Extraction and classification of patient specific features from raw RNA-Seq data

Iuhaniak Andreea-Maria

Introduction

This project analyzes raw RNA-Seq data to explore gene expression patterns, co-expression and co-splicing networks.

First part was performed entirely on a high-performance cluster (HPC) while the second part was performed in R.

Prerequisites

- High-performance computing (HPC) system

- R and RStudio

- Basic knowledge of RNA-Seq data processing and R programming

- Understanding of co-expression network analysis (e.g. WGCNA)

- Conda: required to manage environments

Dependencies

- Edirect: required for accessing and querying NCBI's databases

- SRA-Toolkit: required for downloading RNA-Seq data from NCBI SRA

- FastQC: required for reads quality control

- Cutadapt: required for read processing stages

- Kallisto: required for pseudoalignment and transcript quantification

- R libraries (e.g. WGCNA, dplyr, ggplot)

Data requirements

-raw RNA-Seq data

Installation

1. EDirect Tools

- install by running the following command in your Software directory:

```
sh -c "$(wget -q https://ftp.ncbi.nlm.nih.gov/entrez/entrezdirect/install-edirect.sh -O -)
```

- activate EDirect in your terminal session (export in home):

```
export PATH=$\{HOME\}/edirect:$\{PATH\}
```

2. SRA-Toolkit

- install SRA-Tools from git in your Software directory:

```
git clone https://github.com/ncbi/sra-tools.git\`
```

- check README.file to find download page for pre-built binaries and run the following command:

```
wget https://github.com/ncbi/sra-tools/wiki/01.-Downloading-SRA-Toolkit)`
```

- unzip the tar.gz file you just downloaded

```
tar -xvzf sratoolkit.3.1.1-centos_linux64.tar.gz
```

- check folder for fasterq-dump command

- run `fasterq-dump` in Bin to check if fasterq-dump is working

- load sra-tools module before running fasterq-dump

Optional

- add fasterq-dump to your PATH environment to run it from any directory (no more need for module load)

- go back to home directory and run `ls -a` to reveal hidden files

- enter .bashrc using a command-line text editor (e.g. nano, vim) and paste the path to Bin

- run command source .bashrc to save changes; now fasterq-dump command should work no matter the directory

3. Conda

- HPCs should normally have this module installed (e.g. check it by running `which conda`);

- if installed, load them by running the following command:

```
module load conda
```

- conda needs to have miniforge module loaded first

4. FastQC, Cutadapt, Kallisto

- if installed on cluster, load them before running your command

- if not installed on cluster, install them globally in your base environment or in your project environment.

- to install cutadapt in your base environment run:

```
conda install -c bioconda cutadapt
```

- if you want to install it in your project environment, first load conda, then create your project environment by running the following command:

```
conda create -n rna-seq-env
```

-replace replace rna-seq-env with the name of your project

- install cutadapt:

```
conda install -c bioconda cutadapt
```

- verify installation:

```
cutadapt –version
```

- your project environemnt can contain as many command-line programs as you need.

- creating such an environment is useful when your project requires specific programs and versions. Moreover, it enables exact reproduction of your analysis by using the same tools versions every time.

- activate/deactivate environment:

```
conda activate rna-seq-env/ conda deactivate
```

5. Required R libraries

- install them by running `install.packages()` or `BiocManager::install()`

Workflow

Part One

Download SRR files for CD14 monocytes in fastq format from Bioproject PRJNA577035 (GEO Accession number: GSE138746).

esearch searches sra and retrieves a list of unique identifiers that match the query

efetch retrived metadata table in csv format

cut and **grep** extract SRR numbers

awk and **sort** filteres and sorts the SRR numbers in the given range

echo saves filtered list to a .txt file

cat reads the list

fasterq-dump downloads FASTQ files and gzip compresses them

--split-files is chosen because of paired data

```
# Search SRR numbers associated with PRJNA577035

SRR_LIST=$(esearch -db sra -query "$PROJECT_ID" | efetch -format runinfo | cut -d',' -f1 | gre
p SRR)

# Save the full SRR list to a file and filter CD14 cells

echo "$SRR_LIST" | sort > "$SCRATCH_DIR/SRR_list.txt"

FILTERED_SRR_LIST=$(echo "$SRR_LIST" | awk '$1 >= "SRR10260429" && $1 <= "SRR1
0260508"'| sort)
```

```
# Save the filtered SRR list to a file in scratch
echo "$FILTERED_SRR_LIST" > "$SCRATCH_DIR/CD14_SRR_list.txt"
# Download and zip files
for SRR in $(cat "$SCRATCH_DIR/CD14_SRR_list.txt"); do
    echo "Downloading $SRR..."
    fasterq-dump --temp "$TEMP_DIR" --outdir "$OUTPUT_DIR" --split-files --details "$SRR"
    gzip "$OUTPUT_DIR/${SRR}"_*.fastq
done
```

Perform quality check control (QC) on .fasta files:

```
for fastq_file in "$OUTPUT_DIR"/*.fastq.gz; do
   fastqc -o "$QC_DIR" "$fastq_file"
done
```

Perform 3' and 5' end trimming on fasta files:

cutadapt trims bases from both forward and reverse reads

```
# Process all paired-end files in the input directory
for R1 in "$INPUT_DIR"/*_1.fastq.gz; do
    R2="${R1/_1.fastq.gz/_2.fastq.gz}"
    # Extract base name for output files
    BASENAME=$(basename "$R1" _1.fastq.gz)
    # Define output file paths
    TRIMMED_R1="$OUTPUT_DIR/${BASENAME}_trimmed_1.fastq"
    TRIMMED_R2="$OUTPUT_DIR/${BASENAME}_trimmed_2.fastq"
    # Run Cutadapt for trimming
  ~/.conda/envs/cutadapt_env/bin/cutadapt -u 10 -u -20 -U 10 -U -20 -o "$TRIMMED_R1" -p "$
TRIMMED_R2" "$R1" "$R2"
  gzip "$TRIMMED_R1"
  gzip "$TRIMMED_R2"
```

```
done
```

Perform second QC on trimmed files:

```
for trimmed_file in "$OUTPUT_DIR"/*.fastq.gz; do
   fastqc -o "$QC_DIR" "$trimmed_file"
done
```

Build index for pseudoalignment with Kallisto: get the file, unzip it, build index

```
wget ftp://ftp.ensembl.org/pub/release-110/fasta/homo_sapiens/cdna/Homo_sapiens.GRCh38.cdna.all.fa.gz -O /u/scratch/t/tosevsa2/kallisto/Homo_sapiens.GRCh38.cdna.all.fa.gz

gunzip Homo_sapiens.GRCh38.cdna.all.fa.gz

kallisto index -i /u/scratch/t/tosevsa2/kallisto/transcriptome.idx /u/scratch/t/tosevsa2/kallisto/Homo_sapiens.GRCh38.cdna.all.fa
```

Run Kallisto on trimmed files:

quant estimates transcript abundances through pseudoalignment

-b means boostrap replicates will be performed

last part renames abundance.tsv tables

```
# Define input files
R1="$INPUT_DIR/${SRR}_trimmed_1.fastq.gz"
R2="$INPUT_DIR/${SRR}_trimmed_2.fastq.gz"
# Define sample name and output directory
OUTPUT_DIR="$OUTPUT_BASE/$SRR"
mkdir -p "$OUTPUT_DIR"
# Run Kallisto and rename abundance.tsv to include the sample name
~/.conda/envs/kallisto/bin/kallisto quant -i "$INDEX_FILE" -o "$OUTPUT_DIR" -b 100 "$R1" "$R2"
   if [[ -f "$OUTPUT_DIR/abundance.tsv" ]]; then
      mv "$OUTPUT_DIR/abundance.tsv" "$OUTPUT_DIR/${SRR}_abundance.tsv"
   fi
```

Create transcript counts table:

```
# Initialize temporary file
TEMP_FILE="temp_counts.tsv"
# Find all SRR directories and sort them
SRR_DIRS=($(find "$KALLISTO_DIR" -maxdepth 1 -type d -name "SRR*" | sort))
# Loop through SRR directories, find abundance files, and extract counts
for SRR_DIR in "${SRR_DIRS[@]}"; do
    # Get the sample ID
    SRR_ID=$(basename "$SRR_DIR")
    # Find the abundance file
    ABUNDANCE_FILE=$(find "$SRR_DIR" -maxdepth 1 -type f -name "*_abundance.tsv" | he
ad -n 1)
    # If this is the first file, extract the target_id column for the header
    if [[ -z "$FIRST_FILE" ]]; then
        cut -f1 "$ABUNDANCE_FILE" > "$TEMP_FILE"
        FIRST_FILE="$ABUNDANCE_FILE"
    fi
```

Extract the counts columns and add them to the temporary file:

cut extracts transcript counts from the 4th column for each sample

paste combines the extracted counts with the existing temporary table

mv updates the temporary counts table

```
    cut -f4 "$ABUNDANCE_FILE" | tail -n +2 > "${SRR_ID}_counts.tmp"
    paste "$TEMP_FILE" "${SRR_ID}_counts.tmp" > "${TEMP_FILE}_new"
    mv "${TEMP_FILE}_new" "$TEMP_FILE"
    rm "${SRR_ID}_counts.tmp"
done
```

Add header to the table:

first part loops through the folder where SRR directories are and extracts their names

> creates the output file and writes the header row to it

cat displays the content of the file that stores transcript counts

>> apends the count data to the output file

```
HEADER="Transcript_ID"
for SRR_DIR in "${SRR_DIRS[@]}"; do
    HEADER+="\t$(basename "$SRR_DIR")"
done
echo -e "$HEADER" > "$OUTPUT_FILE"
cat "$TEMP_FILE" >> "$OUTPUT_FILE"
# Remove temporary file
rm "$TEMP_FILE"
```

Part two

Load libraries:

```
library(readr)
library(rtracklayer)
library(dplyr)
library(R.utils)
library(WGCNA)
library(DESeq2)
library(ggplot2)
library(reshape2)
library(GEOquery)
library(tidyr)
library(genefilter)
```

```
library(ggrepel)

library(matrixStats)

library(ComplexHeatmap)

library(circlize)

library(tibble)

library(pheatmap)

library(DescTools)
```

Import data into R and download GTF file (annotation file) which - provides information about genes in reference genome, such as gene_name, gene_id, and transcript_id.

```
# Import table

df2 <-readr::read_tsv("/Users/andreeaiuhaniak/Desktop/Software-Project/table_counts.tsv")

# Download GTF file

gtf_url <- "https://ftp.ensembl.org/pub/release-110/gtf/homo_sapiens/Homo_sapiens.GRCh38.110.gtf.gz"

dest_dir <- "/Users/andreeaiuhaniak/Desktop/Software-Project/"

dest_file <- paste0(dest_dir, "Homo_sapiens.GRCh38.110.gtf.gz")

download.file(gtf_url, destfile = dest_file, mode = "wb")

gunzip(dest_file, overwrite = TRUE)

# Load GTF file

gtf_data <- import("/Users/andreeaiuhaniak/Desktop/Software-Project/Homo_sapiens.GRCh38.110.gtf")

gtf_df <- as.data.frame(gtf_data)
```

Extract columns from GTF file and append them to a variable

```
# Extract columns from GTF file

transcript_to_gene <- gtf_data %>%

  as.data.frame() %>%

  filter(type == "transcript") %>%

  select(seqnames, transcript_id, gene_name, gene_id) %>%
```

```
  distinct()

colnames(transcript_to_gene) <- c("chromosome", "transcript_id", "gene_name", "gene_id")
```

Prepare gene counts data frame with names of genes as first column

```
# Adjust format of column Transcript ID

df2 <- df2 %>%

  mutate(Transcript_ID = sub("\\..*", "", Transcript_ID))

# Join data frames

df2 <- left_join(df2, transcript_to_gene, by= c("Transcript_ID" = "transcript_id"))

# Get gene counts values

gene_counts_name <- df2 %>% group_by (gene_name) %>% summarise(across(where(is.numeric), sum, na.rm = TRUE))
```

Import metadata table corresponding to Project_ID, amend it, and use it for the normalization step

```
# Meta_data table

gse <- getGEO(filename = "/Users/andreeaiuhaniak/Downloads/GSE138746_series_matrix.txt")

meta_data_table <- data.frame(Sample_ID = gse$geo_accession, Sex = gse$`Sex:ch1`, Age = gse$`age:ch1`, DiseaseState = gse$`disease state:ch1`, Drug = gse$`drug:ch1`, Response = gse$`response:ch1`, CellType = gse$`cell type:ch1`)

 # Create a vector of SRR numbers

srr_numbers <- paste0("SRR10260", 429:508) #sample number of monocytes

 # Initialize a new column with NA

meta_data_table$SRR_ID <- NA

 # Assign SRR numbers only to rows where CellTypes == "monocytes"

meta_data_table$SRR_ID[meta_data_table$CellType == "Monocytes"] <- srr_numbers

meta_data_table_monocytes <- meta_data_table[81:160, ] # keep only rows for monocytes

 # Adjust table

meta_data_table_monocytes <- meta_data_table_monocytes[ ,-1] #removes first column Sample ID
```

```
meta_data_table_monocytes <- meta_data_table_monocytes[, c(ncol(meta_data_table_monocytes), 1:(ncol(meta_data_table_monocytes) - 1))] #moves last column SRR ID to first position

meta_data_table_monocytes <- as.data.frame(meta_data_table_monocytes) # from tibble to df to set row names

row.names(meta_data_table_monocytes) <- meta_data_table_monocytes$SRR_ID #sets row names using SRR_IDs

meta_data_table_monocytes <- meta_data_table_monocytes[, -1] #removes SRR_ID col bc now it's stored as row names

meta_data_table_monocytes <- meta_data_table_monocytes[row.names(meta_data_table_monocytes) != "SRR10260461", ] #removes row with specific SRR ID
```

Set gene_names as row names and turn df into matrix

```
# Set gene_names as row names

gene_counts_name <- as.data.frame(gene_counts_name)

row.names(gene_counts_name) <- gene_counts_name$gene_name # sets gene_name column as row names

gene_counts_name <- gene_counts_name %>% select(-gene_name) #deletes column gene_name

# change to matrix format

gene_counts_name_m <- as.matrix(gene_counts_name)
```

Gene count table normalization

```
dds_1 <- DESeqDataSetFromMatrix(round(gene_counts_name_m),

                        meta_data_table_monocytes,

                        design = ~Drug + Response)

dds_1 <- DESeq(dds_1)

norm_counts_1 <- counts(dds_1, normalized = TRUE)  # normalized counts table

norm_counts_1_t <- t(norm_counts_1)  # prepare for WGCNA
```

Determine best soft-threshold power and plot results

```
powers <- c(1:20)  # Range of soft-thresholding powers

sft <- pickSoftThreshold(networkType= "signed", norm_counts_1_t, powerVector=powers, verbose=5)
```

```
 # plot mean connectivity and scale independence
par(mfrow = c(1,2));
cex1 = 0.9;
plot(sft$fitIndices[, 1],
    -sign(sft$fitIndices[, 3]) * sft$fitIndices[, 2],
    xlab = "Soft Threshold (power)",
    ylab = "Scale Free Topology Model Fit, signed R^2",
    main = paste("Scale independence")
)
text(sft$fitIndices[, 1],
    -sign(sft$fitIndices[, 3]) * sft$fitIndices[, 2],
    labels = powers, cex = cex1, col = "red"
)
abline(h = 0.90, col = "red")
plot(sft$fitIndices[, 1],
    sft$fitIndices[, 5],
    xlab = "Soft Threshold (power)",
    ylab = "Mean Connectivity",
    type = "n",
    main = paste("Mean connectivity")
)
text(sft$fitIndices[, 1],
    sft$fitIndices[, 5],
    labels = powers,
    cex = cex1, col = "red")
```

Run WGCNA

```
picked_power <- 9
```

```
netwk <- blockwiseModules(norm_counts_1_t,
                          power = picked_power,
                          networkType = "signed",
                          deepSplit = 3,
                          corType = "pearson",
                          maxBlockSize = 5000,
                          minModuleSize = 30,
                          mergeCutHeight = 0.25,
                          saveTOMs = TRUE,
                          saveTOMFileBase = "PW_9_30k_TOM",
                          numericLabels = TRUE,
                          verbose = 3)
# Convert labels to colors
mergedColors = labels2colors(netwk$colors)
```

Plot cluster dendrogram and create df with gene names and corresponding module color

```
# Plot the dendrogram and the module colors underneath
plotDendroAndColors(
  netwk$dendrograms[[1]],
  mergedColors[netwk$blockGenes[[1]]],
  "Module colors",
  dendroLabels = FALSE,
  hang = 0.03,
  addGuide = TRUE,
  guideHang = 0.05)
# Relate modules to samples
module_df <- data.frame(
  gene_name = names(netwk$colors),
```

```
  colors = labels2colors(netwk$colors)
)
```

Module with similar expression patterns are merged and the new module assignment plotted

```
# Merging of modules whose expression profiles are very similar
MEList <- moduleEigengenes(norm_counts_1_t, colors = netwk$colors, impute = TRUE)
MEs <- MEList$eigengenes
MEDissThres = 0.9  # Modules with a correlation ≥ (1 - MEDissThres) are merged
merge = mergeCloseModules(norm_counts_1_t, netwk$colors, cutHeight = MEDissThres, verbo
se = 3)
mergedColors_2 <- labels2colors(merge$colors)  # convert labels to colors
# Plot dendrogram
plotDendroAndColors(netwk$dendrograms[[1]], cbind(mergedColors[netwk$blockGenes[[1]]],
mergedColors_2[netwk$blockGenes[[1]]]),
                    c("Module Colors", "Merged Modules"), dendroLabels = FALSE, hang = 0.03, add
Guide = TRUE, guideHang = 0.05)
# Relate new modules to samples
module_df_merged <- data.frame(
  gene_name = names(merge$colors),
  colors = labels2colors(merge$colors) # base for enrichment analysis
)
```

Plot number of genes in each module

```
# Determine number of genes in modules
module_counts_new <- table(module_df_merged$colors)
module_counts_new_df <- as.data.frame(module_counts_new)
colnames(module_counts_new_df) <- c("Module", "Gene_Count")
# Bar plot all modules
ggplot(module_counts_new_df, aes(x=reorder(Module, -Gene_Count), y=Gene_Count, fill=Mod
ule)) +
```

```
  geom_bar(stat="identity", color="black") +  # Create bars with black border

  theme_minimal() +

  theme(axis.text.x = element_text(angle=45, hjust=1), legend.position = "none") +  # Rotate x-a
xis labels for readability

  labs(title="Number of Genes per Module", x="Module", y="Gene Count") +

  scale_fill_identity()

# Bar plot modules with <2000 genes

module_counts_new_df_1 <- module_counts_new_df[-c(14, 23, 26, 33, 34), ] #delete rows beca
use of high value

ggplot(module_counts_new_df_1, aes(x=reorder(Module, -Gene_Count), y=Gene_Count, fill=M
odule))+

  geom_bar(stat="identity", color="black") +  # Create bars with black border

  theme_minimal() +

  theme(axis.text.x = element_text(angle=45, hjust=1), legend.position = "none") +  # Rotate x-a
xis labels for readability

  labs(title="Number of Genes per Module", x="Module", y="Gene Count") +

  scale_fill_identity()
```

Determine Module-Trait relationship

```
nGenes <- ncol(norm_counts_1_t)

nSamples <- nrow(norm_counts_1_t)

# Calculate eigengene values

MEs0 <- moduleEigengenes(norm_counts_1_t, mergedColors_2)$eigengenes

MEs <- orderMEs(MEs0)

# Compute Pearson correlation between module eigengenes (MEs) and traits

modOXCor = cor(MEs, meta_table_monocytes_encoded[, colnames(meta_table_monocytes_enc
oded)[1:4]], use = "p")

# Format correlation for display

textMatrix1 = paste(signif(modOXCor, 2))

dim(textMatrix1) = dim(modOXCor)
```

```
# Generate correlation heatmap (ComplexHeatmap)
col_fun = colorRamp2(c(-1, 0, 1), c("blue", "white", "red"))
Heatmap(modOXCor, name = "Correlation",
        col = col_fun,
        column_names_side = "top",
        row_names_side = "left",
        row_names_gp = gpar(fontsize = 10),  # Adjust row label size
        column_names_gp = gpar(fontsize = 10),  # Adjust column label size
        cell_fun = function(j, i, x, y, width, height, fill) {
          grid.text(sprintf("%.2f", modOXCor[i, j]), x, y, gp = gpar(fontsize = 8))
        })
```

Visualize expression of genes in red module for response

```
# Box plot eigengenevalues red based on response
MEs_red <- MEs$MEred
eigengene_red <- data.frame(
  Sample_ID <- rownames(meta_data_table_monocytes),
  Eigengene_val <- MEs_red,
  Response <- meta_data_table_monocytes$Response
)
ggplot(eigengene_red, aes(x = Response, y = Eigengene_val, fill = Response)) +
  geom_boxplot() +
  theme_minimal() +
  labs(title = "Red Module Eigengene Values by Response",
       x = "Response", y = "Eigengene Value") +
  scale_fill_manual(values = c("no" = "blue", "good" = "magenta", "moderate" = "purple"))
```

Perform WGCNA on genes in the red module only

```
red_genes <- module_df_merged %>%
  filter(colors == "red") %>%
  pull(gene_name)
print(red_genes) # select genes in red module
red_genes_df <- data.frame(red_genes)
united_df <- red_genes_df %>%
  left_join(df2, by = c("red_genes" = "gene_name")) # merges df2 and red_genes_df based on ge
nes present in red_genes_df
united_df <- united_df[, -((ncol(united_df) - 1):ncol(united_df))] # deletes last two columns
row.names(united_df) <- united_df$Transcript_ID  # sets Transcript ID names as row names
united_df$Transcript_ID <- NULL # deletes Transcript ID
united_df$red_genes <- NULL # deletes column red_genes
```

Normalization of transcript counts

```
dds_2 <- DESeqDataSetFromMatrix(round(united_df),
                                meta_data_table_monocytes,
                                design = ~Drug + Response)
dds_2 <- DESeq(dds_2)
norm_counts_2 <- counts(dds_2, normalized = TRUE)
norm_counts_2_t <- t(norm_counts_2) #WGCNA input
```

Run WGCNA and plot cluster dendrogram

```
picked_power <- 9
netwk_1 <- blockwiseModules(norm_counts_2_t,
                            power = picked_power,
                            networkType = "signed",
                            deepSplit = 4,
```

```
                    corType = "pearson",
                    maxBlockSize = 4000,
                    minModuleSize = 30,
                    mergeCutHeight = 0.25,
                    saveTOMs = TRUE,
                    saveTOMFileBase = "PW_9_25k_TOM",
                    numericLabels = TRUE,
                    verbose = 3)
mergedcolors = labels2colors(netwk_1$colors)
plotDendroAndColors(
 netwk_1$dendrograms[[1]],
 mergedcolors[netwk_1$blockGenes[[1]]],
 "Module colors",
 dendroLabels = FALSE,
 hang = 0.03,
 addGuide = TRUE,
 guideHang = 0.05)
```

Merging of modules with similar expression patterns

```
MEList1 <- moduleEigengenes(norm_counts_1_t, colors = netwk$colors, impute = TRUE)
MEs1 <- MEList1$eigengenes
MEDissThres1 = 0.9  # modules with >= 0.1 correlation coefficient will be merged into one mod
ule
merge1 = mergeCloseModules(norm_counts_2_t, netwk_1$colors, cutHeight = MEDissThres1,
verbose = 3)  # identifies modules that are highly correlated and combines them into a single mo
dule
mergedColors_red <- merge1$colors  # New merged module assignments
mergedColors_red1 <- labels2colors(merge1$colors)
```

```
plotDendroAndColors(netwk_1$dendrograms[[1]], cbind(mergedcolors[netwk_1$blockGenes[[1]]], mergedColors_red1[netwk_1$blockGenes[[1]]]),
                    c("Module Colors", "Merged Modules"), dendroLabels = FALSE, hang = 0.03, addGuide = TRUE, guideHang = 0.05)
```

Relate samples to new modules and determine number of genes in each module

```
 # Relate new modules to sample
red_module_df_merged <- data.frame(
  gene_id = names(merge1$colors),
  colors = labels2colors(merge1$colors)
)
# Number of genes in modules
red_module_counts_new <- table(red_module_df_merged$colors)
red_module_counts_new_df <- as.data.frame(red_module_counts_new)
colnames(red_module_counts_new_df) <- c("Module", "Gene_Count")
```

Determine Module-Trait relationship for submodules of red module

```
nGenes1 <- ncol(norm_counts_2_t)
nSamples1 <- nrow(norm_counts_2_t)


# Calculate eigengene values
MEs0_ <- moduleEigengenes(norm_counts_2_t, mergedColors_red1)$eigengenes
MEs_ <- orderMEs(MEs0_)


# Compute Pearson correlation between module eigengenes (MEs) and traits
modOXCor1 = cor(MEs_, meta_table_monocytes_encoded[, colnames(meta_table_monocytes_encoded)[1:4]], use = "p")
# Format the correlation for display
textMatrix2 = paste(signif(modOXCor1, 2))
dim(textMatrix2) = dim(modOXCor1)
```

```
# Generate correlation heatmap for red module (ComplexHeatmap)
col_fun = colorRamp2(c(-1, 0, 1), c("blue", "white", "red"))
Heatmap(modOXCor1, name = "Correlation",
        col = col_fun,
        column_names_side = "top",
        row_names_side = "left",
        row_names_gp = gpar(fontsize = 10),  # Adjust row label size
        column_names_gp = gpar(fontsize = 10),  # Adjust column label size
        cell_fun = function(j, i, x, y, width, height, fill) {
          grid.text(sprintf("%.2f", modOXCor1[i, j]), x, y, gp = gpar(fontsize = 8))
        })
```

Visualize number of genes in submodules of red module

```
# Bar plot all submodules in red modules
ggplot(red_module_counts_new_df, aes(x=reorder(Module, -Gene_Count), y=Gene_Count, fill=
Module)) +
  geom_bar(stat="identity", color="black") +  # Create bars with black border
  theme_minimal() +
  theme(axis.text.x = element_text(angle=45, hjust=1), legend.position = "none", plot.title = elem
ent_text(hjust = 0.5)) +  # Rotate x-axis labels for readability
  labs(title="Number of Genes per Module", x="Module", y="Gene Count") +
  scale_fill_identity()


# Bar plot modules with <2000 genes
red_module_counts_new_df_1 <- red_module_counts_new_df[-c(16, 30, 26, 12, 37), ] #delete ro
ws because of high value
ggplot(red_module_counts_new_df_1, aes(x=reorder(Module, -Gene_Count), y=Gene_Count, fi
ll=Module)) +
  geom_bar(stat="identity", color="black") +  # Create bars with black border
```

```
  theme_minimal() +
  theme(axis.text.x = element_text(angle=45, hjust=1), legend.position = "none", plot.title = elem
ent_text(hjust = 0.5)) +  # Rotate x-axis labels for readability
  labs(title="Number of Genes per Module", x="Module", y="Gene Count") +
  scale_fill_identity()
```

Check likelihood of transcripts from the same gene to be in same module color

```
# Count pairs of transcripts per gene and module
same_module_pairs <- united_df_1_filtered %>%
  group_by(red_genes, colors) %>%
  summarise(PairsInSameModule = choose(n(), 2)) %>%
  ungroup() %>%
  summarise(TotalSameModulePairs = sum(PairsInSameModule, na.rm = TRUE))
# Calculate total transcript pairs per gene
total_pairs <- united_df_1_filtered %>%
  group_by(red_genes) %>%
  summarise(TotalPairs = choose(n(), 2)) %>%
  ungroup() %>%
  summarise(TotalPairsSum = sum(TotalPairs, na.rm = TRUE))
# Calculate the expected pairs in the same module
num_modules <- length(unique(united_df_1_filtered$colors)) # numbers of module colors prese
nt
expected_pairs <- total_pairs$TotalPairsSum / num_modules
# Contingency Table
contingency_table <- matrix(c(same_module_pairs$TotalSameModulePairs,
                              total_pairs$TotalPairsSum - same_module_pairs$TotalSameModulePairs,
                              expected_pairs,
                              total_pairs$TotalPairsSum - expected_pairs),
                            nrow = 2,
```

```
                        byrow = TRUE)
colnames(contingency_table) <- c("Same Module", "Different Module")
rownames(contingency_table) <- c("Observed", "Expected")
chisq_test <- chisq.test(contingency_table)
print(chisq_test) # very significant; reject H0 which means strong likelihood that transcripts that come from the same gene are in the same module
```

Visualize expression of transcripts of HGS (gene with most transcripts in our dataset)

```
gene_most_transcript <- which.max(united_df_1_filtered$TranscriptCount) #row 9831
subset_united_df_filtered <- united_df_1_filtered[united_df_1_filtered$red_genes == "HGS", c( "red_genes", "colors")]
transcript_number <- as.data.frame(table(subset_united_df_filtered$colors))
ggplot(transcript_number, aes(x = Var1, y = Freq, fill = Var1)) +
 geom_bar(stat = "identity") +
 scale_x_discrete(guide = guide_axis(angle = 45)) + #names on X axis tilted
 labs(
  title = paste("Transcript Module Distribution for HGS"),
  x = "Module Color",
  y = "Transcript Number"
 ) +
 theme_minimal() +
 theme(plot.title = element_text(hjust = 0.5)) + #title in the middle
 scale_fill_identity()
```

Check the expression of HGS transcripts that are assigned to magenta module; the same was done for transcripts assigned to midnightblue module

```
 # Filter HGS genes
subset_united_df_filtered_expression <- united_df_1_filtered[united_df_1_filtered$red_genes == = "HGS", ]
  # Filter rows for magenta transcripts & drop unwanted columns
```

```
magenta_module <- subset_united_df_filtered_expression %>%
  ungroup() %>%
  filter(colors == "magenta") %>%
  select(-'red_genes', -'TranscriptCount', -'colors')
magenta_module_long <- magenta_module %>% #pivot data to long format
  pivot_longer(
    cols = starts_with("SRR"),
    names_to = "sample",
    values_to = "expression"
  )
   # Calculate average expression
average_expression_magenta <- magenta_module_long %>% group_by(Transcript_ID) %>% su
mmarise(avg_expression = mean(expression, na.rm = TRUE))
   # Plot the average expression for each transcript id
ggplot(average_expression_magenta, aes(x = Transcript_ID, y = avg_expression)) +
  geom_bar(stat = "identity", fill = "magenta") +
  labs(title = "Average Expression of Transcripts in Module Magenta",
       x = "Transcript ID",
       y = "Average Expression") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Download and import splicing factors for co-splicing analysis

```
url <- "http://srv00.recas.ba.infn.it/SpliceAidF/download.php?t=f&s=all"
destination <- "/Users/andreeaiuhaniak/Desktop/Software-Project/"
destination_file <- paste0(destination, "spliceaidf_data.tsv") # create full path
download.file(url, destfile = destination_file, mode = "w")
splice_data <- read_tsv("/Users/andreeaiuhaniak/Desktop/Software-Project/spliceaidf_data.tsv")
```

```
splice_data_df <- as.data.frame(splice_data)
```

Checks if there are any splicing factors in our dataset

```
matching_genes_all_modules <- subset(splice_data_df, Gene %in% rownames(norm_counts_1_df))

# matching splicing factors in all modules

norm_counts_1_df <- as.data.frame(norm_counts_1)

norm_counts_3 <- rownames_to_column(norm_counts_1_df, var = "Gene_name") # row names to column "gene_name"

# keeps rows where gene_name column is present in gene column of matching_genes_all_modules

norm_counts_3 <- subset(norm_counts_3, Gene_name %in% matching_genes_all_modules$Gene)

# identifies which modules the selected genes belong to

which_modules <- subset(module_df_merged, gene_id %in% matching_genes_all_modules$Gene)
```

Data preparation for WGCNA

```
row.names(norm_counts_3) <- norm_counts_3$Gene_name  # turns gene_name values into row names

norm_counts_3 <- norm_counts_3 %>% select(-Gene_name) # deletes column gene_name

norm_counts_3_t <- t(norm_counts_3) # transposes df; WGCNA input
```

Perform co-splicing analysis using WGCNA

```
picked_power <- 9

netwk_2 <- blockwiseModules(norm_counts_3_t,

                            power = picked_power,

                            networkType = "signed",

                            deepSplit = 3,

                            corType = "pearson",

                            maxBlockSize = 5000,

                            minModuleSize = 10,
```

```
                    mergeCutHeight = 0.25,
                    saveTOMs = TRUE,
                    saveTOMFileBase = "PW_8_366_TOM",
                    numericLabels = TRUE,
                    verbose = 3).
mergedcolors1 <- labels2colors(netwk_2$colors)
# Plot cluster dendrogram
plotDendroAndColors(
 netwk_2$dendrograms[[1]],
 mergedcolors1[netwk_2$blockGenes[[1]]],
 "Module colors",
 dendroLabels = FALSE,
 hang = 0.03,
 addGuide = TRUE,
 guideHang = 0.05)
```

Visualize splicing factors expression correlation with traits (e.g. age, drug, response, sex)

```
# Compute correlation matrix
cor_matrix <- cor((norm_counts_3_t), meta_table_monocytes_encoded, method = "pearson", use = "pairwise.complete.obs")
# Heatmap
pheatmap(cor_matrix,
        cluster_rows = TRUE,
        cluster_cols = TRUE,
        display_numbers = TRUE,
        color = colorRampPalette(c("blue", "white", "red"))(50),
        main = "Correlation between splicing factors expression and traits")
```

Determine which module do genes belong to

```
ggplot(which_modules, aes(x = gene_id, y = colors, fill = colors)) +
  geom_tile() +
  theme_minimal() +
  labs(title = "Splicing factors-Module Association",
       x = "Gene",
       y = "Module Color") +
  scale_fill_identity() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1),
        legend.position = "none", plot.title = element_text(hjust = 0.5))
```

Summary

This guide outlines a **comprehensive user-level documentation** template tailored for an **RNA-Seq co-expression and co-splicing analysis workflow.**