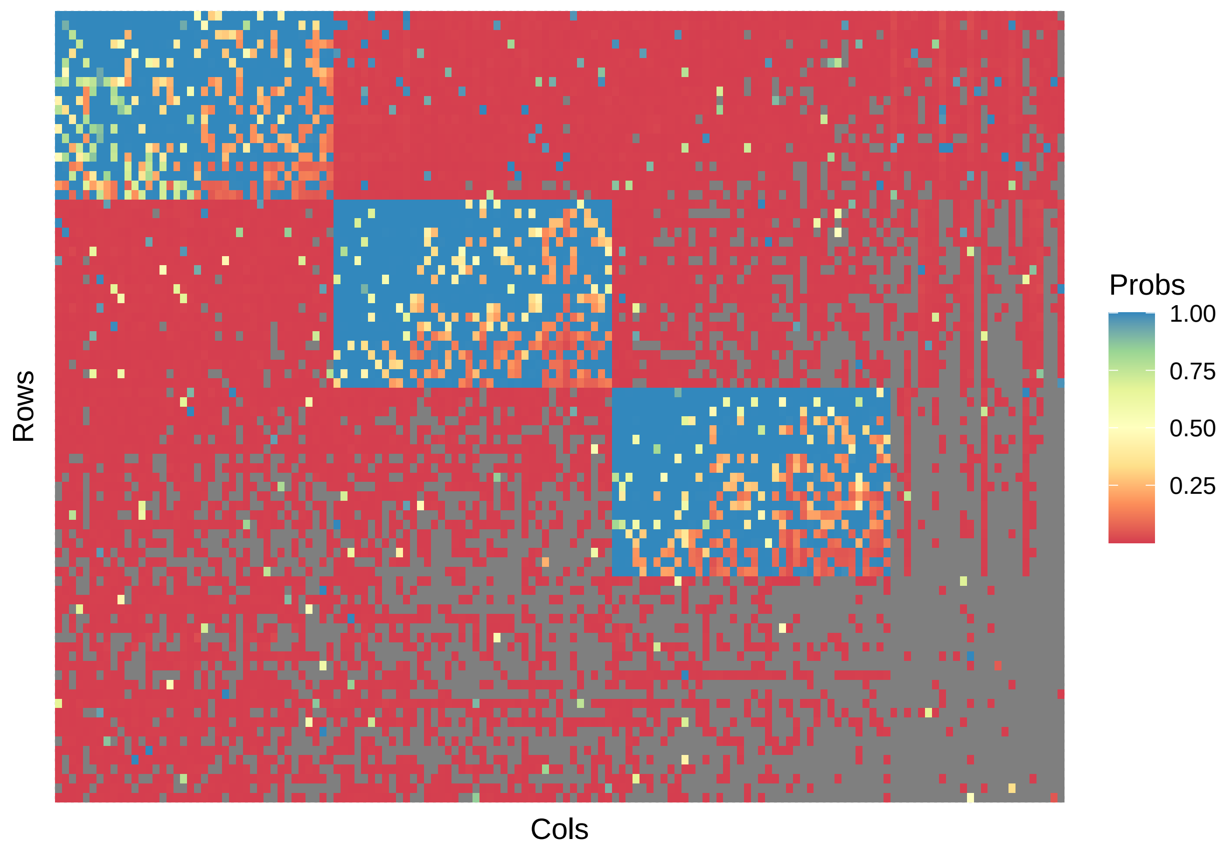
```r
# Plot the matrix with spurious
cols <- c("white", "red4")
dat_spur <- myReconst$data
new_spur <- which(myReconst$data == 1 & myReconst$new_mat == 0)
new_missing <- which(myReconst$data == 0 & myReconst$new_mat == 1)
if(length(new_spur) != 0){
    dat_spur[new_spur] <- 2
    cols <- c(cols, "blue4")
}
if(length(new_missing)!=0){
    dat_spur[new_missing] <- 3
    cols <- c(cols, "green4")
}
plot_interaction_matrix(dat_spur,
                        col = cols, order_mat = FALSE)
```



The heatmap of the probabilities below show that higher probabilities are mostly contained within the structured modules. Nodes outside one of the three modules (bottom rows and right columns) have a lot of non-sampled interactions (colored grey) possibly representing unlikely interactions.

```
library(reshape2)
library(ggplot2)
library(RColorBrewer)
# Create a matrix of averaged probabilities over all folds
mat_avg <- apply(simplify2array(myReconst$pred_mats), 1:2, mean,
                 na.rm = TRUE)

melted_matrix <- melt(mat_avg)
row_names <- rownames(mat_avg)
colnames(melted_matrix) <- c("Rows", "Cols", "Probs")
ggplot(melted_matrix, aes(Cols, Rows, fill= Probs)) +
  geom_tile() +
  scale_y_discrete(limits = rev(row_names)) +
  scale_fill_distiller(palette = "Spectral", direction = 1) +
  theme_minimal() +
  theme(axis.text = element_blank())
```



In most real world networks the threshold of `0.5` is too conservative. Thus, the inspection of likely spurious can be done by ranking documented edges by their probabilities.

Let's now examine the top 10 predicted links that are most likely to be spurious (false positives) by visualizing the probabilities of "documented" links in ascending order.

```
# Visualize the top most likely spurious links
top_links_spurious <- top_links(myReconst,
                                n = 10,
                                edge_type = "documented")
```

```r
print(top_links_spurious)
```

```
##    v1_names v2_names          p         sd
## 1     row84   col144 0.04833817 0.08372416
## 2     row70   col136 0.05813915 0.10069996
## 3     row59    col71 0.23712371 0.41071032
## 4     row83   col138 0.33333333 0.57735027
## 5     row58    col50 0.42364236 0.51719577
## 6     row80    col91 0.43384338 0.51296208
## 7     row63    col10 0.45224522 0.43032235
## 8     row27    col25 0.45444544 0.50451947
## 9     row53    col53 0.47314731 0.45207354
## 10    row66   col105 0.48731540 0.50048246
```

We show how we can rank the most likely missing links according to the model.

```r
# Visualize the top most likely spurious links
top_links_missing <- top_links(myReconst,
                               n = 10,
                               edge_type = "undocumented")
print(top_links_missing)
```

```
##    v1_names v2_names         p         sd
## 1      row2    col19 0.9209254 0.03741822
## 2     row41    col90 0.9095576 0.05295710
## 3     row30    col45 0.9045571 0.08043455
## 4     row55    col81 0.8150148 0.11143835
## 5     row47    col87 0.8084142 0.05019549
## 6     row26    col42 0.7853119 0.05090011
## 7      row2    col28 0.7826449 0.10219928
## 8     row50    col81 0.7698770 0.11708609
## 9     row12    col20 0.7655099 0.05607827
## 10    row55    col94 0.7524752 0.07918487
```

If the task is to do network reconstruction by specifically targetting the insertion of the most plausible missing links, we can use the probabilities from the `marginal_all` method as 'scores' in supervised binary classification, which is implemented in *sabinaHSBM* only for missing links. This is achieved by following the steps in Supporting Information S1, but with "method = 'marginal_all' " in `hsbm.predict()` and optionally "rm_documented = 'FALSE' " in `hsbm.reconstructed()`.

This document demonstrates the use of the *sabinaHSBM* package for network reconstruction. By applying the "marginal_all" method, we showcased how likely spurious and missing links can be identified and addressed in complex networks.

Computing characteristics

```r
# Show processing time and computer characteristics
end_time <- Sys.time()
cat("The processing time of this script took: ",
    round(as.numeric(difftime(end_time, start_time, units = "mins", 1))), "minutes\n")
```

```
## The processing time of this script took:  27 minutes
```

This analysis was performed on a Dell Inspiron notebook with the following characteristics:

- Processor (CPU): Intel Core i7-7500U @ 3.5GHz

- Memory (RAM): 16 GB
- Operating System: Ubuntu 24.04 (run on a docker image)
- R Version: 4.3.3