# Appendix

The appendix provides the code used to reproduce the RNA-Seq analyses in the paper provided.

# Initial set-up of work environment and data

The first step was to set up the work environment and data on the HPC, where we navigated into our scratch place and started an interactive job.

1. Log into the CREATE node and start an interactive job

ssh -i ~/.ssh/ed\_create youremail@hpc.create.kcl.ac.uk

1. Navigate to the scratch directory and star an interactive job

cd /scratch\_tmp/grp/msc\_appbio/group4\_tmp/ #navigating to the scratch directory  
  
srun -p msc\_appbio --pty /bin/bash   
  
#srun - starts an interactive session on a computer node  
# -p specifies the partition used which is the msc\_appbio   
# --pty /bin/bash specifies the shell used in the session is Bash

1. Create new directories to store files

mkdir raw\_data\_rna  
mkdir processed\_data   
mkdir ref\_genome  
mkdir sorted\_and\_indexed\_output  
mkdir index  
mkdir htseq\_counts

1. Install and activate the tools needed for environment setup using Conda in the environment.sh nano file

nano environment.sh #creates nano script

In nano script

#!/bin/bash  
  
echo "start of the pipeline"  
  
# Add conda channels  
conda config --add channels bioconda   
conda config --add channels conda-forge  
  
# Create a new conda environment with specific versions specified in the paper  
conda create -n bioenv fastqc=0.11.5 cutadapt=2.3 faqcs=2.08 star= 2.5.2a fdrtool= 1.2.15 htseq= 0.9.1 -y  
  
# Print a message about how to activate the environment  
echo "To activate your environment, run: conda activate bioenv"   
  
echo "end of the pipeline"  
  
#Activate the Conda environment with specified tool versions  
conda activate bioenv

# Downloading the fastq files and reference genome

1. Create a nano script containing the fastq URLs downloaded from <https://www.ebi.ac.uk/biostudies/arrayexpress/studies/E-MTAB-9499>

nano links2download.txt #contains all the fastq URLs

1. Write the download.sh script using wget command

Bash script (download.sh) was created to automate the download process using the wget command:

nano download.sh #creates nano script  
  
#!/bin/bash  
#SBATCH --job-name=download\_job # Job name  
#SBATCH --output=download\_job\_%j.log # Output log file (%j will be replaced with job ID)  
#SBATCH --error=download\_job\_%j.err # Error log file (%j will be replaced with job ID)  
#SBATCH --ntasks=1 # Number of tasks (processes)  
#SBATCH --time=00:30:00 # Time limit (hh:mm:ss)  
#SBATCH --mem=1G # Memory per node (adjust as needed)  
#SBATCH --partition=standard # Partition to submit to (adjust to your cluster setup)  
  
# Load wget  
module load wget  
  
# Text file containing URLs  
LINK\_FILE="/scratch\_tmp/grp/msc\_appbio/practice/links2download.txt"  
  
# Check if the file exists  
if [[ ! -f "$LINK\_FILE" ]]; then  
echo "Error: $LINK\_FILE does not exist."  
exit 1  
fi  
  
# Create a directory to store the downloaded files  
DOWNLOAD\_DIR="/scratch\_tmp/grp/msc\_appbio/practice/downloads"  
mkdir -p "$DOWNLOAD\_DIR"  
  
# Read each line of URL from the file and download it  
while IFS= read -r URL; do  
if [[ -n "$URL" ]]; then  
echo "Downloading: $URL"  
wget -P "$DOWNLOAD\_DIR" "$URL" || echo "Failed to download: $URL"  
fi  
done < "$LINK\_FILE"  
  
echo "Download complete. Files saved in '$DOWNLOAD\_DIR'."

1. Obtain the reference genome

The NIH webpage was visited, and the name of the reference genome (GCA\_000269885) was pasted in the search bar

The data set was downloaded onto our local terminals and then pushed data into CREATE using sftp:

sftp -i~/.ssh/create\_msc k24088153@hpc.create.kcl.ac.uk #logs into the secure file transfer protocol  
sftp> put GCA\_000269885.1 #file with the reference genome from our local is transported to our ref\_genome directory on the HPC.  
  
#Within GCA\_000269885.1  
  
# GCA\_000269885.1\_ASM26988v1\_genomic.fna   
# genomic.gtf

The output is a list of fastq.gz files from the RNA-seq data and a gtf and .fna file for the reference genome.

# 

# Quality Control of our files, using fastqc

1. Create a Nano script to fastqc the fastq.gz files

nano fastqc\_RNA #creates nano script  
  
In the nano script:  
#!/bin/bash  
  
echo "start of the pipeline"  
  
#Load Fastqc - if not loaded previously   
module load fastqc   
  
# Define base and results directories  
baseDirectory="/scratch\_tmp/grp/msc\_appbio/group4\_tmp/raw\_data\_rna/fastq"  
resultsDirectory="/scratch/grp/msc\_appbio/group4\_tmp/processed\_data/"  
  
# Create the results directory   
mkdir -p "$resultsDirectory"  
  
# Run Fastqc on all .fastq.gz files in basedirectory  
fastqc -o "$resultsDirectory" -t 4 "$baseDirectory"/\*.fastq.gz #(-o represents the output directory -t 4 specifies that 4 threads were used)  
  
echo "end of the pipeline"

The output should be a series of fastqc.html files which can be downloaded and observed for adapter content and for the quality of the fastq files.

# Aligning the Fastq reads to the reference genome of S. cerevisiae CEN.PK113‐7D using STAR

The next step was to align the reads to the reference genome using STAR. In order to use STAR, the reference needed to be indexed.

1. Creating an alignment index

nano star\_index.sh #creates a nano script  
  
In the nano script:  
#!/bin/bash   
  
#Loading STAR if not already loaded  
module load star   
  
#Making the index  
STAR   
--runThreadN 8 #number of threads specified for fast processing  
--runMode genomeGenerate #mode is set to genomeGenerate

--genomeDir /scratch\_tmp/grp/msc\_appbio/group4\_tmp/index #provides genome index directory path   
--genomeFastaFiles /scratch\_tmp/grp/msc\_appbio/group4\_tmp/ref\_genome/GCA\_000269885.1\_ASM26988v1\_genomic.fna #defining path of fasta files   
--sjdbGTFfile /scratch\_tmp/grp/msc\_appbio/group4\_tmp/ref\_genome/genomic.gtf #provides the GTF file

The output of indexing provides STAR the ability to perform alignment and importantly how the reads from RNA-Seq align to the reference genome.

1. Alignment of RNA-Seq raw files to the reference genome using STAR

nano star\_alignment.sh #creates nano script  
  
In the nano script:  
  
#!/bin/bash  
  
#SBATCH --job-name=star\_alignment # Job name  
#SBATCH --output=star\_alignment.log # Output log file  
#SBATCH --error=star\_alignment.err # Error log file  
#SBATCH --time=06:00:00 # Time limit (hh:mm:ss)  
#SBATCH --cpus-per-task=4 # Number of CPU cores  
#SBATCH --mem=32G # Memory pool for the job  
#SBATCH --partition=msc\_appbio # Partition to submit to (adjust based on your cluster)  
#SBATCH --mail-type=END,FAIL # Notifications for job done & fail  
#SBATCH --mail-user=k24023654@kcl.ac.uk # Email for notification  
  
# Parameters - customize these paths  
INDEX\_PATH="../../index/" # Path to STAR indexed genome  
REF\_GENOME\_DIR="/scratch\_tmp/grp/msc\_appbio/group4\_tmp/ref\_genome/GCA\_000269885.1\_ASM26988v1\_genomic.fna" # Path to reference genome  
FASTQ\_DIR="../../raw\_data\_rna/fastq/" #Fastqc directory path  
OUTPUT\_DIR="../../star\_output" # STAR output directory path  
THREADS=4 # Number of threads for STAR  
  
# load star  
module load star/2.7.10b-gcc-13.2.0  
  
  
# Create output directory if it doesn't exist  
mkdir -p $OUTPUT\_DIR  
  
# Loop through each fastq.gz file in the raw\_data\_rna/fastq directory  
for FASTQ in $FASTQ\_DIR/\*.fastq.gz;do  
 BASENAME=$(basename $FASTQ .fastq.gz)  
  
 # Run STAR  
 echo "Running STAR alignment for $BASENAME..."  
  
 STAR --genomeDir $INDEX\_PATH \  
 --readFilesIn $FASTQ \  
 --readFilesCommand zcat \  
 --runThreadN $THREADS \  
 --genomeLoad NoSharedMemory \  
 --outFilterMultimapNmax 20 \  
 --alignSJoverhangMin 8 \  
 --alignSJDBoverhangMin 1 \  
 --outFilterMismatchNmax 999 \  
 --outFilterMismatchNoverReadLmax 0.04 \  
 --alignIntronMin 20 \  
 --alignIntronMax 1000000 \  
 --alignMatesGapMax 1000000 \  
 --outSAMheaderHD @HD VN:1.4 SO:coordinate \  
 --outSAMunmapped Within \  
 --outFilterType BySJout \  
 --outFileNamePrefix ${OUTPUT\_DIR}/${BASENAME}\_ \  
 --outSAMattributes NH HI AS NM MD \  
 --outSAMstrandField intronMotif \  
 --outSAMtype BAM SortedByCoordinate \  
 --quantMode TranscriptomeSAM GeneCounts \  
 --sjdbScore 1 \  
 --limitBAMsortRAM 8000000000  
  
 echo "Alignment completed for $BASENAME. Output saved to ${OUTPUT\_DIR}/${BASENAME}\_"  
done

The output should be \_Aligned.out.bam files for the 20 fastq files

1. Sort and index the BAM files using samtools

This improves the efficiency of the RNA-Seq data for downstream analysis

nano sort\_and\_index\_coord\_bams.sh #creates a nano script  
  
In the nano script:  
  
#!/bin/bash  
#SBATCH --output=sort\_index\_%j.log # Standard output log file  
#SBATCH --error=sort\_index\_%j.err # Standard error log file  
#SBATCH --time=02:00:00 # Time limit (hh:mm:ss)  
#SBATCH --mem=8G # Memory limit  
#SBATCH --ntasks=1 # Number of tasks  
#SBATCH --cpus-per-task=1 # Number of CPU cores  
#SBATCH --partition=msc\_appbio # Partition to use  
  
# Load samtools module  
module load samtools/1.17-gcc-13.2.0-python-3.11.6  
  
# Base directory containing the subfolders with BAM files  
BASE\_DIR="/scratch\_tmp/grp/msc\_appbio/group4\_tmp/sorted\_and\_indexed\_output"  
  
echo "Starting sorting and indexing of sortedByCoord BAM files..."  
  
# Loop through each folder and process the BAM file  
for SEQ\_DIR in "$BASE\_DIR"/\*; do  
 # Check if SEQ\_DIR is a directory  
 if [ -d "$SEQ\_DIR" ]; then  
 # Find the BAM file in the directory  
 BAM\_FILE=$(find "$SEQ\_DIR" -maxdepth 1 -type f -name "\*\_Aligned.sortedByCoord.out.bam")  
   
 # Skip if no BAM file is found  
 if [ -z "$BAM\_FILE" ]; then  
 echo "No BAM file found in $SEQ\_DIR, skipping..."  
 continue  
 fi  
  
 # Extract file and output names  
 BAM\_FILENAME=$(basename "$BAM\_FILE" .bam) # Original BAM file name  
 OUTPUT\_BAM="$SEQ\_DIR/$BAM\_FILENAME"\_sorted.bam # Sorted BAM file  
 echo "Processing file: $BAM\_FILE"  
  
 # Sort BAM file by coordinate (in case further sorting is needed)  
 samtools sort -o "$OUTPUT\_BAM" "$BAM\_FILE"  
  
 # Index the sorted BAM file  
 samtools index "$OUTPUT\_BAM"  
  
 echo "Sorted and indexed: $OUTPUT\_BAM"  
 fi  
done  
  
echo "All BAM files have been processed."

The output should be .sortedByCoord.out.sorted.bam files and should highlight the mapped reads

# Feature counts with HTSeq

Then the mapped reads were used to create gene-level read count tables using HTSeq, essential for downstream analyses.

nano counts.sh #creates a nano script   
  
In the nano script:  
  
#!bin/bash  
  
#Define paths  
baseDir="/scratch\_tmp/grp/msc\_appbio/group4\_tmp"  
bamDir="${baseDir}/sorted\_and\_indexed\_output"  
gtf\_file="${baseDir}/ref\_genome/genomic.gtf"  
outputDir="${baseDir}/htseq\_counts"  
  
#Create output directory if it doesn't exist   
mkdir -p "$outputDir"  
  
#Loop through all BAM files in the subdirectories matching ERR4553\*  
for bamfile in "$bamDir"/ERR4553\*/\*sortedByCoord.out\_sorted.bam; do  
  
 #Extract sample name from the BAM file.  
 sampleName=$(basename "$bamfile" .bam)  
  
 #Define the output counts file   
 outputCounts="${outputDir}/counts\_${sampleName}.txt"  
  
 #Run HTSeq-Count for the BAM files, annotating it with GTF file  
 echo "Running HTSeq-counts for sample" "$sampleName"  
 htseq-count -f bam -r pos -s yes "$bamfile" "$gtf\_file" > "$outputCount"  
  
# -f bam specifies that the input is a bam file  
# -r pos specifies that reads are sorted by position  
# -s yes specifies that the RNA-Seq data is strand specific  
  
 # Check if HTSeq-Count ran successfully  
 if [[ $? -eq 0 ]]; then   
 echo "Counts for $sampleName directed to $outputCounts"  
 else  
 echo "Error: HTSeq-Count failed for $sampleName. Check input files."  
 fi  
done  
  
  
echo "Gene counting with HTSeq is completed for all samples. Results are in $outputDir."

# Differential analysis using DeSeq2 (In R)

Downstream analysis of gene counting was done using the DeSeq2 tool in R. DeSeq2 tells us whether the genes are differentially expressed between the different yeast strains and their evolution patterns.

1. Set the required working directory and install/load packages that are needed

setwd("~/Desktop/MSc\_appbio\_group4/")  
  
#Install the packages needed  
install.packages("BiocManager")  
BiocManager::install("DESeq2")   
BiocManager::install(c("GenomicFeatures", "txdbmaker"))   
  
library(DESeq2)  
library(GenomicFeatures)  
library(txdbmaker)  
library(ggplot2)  
library(Rtsne)  
library(umap)  
library(matrixStats)

1. Read and load the count files and metadata

#List and filter the files collected from HTSeq count data   
count\_file\_list <- list.files(path = "htseq\_counts/", pattern = "\*.txt", full.names = TRUE)  
  
#View the count.txt files  
count\_file\_list  
  
#List the metadata   
metadata <- read.csv(file ="phenotype\_metadata.csv", header = TRUE, row.names = 1)  
  
#View the metadata  
head(metadata)

1. Extract and Save gene-level annotations from GTF file in a GTF format

#Load the GTF file and convert it into a TxDb  
txdb <- makeTxDbFromGFF(file = "genomic.gtf", format = "gtf")  
txdb  
  
# Extract gene data for gene reference list  
genes <- genes(txdb)  
genes\_df <- as.data.frame(genes)  
  
# Extract only the gene IDs (rownames from genes\_df)  
gene\_ids <- rownames(genes\_df)  
  
# View the first few gene IDs  
head(gene\_ids)  
  
  
# Save the gene list to a file  
write.table(gene\_ids, file = "gene\_ids.txt", quote = FALSE, row.names = FALSE, col.names = FALSE)  
  
#Read the gene\_ids.txt into a new data frame   
reference\_genes <- read.table("gene\_ids.txt", header = FALSE, stringsAsFactors = FALSE)  
  
#Extracts first column from reference\_genes, containing gene\_ids  
reference\_genes <- reference\_genes$V1

1. Align and Process Gene count data to the reference genome

#Function to Align Counts to Reference Genes  
read\_and\_align <- function(file, reference\_genes) {  
   
 #Read the HTSeq file  
 df <- read.table(file, header = FALSE, row.names = 1, col.names = c("gene\_id", "count"))  
   
 #Remove summary rows (e.g., \_\_no\_feature, \_\_ambiguous)  
 df <- df[!grepl("^\_\_", rownames(df)), , drop = FALSE]  
  
#Create a full matrix aligned to the reference genes  
 aligned\_df <- data.frame(count = rep(0, length(reference\_genes)), row.names = reference\_genes)  
   
 # Fill in the counts for existing genes  
 aligned\_df[rownames(df), "count"] <- df$count  
   
 # Return the aligned counts  
 colnames(aligned\_df) <- gsub(".txt$", "", basename(file)) #Use the sample name as column name  
 return(aligned\_df)  
}  
  
# Read and align all files to the reference gene list  
aligned\_counts <- lapply(count\_file\_list, function(file) read\_and\_align(file, reference\_genes))  
  
# Combine all aligned data frames into a single matrix  
count\_matrix <- do.call(cbind, aligned\_counts)  
  
# Check the resulting matrix  
head(count\_matrix)  
tail(count\_matrix)  
  
# Check if metadata row names match count matrix column names   
all(rownames(metadata) == colnames(count\_matrix))  
  
# Check for NA values  
sum(is.na(count\_matrix))

1. Creating a DeSeq2 dataset and filtering low count genes

The DeSeq2 data set will manage and store the RNA-Seq data and metadata.

# Creating the deseq2 data set using count matrix and phenotype metadata  
dds <- DESeqDataSetFromMatrix(countData = count\_matrix,  
 colData = metadata,  
 design = ~ phenotype)  
  
#Filtering low-count genes to those with at least 10 reads in at least 2 samples   
dds <- dds[rowSums(counts(dds) >= 10) >= 2, ]

1. Normalize the data and handle missing values from differential gene analysis

#Normalising and fitting the model  
dds <- DESeq(dds) #contains normalised counts   
  
#Extracting and inspecting results  
results <- results(dds)  
head(results)  
  
#Check for NA values in the results data frame  
sum(is.na(results))   
  
#Identify where the missing values are from  
sum(is.na(results$pvalue)) # looks at NAs in p-value (5 NAs)  
  
#Excluding missing values (NA) from the analysis  
exNA\_results <- results[!is.na(results$pvalue), ]  
head(exNA\_results)  
  
#Confirming whether NA values have been excluded  
sum(is.na(exNA\_results))  
sum(is.na(exNA\_results$padj))

# Adjusting the P-values with the FDR tool for Multiple Testing Correction

For differential gene expression analysis in RNA-Seq, each gene will be tested for statistical significance. Loads of statistical tests are performed and these multiple tests should be corrected for. If not, the number of genes being highlighted as differentially expressed will be inaccurate, leading to incorrect conclusions and invalid downstream analysis.

1. Install and load the necessary packages

install.packages("fdrtool")  
library(fdrtool)

1. Multiple Testing Correction with FDR

#Extract p-values from 'results' and assign to pvalues  
pvalues <- results$pvalue  
  
#Applying 'fdrtool' to the pvalues   
fdr\_results <- fdrtool(pvalues, statistic = "pvalue")  
  
#Displays summary of results  
summary(fdr\_results)  
  
# Add q-values to the original results table  
results$qval <- fdr\_results$qval  
  
head(results)  
  
significant\_genes <- subset(results, qval < 0.1)  
head(significant\_genes)  
significant\_genes

1. Saving the data in a portable csv format

#Saving differential expression analysis data to a csv file  
write.csv(as.data.frame(exNA\_results), file = "differential\_expression\_results.csv")  
  
#Saving the significant genes (with q-value < 0.1) to a csv file (from fdrtools)  
write.csv(as.data.frame(significant\_genes), file = "significant\_genes\_results.csv")

# Downstream analysis visualisation

## Creating a heatmap to visualise the differentially expressed genes from DESeq2 data (In R)

* 1. Install and load the packages needed

#Install packages   
install.packages("pheatmap")  
  
#Load packages  
library(DESeq2)  
library(reshape2)  
library(ggplot2)  
library(pheatmap)

* 1. Import DESeq dataset and save it as an RDS

#Read the significant genes csv from DeSeq2 analysis into R  
results\_df <- read.csv("significant\_genes\_results.csv", row.names = 1)  
  
#Save the data frame as an RDS file with the name 'dds\_object.rds'  
saveRDS(results\_df, file = "dds\_object.rds")  
  
#Loading the RDS file  
dds <- readRDS("dds\_object.rds")   
  
#View the data set   
dds

* 1. Transform the data set

Transforming the data set into a regularized log transformation makes RNA-Seq data suitable for visualization and analysis such as in heatmaps

#Apply a regularized log transformation   
rld = rlog(dds, blind = TRUE)   
#blind = TRUE makes sure that variances are stabilized across all genes without taking into account experimental groups.   
  
#View the dataset  
rld

1. Extract phenotype and strain information to annotate the Heatmap

#Extract Column data from Transformed dataset  
col\_data = colData(rld)   
  
#Extract the Phenotype column from the metadata in col\_data  
phenotype = col\_data$phenotype   
  
#Extract the Strain column from the metadata in col\_data  
strain = col\_data$strain

1. Extract Transformed Data for the Heatmap for normalization

#Extract numerical matrix of log-transformed gene expression data   
rld\_matrix = assay(rld)   
  
#View first three rows of the matrix  
head(rld\_matrix,3)

1. Select the 50 most highly expressed genes from the rld\_matrix dataset

#Identify Top 50 highly expressed genes   
top\_genes = head(order(rowMeans(rld\_matrix), decreasing = TRUE), 50) # rowMeans will calculate the average expression level of each row (gene) and decreasing = TRUE, selects the top 50 genes in descending order.  
  
#Extract Top 50 Data for Heatmap  
heatmap\_data = rld\_matrix[top\_genes,] #ensures that the heatmap data has the top 50 gene expression values.   
  
#Display first 3 rows of the heatmap\_data matrix  
head(heatmap\_data,3)

1. Create the heatmap

#Create a matrix of sample annotations (phenotype and strain)  
sample\_annotation <- data.frame(Phenotype = phenotype, Strain = strain)   
  
#Set rownames to align with column names of rld\_matrix  
rownames(sample\_annotation) <- colnames(rld\_matrix)   
  
  
#Creating the heatmap  
pheatmap(rld\_matrix[top\_genes, ],  
 annotation\_col = sample\_annotation, # Add phenotype and strain annotations  
 cluster\_rows = FALSE, # Cluster genes  
 cluster\_cols = TRUE, # Cluster samples  
 show\_rownames = TRUE, # Hide gene names  
 show\_colnames = TRUE, # Show sample names  
 main = "Top 50 Variable Genes",  
 fontsize\_row = 3.5)

A Heatmap showing the top 50 highly expressed genes will be shown, showing the differentially expressed genes across the sample types with different phenotypes and evolutionary patterns

## Create a PCA (Principal Component Analysis) plot

This is vital for RNA-Seq analysis, showing how the samples cluster depending on their gene expression patterns. Those with similar gene expression patterns will cluster together.

1. Install/load necessary packages

install.packages("ggfortify")  
library(ggplot2)  
library(ggfortify)

1. Pre-processing RNA-Seq data and conducting PCA to visualise gene variability

#Use count\_matrix data for PCA data  
head(count\_matrix)  
  
#Establishing PCA variability across columns   
zero\_var\_cols <- apply(count\_matrix, 1, function(x) var(x) == 0) #identifies rows where variance=0 and removes them  
  
print(colnames(count\_matrix)[zero\_var\_cols])  
  
#Filtering constant 0s  
filtered\_matrix <- count\_matrix[apply(count\_matrix, 1, function(x) var(x) > 0), ]  
  
#Applying PCA on transposed and scaled data matrix  
pca <- prcomp(t(filtered\_matrix), scale. = TRUE)  
summary(pca)  
  
#Showing string of phenotype metadata and strain metadata  
metadata$phenotype  
metadata$strain

1. PCA plotting

#Create a PCA plot, phenotype is colored, strain is shaped  
PCAplot <- ggplot(pca, aes(x = PC1, y = PC2, colour = metadata$phenotype, shape = metadata$strain)) + #colour= phenotype and shape=strain  
 geom\_point(size = 3)+ #point size is set for 3   
 labs(title = "Transcriptomics PCA Plot", x = "PC1: 35.5% variance", y = "PC2: 28.6% variance", colour = "Phenotype", shape = "Strain Type") + #add titles and axes labels, explaining %variance in PC1 and 2  
 scale\_color\_manual(values = c("fumarate\_producer" = "indianred2", "malate\_producer" = "olivedrab3", "succinate\_producer" = "turquoise3", "wild\_type" = "orchid2"), labels = c("Fumarate Producer", "Malate Producer", "Succinate Producer", "Wild Type")) + #set the colors for the different phenotypes and apply their labels  
 scale\_shape\_manual(values = c("parental" = 15, "evolved" = 16, "wild\_type" = 17), labels = c("Parental", "Evolved", "Wild Type")) + #set shapes to the different strains and apply their labels  
 theme\_bw() + #set a white background for the figure  
 theme(legend.position = "bottom") #legend is positioned at the bottom of the figure  
  
#Saving plot as image  
ggsave("PCAplot.png", plot = PCAplot, width = 12, height = 8)