Weighted gene co-expression network analysis(WGCNA)

Loading required libraries.

```
library(DESeq2)
```

```
## Warning: package 'matrixStats' was built under R version 4.4.1
```

```
library(WGCNA)
```

```
## Warning: package 'WGCNA' was built under R version 4.4.1
```

```
library(magrittr)
```

```
## Warning: package 'magrittr' was built under R version 4.4.1
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.4.1
```

```
library(genefilter)
```

Read counts data into CSV file.

```
read_count_data <- function(file_path){
  counts_data <- read.csv(file_path, row.names = 1)
  expression_data <- round(counts_data)
  return(expression_data)
}
expression_matrix <- read_count_data("../Network analysis/Data/GSE183947_fpkm.csv")
head(expression_matrix,2)
```

```
##        tumor.rep1 tumor.rep2 tumor.rep3 tumor.rep4 tumor.rep5 tumor.rep6
## TSPAN6          1          2          0          5          5          5
## TNMD            0          0          0          0          0          0
##        tumor.rep7 tumor.rep8 tumor.rep9 tumor.rep10 tumor.rep11 tumor.rep12
## TSPAN6          4          4          6          12           6           4
## TNMD            0          0          0           0           0           0
##        tumor.rep13 tumor.rep14 tumor.rep15 tumor.rep16 tumor.rep17 tumor.rep18
## TSPAN6           8          11           4          14          10           7
## TNMD             0           1           0           0           0           0
##        tumor.rep19 tumor.rep20 tumor.rep21 tumor.rep22 tumor.rep23 tumor.rep24
## TSPAN6           6           7           9           7          10           7
## TNMD             0           0           0           0           0           0
##        tumor.rep25 tumor.rep26 tumor.rep27 tumor.rep28 tumor.rep29 tumor.rep30
## TSPAN6          5           2           5           5           9           2
## TNMD            0           0           0           1           0           1
##        normal.rep1 normal.rep2 normal.rep3 normal.rep4 normal.rep5 normal.rep6
## TSPAN6          12           3          13          15           7           0
## TNMD             6           2           0           2           0           0
##        normal.rep7 normal.rep8 normal.rep9 normal.rep10 normal.rep11
## TSPAN6          10           7           5            6            6
## TNMD             0           0          11            0            3
##        normal.rep12 normal.rep13 normal.rep14 normal.rep15 normal.rep16
## TSPAN6            7           11           16           12           12
## TNMD              0            0            0            0            1
##        normal.rep17 normal.rep18 normal.rep19 normal.rep20 normal.rep21
## TSPAN6           10            9            7            9            7
## TNMD              1            0            0            0            0
##        normal.rep22 normal.rep23 normal.rep24 normal.rep25 normal.rep26
## TSPAN6            8            9            6            6            6
## TNMD              0            1            0            0            0
##        normal.rep27 normal.rep28 normal.rep29 normal.rep30
## TSPAN6            4            5           10            5
## TNMD              9            1            0            0
```

Read metadata into CSV file.

```
read_metadata <- function(file_path){
  coldata <- read.csv(file_path, row.names = 1)
  return (coldata)
}
meta_data <- read_metadata("../Network analysis/Data/metadata.csv")
head(meta_data)
```

```
##            condition description
## tumor rep1     tumor    CA.102548
## tumor rep2     tumor    CA.104338
## tumor rep3     tumor    CA.105094
## tumor rep4     tumor    CA.109745
## tumor rep5     tumor   CA.1906415
## tumor rep6     tumor   CA.1912627
```

Convert condition column in metadata to factor.

```
meta_data$condition <- as.factor(meta_data$condition)
meta_data$description <- as.factor(meta_data$description)
```

Make sure the row names in metadata matches to the column names in expression matrix.

```r
all(rownames(meta_data) %in% colnames(expression_matrix))
```

```
## [1] FALSE
```

Match the row names in metadata to the column names in expression matrix.

```r
rownames(meta_data) = colnames(expression_matrix)
```

Create a new column named accession_code and store the colnames of expression matrix in it

```r
meta_data$accession_code <- colnames(expression_matrix)
```

pre-filtering to keep only genes with 50 or more reads in total across the samples.

```r
pre_filter <- function(){
  # Only keep rows that have total counts above the cutoff
  keep <- expression_matrix %>% rowSums(.) >= 50
  filtered_counts <- expression_matrix[keep,]
  return (filtered_counts)
}
filtered_expression_counts <- pre_filter()
head(filtered_expression_counts,2)
```

```
##        tumor.rep1 tumor.rep2 tumor.rep3 tumor.rep4 tumor.rep5 tumor.rep6
## TSPAN6          1          2          0          5          5          5
## DPM1            0          0          0          3          8          9
##        tumor.rep7 tumor.rep8 tumor.rep9 tumor.rep10 tumor.rep11 tumor.rep12
## TSPAN6          4          4          6          12           6           4
## DPM1            8          8          6           6           7           6
##        tumor.rep13 tumor.rep14 tumor.rep15 tumor.rep16 tumor.rep17 tumor.rep18
## TSPAN6           8          11           4          14          10           7
## DPM1            10           7          13          10           9           8
##        tumor.rep19 tumor.rep20 tumor.rep21 tumor.rep22 tumor.rep23 tumor.rep24
## TSPAN6           6           7           9           7          10           7
## DPM1            18           8           6           9           9           3
##        tumor.rep25 tumor.rep26 tumor.rep27 tumor.rep28 tumor.rep29 tumor.rep30
## TSPAN6           5           2           5           5           9           2
## DPM1             7           3          15           7           4           6
##        normal.rep1 normal.rep2 normal.rep3 normal.rep4 normal.rep5 normal.rep6
## TSPAN6          12           3          13          15           7           0
## DPM1             0           9          11           9           7           0
##        normal.rep7 normal.rep8 normal.rep9 normal.rep10 normal.rep11
## TSPAN6          10           7           5           6           6
## DPM1             8           4           0          10           0
##         normal.rep12 normal.rep13 normal.rep14 normal.rep15 normal.rep16
## TSPAN6             7           11           16           12           12
## DPM1               4            4            0            1            6
##         normal.rep17 normal.rep18 normal.rep19 normal.rep20 normal.rep21
## TSPAN6            10            9            7            9            7
## DPM1               4            6           19            3            5
##         normal.rep22 normal.rep23 normal.rep24 normal.rep25 normal.rep26
## TSPAN6             8            9            6            6            6
## DPM1               7            6            5            9            9
##         normal.rep27 normal.rep28 normal.rep29 normal.rep30
## TSPAN6             4            5           10            5
## DPM1               5            4            7            3
```

Construct a DESeqDataSet.

```
deseqdataset <- function(){
  deseqdataset <- DESeqDataSetFromMatrix(countData = filtered_expression_counts,
                                         colData = meta_data,
                                         design = ~ condition)

  return(deseqdataset)
}
deseqdataset_object <- deseqdataset()
```

```
## converting counts to integer mode
```

```
deseqdataset_object
```

```
## class: DESeqDataSet
## dim: 17172 60
## metadata(1): version
## assays(1): counts
## rownames(17172): TSPAN6 DPM1 ... RP4-583P15.15 ZBTB8B
## rowData names(0):
## colnames(60): tumor.rep1 tumor.rep2 ... normal.rep29 normal.rep30
## colData names(3): condition description accession_code
```

Differential expression analysis

```
diff_expr_analysis <- function(){
  deseq_analysis <- DESeq(deseqdataset_object)
  # Apply Variance Stabilizing Transformation (VST)
  vsd <- vst(deseq_analysis, blind = FALSE)
  return (vsd)
}
dds_norm <- diff_expr_analysis()
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing

## -- replacing outliers and refitting for 854 genes
## -- DESeq argument 'minReplicatesForReplace' = 7
## -- original counts are preserved in counts(dds)

## estimating dispersions

## fitting model and testing
```

```
dds_norm
```

```
## class: DESeqTransform
## dim: 17172 60
## metadata(1): version
## assays(1): ''
## rownames(17172): TSPAN6 DPM1 ... RP4-583P15.15 ZBTB8B
## rowData names(23): baseMean baseVar ... replace dispFit
## colnames(60): tumor.rep1 tumor.rep2 ... normal.rep29 normal.rep30
## colData names(5): condition description accession_code sizeFactor
##   replaceable
```

Extract the VST-transformed data, Filter low variance genes then transpose to have genes as columns

```
norm_counts <- function(){
  norm_transposed_vst <- assay(dds_norm) %>%
              varFilter(var.cutoff = 0.5) %>%
              t()
  return(norm_transposed_vst)
}
normalized_counts <- norm_counts()
```

Determine power soft-threshold

```
pick_s_th <- function(){
# the pickSoftThreshold() function help identify good choices for power parameter
  sft <- pickSoftThreshold(normalized_counts,
    dataIsExpr = TRUE,
    corFnc = cor,
    networkType = "signed")
  return(sft)

}
pick_soft_th <- pick_s_th()
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
##    Power SFT.R.sq  slope truncated.R.sq mean.k. median.k. max.k.
## 1      1   0.7110  5.700          0.793  4770.0   4840.00   5560
## 2      2   0.3910  3.020          0.872  2840.0   2860.00   3900
## 3      3   0.2030  1.540          0.890  1780.0   1760.00   2880
## 4      4   0.0349  0.431          0.865  1170.0   1110.00   2200
## 5      5   0.0414 -0.367          0.830   792.0    719.00   1750
## 6      6   0.2710 -0.877          0.840   555.0    476.00   1420
## 7      7   0.5070 -1.210          0.861   400.0    322.00   1170
## 8      8   0.6560 -1.440          0.885   295.0    225.00    983
## 9      9   0.7240 -1.560          0.896   222.0    160.00    834
## 10    10   0.7680 -1.610          0.911   170.0    116.00    713
## 11    12   0.8100 -1.630          0.928   104.0     62.30    534
## 12    14   0.8330 -1.620          0.943    67.3     34.90    410
## 13    16   0.8360 -1.630          0.949    45.3     20.30    321
## 14    18   0.8330 -1.640          0.953    31.6     12.30    258
## 15    20   0.8320 -1.660          0.959    22.7      7.53    211
```

Calculate a measure of the model fit, the signed R^2

```
calculate_model_fit <- function(){
  sft_df <- data.frame(pick_soft_th$fitIndices) %>%
    dplyr::mutate(model_fit = -sign(slope) * SFT.R.sq)
  return(sft_df)
}
model_fit_df <- calculate_model_fit()
model_fit_df
```

```
##    Power   SFT.R.sq      slope truncated.R.sq    mean.k.   median.k.    max.k.
## 1      1 0.71136021  5.6974248      0.7927414 4765.92918 4842.649638 5559.8410
## 2      2 0.39058473  3.0161402      0.8723373 2835.32680 2863.728388 3899.4809
## 3      3 0.20340679  1.5425168      0.8904587 1779.16029 1757.180497 2877.8090
## 4      4 0.03485874  0.4314162      0.8648870 1165.86496 1111.064309 2199.8846
## 5      5 0.04143091 -0.3667274      0.8301182  792.23125  719.231721 1745.8142
## 6      6 0.27118585 -0.8772812      0.8400870  555.28105  475.805500 1418.8366
## 7      7 0.50706240 -1.2136962      0.8605875  399.75410  322.096223 1173.0804
## 8      8 0.65615762 -1.4436815      0.8845583  294.57086  224.813781  983.3211
## 9      9 0.72397197 -1.5609393      0.8961605  221.54083  160.078081  833.5887
## 10    10 0.76826030 -1.6054086      0.9112820  169.64234  115.559172  713.3297
## 11    12 0.81008120 -1.6268522      0.9284748  104.16417   62.348212  534.4328
## 12    14 0.83301391 -1.6226687      0.9428943   67.31119   34.939850  410.2240
## 13    16 0.83611516 -1.6265376      0.9488156   45.34131   20.322396  320.9467
## 14    18 0.83269547 -1.6423412      0.9529001   31.61272   12.259403  258.1983
## 15    20 0.83182503 -1.6558128      0.9590780   22.69254    7.531113  211.2544
##      model_fit
## 1  -0.71136021
## 2  -0.39058473
## 3  -0.20340679
## 4  -0.03485874
## 5   0.04143091
## 6   0.27118585
## 7   0.50706240
## 8   0.65615762
## 9   0.72397197
## 10  0.76826030
## 11  0.81008120
## 12  0.83301391
## 13  0.83611516
## 14  0.83269547
## 15  0.83182503
```

Plot the model fitting by the power soft threshold so we can decide on a soft-threshold for power.

```r
plot_model_fit <- function(){
jpeg("../Network analysis/outputs/model_fit.jpeg")
p <- ggplot(model_fit_df, aes(x = Power, y = model_fit, label = Power)) +
     geom_point() +
     # We'll put the Power labels slightly above the data points
     geom_text(nudge_y = 0.1) +
     # We will plot what WGCNA recommends as an R^2 cutoff
     geom_hline(yintercept = 0.80, col = "red") +
    # Just in case our values are low, we want to make sure we can still see the 0.80 level
     ylim(c(min(model_fit_df$model_fit), 1.05)) +
     xlab("Soft Threshold (power)") +
     ylab("Scale Free Topology Model Fit, signed R^2") +
     ggtitle("Scale independence") +
     theme_classic()
print(p)
dev.off()
}

plot_model_fit()
```

```
## pdf
##   2
```

Run WGCNA to find gene co-expression modules using 16 for the power argument

```
run_WGCNA <- function(){
bwnet <- blockwiseModules(
  normalized_counts,
  # What size chunks (how many genes) the calculations should be run in
  maxBlockSize = 2000,
  # topological overlap matrix
  TOMType = "signed",
  # soft threshold for network construction
  power = 16,
  # Let's use numbers instead of colors for module labels
  numericLabels = TRUE,
  randomSeed = 1234
  )
return(bwnet)
}
bwnet <- run_WGCNA()
```

Write main WGCNA results to CSV file

```
write_WGCNA <- function(out_path){
  write.csv(bwnet$MEs, out_path)
}
write_WGCNA("../Network analysis/outputs/main_WGCNA_results.csv")
```

Explore WGCNA results

```
# Explore eigengene modules for each sample
mod_eigengenes<- function(){
  module_eigengenes <- bwnet$MEs
  return(module_eigengenes)
}
module_eigengenes <- mod_eigengenes()
head(module_eigengenes,2)
```

```
##                     ME8       ME2         ME9        ME13        ME6       ME11
## tumor.rep1 -0.05752606 0.2743180 -0.12421496 -0.06991644 -0.1057725 -0.1832143
## tumor.rep2 -0.06828790 0.2239259 -0.09950691 -0.08196301 -0.1050379 -0.1424683
##                   ME7         ME1        ME3       ME14        ME4        ME5
## tumor.rep1 -0.1067638 -0.08753122 -0.2540542 -0.1238199 -0.2284450 -0.3809287
## tumor.rep2 -0.1391291 -0.05392580 -0.2005276 -0.1262192 -0.2065075 -0.3344554
##                  ME10       ME12        ME0
## tumor.rep1 -0.4174163 -0.1978610 -0.2800349
## tumor.rep2 -0.3408421 -0.1926129 -0.2515502
```

Which modules have biggest differences across two condition groups?

Run linear model on each module

```r
fit_linear_model <- function(){
  # Create the design matrix from the `condition` variable
  des_mat <- model.matrix(~ meta_data$condition)
  # lmFit() needs a transposed version of the matrix
  fit <- limma::lmFit(t(module_eigengenes), design = des_mat)
  # Apply empirical Bayes to smooth standard errors
  fit <- limma::eBayes(fit)
  return(fit)
}

fit <- fit_linear_model()
```

Apply multiple testing correction and obtain stats in a dataframe

```r
dataframe_stats <- function(){
stats_df <- limma::topTable(fit, number = ncol(module_eigengenes)) %>%
  tibble::rownames_to_column("module")
return(stats_df)
}
stats_df <- dataframe_stats()
```

```
## Removing intercept from test coefficients
```

```r
stats_df
```

```
##    module       logFC       AveExpr          t      P.Value    adj.P.Val
## 1     ME5 -0.20957380 -2.059984e-18 -8.7649414 8.093550e-14 1.214032e-12
## 2     ME6  0.18917581 -3.350185e-18  7.3706041 6.699936e-11 5.024952e-10
## 3    ME13  0.16950234  9.396419e-18  6.2552758 1.183287e-08 5.916433e-08
## 4     ME4 -0.12647705 -2.855066e-18 -4.2865945 4.408968e-05 1.653363e-04
## 5     ME8 -0.12290923  6.389565e-18 -4.1430760 7.519480e-05 2.162929e-04
## 6    ME11  0.12118016 -3.216466e-19  4.0743500 9.672494e-05 2.162929e-04
## 7     ME9 -0.12088440  7.950816e-20 -4.0626470 1.009367e-04 2.162929e-04
## 8    ME14 -0.11434469 -5.522203e-18 -3.8076245 2.507535e-04 4.701628e-04
## 9     ME1  0.09144734  1.795439e-17  2.9634257 3.859165e-03 6.431942e-03
## 10   ME12 -0.08486184 -1.420305e-18 -2.7322486 7.522004e-03 1.128301e-02
## 11    ME7  0.06151967  5.341503e-18  1.9442081 5.487821e-02 7.483393e-02
## 12    ME2 -0.05713576 -6.454617e-18 -1.8006644 7.498206e-02 9.372758e-02
## 13   ME10 -0.04707365  6.649773e-19 -1.4753524 1.434795e-01 1.599029e-01
## 14    ME3  0.04641367  2.411988e-17  1.4541963 1.492427e-01 1.599029e-01
## 15    ME0 -0.02506584 -6.412694e-17 -0.7791228 4.378762e-01 4.378762e-01
##             B
## 1  20.9466402
## 2  14.3220446
## 3   9.2429763
## 4   1.2660206
## 5   0.7559701
## 6   0.5159477
## 7   0.4753534
## 8  -0.3885874
## 9  -2.9406577
## 10 -3.5488761
## 11 -5.2946050
## 12 -5.5547996
## 13 -6.0749808
## 14 -6.1054115
## 15 -6.8518385
```

Module 5 seems to be the most differentially expressed across condition groups

Let's make plot of module 5

```
modules_dataframe <- function(){
  module_5 <- module_eigengenes %>%
  tibble::rownames_to_column("accession_code") %>%
  dplyr::inner_join(
    meta_data %>%
      dplyr::select(accession_code, condition),
    by = "accession_code"
  )
  return(module_5)
}
modules_df <- modules_dataframe()
head(modules_df,2)
```

```
##   accession_code         ME8       ME2         ME9         ME13        ME6
## 1     tumor.rep1 -0.05752606 0.2743180 -0.12421496 -0.06991644 -0.1057725
## 2     tumor.rep2 -0.06828790 0.2239259 -0.09950691 -0.08196301 -0.1050379
##         ME11        ME7         ME1        ME3       ME14        ME4        ME5
## 1 -0.1832143 -0.1067638 -0.08753122 -0.2540542 -0.1238199 -0.2284450 -0.3809287
## 2 -0.1424683 -0.1391291 -0.05392580 -0.2005276 -0.1262192 -0.2065075 -0.3344554
##         ME10       ME12        ME0 condition
## 1 -0.4174163 -0.1978610 -0.2800349     tumor
## 2 -0.3408421 -0.1926129 -0.2515502     tumor
```

Boxplot of module 5

```
boxplot_mod_5 <- function(){
jpeg("../Network analysis/outputs/boxplot_of_module_5.jpeg")
p <- ggplot(modules_df,aes(x = condition,
                             y = ME5,
                             color = condition)) +
    # a boxplot with outlier points hidden (they will be in the sina plot)
    geom_boxplot(width = 0.2, outlier.shape = NA) +
    # A sina plot to show all of the individual data points
    ggforce::geom_sina(maxwidth = 0.3) +
    theme_classic()

print(p)
dev.off()
}
boxplot_mod_5()
```

```
## pdf
##   2
```

Boxplot of module 6

```
boxplot_mod_6 <- function(){
jpeg("../Network analysis/outputs/boxplot_of_module_6.jpeg")
p <- ggplot(modules_df,aes(x = condition,
                             y = ME6,
                             color = condition)) +
    # a boxplot with outlier points hidden (they will be in the sina plot)
    geom_boxplot(width = 0.2, outlier.shape = NA) +
    # A sina plot to show all of the individual data points
    ggforce::geom_sina(maxwidth = 0.3) +
    theme_classic()
print(p)
dev.off()
}
boxplot_mod_6()
```

```
## pdf
##   2
```

What genes are a part of module 5

Genes corresponding to each module

```
gene_module <- function(module_name){
 gene_module<- tibble::enframe(bwnet$colors, name = "gene",
                                    value = "module") %>%
# Let's add the `ME` part so its more clear what these numbers are and it matches elsewhere
dplyr::mutate(module = paste0(module_name, module))
 return(gene_module)
}
gene_module_key <- gene_module("ME")
```

Genes that part of module 5

```r
gene_module_key %>% dplyr::filter(module == "ME5")
```

```
## # A tibble: 192 x 2
##    gene     module
##    <chr>    <chr>
##  1 TMEM132A ME5
##  2 CACNA1G  ME5
##  3 GAS7     ME5
##  4 TENM1    ME5
##  5 IDS      ME5
##  6 PREX2    ME5
##  7 ARHGAP6  ME5
##  8 LAMA3    ME5
##  9 PRR11    ME5
## 10 GPR116   ME5
## # i 182 more rows
```

Extract ME5 eigengene module values

```r
mod_eigengene <- function(module_name){
module_eigengene <- module_eigengenes %>%
    dplyr::select(all_of(module_name)) %>%
    tibble::rownames_to_column("accession_code")

return(module_eigengene)
}
module_eigengene <- mod_eigengene("ME5")
head(module_eigengene,10)
```

```
##    accession_code        ME5
## 1      tumor.rep1 -0.3809287
## 2      tumor.rep2 -0.3344554
## 3      tumor.rep3 -0.3568427
## 4      tumor.rep4 -0.2012686
## 5      tumor.rep5 -0.1473240
## 6      tumor.rep6 -0.1750315
## 7      tumor.rep7 -0.1103692
## 8      tumor.rep8 -0.1186162
## 9      tumor.rep9 -0.1376516
## 10    tumor.rep10 -0.1160866
```

Create dataframe that contain condition and ME5 columns

```r
col_annotation_df <- function(){
  # Set up column annotation from metadata
  col_annot_df <- meta_data %>%
  # Only select the condition and sample ID columns
  dplyr::select(accession_code, condition) %>%
  # Add on the eigengene expression by joining with sample IDs
  dplyr::inner_join(module_eigengene, by = "accession_code") %>%
  # Arrange by condition
  dplyr::arrange(condition) %>%
  # Store sample
  tibble::column_to_rownames("accession_code")

  return(col_annot_df)
}

col_annot_df <- col_annotation_df()
head(col_annot_df,10)
```

```
##              condition         ME5
## normal.rep1     normal  0.02251745
## normal.rep2     normal  0.08984557
## normal.rep3     normal  0.06511319
## normal.rep4     normal  0.07445223
## normal.rep5     normal  0.09020557
## normal.rep6     normal -0.06160411
## normal.rep7     normal  0.09159700
## normal.rep8     normal  0.09701780
## normal.rep9     normal  0.09292273
## normal.rep10    normal  0.14064270
```

Create the ComplexHeatmap column annotation function

```r
ComplexHeatmap_col_annotation <- function(module_name){
  # Create the ComplexHeatmap column annotation object
  col_annot <- ComplexHeatmap::HeatmapAnnotation(
  # Supply condition labels
  condition = col_annot_df$condition,
  # Add annotation barplot
  module_eigengene = ComplexHeatmap::anno_barplot(dplyr::select(col_annot_df, module_name)),
  # Pick colors for each experimental group in condition
  col = list(condition = c("tumor" = "#f1a340", "normal" = "#998ec3"))
  )

  return(col_annot)
}

col_annot <- ComplexHeatmap_col_annotation("ME5")
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use ‘all_of()‘ or ‘any_of()‘ instead.
##   # Was:
##   data %>% select(module_name)
##
##   # Now:
##   data %>% select(all_of(module_name))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

Get a vector of the gene IDs that correspond to this module

```
get_module_genes <- function(module_name){
  module_genes <- gene_module_key %>%
      dplyr::filter(module == module_name) %>%
      dplyr::pull(gene)
  return(module_genes)
}

module_genes <- get_module_genes("ME5")
```

Set up the gene expression data frame

```
module_matrix <- function(){
mod_mat <- normalized_counts %>%
    t() %>%
    as.data.frame() %>%
    # Only keep genes from ths module
    dplyr::filter(rownames(.) %in% module_genes) %>%
    # Order the samples to match col_annot_df
    dplyr::select(rownames(col_annot_df)) %>%
    # Data needs to be a matrix
    as.matrix()
return(mod_mat)
}

module_mat <- module_matrix()
```

Normalize the gene expression values

```
norm_module_matrix <- function(){
  mod_mat <- module_mat %>%
    # Scale can work on matrices, but it does it by column so we will need to
```

```r
    # transpose first
    t() %>%
    scale() %>%
    # And now we need to transpose back
    t()
return(mod_mat)
}

mod_mat <- norm_module_matrix()
```

Create a color function based on standardized scale

```r
color <- function(){
  color_func <- circlize::colorRamp2(c(-4, 0, 4),
                                     c("#67a9cf", "#f7f7f7", "#ef8a62"))
  return(color_func)
}

color_func <- color()
```

Plot the Heatmap

```r
plot_heatmap <- function(module_name){
  set.seed(432)
  jpeg("../Network analysis/outputs/Heatmap_of_largest_DE_Module.jpeg")
  heatmap <- ComplexHeatmap::Heatmap(mod_mat,
    name = module_name,
    # Supply color function
    col = color_func,
    # Supply column annotation
    bottom_annotation = col_annot,
    # We don't want to cluster samples
    cluster_columns = FALSE,
    # We don't need to show sample or gene labels
    show_row_names = FALSE,
    show_column_names = FALSE
  )
  print(heatmap)
  dev.off()
}
plot_heatmap("ME5")
```

```
## pdf
##   2
```