The repository contains the following files:

- **Gene expression analysis.pdf** —The resulting gene expression analysis report in PDF format.
- **GeneExpressionAnalysis.qmd** —Quarto document containing the necessary code to perform the analysis and its explanation. Used to generate reports in various formats (HTML, PDF, etc.).
- **GeneExpressionAnalysis.R** —R script containing all the code
- **books.bib** —BibTeX file containing a list of references.
- **GeneExpressionAnalysis.html** —HTML report of gene expression analysis (with code and its explanation). Interactive document that can be opened in a browser to view the results.

I recommend using files **Gene expression analysis.pdf**, and **GeneExpressionAnalysis.qmd**, as they contain all the necessary information.

Below is an explanation of the code execution (by code (labeled as chunks) corresponding to the GeneExpressionAnalysis.R file).

Chunk 1

The following code loads data from the archive, unpacks it, and prepares it for further analysis. Specify the path to the folder where the archive with data is stored. Next, unpack the archive using the `untar()` function and place it in the current directory. The full path to the `brca_tcga_pan_can_atlas_2018` folder where the archive was unzipped is created, where `getwd()` returns the current working directory. Thus, paths to the main files are created: RNA-seq file, patient data file, and copy number aberrations data. We read text files using the `read.delim()` function. Next, I deleted the first 4 lines from `data_patient`, as they contained a description of the columns.

Chunk 2

This code does several key things to prepare the metadata that will be used in the data analysis. It takes the column names from the `data_cna` file, starting with column 3 (the first two columns contain the gene names). `substr(..., 1, 12)` truncates the identifiers to the first 12 characters, because patient identifiers are usually unique only by the first 12 characters. Since the patient identifiers in the `data_patient` file are different, they were converted to the same format as the other two files. `gsub("-", "̈", ...)` replaces hyphens (-) with periods (.), because `data_cna` stores identifiers in dotted format. The `intersect()` function returns identifiers that are present in both data sets (`cna_ids` and `patient_ids`).

From the RNA-seq data, we remove the first 2 columns that contain gene names and convert the remaining data into a numeric matrix `assay`. The RNA-seq values are rounded to the nearest integer since they must be integers for `DESeq()`. The names of the matrix rows that correspond to the gene names are set. A metadata matrix of size [number of columns (number of patients) in the assay, 2] is created. Next, we find the row corresponding to the `ERBB2` gene in the CNA data. We loop through all columns of the assay expression matrix (for each patient), extract the patient ID, check if this ID is in `common_ids` (thereby

matching the IDs in the three files). Next, we check if this ID is in `common_ids` (i.e. in both datasets), find the index of this ID in the CNA data.

If the index is found, then:
`metadata[i, 1] <- pat_barcode`: set the patient ID to the first column of metadata;
`metadata[i, 2] <- ifelse(...)` Check HER2 (ERBB2) value in CNA:

- If > 0, status = 1 (Amplified).
- If ≤ 0, status = 0 (Not Amplified).

If ID not found in `common_ids`, HER2 status = NA. Converts the `metadata` matrix to data.frame and the `HER2_Status` column to a factor for convenience. `metadata <- metadata[complete.cases(metadata), ]` checks rows for NA and removes them.

Chunk 3

For differential gene expression analysis, we need the `BiocManager` and `DESeq2` packages. We check if the packages are installed, and if they are not installed, the code automatically downloads and installs them.

Chunk 4

All negative values in the assay matrix are replaced with 0. This is important because gene expression reads cannot be negative. Negative values may be the result of errors or incorrect data processing. Next, we remove columns (patient ids) that are not in the metadata but are in the `assay` matrix. `which(... %in% ids_to_remove)` defines the indexes of the columns corresponding to the missing ids. You can also check if the order of the column names in the assay and the patient ids in the `metadata$Patient_ID` match. It is also important to filter out genes with low expression. A minimum number of samples (groups) is set, and then the number of samples for each gene with expression ≥ 10 is counted. `filtered_assay[keep, ]` removes genes that did not pass the filter.

I found a bit challenging deciding what to do with NA (convert all to 0 or remove). Converting to 0 is convenient for preserving all data, including patient data and genes with low expression or insufficient information. Since the task also requires creating metadata using the CNA level of ERBB2+, I did not include in the matrix assay those patients who are not in the `fcna_ids`. If NA values mean no expression, it is logical to interpret them as zeros. But if NAs are due to missing data or low quality, interpreting them as 0 may be incorrect (too many genes with "zero" expression can increase the noise level). Removing NAs excludes genes with insufficient information, which increases confidence in the statistics. `flog2FoldChange`, `p-value`, and `fpadj` are calculated correctly only for genes with full information, which makes the results more reliable. And given that the number of NAs is small, and the goal is to identify more expressed genes, I removed them to perform with minimal noise.

Chunk 5

`DESeqDataSetFromMatrix()` creates a DESeqDataSet object that is used for analysis with `DESeq2()`. It is passed a matrix of gene reads (`filtered_assay`) (where the rows are genes and the columns are patient IDs), along with metadata and an experimental design formula (dividing samples into HER2-amplified (1) versus HER2-not-amplified (0) groups, between

which DESeq2 will compare genes). Subsequently, DESeq() performs the following: read normalisation for each sample taking into account the total number of reads, calculation of logarithmic expression changes (log2FoldChange), statistical analysis (Wald test to assess the significance of expression changes, generation of p-value and adjusted p-values (padj)).

Chunk 6

resultsNames() allows you to see which groups are being compared and what coefficients can be extracted. In this case, the analysis is performed for the HER2_Status variable, where two groups are compared: 1 and 0. results() retrieves the results of differential gene expression analysis. The function order(res$log2FoldChange) returns the indices of rows in res, sorted by log2FoldChange in ascending order. Using decreasing = TRUE sorts the rows in descending order.

Chunk 7

The code checks for and installs missing packages that are used to visualize differential gene expression results and create graphs.

Chunk 8

The code creates a volcano plot that visualizes the results of differential gene expression analysis between two groups: HER2 Amplified and Not Amplified. The plot shows the log2FoldChange expression change on the X-axis and the statistical significance (adjusted p-value) on the Y-axis. pdf()/dev.off() opens/closes the PDF plotter to save it in vector format. The main plotting function is EnhancedVolcano(), where res is the results of differential expression analysis (an object returned by DESeq2). Significance cutoffs: pCutoff = 0.05: p-value (genes with padj < 0.05 are considered significant), FCcutoff = 1.5: expression change (log2FoldChange > 1.5 or < -1.5). This is equivalent to a fold change ratio of $2^{1.5} \approx 2.8$.

Chunk 9

vst() is a function to perform a variational stabilizing transformation, i.e. transforms the expression data to reduce the influence of genes with high variance. PCA is sensitive to data with high variance. If stabilization is not performed, genes with high variability may dominate, making interpretation difficult. The result is a vsd object that contains the normalized expression data. plotPCA() is a function from the DESeq2 package that plots the Principal Component Analysis (PCA) plot. Since I previously converted the HER2_Status data to a factor, the points will be coloured by category, not by gradient (as in the numeric one). Since in the task we need to analyse between the Amplified and Not Amplified categories, even if the type were numeric, the result would still only be coloured in two colours.

Chunk 10

The code checks if the required packages are installed: fgsea (a gene set enrichment analysis (GSEA) that helps determine whether certain genes are associated with biological processes, pathways, or phenotypes), clusterProfiler (enrichment analyses of KEGG (Kyoto Encyclopedia of Genes and Genomes) pathways, GO (Gene Ontology) terms), org.Hs.eg.db

(a human annotation database containing information about genes), and `enrichplot` (a visualisation of enrichment analysis results), and if not, installs them.

Chunk 11

The code filters differentially expressed genes, separates them into over- and underexpressed genes, performs Gene Ontology (GO) Enrichment Analysis using the "Biological Process" ontology for over- and underexpressed genes, and plots `dotplot()` scatter plots to show the top 10 enriched biological processes for each gene group.

Chunk 12

Required packages in R: `pathview` (visualization of metabolic and signaling pathways using KEGG data), `ReactomePA` (pathway analysis based on the Reactome database).

Chunk 13

The `bitr()` function converts gene symbols (SYMBOL) into ENTREZID unique numeric identifiers for later use. The `enrichKEGG()` function performs KEGG (Kyoto Encyclopedia of Genes and Genomes) enrichment on the supplied genes. Some results and a visualization of the top 10 enriched KEGG pathways for the two groups are also output.

Chunk 14

The `enrichPathway()` function performs Reactome Pathway Enrichment Analysis for up- and downregulated genes based on gene identifiers (ENTREZ IDs). For each group, it saves a PDF file visualizing the 10 most enriched pathways.

Chunk 15

The code calculates the similarity of GO terms and enriched KEGG/Reactome pathways for overexpressed and underexpressed genes using the `pairwise_termsim()` function. It plots treeplots (`treeplot()`) to visualize the hierarchical structure.

Chunk 16

We check for the presence of package `pheatmap`, and install it if it is not there.

Chunk 17

The code uses the `assay()` function to extract a matrix of normalized expression data from the `vsd` object, selects the top 20 genes with the lowest `padj`, builds a heatmap of their expression using the `pheatmap()` function, adds row (genes) and column (samples) clustering, applies annotations for HER2 status, and sets colors for visualization. The resulting heatmap is saved as a PDF.

Chunk 19

We select DE genes that are statistically significant based on the adjusted p-value (`padj < 0.05`), indicating significant expression changes between the compared groups. Only these genes are extracted from the expression matrix (`vsd`) and then transposed for ease of analysis (rows now represent patients, columns - genes). Next, we prepare the data for survival analysis: columns with patient IDs, survival status (DECEASED/ALIVE), and survival time (in months) are extracted from the clinical data table. We convert patient IDs to a unified format (using `gsub()` and `substr()`) so that they can be matched with IDs in the expression matrix. Survival status is converted to binary format (1 for deceased, 0 for alive), and patients with invalid time data are removed. A surv object is created that contains information about survival time and status for each patient. The `glmnet()` function constructs the Lasso-regularised Cox model. `cv.glmnet()` performs cross-validation to select the optimal parameter λ. `lambda.min` calculates the value of λ that minimises the error (deviance). For each patient, the predicted risk (`predict()`) of death is calculated based on the model. Patients are divided into two groups (high and low risk) by the median value of the predicted risk. Survival curves for the high- and low-risk groups are plotted, where `survfit()` calculates the Kaplan-Meier survival curves and `ggsurvplot()` creates a graph.

I chose median grouping as the grouping method. Grouping patients by HER2 status makes limited sense in this case, since the `DESeq2` model was set to `design =~HER2_Status`. That is, gene expression analysis was already performed taking into account the HER2 status of the patients, and the differentially expressed genes (DE genes) were selected based on the comparison of two groups: HER2 Amplified and HER2 Not Amplified. Therefore, grouping by HER2 status does not provide new information, since your data is already orientated to identify differences between the groups.

Chunk 20

The code is almost the same as the previous one, but with a change in risk groups. Additionally, rows from metadata that correspond to patients with incorrect survival time data are removed. A categorical variable `risk_groups` is created, in which patients are divided into two groups depending on their HER2 status: with amplification (`HER2_Status == 1`), the rest of the patients without (`HER2_Status == 0`).