Comparison of Log transform and Scaling

Stephen Coleman

06/02/2020

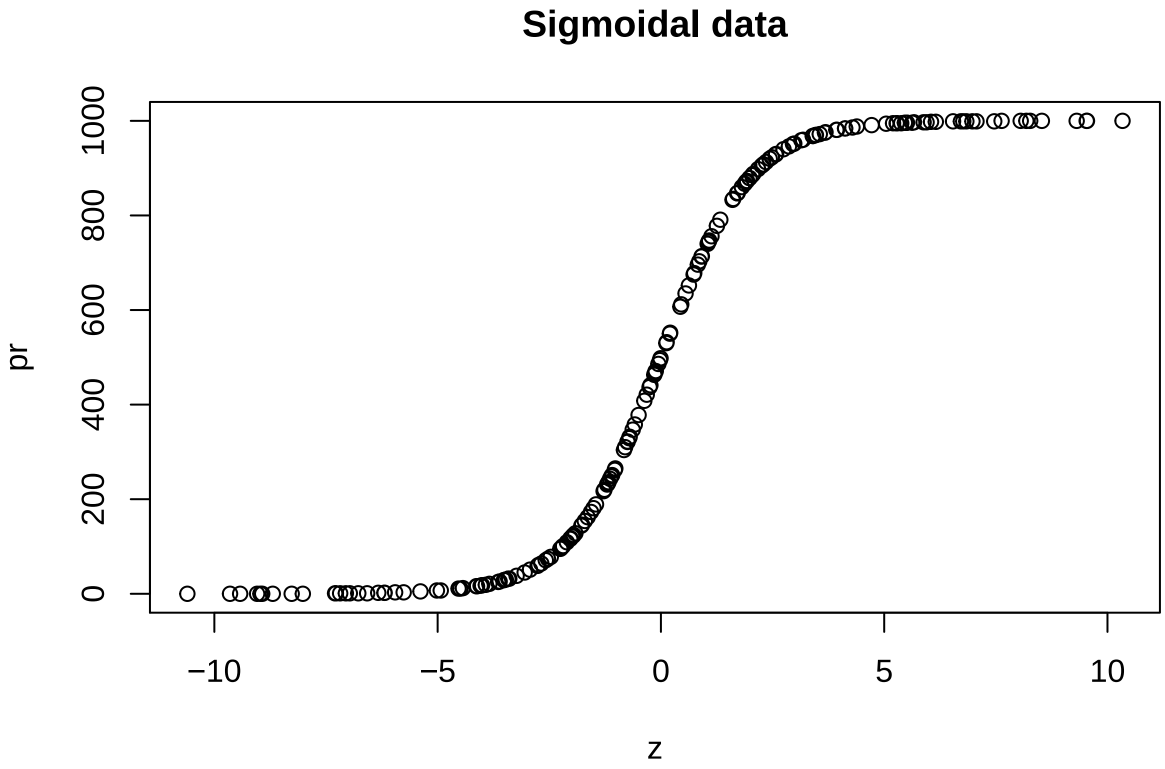
Generate count data

We wish to generate count data. First we create 5 subpopulations with some peturbation about a mean.

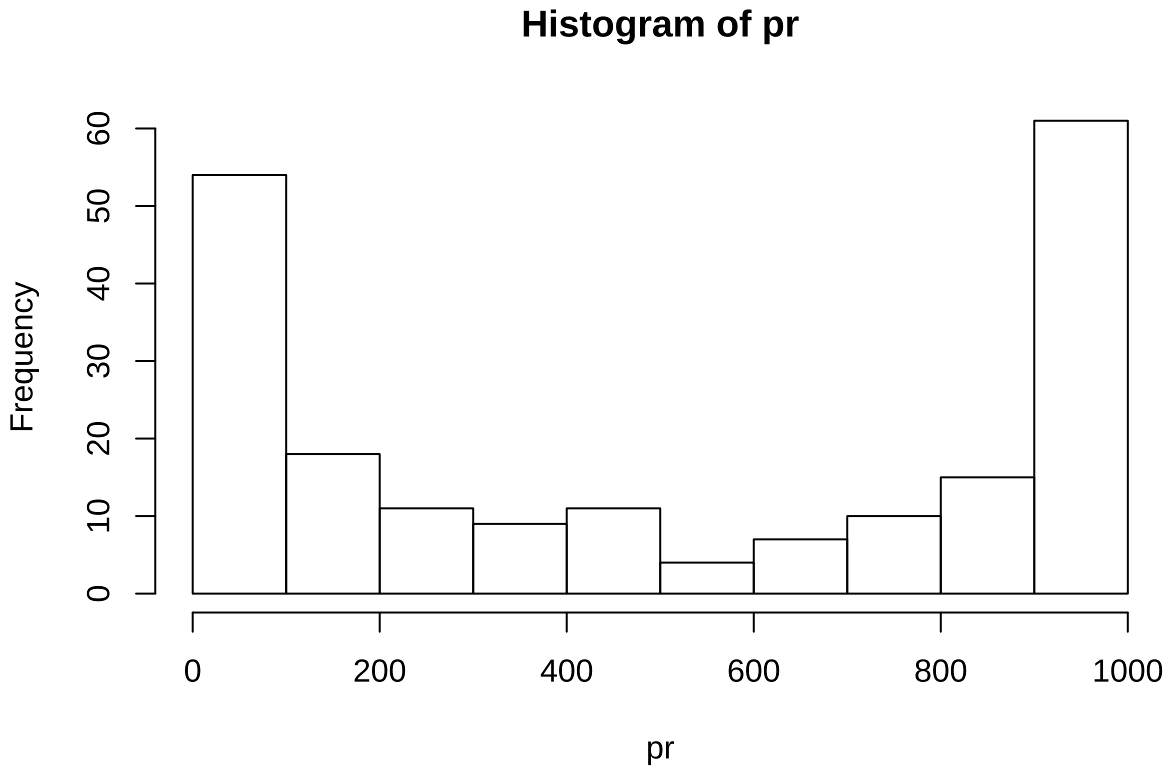
```
# Example of transforms on poission data
n <- 100
beta_0 <- c(1, 1, 5, 6, 2, 2)
beta_1 <- c(0.2, 0.2, 1, 2.5, 2, 2)

# Generate random data
x <- runif(n = n, min = 0, max = 2.0)

# Generate data from 4 poisson regression models
beta_1_mat <- sapply(beta_1, `*`, x)
exponent_mat <- t(apply(beta_1_mat, 1, `+`, t(beta_0)))

poisson_data <- apply(exp(exponent_mat), 2, function(x) {
  rpois(n = n, lambda = x)
})

# Put this data into a data.frame
poisson_df <- data.frame(
  Count_1 = c(poisson_data[, 1], poisson_data[, 3]),
  Count_2 = c(poisson_data[, 2], poisson_data[, 4]),
  Count_3 = c(poisson_data[, 5], poisson_data[, 6])
)
```

Create some data that follows a sigmoidal curve:

```
# some continuous variables
# x1 <- rnorm(n)
x1 <- runif(2 * n, -2, -1)
x2 <- runif(2 * n, 1, 2)
x3 <- runif(2 * n, -1, 1)
# x2 <- rnorm(n)

# linear combination with a bias
z <- 1 + 8 * x1 + 7.5 * x2 + 5 * x3

# pass through an inv-logit function and move to a scale similar to a count
pr <- round(1 / (1 + exp(-z)) * 1000)

plot(z, pr, main = "Sigmoidal data")
```

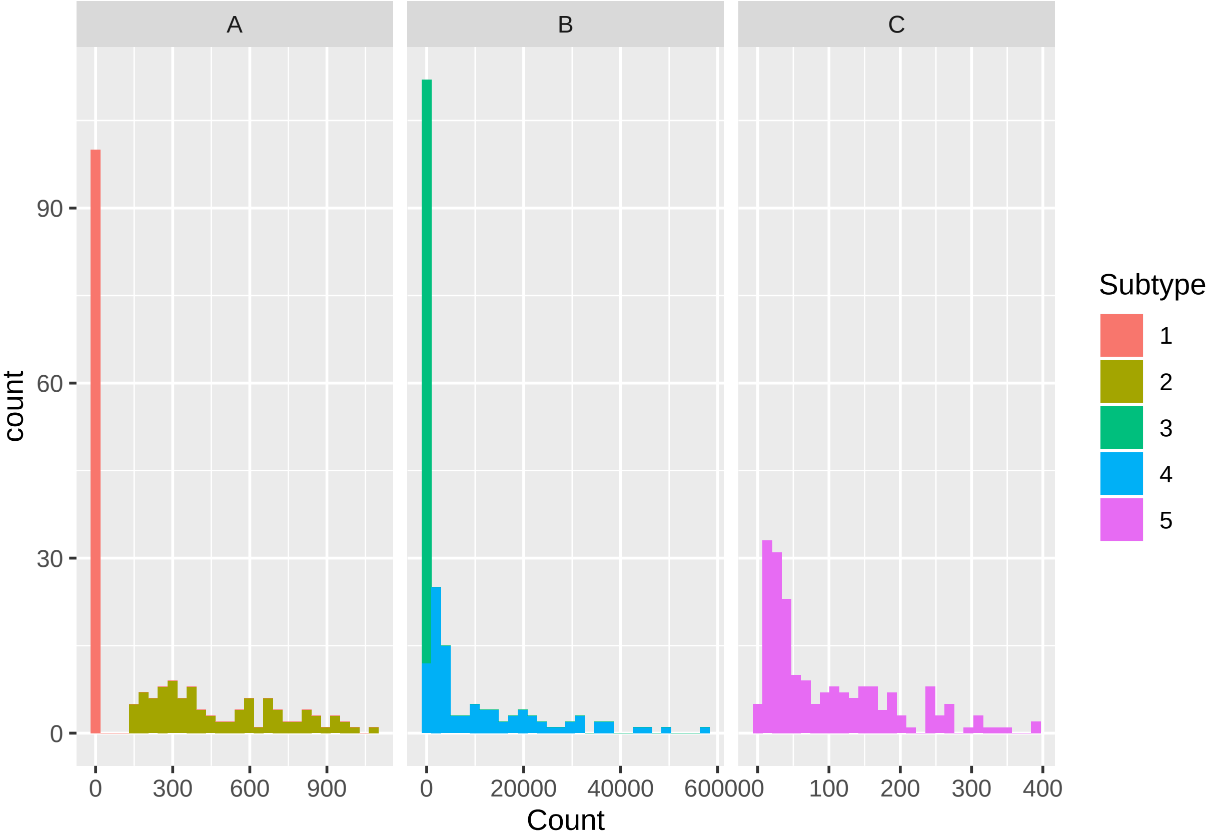

Sigmoidal data
pr
0
200
400
600
800
1000
−10
−5
0
5
10
z

```
hist(pr)
```


Histogram of pr
Frequency
0
10
20
30
40
50
60
0
200
400
600
800
1000
pr

Exploratory data analysis

We visualise the data to see how distinct the sub-populations are:



Now we combine these datasets and apply our different trasnformations:

```
# Bad hack to have the high counts align in each dataset imperfectly but well
# enough to have less sub-populations emerge from the combined dataset
fractions <- 8
new_order <- order(pr)
flag <- as.numeric(cut(pr,
  breaks = quantile(pr, probs = seq(0, 1, 1 / fractions)),
  include.lowest = T,
  labels = 1:fractions
))

flag[flag %in% 3:4] <- fractions + 1

# Combine the generated data
my_data <- as.data.frame(cbind(poisson_df, pr[order(flag)]))

# Assign row and column names
colnames(my_data) <- c(paste0("Count_", 1:(ncol(poisson_df) + 1)))
row.names(my_data) <- paste("Person", 1:nrow(my_data))

# Of some use later
n_var <- ncol(my_data)
```

```
head(my_data)
```

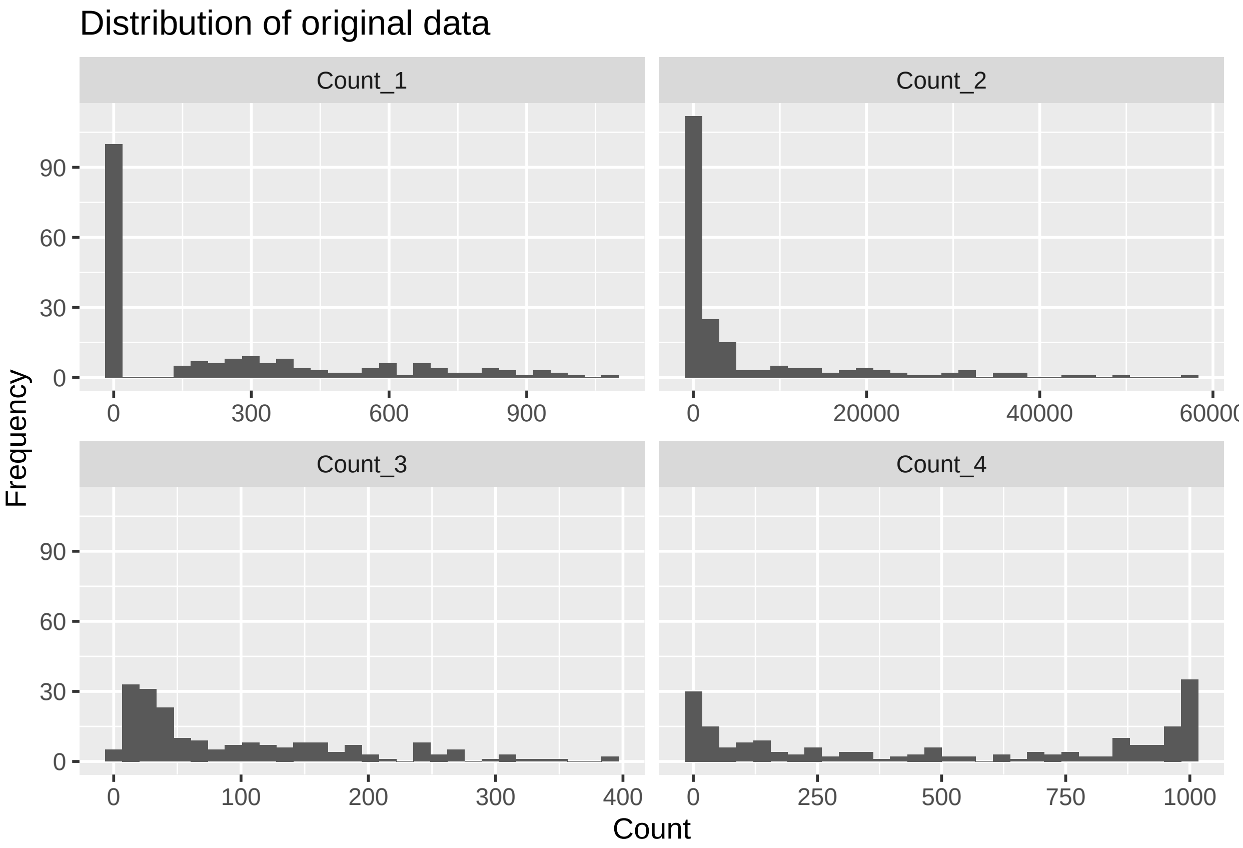
```
##          Count_1 Count_2 Count_3 Count_4
## Person 1       4       2      18       0
## Person 2       2       2      30       3
## Person 3       2       3      89      11
## Person 4       9       3     269       0
## Person 5       3       1      23       2
## Person 6       2       4     238       0
```

```
# Log transform
log_data <- log(1 + my_data) %>% as.data.frame()

# Mean centre and standardise the standard deviation within each variable
scaled_data <- scale(my_data) %>% as.data.frame()
```

Let us look at the distributions described by each variable for each dataset. We expect there to be two subpopulations present under variables "Count_1", "Count_2" and "Count_4", and a single population under "Count_3".

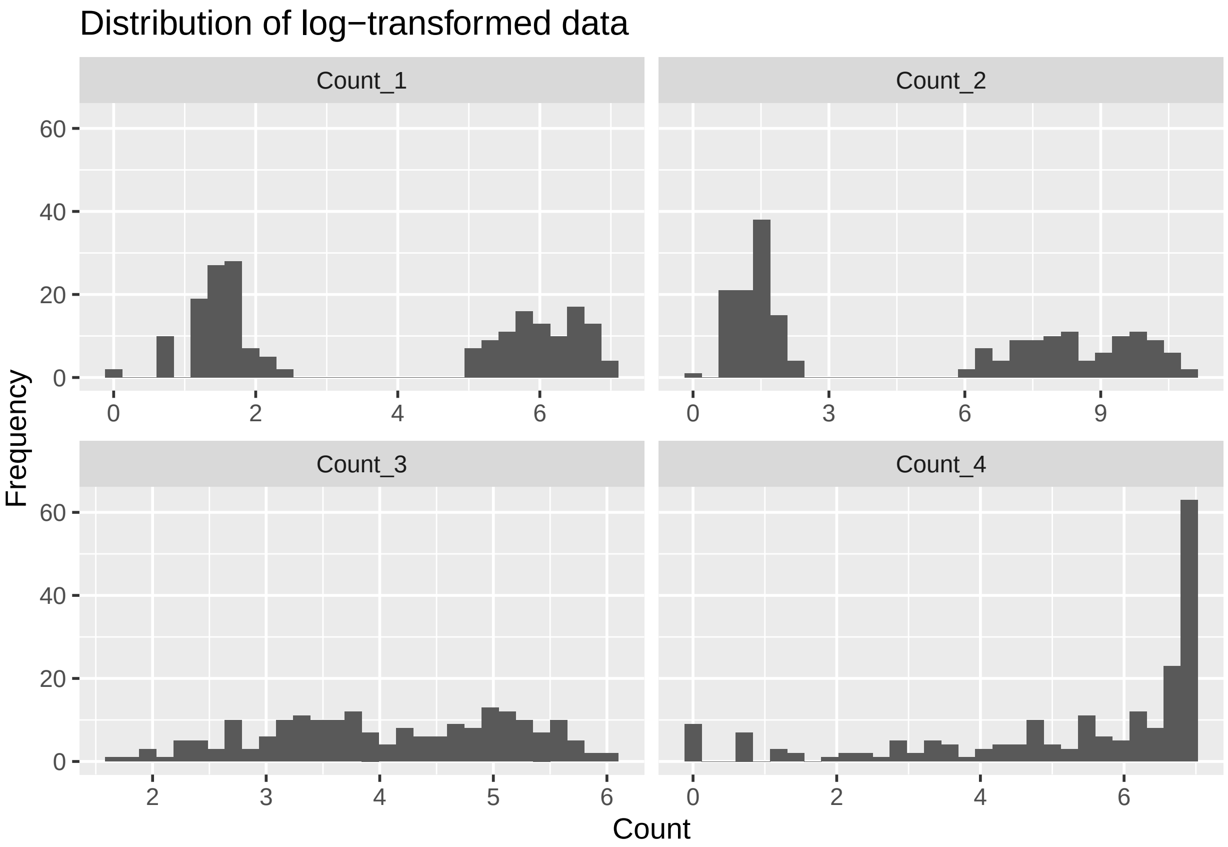
```
ggplot(gather(my_data), aes(value)) +
  geom_histogram() +
  facet_wrap(~key, scales = "free_x") +
  labs(
    title = "Distribution of original data",
    x = "Count",
    y = "Frequency"
  )
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



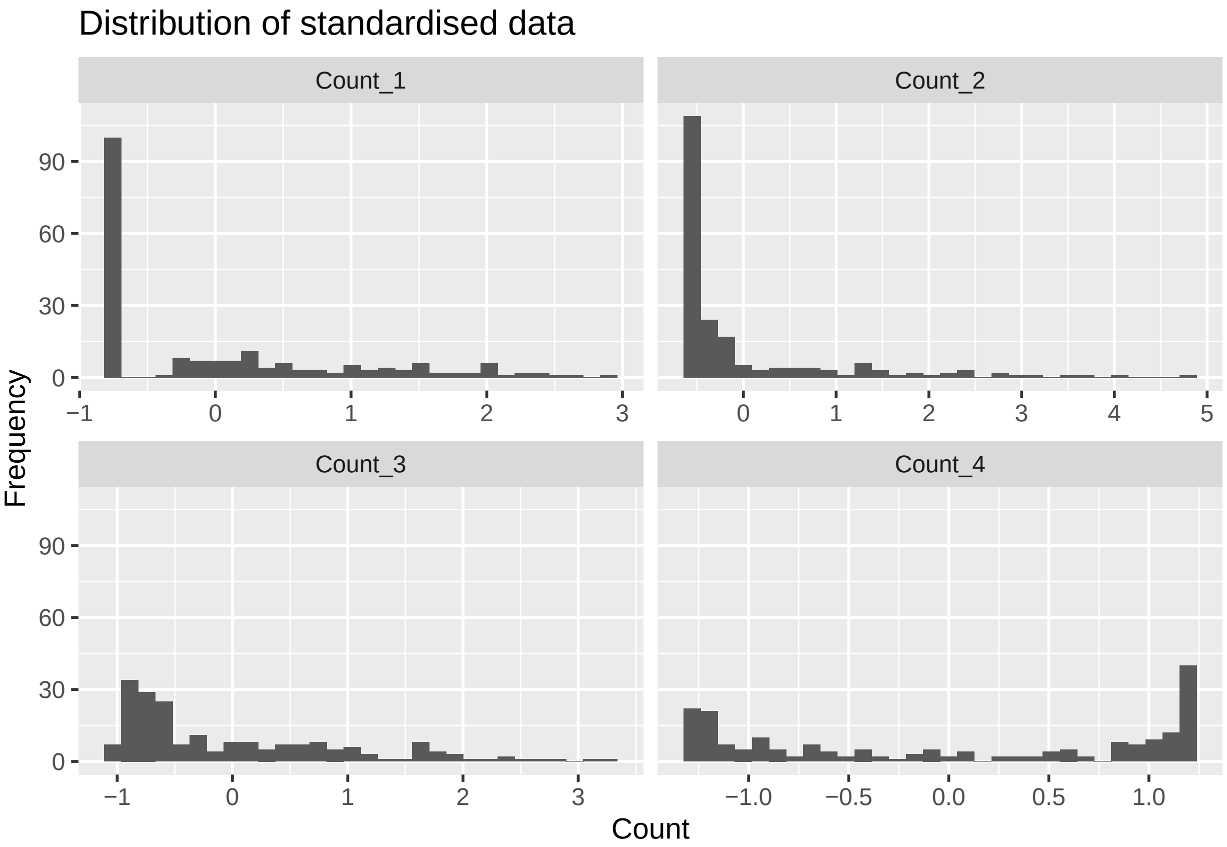
```r
ggplot(gather(log_data), aes(value)) +
  geom_histogram() +
  facet_wrap(~key, scales = "free_x") +
  labs(
    title = "Distribution of log-transformed data",
    x = "Count",
    y = "Frequency"
  )
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
ggplot(gather(scaled_data), aes(value)) +
  geom_histogram() +
  facet_wrap(~key, scales = "free_x") +
  labs(
    title = "Distribution of standardised data",
    x = "Count",
    y = "Frequency"
  )
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

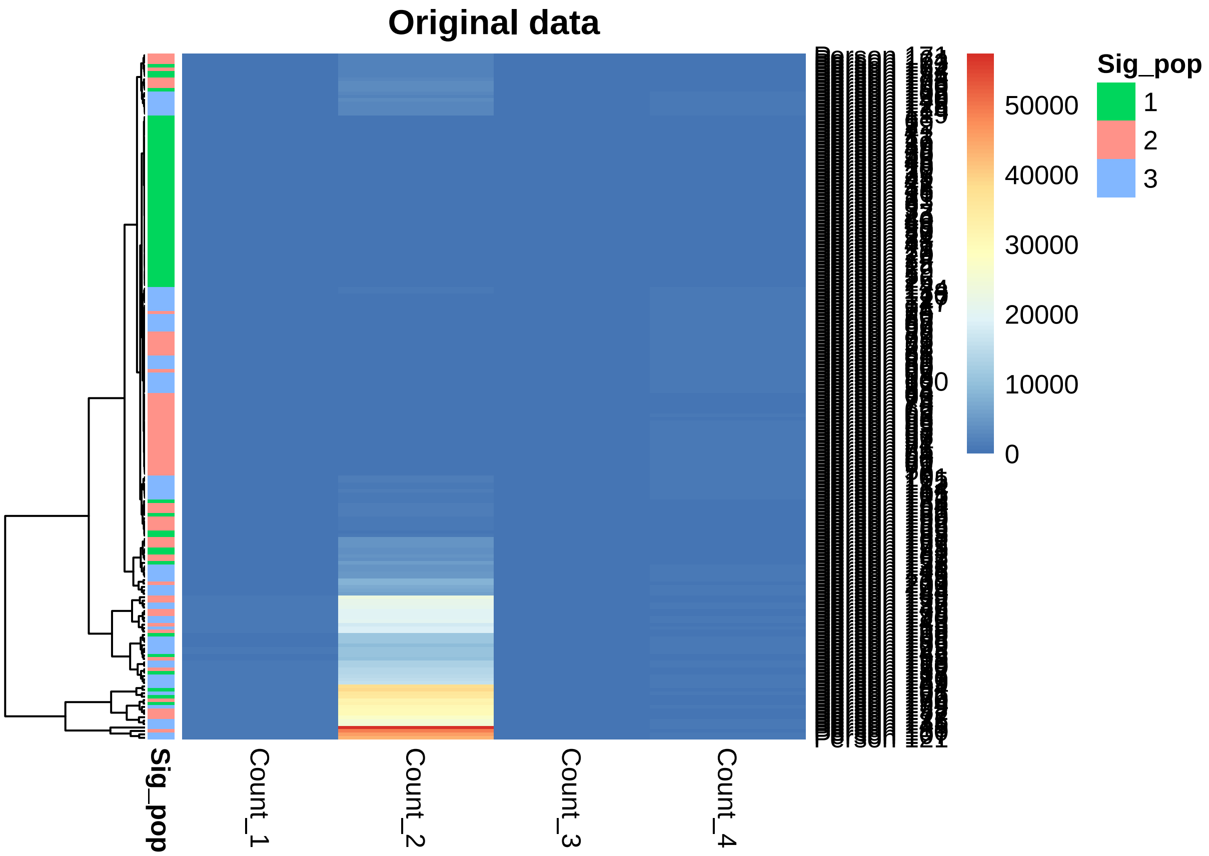


It appears that the `log`-transform has worked most effectively for the virst two variables, but does not succeed with the sigmoidal data.

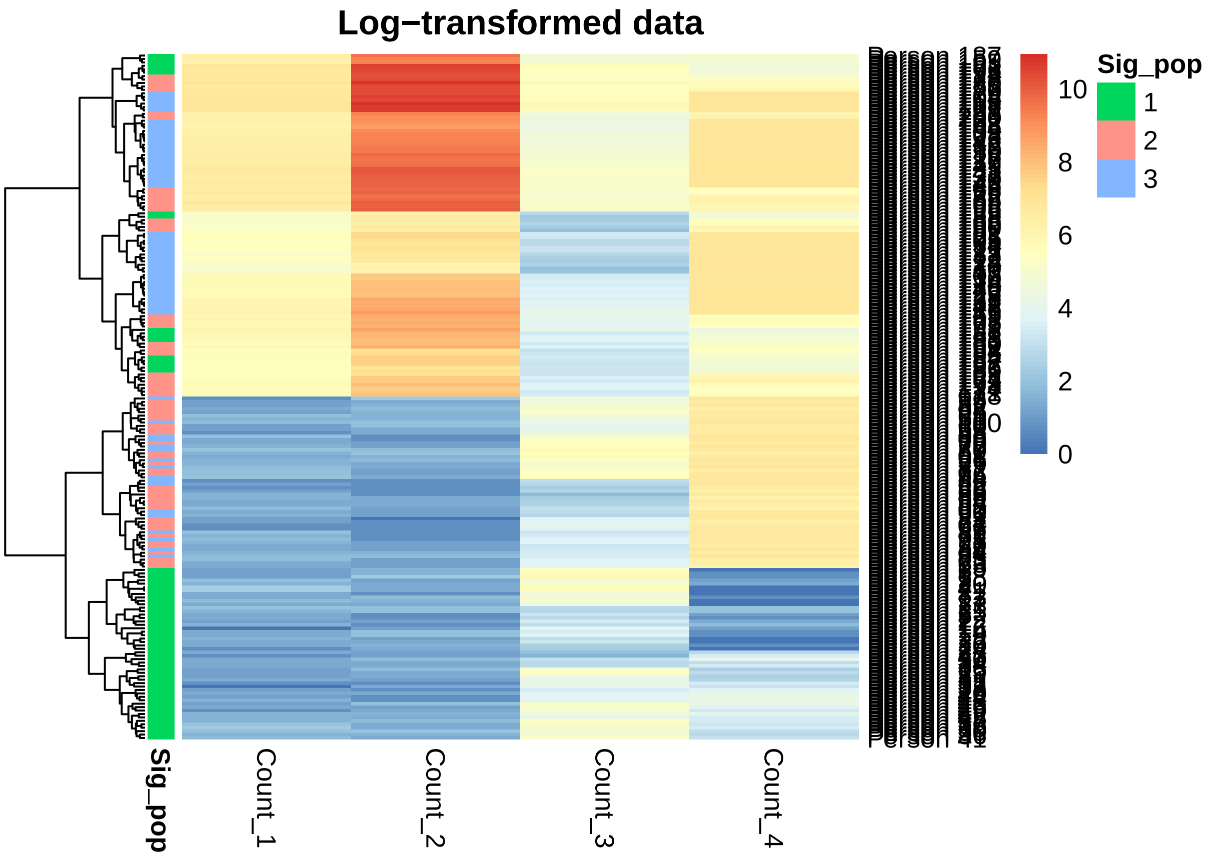
We will now attempt to infer the latent clsutering labels using mixture models. I use the `Mclust` function from the `mclust` package and the `mvnormalmixEM` from `mixtools` to create Gaussian mixture models. We will look at the clustering predicted for the data using `pheatmap`.

I create some labels for the sigmoidal data to keep track of the tails - we hope these are allocated correctly and are probably the hardest sub-populations to untangle for the `log`-transformed data. These labels are based on which tertiles the sigmoidal data falls into and is intended as a rough guide of how well the models deconstruct the sigmoidal data.

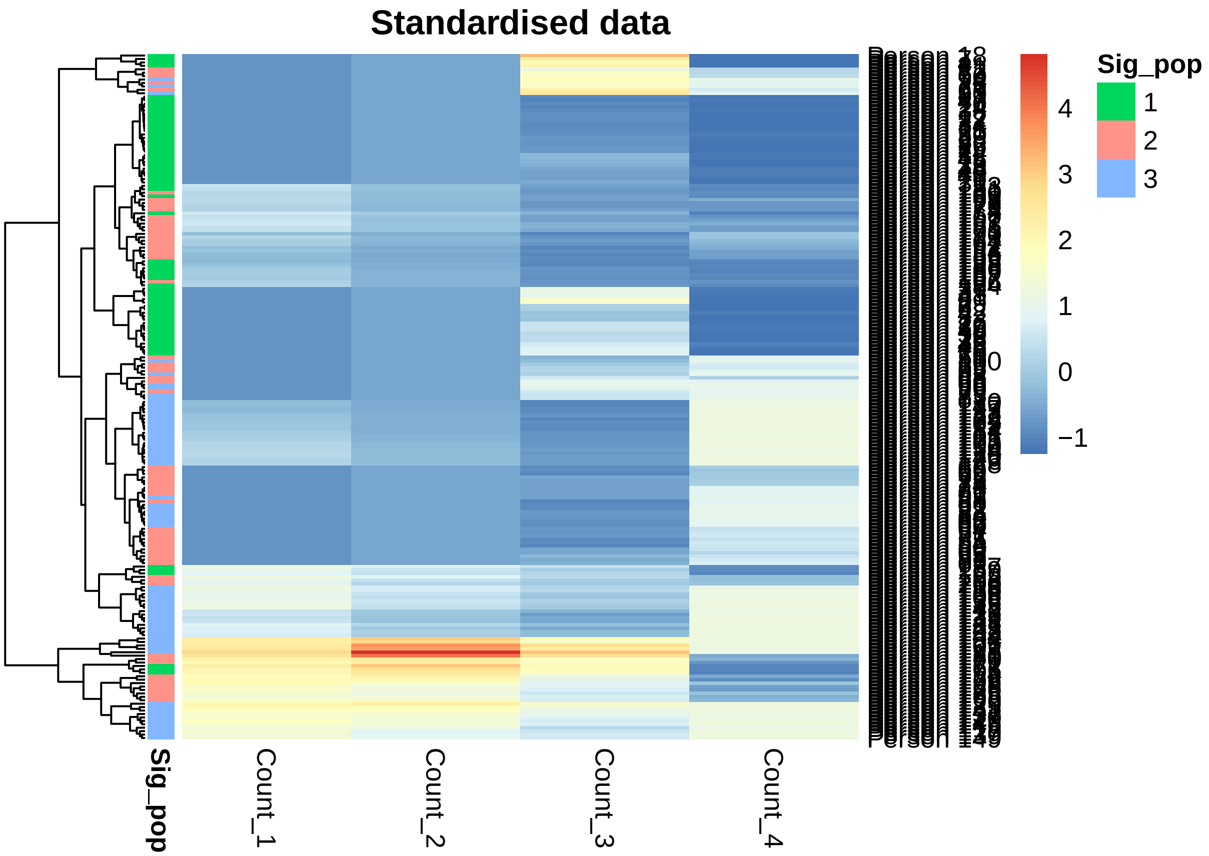
```r
# Keep track of the sigmoidal data by assigning a label based on quantiles
n_labels <- 3
sig_labels <- cut(my_data[, n_var],
  breaks = quantile(my_data[, n_var], probs = seq(0, 1, 1 / n_labels)),
  include.lowest = T,
  labels = 1:n_labels
)

# Create a data.frame to annotate the heatmaps
annotation_row <- data.frame(Sig_pop = as.factor(sig_labels))
row.names(annotation_row) <- row.names(my_data)

# We see that in the
pheatmap(my_data,
  annotation_row = annotation_row,
  cluster_cols = F,
  main = "Original data"
)
```



```
pheatmap(log_data,
  annotation_row = annotation_row,
  cluster_cols = F,
  main = "Log-transformed data"
)
```



```
pheatmap(scaled_data,
  annotation_row = annotation_row,
  cluster_cols = F,
  main = "Standardised data"
)
```



Model fitting

Now attempt to fit models. We find that `mixtools` is less robust to `mclust` (struggles to solve the same datasets). For this reason I recommend use of the `mclust` package and comment out the code for `mixtools` (as it crashes).

```
model_functions <- c(
  "Mclust",
  "mvnormalmixEM" # ,
  # "poisregmixEM"
)

transforms <- c(
  "Original",
  "Log",
  "Standardise"
)

datasets <- list(my_data, log_data, scaled_data) %>%
  set_names(transforms)

n_datasets <- length(datasets)
n_models <- length(model_functions)

model_out <- vector("list", n_models) %>%
  set_names(model_functions)
```

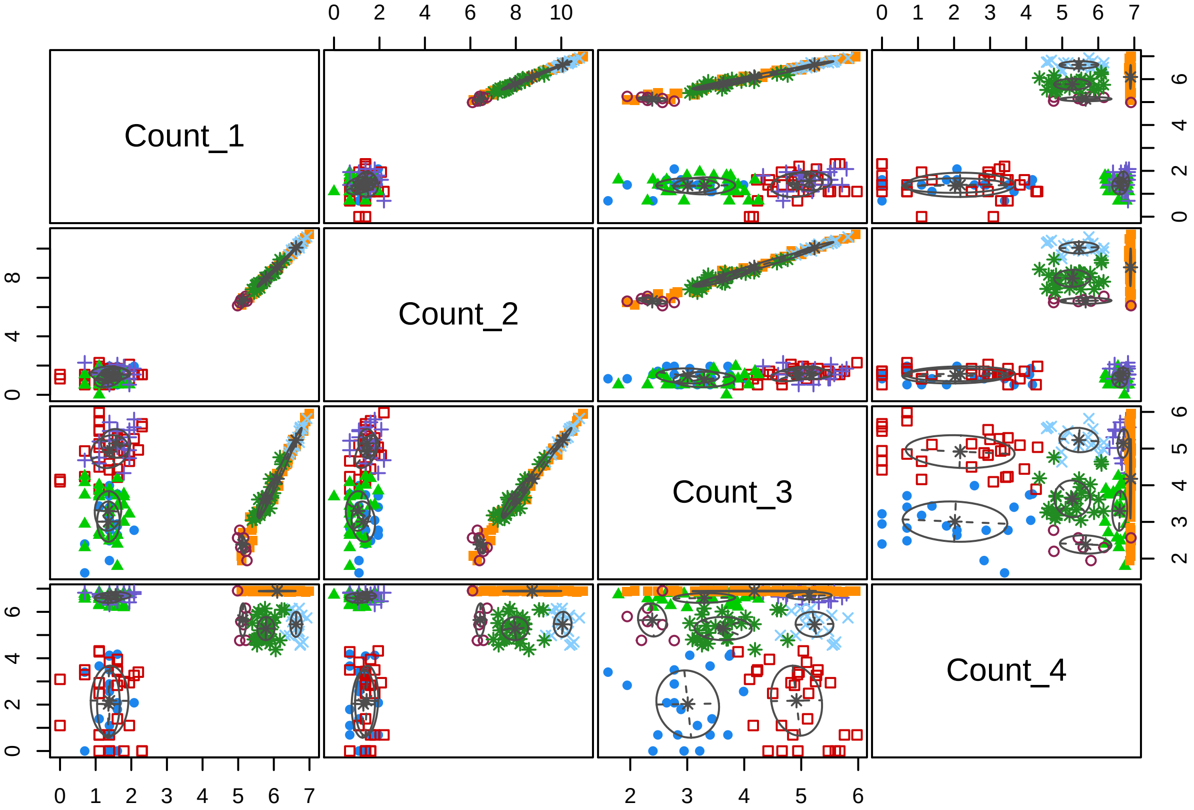
```
model_bic <- model_out

for (i in 1:n_models) {
  model_out[[i]] <- vector("list", n_datasets) %>%
    set_names(transforms)

  model_bic[[i]] <- model_out[[i]]

  if (model_functions[i] == "Mclust") {
    for (j in 1:n_datasets) {
      # do.call(model_functions[i], list(datasets[[j]])
      model_out[[i]][[j]] <- Mclust(datasets[[j]], G = 2:15)
      model_bic[[i]][[j]] <- mclustBIC(datasets[[j]])
    }
  }
  # if (model_functions[i] == "mvnormalmixEM") {
  #   for (j in 1:n_datasets) {
  #     # do.call(model_functions[i], list(datasets[[j]]), k = 15)
  #     model_out[[i]][[j]] <- mvnormalmixEM(datasets[[j]], k = 10)
  #   }
  # }
}
```

We can now inspect the model using several different visualisations. We can investigate the optimal number of components under the Bayesian Information Criterion (BIC) and we can look at the clusterings as defined by pairs of variables. I will look at the models defined on the scaled data and the `log`-transformed data. The BIC vs k plot also shows which of the possible types of covariance structure allowed by `Mclust` is optimal (this is the difference models listed and plotted).

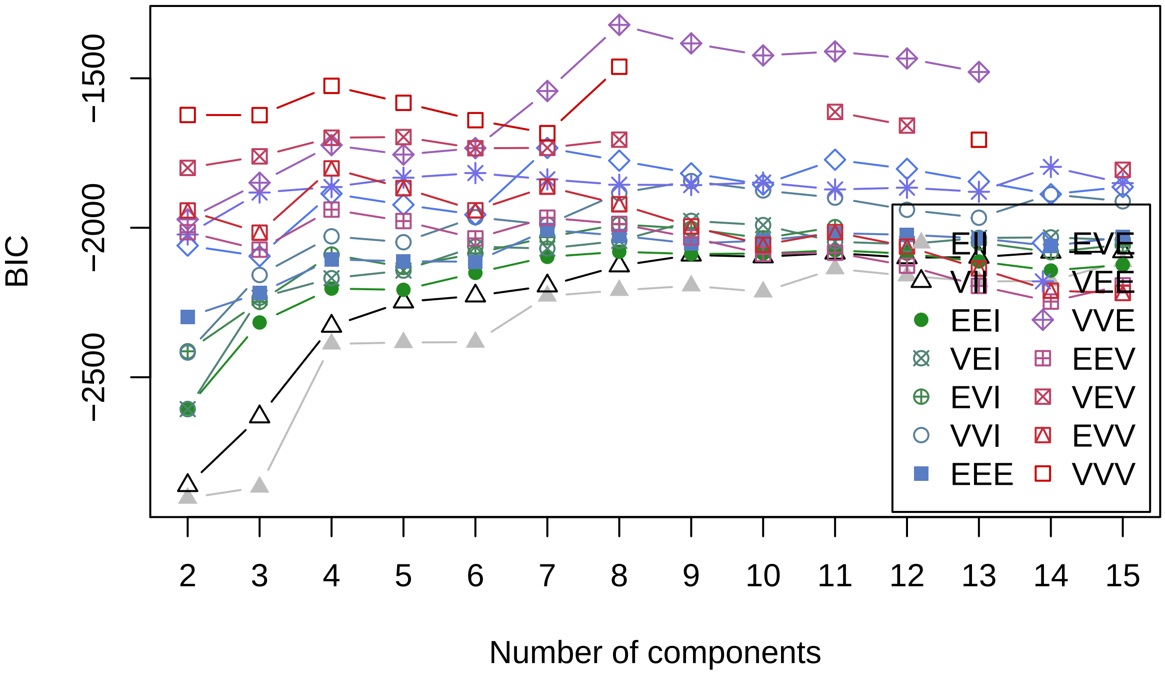
```
summary(model_out[[1]][[2]])
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVE (ellipsoidal, equal orientation) model with 8 components:
##
##  log-likelihood   n df       BIC       ICL
##       -456.7326 200 77 -1321.436 -1331.303
##
## Clustering table:
##  1  2  3  4  5  6  7  8
## 21 29 30 20 49 17  7 27
```

```
# mclustModel(datasets[[3]], model_bic[[1]][[2]])
plot(model_out[[1]][[2]], what = "classification")
```



```
plot(model_out[[1]][[2]], what = "BIC")
```

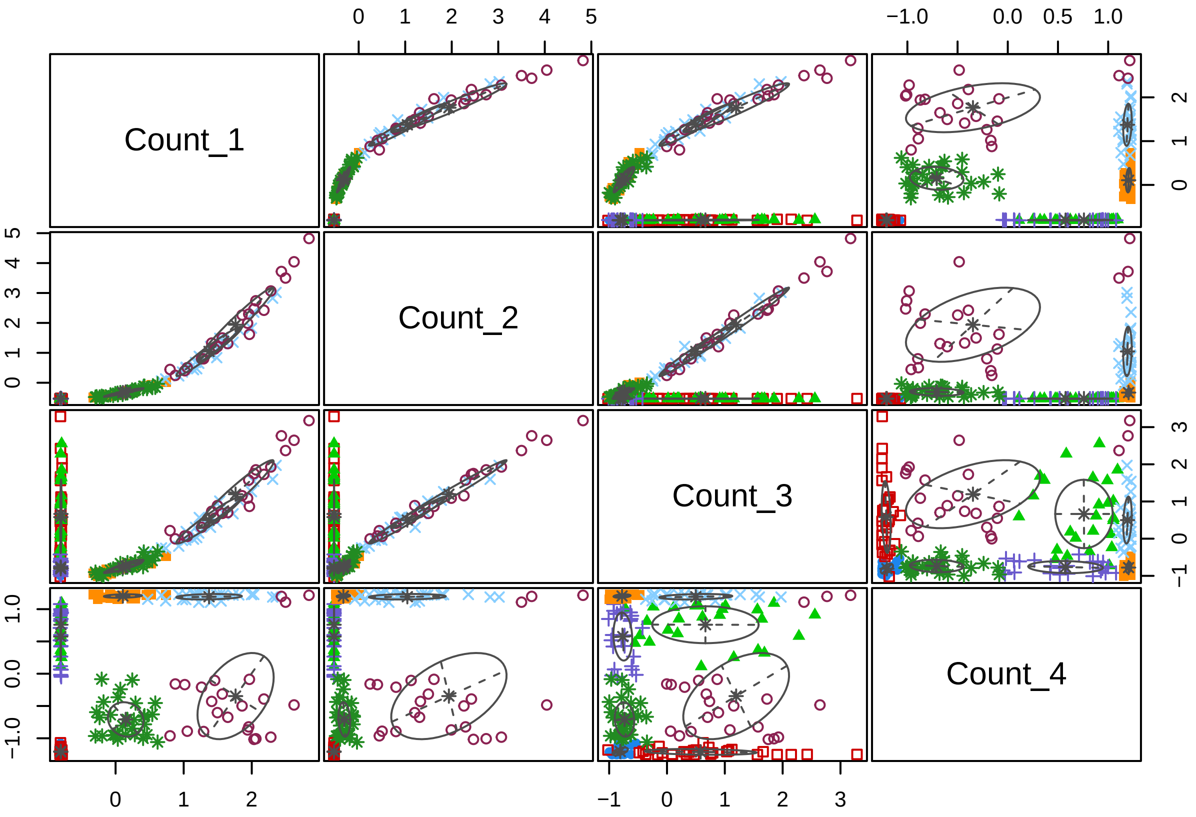


```
summary(model_out[[1]][[3]])
```

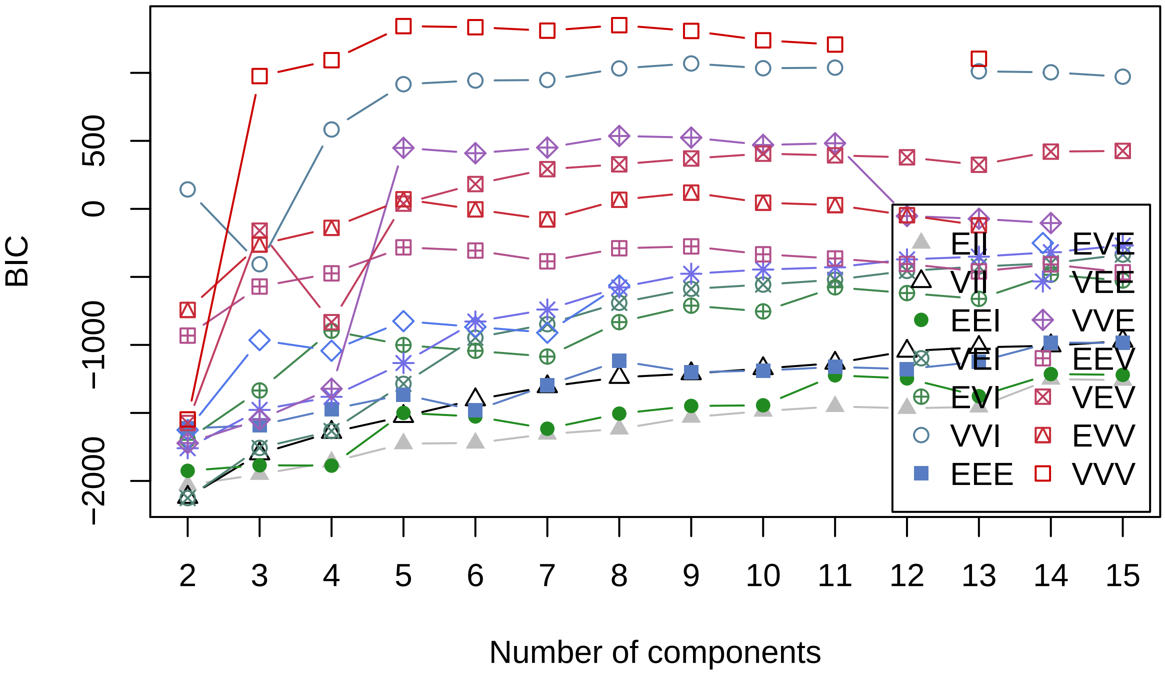
```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
```

```
##
## Mclust VVV (ellipsoidal, varying volume, shape, and orientation) model with 8
## components:
##
##  log-likelihood   n  df      BIC      ICL
##        990.4389 200 119 1350.378 1344.508
##
## Clustering table:
##  1  2  3  4  5  6  7  8
## 20 30 25 25 23 24 23 30
```

```
# mclustModel(datasets[[3]], model_bic[[1]][[3]])
plot(model_out[[1]][[3]], what = "classification")
```



```
plot(model_out[[1]][[3]], what = "BIC")
```



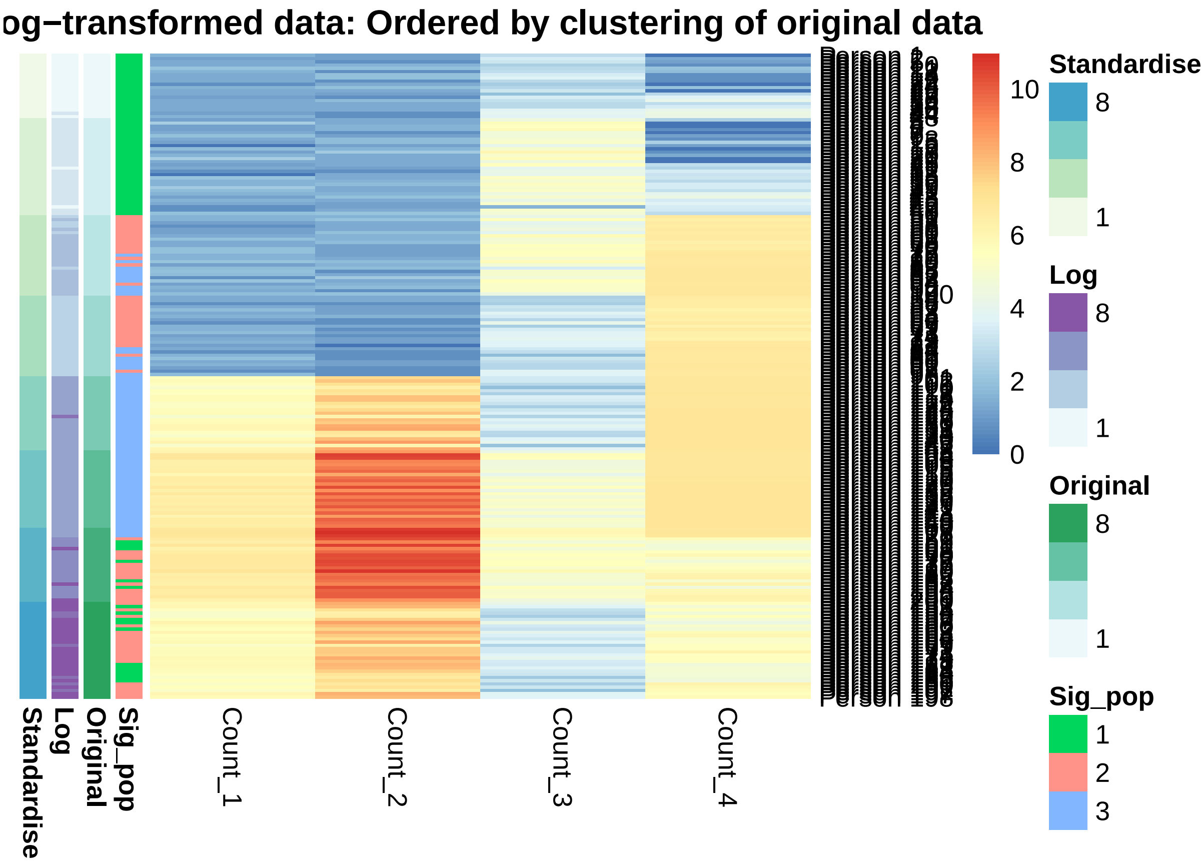
We can see that by using a vector of possible components (the input `G = 2:15` in the call to `Mclust`), we have captured the optimal value. This is the global maximum in the plot comparing BIC to number of components for each model allowed by `Mclust`. One thing to notice here is that the two models do agree on the optimal number of components (8), but that this is not expected to occurr, particularly for datasets of greater dimension. Another point is that the type of model that best fits the scaled data is the "EVE" model in comparison to the "VVV" model. This is a simpler model where each cluster has more restrictions on its parameters - thus the EVE model is easier to run (this is worth remembering if `log`-transformed data demands to complex a model computationally).

In the plots comparing clusterings across variables, we can see that the model defined on the `log`-transformed data separates the two sub-populations in $Count_1$ and $Count_2$ with much greater confidence than the model defined on the standardised data. One can also see that $Count_3$, which has no sub-population structure, is not contributing significantly to the cluster allocaitons - the data is pretty uniformly distributed across the axis defined by $Count_3$, no clear partitions emerging. One can see that the sigmoidal structure in $Count_4$ is captured quie well by the `log` model.

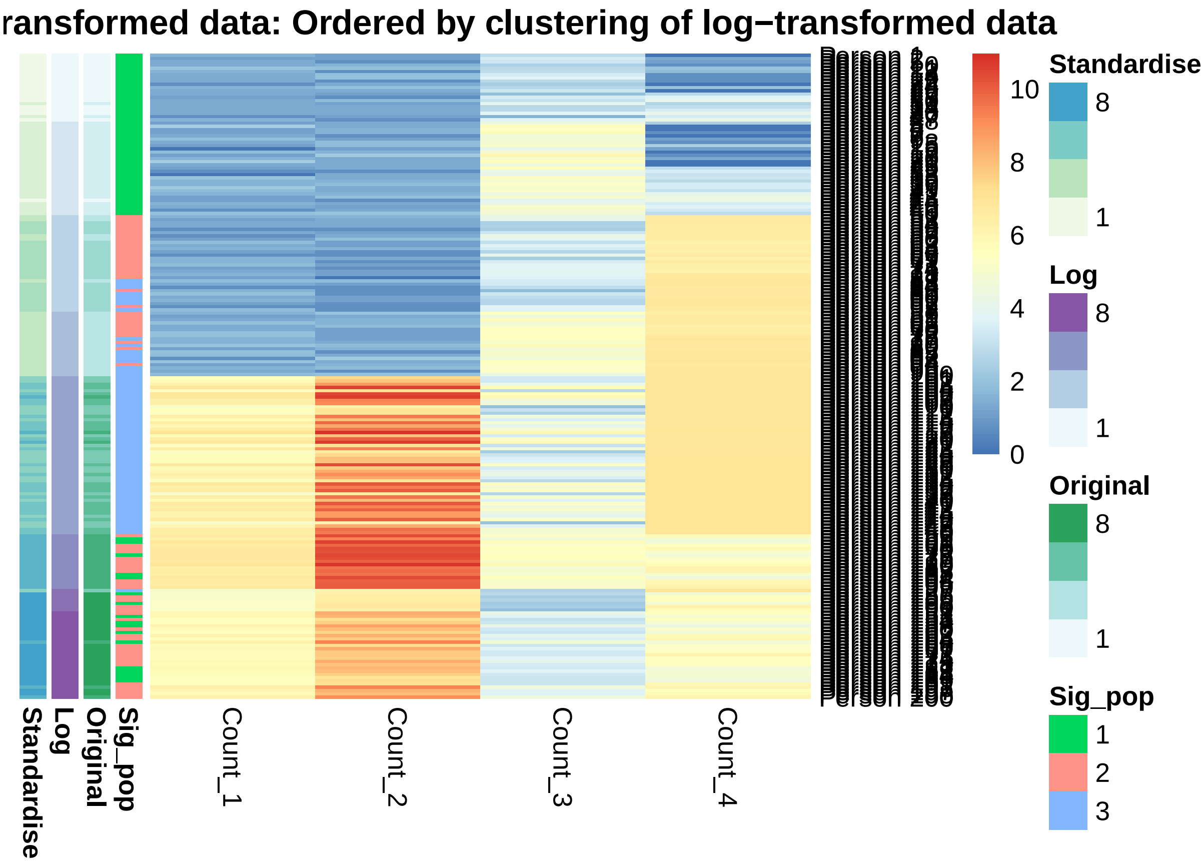
Results

Let us know inspect the clustering provided. I ignore the original data as the scaling makes inspecting the data unfeasible.

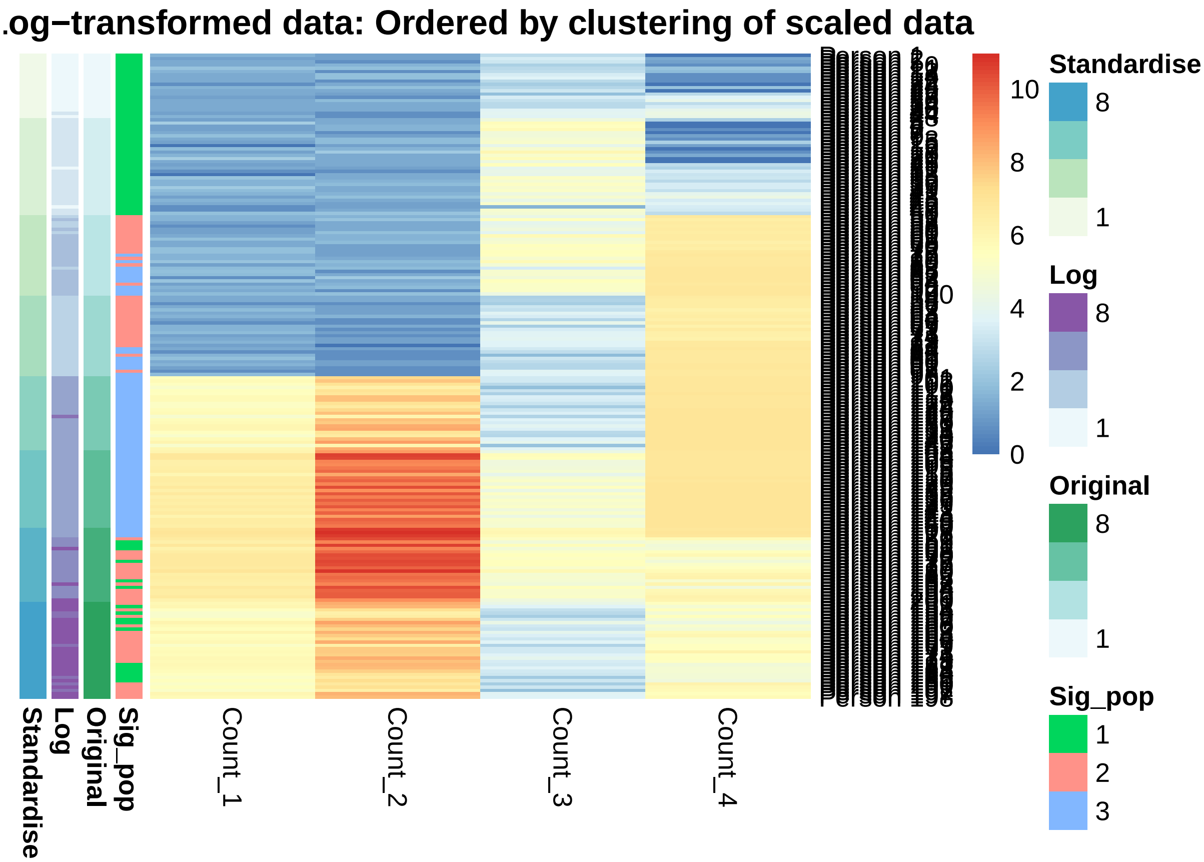
```r
labelling <- lapply(model_out$Mclust, function(x) {
  x$classification
})

annotation_row <- cbind(annotation_row, labelling)

# pheatmap(my_data[order(annotation_row[, 1]), ],
#   cluster_rows = F,
#   cluster_cols = F,
#   annotation_row = annotation_row,
#   main = "Original data: Ordered by Sigmoid tertiles"
# )
#
# pheatmap(my_data[order(annotation_row[, 2]), ],
#   cluster_rows = F,
#   cluster_cols = F,
#   annotation_row = annotation_row,
#   main = "Original data: Ordered by clustering of original data"
# )
#
# pheatmap(my_data[order(annotation_row[, 3]), ],
#   cluster_rows = F,
#   cluster_cols = F,
#   annotation_row = annotation_row,
#   main = "Original data: Ordered by clustering of log-transformed data"
# )
#
# pheatmap(my_data[order(annotation_row[, 4]), ],
#   cluster_rows = F,
#   cluster_cols = F,
#   annotation_row = annotation_row,
#   main = "Original data: Ordered by clustering of scaled data"
# )
#
# pheatmap(log_data[order(annotation_row[, 1]), ],
#   cluster_rows = F,
#   cluster_cols = F,
#   annotation_row = annotation_row,
#   main = "Log-transformed data: Ordered by Sigmoid tertiles"
# )

pheatmap(log_data[order(annotation_row[, 2]), ],
  cluster_rows = F,
  cluster_cols = F,
  annotation_row = annotation_row,
  main = "Log-transformed data: Ordered by clustering of original data"
)
```



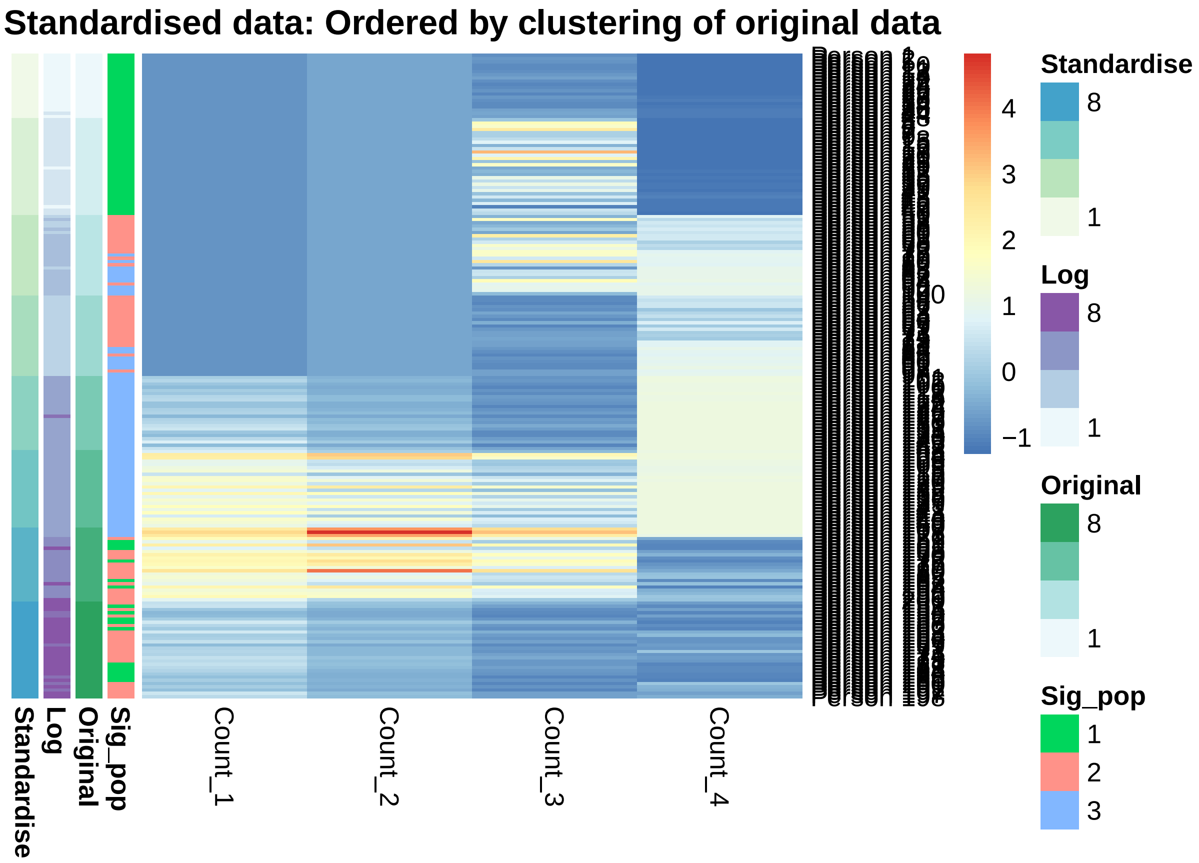
```r
pheatmap(log_data[order(annotation_row[, 3]), ],
  cluster_rows = F,
  cluster_cols = F,
  annotation_row = annotation_row,
  main = "Log-transformed data: Ordered by clustering of log-transformed data"
)
```



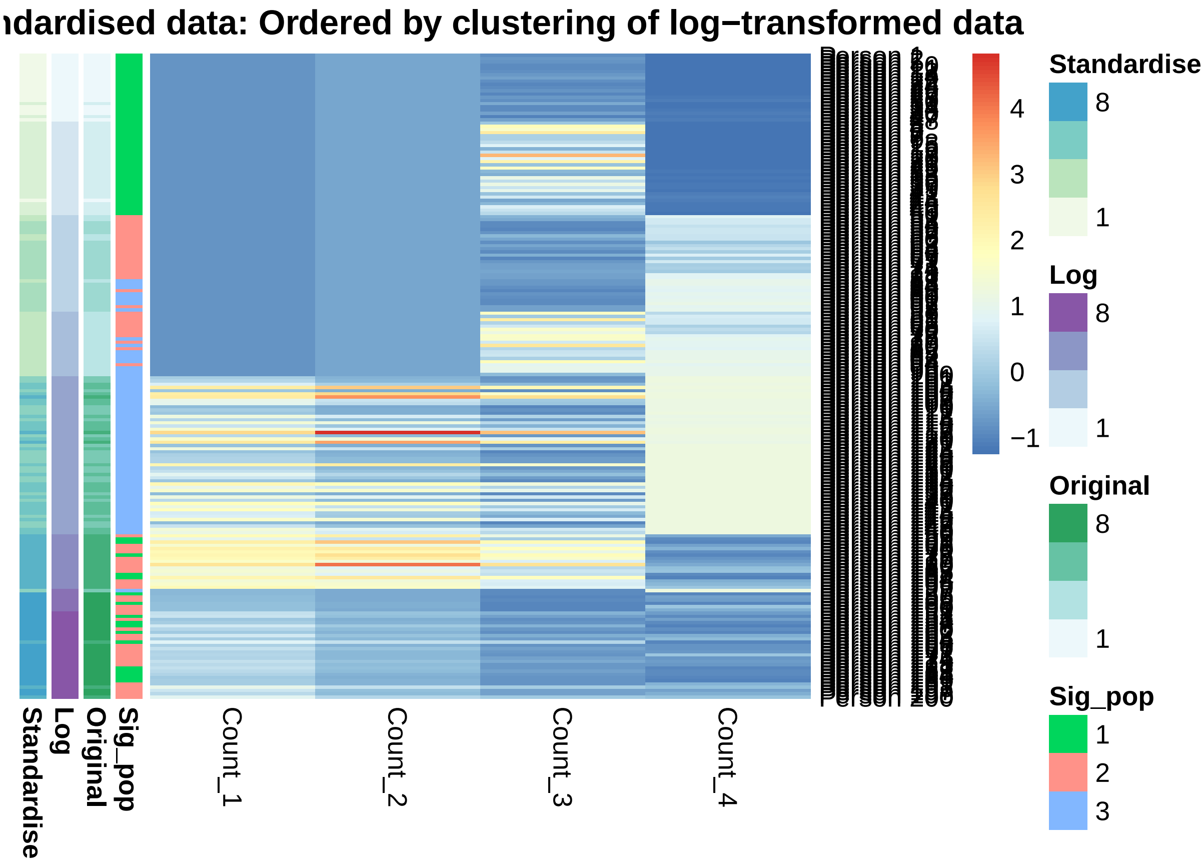
```
pheatmap(log_data[order(annotation_row[, 4]), ],
  cluster_rows = F,
  cluster_cols = F,
  annotation_row = annotation_row,
  main = "Log-transformed data: Ordered by clustering of scaled data"
)
```



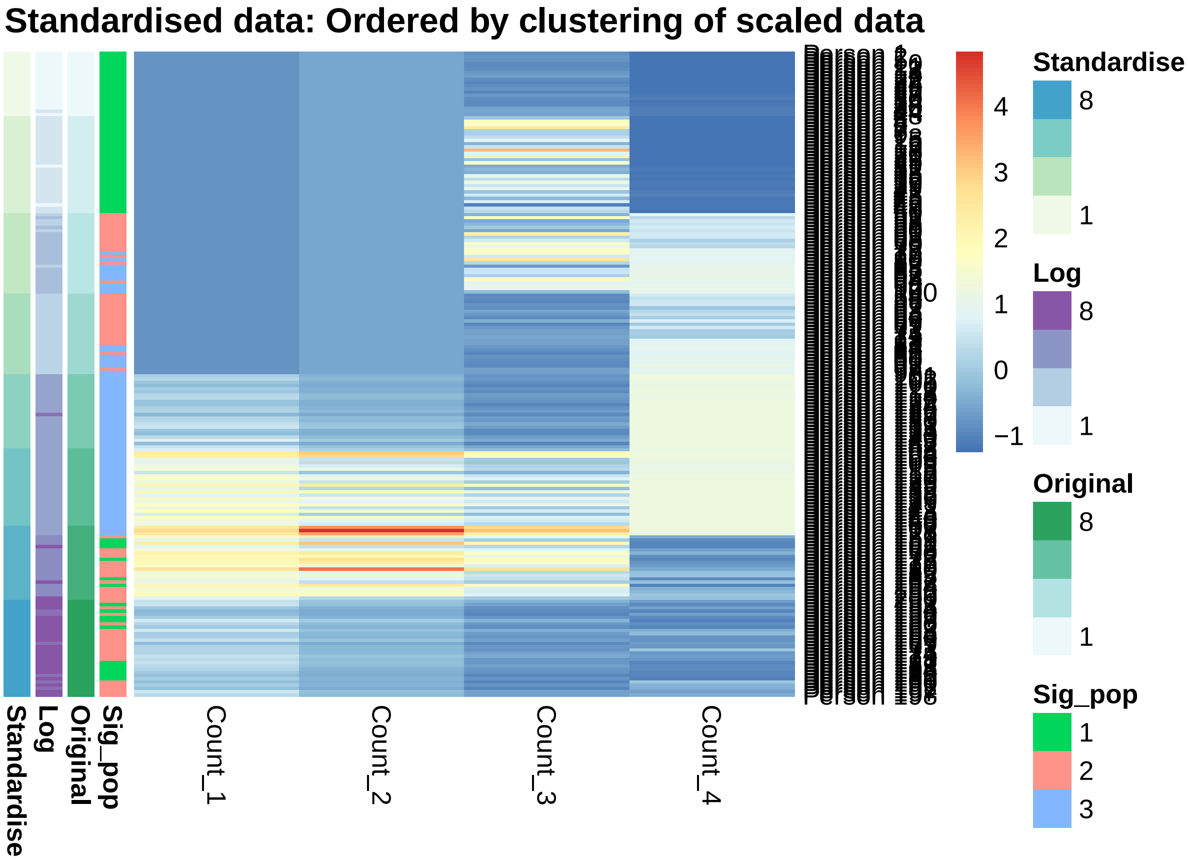
```
# # Standardised data
# pheatmap(scaled_data[order(annotation_row[, 1]), ],
#   cluster_rows = F,
#   cluster_cols = F,
#   annotation_row = annotation_row,
#   main = "Standardised data: Ordered by Sigmoid tertiles"
# )

pheatmap(scaled_data[order(annotation_row[, 2]), ],
  cluster_rows = F,
  cluster_cols = F,
  annotation_row = annotation_row,
  main = "Standardised data: Ordered by clustering of original data"
)
```



```
pheatmap(scaled_data[order(annotation_row[, 3]), ],
  cluster_rows = F,
  cluster_cols = F,
  annotation_row = annotation_row,
  main = "Standardised data: Ordered by clustering of log-transformed data"

)
```



```r
pheatmap(scaled_data[order(annotation_row[, 4]), ],
  cluster_rows = F,
  cluster_cols = F,
  annotation_row = annotation_row,
  main = "Standardised data: Ordered by clustering of scaled data"
)
```



Here I have not investigated the best fit; this analysis is to see if the structure in the different distributions can be recognised in each transformed dataset. The scaled data and the original data have no difference in behaviour. The log-transformed data disagrees more so with the final clustering, with approximately 1 in 4 points allocated to a different cluster than in the scaled data; however one of these is a cluster that is split into two components, these components align very well with one larger component in the clustering from the scaled data.

Summary

We have seen in the plots that the `log` transform most successfully deconstruts the count data into the sub-populations we used to generate the data. In our clustering results we see that the allocations provided by a mixture model in the `log`-transformed data do align quite well with the labels from the sigmoidal tertiles (in combination with the signals from the other variables). Based on these results I believe that the `log` transform is probably optimal in this data setting when combined with mixture models.

We also saw that the `Mclust` function could run where the `mvnormalmixEM` could not, suggesting a more robust piece of software.

Thus, I recommend use of `mclust` and the `log` transform.