WGCNA

Amulya Saini

2024-03-08

Load required packages

```
library("WGCNA")
library("tidyverse")
library("dplyr")
library("magrittr")
library("tidyr")
```

Loading the gene expression data and the traits data from the csv files

```
# loading the expression data
exp_data <- read.csv("C:\\Users\\saini\\Downloads\\HW03_expression.csv", header = TRUE)
dim(exp_data)
```

```
## [1] 3055  136
```

```
# loading the Traits data
traits_data <- read.csv("C:\\Users\\saini\\Downloads\\HW03_traits.csv", header = TRUE)
dim(traits_data)
```

```
## [1] 135   3
```

Pre-processing and removing of NA values from both the dataframes

```
# dropping the NA values from traits data
traits_data <- na.omit(traits_data)

# setting the "ID" column as row names
traits_data <- transform(traits_data, row.names = ID)

# removing the original column "ID"
traits_data <- traits_data[, -1]

# extracting the row names from the dataset
traits_data_rownames <- rownames(traits_data)

# getting the column names of exp_data besides gene_symbol
exp_data_colnames <- colnames(exp_data)[-1]

# extracting the column names that are not present in traits data
columns_to_drop <- setdiff(exp_data_colnames, traits_data_rownames)

# dropping the columns that we donot have traits data for
exp_data <- exp_data[, !(colnames(exp_data) %in% columns_to_drop)]

# Drop rows with NA values
exp_data <- na.omit(exp_data)
dim(exp_data)
```

```
## [1] 2418  132
```

```
# Temporarily store the original 'cor' function from WGCNA pipeline
temp_cor <- cor  # Store the current correlation function in a temporary variable
cor <- WGCNA::cor  # Replace the 'cor' function with the one from the WGCNA package
```

Checking if the gene expression data is normalized.In a normalized distribution, the mean is usually close to the median, and the standard deviation is around 1. When a histogram is plotted with normalized data, the distribution often resembles a bell-shaped curve.

```r
# getting the column names
variables <- names(exp_data)
# setting the number of histograms per sheet
histograms_per_sheet <- 4
# Calculating the number of sheets needed
num_sheets <- ceiling(length(variables) / histograms_per_sheet)
# Looping through each variable and creating a histogram
for (i in 1:length(variables)) {
  # Exclude the "gene_symbol" column
  if (variables[i] != "gene_symbol") {
    # Checking if the variable is numeric
    if (is.numeric(exp_data[[variables[i]]])) {
      # Calculating the position in the layout
      sheet_position <- ((i - 1) %/% histograms_per_sheet) + 1
      plot_position <- (i - 1) %% histograms_per_sheet + 1
      # Setting up the layout for each sheet
      if (plot_position == 1) {
        par(mfrow = c(4, 4))
      }
      # Creating a new plot for each numeric variable
      hist(exp_data[[variables[i]]], main = paste("Histogram of", variables[i]), xlab = variables[i])
      # Adding a newline after every 4 histograms
      if (plot_position %% histograms_per_sheet == 0 || i == length(variables)) {
        cat("\n")
        par(mfrow = c(1, 1))  # Reset the layout for each sheet
      }
    } else {
      # Print a message if the variable is not numeric
      cat("Variable", variables[i], "is not numeric and cannot be plotted as a histogram.\n")
    }
  }
}
```

From the histograms, the data looks normalized.

The data needs to be transposed for WGCNA, meaning genes are columns and rows are samples/conditions/treatment.

```r
# making the gene_symbol values unique using make.names
exp_data$gene_symbol <- make.names(exp_data$gene_symbol, unique = TRUE)

# setting the first column as row names
row.names(exp_data) <- exp_data[, 1]

# Remove the first column from the dataframe
exp_data <- exp_data[, -1]

# Transposing the data
input_mat = t(exp_data)
```

The pre-processing of gene expression data is done.

In the context of Weighted Gene Co-expression Network Analysis (WGCNA), a soft threshold is a parameter used to transform the pairwise gene expression correlation values into a measure of connection strength between genes. This transformation is crucial for constructing a weighted network, where edges represent the strength of the relationship between genes.The choice of the soft thresholding power is crucial, as it influences the sensitivity of the analysis to different types of gene-gene relationships.

Choosing a Soft threshold

```r
# Choosing a set of soft-thresholding powers
powers <- c(c(1:10), seq(from = 12, to = 20 , by = 2))
powers
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 12 14 16 18 20
```

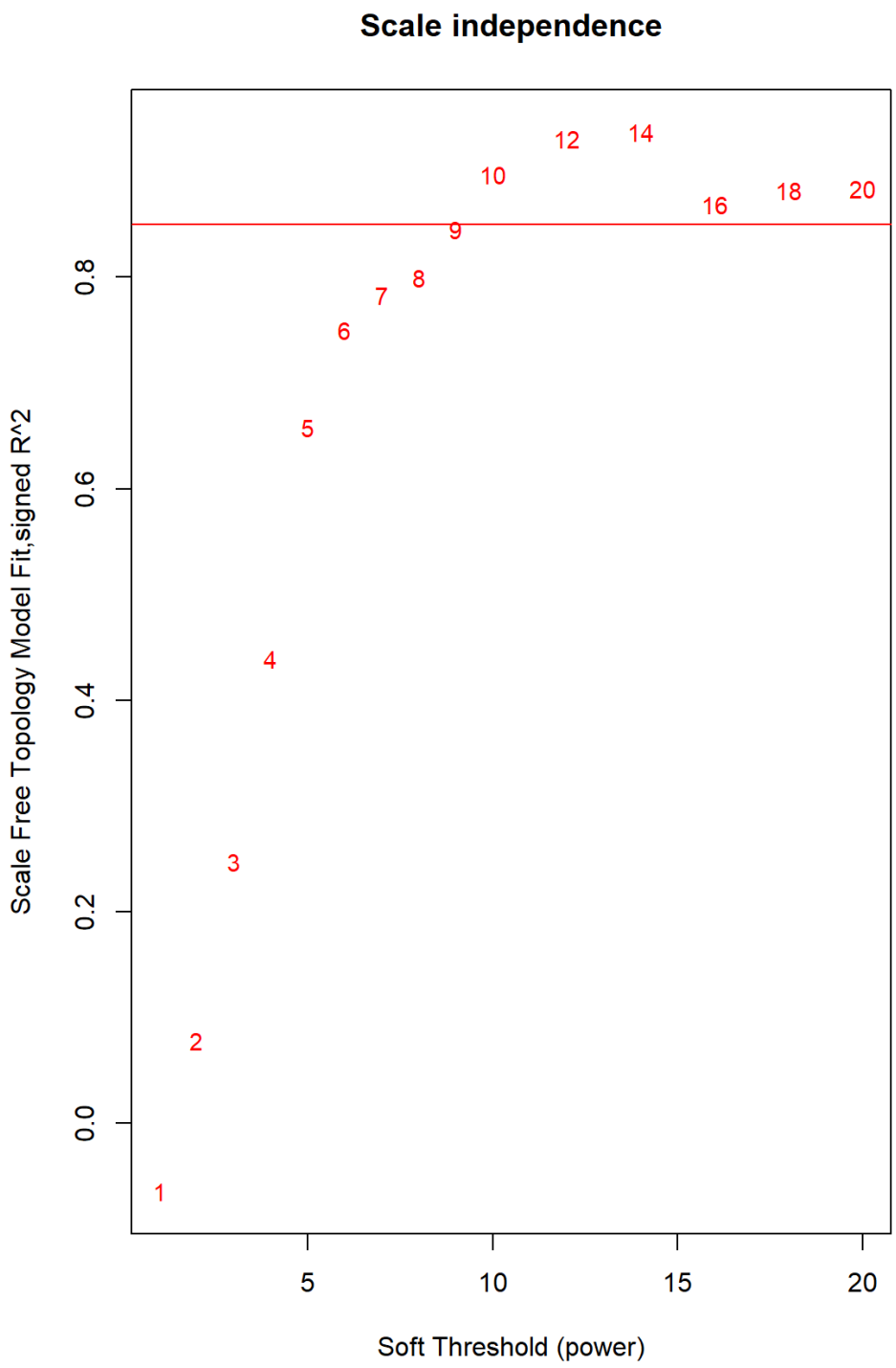
```r
# Call the network topology analysis function
sft <- pickSoftThreshold(input_mat, powerVector = powers, verbose = 5)
```

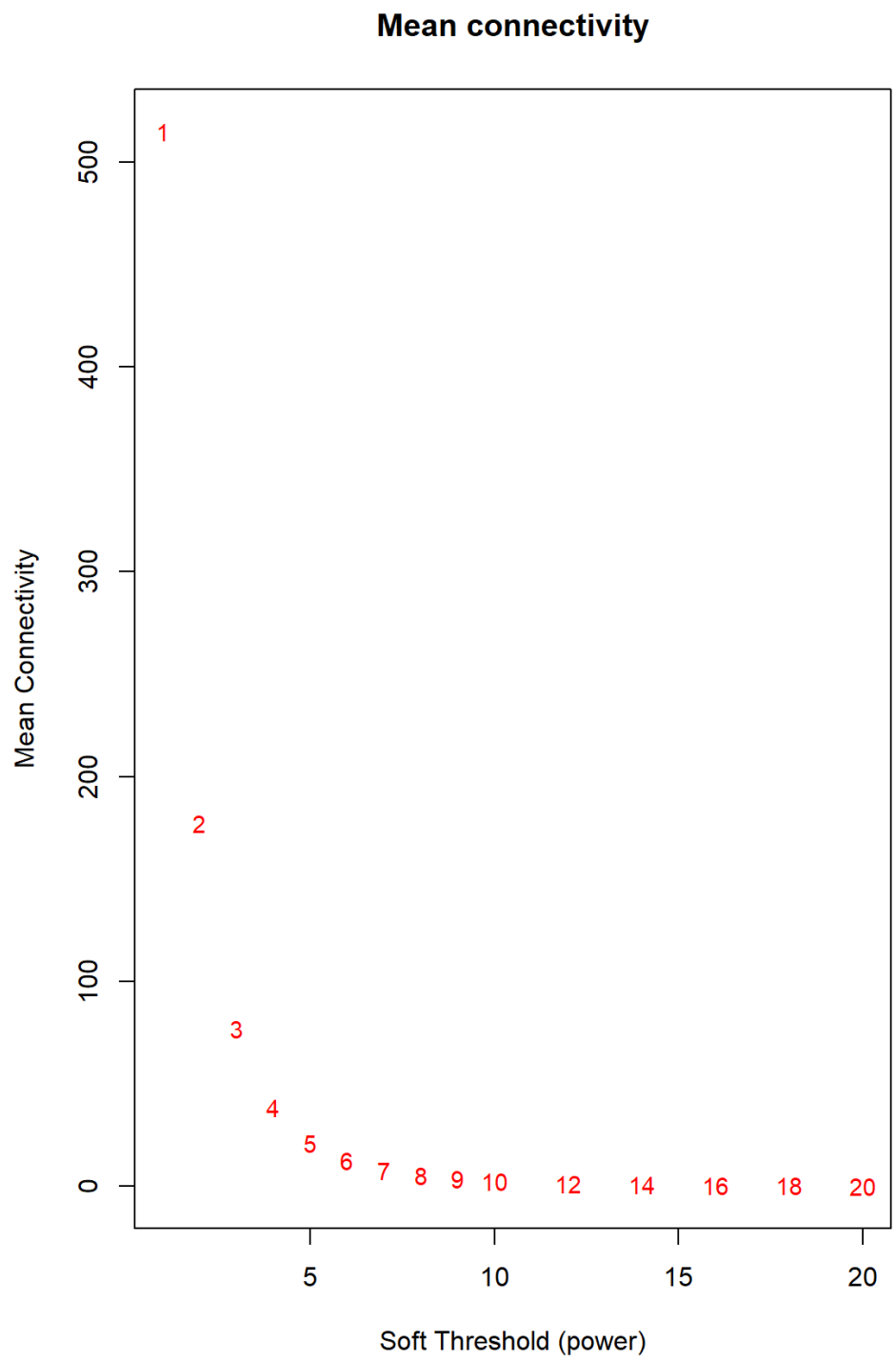
```
## pickSoftThreshold: will use block size 2418.
##  pickSoftThreshold: calculating connectivity for given powers...
##    ..working on genes 1 through 2418 of 2418
##    Power SFT.R.sq  slope truncated.R.sq mean.k. median.k. max.k.
## 1      1   0.0643  0.515          0.408 515.000  527.0000 813.00
## 2      2   0.0781 -0.411          0.834 177.000  174.0000 379.00
## 3      3   0.2470 -0.762          0.968  77.100   71.8000 206.00
## 4      4   0.4390 -1.200          0.983  38.600   33.4000 126.00
## 5      5   0.6580 -1.700          0.982  21.400   17.6000  82.70
## 6      6   0.7500 -1.890          0.986  12.700   10.0000  56.70
## 7      7   0.7830 -1.970          0.986   8.050    5.9500  40.20
## 8      8   0.8000 -1.960          0.987   5.330    3.7500  29.30
## 9      9   0.8450 -1.800          0.987   3.680    2.4500  21.80
## 10    10   0.8980 -1.720          0.989   2.620    1.6100  17.30
## 11    12   0.9310 -1.880          0.965   1.450    0.7550  14.50
## 12    14   0.9370 -1.870          0.929   0.879    0.3710  12.60
## 13    16   0.8690 -1.790          0.836   0.574    0.1940  11.10
## 14    18   0.8820 -1.720          0.867   0.398    0.1040   9.97
## 15    20   0.8840 -1.610          0.890   0.290    0.0583   9.06
```

```
sft_data <- sft$fitIndices
```

Plotting the above results

```
# arranging the plots in a single row with two columns
par(mfrow = c(1,2))

# Setting some parameters for text size
cex1 = 0.9

# plotting the scale free topology fit index aganist the soft thresold power
plot(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2],
     xlab="Soft Threshold (power)",
     ylab="Scale Free Topology Model Fit,signed R^2",
     type="n",
     main = paste("Scale independence"))

# adding labels to the points on the plot indicating the soft-thresholding powers
text(sft$fitIndices[,1], -sign(sft$fitIndices[,3]) * sft$fitIndices[,2],
     labels=powers,
     cex=cex1,
     col="red")

# adding a horizontal line at R^2 cut-off of 0.90
abline(h=0.85,col="red")

# alotting Mean Connectivity against the soft-thresholding power
plot(sft$fitIndices[,1], sft$fitIndices[,5],
     xlab="Soft Threshold (power)",
     ylab="Mean Connectivity",
     type="n",
     main = paste("Mean connectivity"))

# adding labels to the points on the plot indicating the soft-thresholding powers
text(sft$fitIndices[,1], sft$fitIndices[,5],
     labels=powers,
     cex=cex1,
     col="red")
```





we need to pick a soft threshold that has maximum Rsquare and minimum mean connectivity.I picked 10 here.

```
# the selected soft threshold power
picked_power = 10
```

The gene co-expression data is subjected to soft thresholding to construct an adjacency matrix, emphasizing strong gene relationships.The adjacency matrix is then transformed into a Topological Overlap Matrix (TOM), capturing the similarity in overall connection patterns.

A dissimilarity matrix is derived from the TOM, measuring how dissimilar genes are in terms of their co-expression patterns (dissTOMadj <- 1- TOMadj).

Hierarchical clustering is applied to the dissimilarity matrix, grouping genes based on their co-expression relationships.The resulting hierarchical clustering tree (dendrogram) is plotted, providing insights into how genes cluster together based on their co-expression profiles.

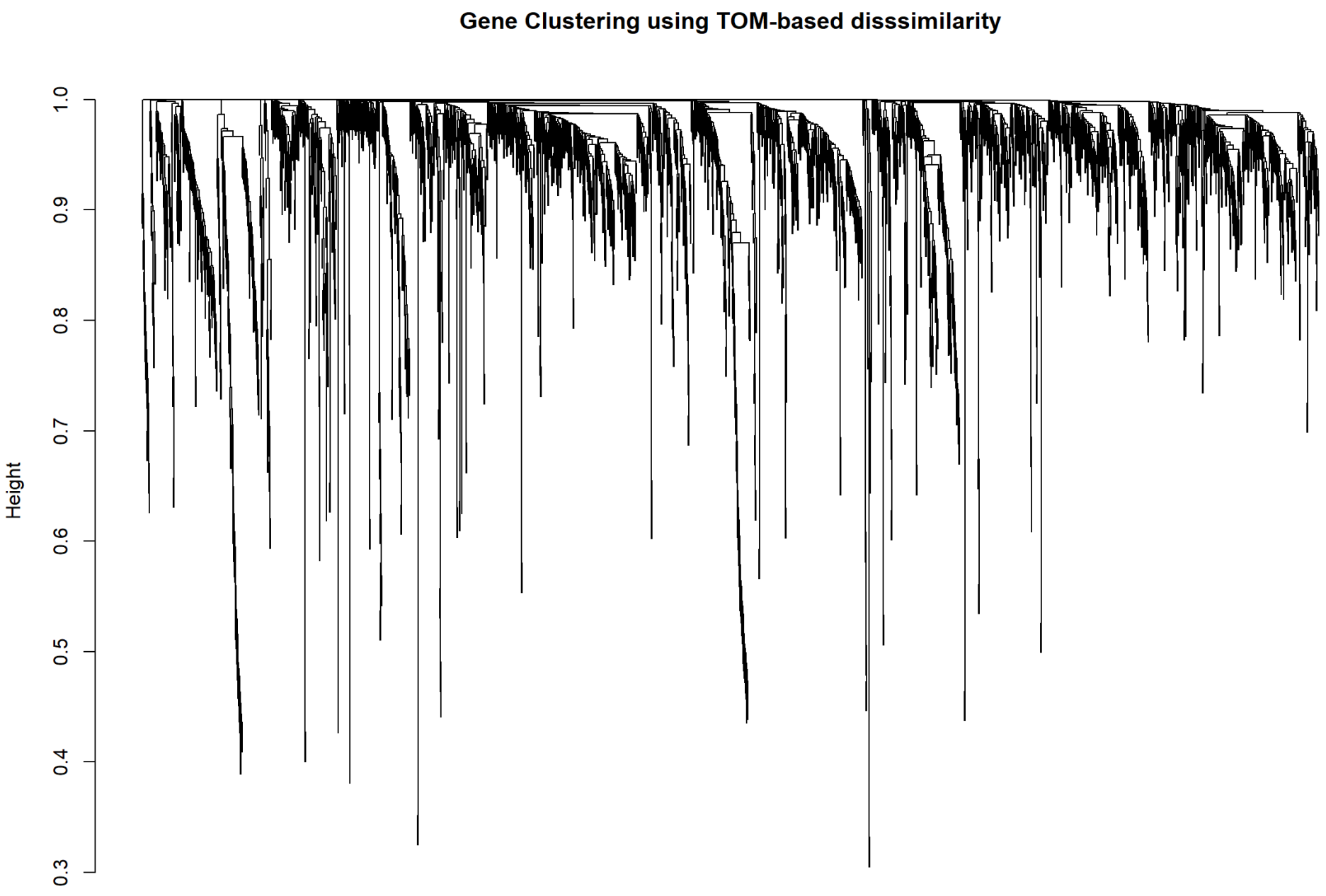
Gene clustering on TOM based dissimilarity

```
# constructing an adjacency matrix, using the predetermined soft threshold
adjacency <- adjacency(input_mat,
                       power = picked_power)

# Calculating the similarity matrix using topological overlap
TOMadj <- TOMsimilarity(adjacency)
```

```
## ..connectivity..
## ..matrix multiplication (system BLAS)..
## ..normalization..
## ..done.
```

```
# Generating the dissimilarity matrix from the topological overlap matrix
dissTOMadj <- 1- TOMadj

# Performing hierarchical clustering using the average linkage method
hclustGeneTree <- hclust(as.dist(dissTOMadj),
                         method = "average")
```

```
# Plotting the resulting hierarchical clustering tree (dendrogram)
plot(hclustGeneTree,
     xlab = "",
     sub = "",
     main = "Gene Clustering using TOM-based disssimilarity",
     labels = FALSE,
     hang = 0.04)
```



Performing module identification and visualization on a hierarchical clustering dendrogram (hclustGeneTree) using a dynamic tree cut approach.The resulting module IDs are converted into colors for visualization, and tables are generated to show the counts of genes in each module. Finally, the dendrogram is plotted with modules highlighted by different colors, providing an informative visualization of gene clustering.

```
# Setting the minimum size a module should have; a higher value excludes smaller modules
minModuleSize <- 30

# Using dynamic tree cut to assign module IDs based on hierarchical clustering
dynamicMods <- cutreeDynamic(dendro = hclustGeneTree,
                             distM = dissTOMadj,
                             deepSplit = 2,
                             pamRespectsDendro = FALSE,
                             minClusterSize = minModuleSize)
```

```
##  ..cutHeight not given, setting it to 0.997  ===>  99% of the (truncated) height range in dendro.
##  ..done.
```

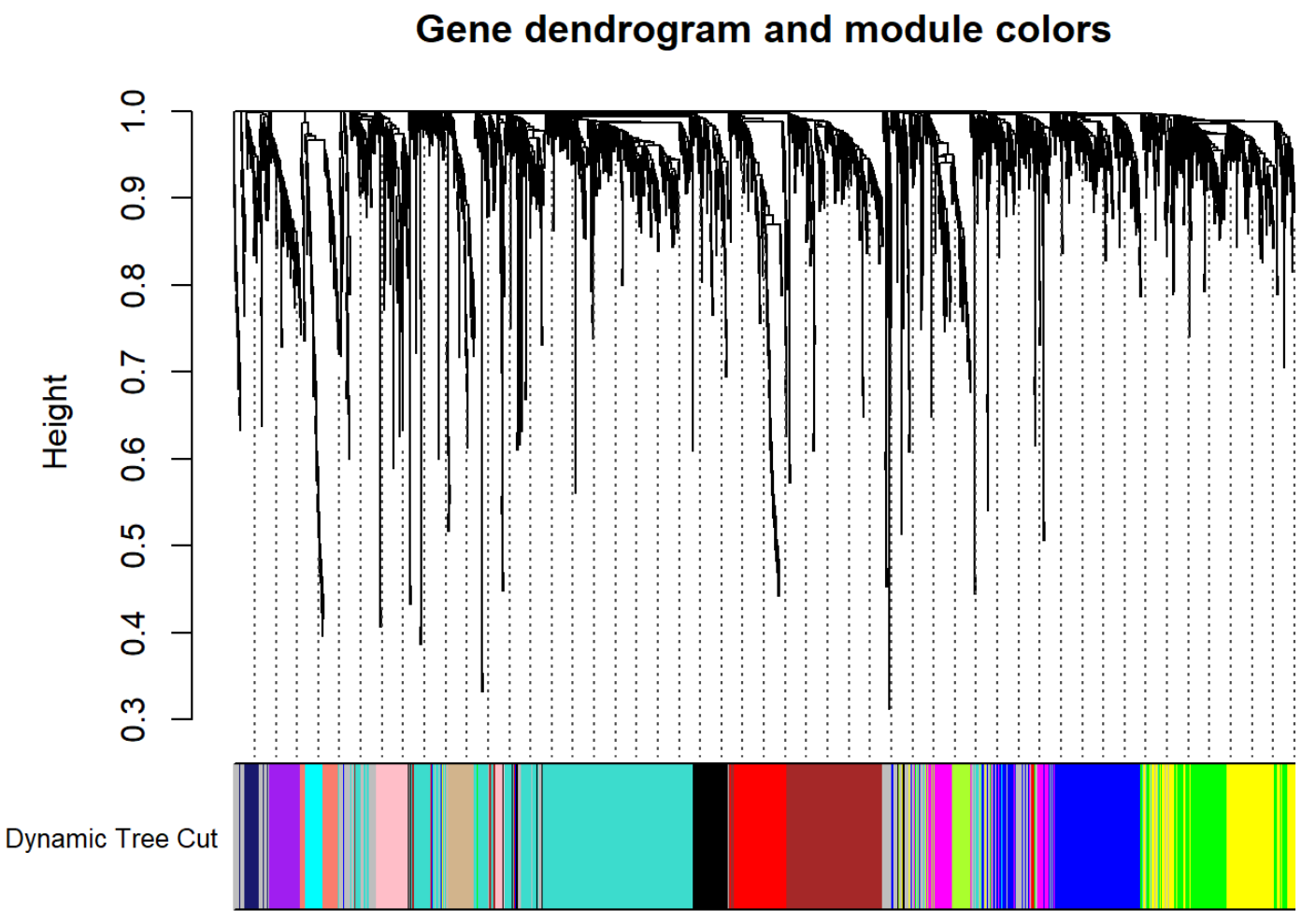
```
table(dynamicMods)
```

```
## dynamicMods
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
## 255 535 272 246 204 155 136 102  99  89  71  70  60  46  42  36
```

```
# Converting numeric labels obtained from dynamic tree cut into colors for visualization
dynamicColors <- labels2colors(dynamicMods)

# Displaying the count of genes in each identified module
table(dynamicColors)
```

```
## dynamicColors
##        black         blue        brown         cyan        green  greenyellow
##          102          272          246           42          155           70
##         grey      magenta midnightblue         pink       purple          red
##          255           89           36           99           71          136
##       salmon          tan    turquoise       yellow
##           46           60          535          204
```

```
# Plotting the dendrogram with modules highlighted by different colors (not merged yet)
plotDendroAndColors(hclustGeneTree,
                    dynamicColors, "Dynamic Tree Cut",
                    dendroLabels = FALSE,
                    hang = 0.03,
                    addGuide = TRUE,
                    guideHang = 0.05,
                    main = "Gene dendrogram and module colors")
```

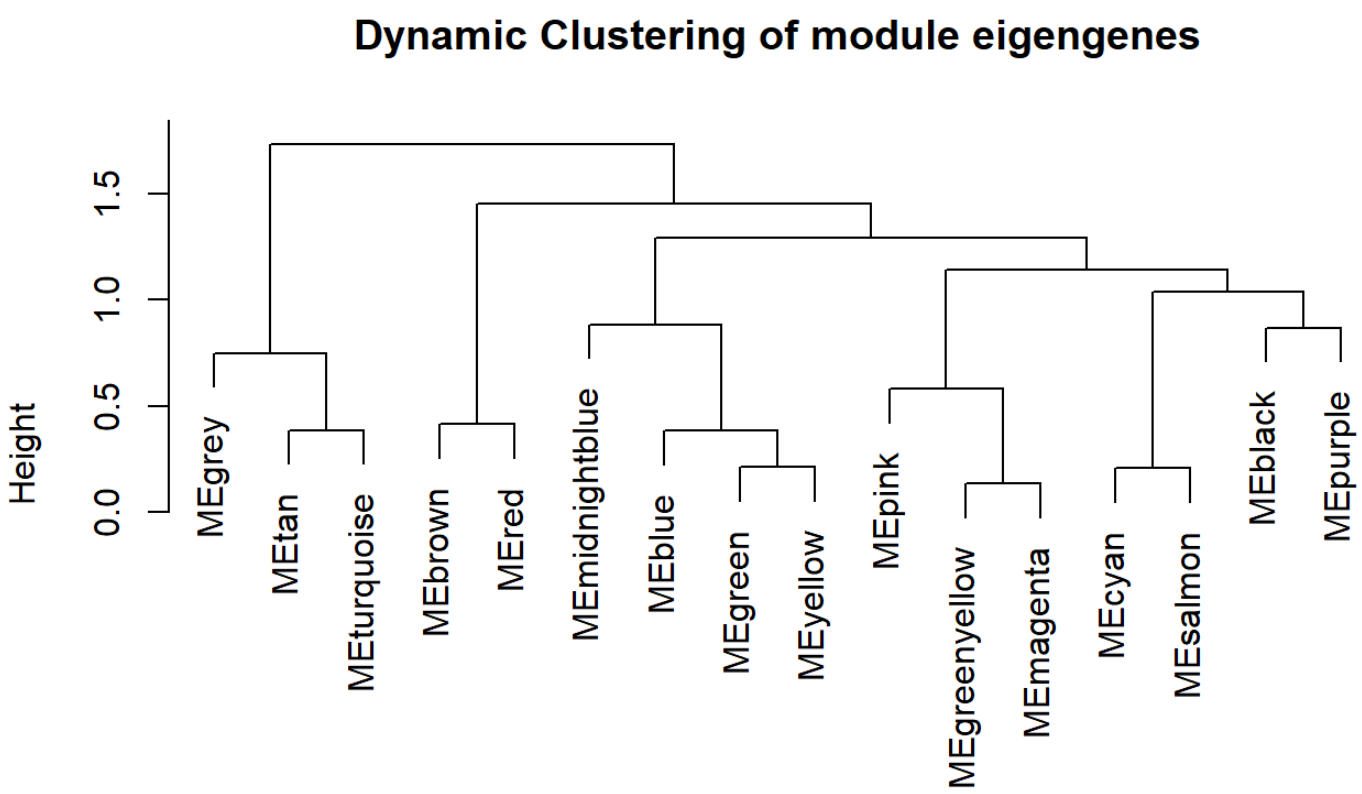


calculating Module Eigengenes

Calculating module eigengenes,which represent the overall gene expression pattern within each module. The eigengenes are extracted, and their dissimilarity is computed. A new hierarchical clustering tree is constructed based on this dissimilarity, visualizing the relationships between module eigengenes. Finally, a plot is generated to illustrate the dynamic clustering of module eigengenes, providing insights into the functional relationships between different gene modules.

```
# Calculating eigengenes representing the gene expression of each module across all samples
dynamic_MEList <- moduleEigengenes(input_mat,
                                   colors = dynamicColors)

# Extracting module eigengenes from the dynamic_MEList and assigning them to dynamic_MEs
dynamic_MEs <- dynamic_MEList$eigengenes

# Calculating dissimilarity of module eigengenes
dynamic_MEDiss <- 1-cor(dynamic_MEs)

# creating a new hierarchial clustering tree based on the Dissimilarity between module eigengenes
dynamic_METree <- hclust(as.dist(dynamic_MEDiss))
```

```
# Plotting the hierarchical clustering tree of module eigengenes
plot(dynamic_METree,
     main = "Dynamic Clustering of module eigengenes",
     xlab = "",
     sub = "")
```



Merging gene modules based on a dissimilarity threshold

The dissimilarity threshold, dynamic_MEDissThres, is set to 0.25, determining the level below which modules are merged. (A visual representation of the threshold can be added to the dendrogram.) The mergeCloseModules function is then employed to automatically merge gene modules in the input matrix (input_mat) based on the specified dissimilarity threshold. The resulting merged modules and their corresponding eigengenes are extracted and visualized through a dendrogram, providing insights into the hierarchical structure and relationships within the gene expression data.

```
# defining a module dissimilarty threshold below which the modules are merged
dynamic_MEDissThres <- 0.25

# Plot the cut line
# abline(h = dynamic_MEDissThres, col = "red")

# Call an automatic merging function
merge_dynamic_MEDs <- mergeCloseModules(input_mat,
                                        dynamicColors,
                                        cutHeight = dynamic_MEDissThres,
                                        verbose = 3)
```

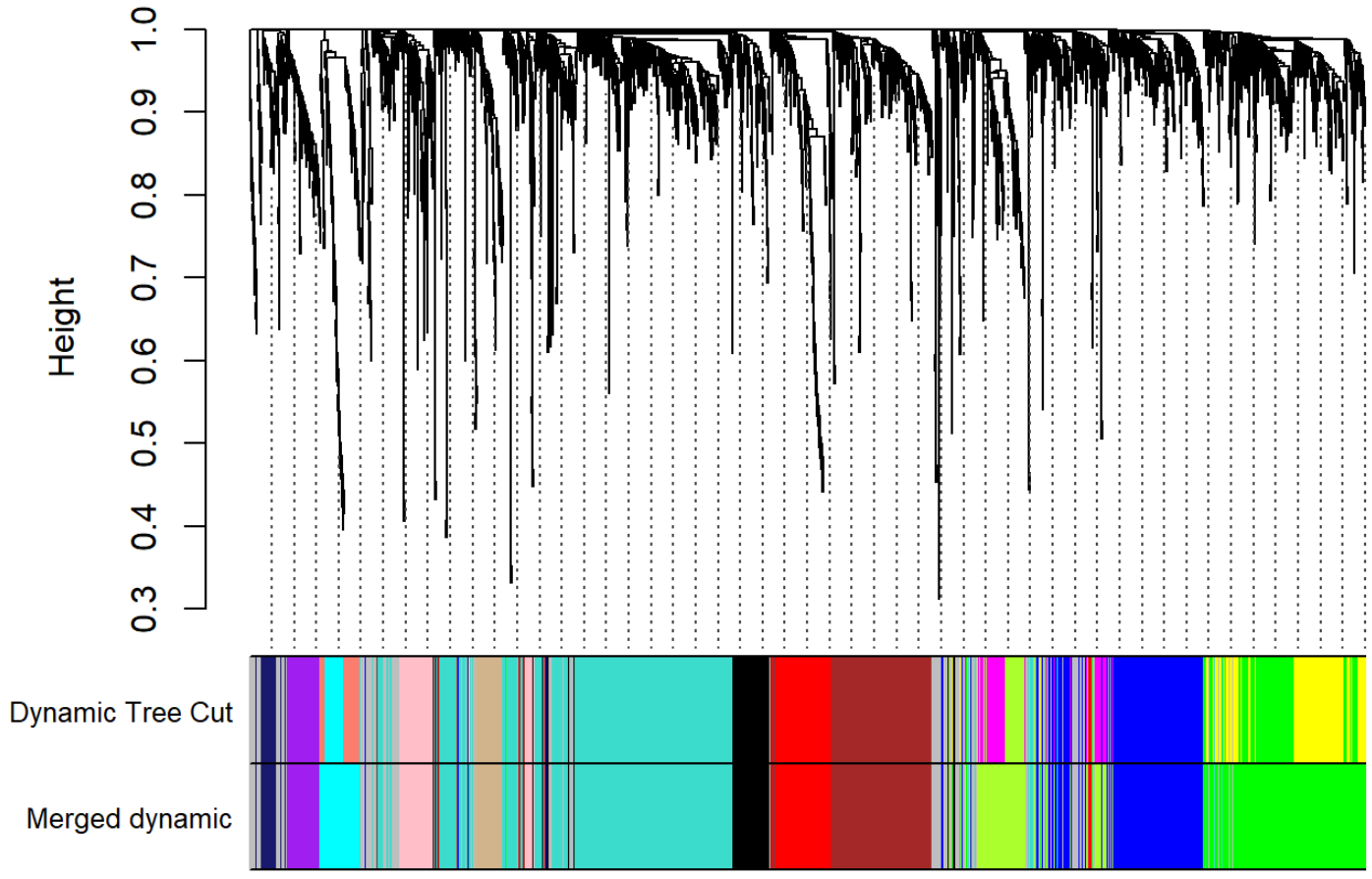
```
##  mergeCloseModules: Merging modules whose distance is less than 0.25
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 16 module eigengenes in given set.
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 13 module eigengenes in given set.
##    Calculating new MEs...
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 13 module eigengenes in given set.
```

```
# The Merged Colors
dynamic_mergedColors <- merge_dynamic_MEDs$colors
table(dynamic_mergedColors)
```

```
## dynamic_mergedColors
##        black         blue        brown         cyan        green  greenyellow
##          102          272          246           88          359          159
##         grey midnightblue         pink       purple          red          tan
##          255           36           99           71          136           60
##    turquoise
##          535
```

```
# Eigen genes of the new merged modules
mergedMEs <- merge_dynamic_MEDs$newMEs
```

```
# plotting a dendrogram with merged module colors
plotDendroAndColors(hclustGeneTree,
                    cbind(dynamicColors, dynamic_mergedColors),
                    c("Dynamic Tree Cut", "Merged dynamic"),
                    dendroLabels = FALSE,
                    hang = 0.03,
                    addGuide = TRUE,
                    guideHang = 0.05)
```



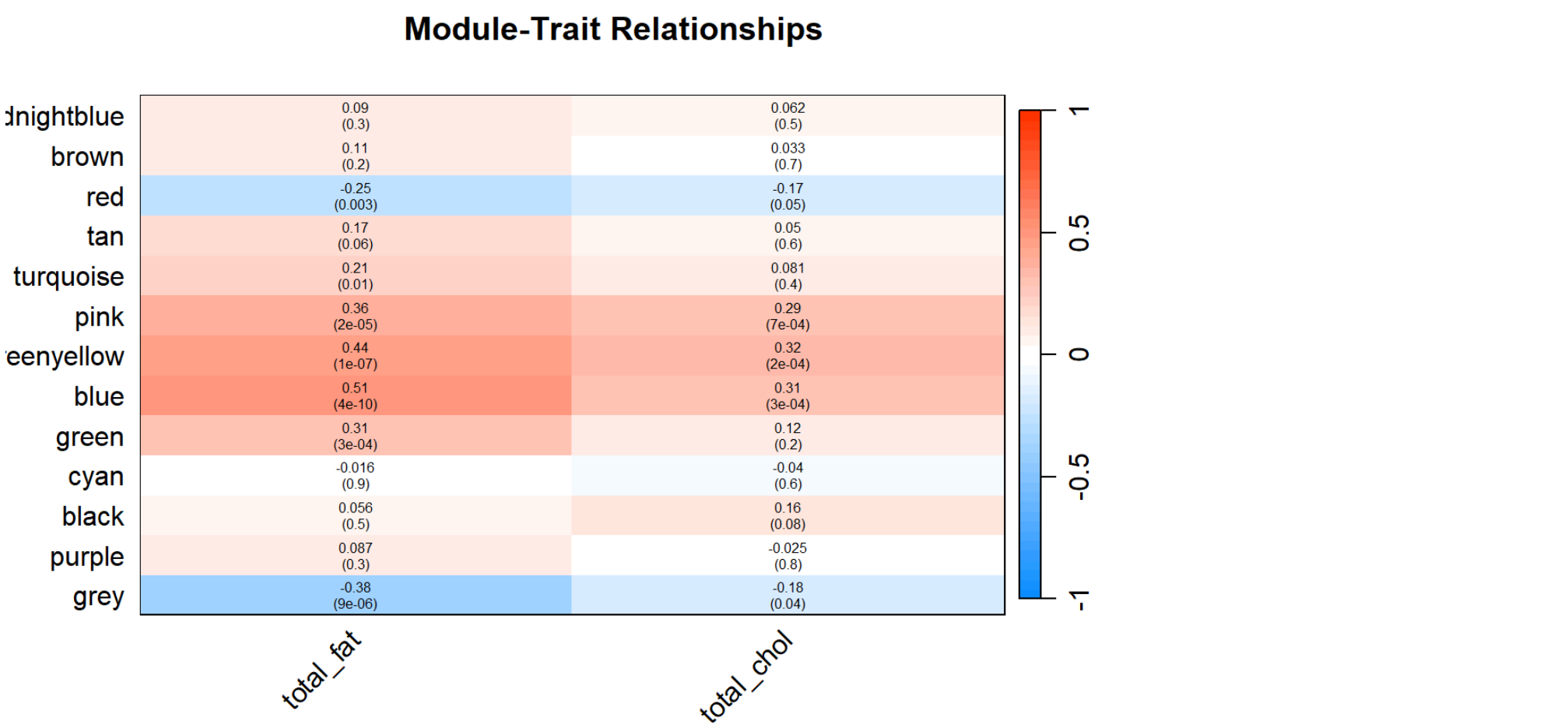
Module-Trait Relationship Analysis

In this section, we perform a Module-Trait Relationship analysis. The code below defines the number of genes and samples, recalculates module eigengenes (MEs) with color labels, and calculates the correlation between these MEs and trait data. The resulting heatmap visually represents the relationships between gene modules and external traits.

This analysis is crucial for understanding how gene modules are associated with specific traits, shedding light on potential biological significance and providing insights into the underlying regulatory mechanisms.

```r
# Defining the number of genes and samples
n_genes <- ncol(input_mat)
n_samples <- nrow(input_mat)

# Recalculating the module eigengenes (MEs) with color labels
MEs0 <- moduleEigengenes(input_mat, dynamic_mergedColors)$eigengenes
MEs <- orderMEs(MEs0)
names(MEs) <- substring(names(MEs), 3)

# Calculate the correlation between MEs and trait data
moduleTraitCor <- cor(MEs, traits_data, use = 'p')
moduleTraitPvalue <- corPvalueStudent(moduleTraitCor, n_samples)

# Create a text matrix for annotated heatmaps
textMatrix <- paste(signif(moduleTraitCor, 2), "\n(", signif(moduleTraitPvalue, 1), ")", sep = "")
dim(textMatrix) <- dim(moduleTraitCor)
```

```r
# Generate a labeled heatmap visualizing Module-Trait Relationships
labeledHeatmap(Matrix = moduleTraitCor,
               xLabels = names(traits_data),
               yLabels = names(MEs),
               ySymbols = names(MEs),
               colorLabels = FALSE,
               colors = blueWhiteRed(50),
               textMatrix = textMatrix,
               setStdMargins = FALSE,
               cex.text = 0.5,
               zlim = c(-1,1),
               main = paste("Module-Trait Relationships"))
```




```r
# Restore the original 'cor' function
cor <- temp_cor
```