Appendix A6.1: Prepare gene expression data for TTCA

TTCA is a time series analysis method used to identify differentially expressed genes over time in response to various treatments or conditions.

```
###################################################
## Project: Time Series - Gene Expression - E-MEXP-3850
## Script purpose: Prepare gene expression data
###################################################

#This R code describes a process of downloading, processing
#and annotating gene expression data from the ArrayExpress repository.

# installations ---------------------------------------------------------
# ArrayExpress: provides access to ArrayExpress and EBI Gene Expression Atlas data
## to install ArrayExpress
# BiocManager::install("ArrayExpress")

## to install the annotation R packages
# Affycoretools tool for analyzing Affymetrix GeneChip data
# pd.hugene.1.0.st.v1: Platform design info for the Affymetrix Human Gene 1.0 ST array
# BiocManager::install(c("affycoretools", "pd.hugene.1.0.st.v1"))

# load libraries --------------------------------------------------------
# Load the installed packages into the R session
library(ArrayExpress)
library(affycoretools)
library(pd.hugene.1.0.st.v1)

# Process expression data ----------------------------------------------

### Building an R object from the processed data (only processed available)
# A directory data/raw_data is created to store downloaded files
# The dataset with the identifier E-MEXP-3850 is downloaded from ArrayExpress
# to the created directory using the getAE function.
# download files
dir.create("data/raw_data", showWarnings = FALSE)
mexp3850 <- getAE("E-MEXP-3850", path = "data/raw_data")



# To fix an error, add a new column called "Array Data File", copying
# An error related to the missing "Array Data File" column is addressed by copying values
# from the "Sample.Name" column.
# This is a common issue where certain functions might expect specific columns to be present.
# The fixed data is written back to the same file.
# the "Sample.Name" column in the file "E-MEXP-3850.sdrf.txt"
sdrf_df <- read.delim("data/raw_data/E-MEXP-3850.sdrf.txt")
sdrf_df$`Array Data File` <- sdrf_df$Sample.Name
write.table(sdrf_df, "data/raw_data/E-MEXP-3850.sdrf.txt", sep = "\t", row.names = FALSE)

### identify column to extract
# The getcolproc function identifies which columns from the dataset are required for processing
# The result is stored in the cn variable
# The second column (assumed to be the "log2 ratio") is then used in the subsequent steps
```

```
    cn <- getcolproc(mexp3850)
    cn
    # extracting 2nd (log2 ratio)

    ### create object
    # The procset function is used to generate a processed set of the data, selecting the relevant
column
    proset <- procset(mexp3850, cn[2])

    ### annotate probes
    # The processed set (proset) is annotated with probe details such as PROBEID, ENTREZID,
SYMBOL, and GENENAME
    # This information is essential for downstream analysis where gene symbols and other identifiers
are required
    # The annotateEset function does this using the pd.hugene.1.0.st.v1 package for the specific
microarray platform
    proset <- annotateEset(proset, pd.hugene.1.0.st.v1, columns = c("PROBEID", "ENTREZID",
"SYMBOL", "GENENAME"),      multivals = "first")

    ### check normalization
    # A boxplot is created to visualize the distribution of expression values across samples.
    # This is a common way to check if data is well-normalized.
    # If the boxes (representing interquartile ranges) are at roughly the same level for all samples
    # it indicates consistent normalization.
    boxplot(exprs(proset))

    ### save for later use
    # The final processed and annotated dataset is saved to a file
    # data/mexp3850_processed.RDS for future use.

    saveRDS(proset, "data/mexp3850_processed.RDS")
```

This R code is used to prepare gene expression data from the ArrayExpress repository for further analysis, specifically for time series gene expression analysis (TTCA). Here's a step-by-step explanation of what the code does:

1. **Install Required Packages (commented out):**
   - The code includes commands to install necessary R packages: "ArrayExpress" and other annotation-related packages such as "affycoretools" and "pd.hugene.1.0.st.v1." These lines are commented out, which means they won't be executed unless the comment symbols ('#') are removed. It's assumed that the packages are already installed.

2. **Load Libraries:**
   - The code loads the installed packages, including "ArrayExpress," "affycoretools," and "pd.hugene.1.0.st.v1," into the R session.

3. **Process Expression Data:**
   - Building an R object from the processed data:
     - A directory named "data/raw_data" is created to store downloaded files.
     - The dataset with the identifier "E-MEXP-3850" is downloaded from ArrayExpress and stored in the created directory using the `getAE` function.

   - Fix an error related to the "Array Data File" column:
     - An error related to the missing "Array Data File" column is addressed by copying values from the "Sample.Name" column in the file "E-MEXP-3850.sdrf.txt." This step is necessary to resolve issues where specific columns are expected by certain functions.

   - Identify the column to extract:
     - The `getcolproc` function identifies which columns from the dataset are required for processing. The result is stored in the `cn` variable, and the second column (assumed to be the "log2 ratio") is selected for further processing.

   - Create an object:
     - The `procset` function generates a processed set of the data, selecting the relevant column identified earlier. This processed set is stored in the variable `proset`.

   - Annotate probes:
     - The processed set `proset` is annotated with probe details such as PROBEID, ENTREZID, SYMBOL, and GENENAME. This information is essential for downstream analysis where gene symbols and other identifiers are required. The `annotateEset` function is used for this purpose, and it uses the "pd.hugene.1.0.st.v1" package specific to the microarray platform.

   - Check normalization:
     - A boxplot is created to visualize the distribution of expression values across samples. This is a common way to check if the data is well-normalized. If the boxes (representing interquartile ranges) are at roughly the same level for all samples, it indicates consistent normalization.

   - Save for later use:
     - The final processed and annotated dataset (`proset`) is saved to an RDS file named "data/mexp3850_processed.RDS" for future use in subsequent analyses.

Overall, this code performs data acquisition, preprocessing, annotation, and quality checking to prepare gene expression data for time series gene expression analysis using TTCA. It ensures that the data is in a suitable format for downstream analysis, including the identification of differentially expressed genes over time in response to various treatments or conditions.This entire process ensures that the gene expression data is downloaded, properly formatted, processed, annotated, checked for normalisation, and then saved for further analyses.

A6.2: Perform TTCA analysis

```
####################################################
## Project: Time Series - Gene Expression - E-MEXP-3850
## Script purpose: Perform TTCA analysis
####################################################

# This code is designed to perform a Time-series Two-Cluster Analysis (TTCA) using the TTCA library
# on a set of gene expression data.
# TTCA helps to identify genes that show different temporal expression patterns between two groups.

# installations -------------------------------------------------------
# The code has commands to install required packages, which are commented out
A specific version of TTCA package from a GitHub repository.
## install Biobase
# Biobase package from the Bioconductor.
# BiocManager::install("Biobase")

## install TTCA (fixed version from my GitHub repository)
# devtools::install_github("egeulgen/TTCA")


# load libraries -------------------------------------------------------
# TTCA: For performing the Time-series Two-Cluster Analysis.
# Biobase: Provides general functionality for Bioconductor.
library(TTCA)
library(Biobase)

# TTCA analysis --------------------------------------------------------

### a. Reading Data:
#The expression data is read from an RDS file (mexp3850_processed.RDS) into the variable proset.
###b. Determine Control Samples:
#The phenotypic data of the samples is extracted.
#Control samples are identified based on the time since admission to a pediatric intensive care unit being equal to 48 hours.
####c. Create Expression Data:
#Expression data is extracted for all samples.
#The expression data for control and case samples is separated.
#The control expression data is repeated five times, likely to match the number of case samples at different time points.
####d. Times Vectors:
#times_case: This extracts the time since admission to the pediatric intensive care unit for the case samples.
#times_control: This sets up a repeating pattern of times (0, 4, 8, 12, 24 hours) for control
####e. Annotation Data Frame:
#Annotation information for the probes is extracted from proset. This includes the probeset ID and the gene symbol.
####f. Run TTCA:
# The TTCA function is run using the extracted case and control expression data
# Their respective times, and the annotation data.
# The time interval being analyzed is specified by timeInt = c(8, 12)
# which represents the middle time period.
# This function is expected to take a long time to run as mentioned in the comment.
```

```
####g. Saving the Result:
#The result from the TTCA analysis is saved into an RDS file (TTCA_result.RDS) in the output
directory.

proset <- readRDS("data/mexp3850_processed.RDS")


### determine control samples - last samples (48h)
pheno_data <- pData(proset)
control_samples <- rownames(pheno_data)
[pheno_data$Factor.Value.time.since.admission.to.paediatric.intensive.care.unit. ==
48]

### create expression data
all_expr <- exprs(proset)
# for controls
control_expr <- all_expr[, control_samples]
# for cases
case_expr <- all_expr[, !colnames(all_expr) %in% control_samples]

# repeat control data for 0, 4, 8, 12, and 24 hours
control_expr <- cbind(control_expr,
                      control_expr,
                      control_expr,
                      control_expr,
                      control_expr)
colnames(control_expr) <- paste0("Control", 1:25)


### times vectors
# times of cases
times_case <- pheno_data$Characteristics.time.since.admission.to.paediatric.intensive.care.unit.
[match(colnames(case_expr) ,   rownames(pheno_data))]
# times of controls
times_control <- rep(c(0, 4, 8, 12, 24), each = 5)


### annotation data frame
feat_data <- fData(proset)
annotation <- data.frame(probeset_id = feat_data$PROBEID, gene_name = feat_data$SYMBOL)


### run TTCA - takes a long time!
TTCA_result <- TTCA(grp1 = case_expr, grp1.time = times_case,
                    grp2 = control_expr, grp2.time = times_control,
                    annot = annotation,
                    timeInt = c(8, 12)) # may change (this is the middle time period)

saveRDS(TTCA_result, "output/TTCA_result.RDS")
```

This R code is designed to perform a Time-series Two-Cluster Analysis (TTCA) on a set of gene expression data. TTCA is used to identify genes that exhibit different temporal expression patterns between two groups. Here's a step-by-step explanation of what the code does:

1. **Install Required Packages (commented out):**
   - The code includes commands to install two required R packages: "Biobase" from Bioconductor and a specific version of "TTCA" from a GitHub repository. These lines are commented out, which means they won't be executed unless the comment symbols ('#') are removed. It's assumed that the packages are already installed.

2. **Load Libraries:**
   - The code loads two libraries, "TTCA" and "Biobase," which are necessary for performing the TTCA analysis and providing general functionality for Bioconductor, respectively.

3. **TTCA Analysis Steps:**
   a. **Reading Data (Expression Data):**
      - Expression data is read from an RDS file named "mexp3850_processed.RDS" and stored in the variable `proset`.

   b. **Determine Control Samples:**
      - The phenotypic data of the samples is extracted from `proset`.
      - Control samples are identified based on the time since admission to a pediatric intensive care unit being equal to 48 hours.

   c. **Create Expression Data:**
      - Expression data is extracted for all samples.
      - The expression data for control and case samples is separated.
      - The control expression data is repeated five times, likely to match the number of case samples at different time points.

   d. **Times Vectors:**
      - `times_case`: This vector extracts the time since admission to the pediatric intensive care unit for the case samples.
      - `times_control`: This vector sets up a repeating pattern of times (0, 4, 8, 12, 24 hours) for control samples.

   e. **Annotation Data Frame:**
      - Annotation information for the probes is extracted from `proset`, including the probeset ID and the gene symbol.

   f. **Run TTCA:**
      - The `TTCA` function is run using the extracted case and control expression data, their respective times, and the annotation data.
      - The time interval being analyzed is specified by `timeInt = c(8, 12)`, which represents the middle time period.
      - A comment suggests that this function is expected to take a long time to run due to the complexity of the analysis.

   g. **Saving the Result:**
      - The result from the TTCA analysis is saved into an RDS file named "TTCA_result.RDS" in the "output" directory.

The code prepares the expression data, divides it into control and case groups, and then uses this data to perform a Time-series Two-Cluster Analysis to identify genes that have different temporal expression patterns between the two groups. Overall, this code is a comprehensive pipeline for

Appendix A6.3: Perform TTCA analysis Antigen Dose Studies

```r
####################################################
## Project: Time Series - Gene Expression - GSE33266
## Script purpose: Perform TTCA analyses
####################################################


# installations ---------------------------------------------------------
## install TTCA (fixed version from GitHub repository)
# devtools::install_github("egeulgen/TTCA")
# load libraries -------------------------------------------------------
library(TTCA)

# read data ------------------------------------------------------------
gset <- readRDS("data/GSE33266_data.RDS")
pheno_data <- pData(gset)
table(pheno_data$`treatment:ch1`)

# 10^2 -----------------------------------------------------------------
### determine case and control samples
case_samples <- pheno_data$geo_accession[pheno_data$`treatment:ch1` == "SARS CoV MA15 infected with 10^2 PFU"]
control_samples <- pheno_data$geo_accession[pheno_data$`treatment:ch1` == "Time-matched mock"]

## create expression data
all_expr <- exprs(gset)
all_expr <- all_expr[, c(case_samples, control_samples)]

# for cases
case_expr <- all_expr[, case_samples]

# for controls
control_expr <- all_expr[, control_samples]

### times vectors
# times of cases
times_case <- pheno_data$`time (post-infection):ch1`[match(case_samples, pheno_data$geo_accession)]
times_case <- as.numeric(sub(" day", "", times_case))

# times of controls
times_control <- pheno_data$`time (post-infection):ch1`[match(control_samples, pheno_data$geo_accession)]
times_control <- as.numeric(sub(" day", "", times_control))

### annotation data frame
feat_data <- fData(gset)
annotation <- data.frame(probeset_id = feat_data$ID, gene_name = feat_data$GENE_SYMBOL)

### run TTCA - takes a long time!
TTCA_result <- TTCA(grp1 = case_expr, grp1.time = times_case,
            grp2 = control_expr, grp2.time = times_control,
            annot = annotation)
```

```
    saveRDS(TTCA_result, "output/10_2_TTCA_result.RDS")

    # 10^3 -------------------------------------------------------------------
    ### determine case and control samples
    case_samples <- pheno_data$geo_accession[pheno_data$`treatment:ch1` == "SARS CoV MA15
infected with 10^3 PFU"]
    control_samples <- pheno_data$geo_accession[pheno_data$`treatment:ch1` == "Time-matched
mock"]

    ## create expression data
    all_expr <- exprs(gset)
    all_expr <- all_expr[, c(case_samples, control_samples)]

    # for cases
    case_expr <- all_expr[, case_samples]

    # for controls
    control_expr <- all_expr[, control_samples]

    ### times vectors
    # times of cases
    times_case <- pheno_data$`time (post-infection):ch1`[match(case_samples,
pheno_data$geo_accession)]
    times_case <- as.numeric(sub(" day", "", times_case))

    # times of controls
    times_control <- pheno_data$`time (post-infection):ch1`[match(control_samples,
pheno_data$geo_accession)]
    times_control <- as.numeric(sub(" day", "", times_control))

    ### annotation data frame
    feat_data <- fData(gset)
    annotation <- data.frame(probeset_id = feat_data$ID, gene_name = feat_data$GENE_SYMBOL)

    ### run TTCA - takes a long time!
    TTCA_result <- TTCA(grp1 = case_expr, grp1.time = times_case,
                grp2 = control_expr, grp2.time = times_control,
                annot = annotation)

    saveRDS(TTCA_result, "output/10_3_TTCA_result.RDS")

    # 10^4 -------------------------------------------------------------------
    ### determine case and control samples
    case_samples <- pheno_data$geo_accession[pheno_data$`treatment:ch1` == "SARS CoV MA15
infected with 10^4 PFU"]
    control_samples <- pheno_data$geo_accession[pheno_data$`treatment:ch1` == "Time-matched
mock"]

    ## create expression data
    all_expr <- exprs(gset)
    all_expr <- all_expr[, c(case_samples, control_samples)]

    # for cases
    case_expr <- all_expr[, case_samples]
    # for controls
    control_expr <- all_expr[, control_samples]

    ### times vectors
    # times of cases
    times_case <- pheno_data$`time (post-infection):ch1`[match(case_samples,
pheno_data$geo_accession)]
    times_case <- as.numeric(sub(" day", "", times_case))

    # times of controls
    times_control <- pheno_data$`time (post-infection):ch1`[match(control_samples,
pheno_data$geo_accession)]
    times_control <- as.numeric(sub(" day", "", times_control))

    ### annotation data frame
    feat_data <- fData(gset)
    annotation <- data.frame(probeset_id = feat_data$ID, gene_name = feat_data$GENE_SYMBOL)

    ### run TTCA - takes a long time!
    TTCA_result <- TTCA(grp1 = case_expr, grp1.time = times_case,
                        grp2 = control_expr, grp2.time = times_control,
                        annot = annotation)

    saveRDS(TTCA_result, "output/10_4_TTCA_result.RDS")

    # 10^5 ----------------------------------------------------------------------
    ### determine case and control samples
    case_samples <- pheno_data$geo_accession[pheno_data$`treatment:ch1` == "SARS CoV MA15
infected with 10^5 PFU"]
    control_samples <- pheno_data$geo_accession[pheno_data$`treatment:ch1` == "Time-matched
mock"]

    ## create expression data
    all_expr <- exprs(gset)
    all_expr <- all_expr[, c(case_samples, control_samples)]

    # for cases
    case_expr <- all_expr[, case_samples]

    # for controls
    control_expr <- all_expr[, control_samples]

    ### times vectors
    # times of cases
    times_case <- pheno_data$`time (post-infection):ch1`[match(case_samples,
pheno_data$geo_accession)]
    times_case <- as.numeric(sub(" day", "", times_case))

    # times of controls
    times_control <- pheno_data$`time (post-infection):ch1`[match(control_samples,
pheno_data$geo_accession)]
    times_control <- as.numeric(sub(" day", "", times_control))

    ### annotation data frame
    feat_data <- fData(gset)
    annotation <- data.frame(probeset_id = feat_data$ID, gene_name = feat_data$GENE_SYMBOL)
```

```
### run TTCA - takes a long time!
TTCA_result <- TTCA(grp1 = case_expr, grp1.time = times_case,
                    grp2 = control_expr, grp2.time = times_control,
                    annot = annotation)

saveRDS(TTCA_result, "output/10_5_TTCA_result.RDS")
```

CODE EXPLANATION

This R script is part of a bioinformatics analysis workflow for time series gene expression data from a dataset called "GSE33266." The script's main purpose is to perform TTCA (Time Series Differential Expression Analysis) on different conditions or treatments at different levels of viral infection (10^2, 10^3, 10^4, and 10^5 PFU) and compare them to a control group. Below is a breakdown of what each section of the code does:

1. Installation of TTCA Package (Commented Out):
   - The script starts with comments mentioning that you can install the TTCA R package from a specific GitHub repository using the `devtools::install_github` function. However, this part of the code is commented out, suggesting that TTCA might have been installed previously or from another source.

2. Loading Required Libraries:
   - The script loads the TTCA library using `library(TTCA)` to make its functions and methods available for analysis.

3. Reading Data:
   - The gene expression data from the GSE33266 dataset is read from an RDS file named "GSE33266_data.RDS" and stored in the variable `gset`.

4. Sample Selection (10^2, 10^3, 10^4, 10^5 PFU):
   - The code then repeats a similar set of operations for different levels of viral infection (10^2, 10^3, 10^4, and 10^5 PFU), where it identifies case and control samples based on the treatment conditions.
   - `case_samples` and `control_samples` are identified based on the treatment condition columns in the pheno_data dataframe.
   - Expression data for all samples (`all_expr`) is subsetted to include only the relevant case and control samples for the current level of infection.
   - Separate expression data matrices (`case_expr` and `control_expr`) are created for cases and controls.
   - Time vectors (`times_case` and `times_control`) are created by extracting and processing time information from the pheno_data dataframe.

5. Annotation Data:
   - The script extracts annotation data from `gset` to create an annotation data frame (`annotation`) containing probeset IDs and gene names.

6. Running TTCA:
   - The TTCA function is then called with the case and control expression data, time vectors, and annotation data for the current level of viral infection.
   - The results of TTCA are stored in `TTCA_result`.

7. Saving Results:
   - Finally, the results of TTCA for each level of viral infection are saved as RDS files with specific names, such as "10_2_TTCA_result.RDS" for 10^2 PFU, "10_3_TTCA_result.RDS" for 10^3 PFU, and so on.

This script automates the TTCA analysis for different viral infection levels and saves the results in separate files, making it easier to compare and analyse the differential gene expression patterns under different conditions.