Programming Approaches for Bioinformatics Exam

Maiolino Aurelio - 923271

27-06-2025

Index

Introduction

In this project I will provide a workflow for an analysis of multi-omic data using R in a reproducible environment. The primary focus is the processing and analysis of a dataset of peripheral blood mononuclear cells (PBMCs) from a healthy donor, combining gene expression and chromatin accessibility (ATAC-seq) data.

This project is divided into three key parts:

- **Part 1: The Dockerized Environment**
  Construction of a Docker container based on Ubuntu to ensure a standardized and reproducible computational environment. All software dependencies, including required R packages, are installed and configured within this container.
- **Part 2: Data processing and analysis**
  The PBMC dataset is processed stepwise:
  - Conversion of sparse matrices to dense formats
  - Separation of gene expression and ATAC-seq data
  - Summarization and annotation of both data types using GenomicRanges
  - Integration and normalization of expression and accessibility profiles
- **Part 3: Report**
  I build a Report using Rmarkdown to illustrate the steps i took in the analysis.

The repository for this exam can be found on my GitHub (https://github.com/Maiolino-Au/ProgrAppBioinfo_Exam)

Part 1: The Dockerized Environment

Docker is an open-source platform that enables developers to package applications and their dependencies into lightweight, portable containers. These containers can run consistently across different computing environments, making it easier to develop, test, and deploy software efficiently and reproducibly.

A Docker Image can be uploaded to Docker Hub so that everyone is able to download it. The Image for this project can be pulled by:

```
docker pull maiolinoaurelio/pab_exam2025
```

Building the Docker: the Dockerfile

A docker is build from a Dockerfile, a set of layered instruction that build the desired work environment.

Build a container from a base Ubuntu image, v20.04 was chosen due to its proven stability.

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND=noninteractive

LABEL maintainer="aurelio.maiolino@edu.unito.it"
```

The base Ubuntu image is just that, an Ubuntu installation with not a lot more inside it. It is therefore necessary to install various packages.

```
RUN apt-get update && apt-get install -y --no-install-recommends \
    software-properties-common \
    dirmngr \
    gpg \
    curl \
    build-essential \
    libcurl4-openssl-dev \
    libssl-dev \
    libxml2-dev \
    libfontconfig1-dev \
    libfreetype6-dev \
    libpng-dev \
    libtiff5-dev \
    libjpeg-dev \
    libharfbuzz-dev \
    libfribidi-dev \
    make \
    cmake \
    gfortran \
    libxt-dev \
    liblapack-dev \
    libblas-dev \
    sudo \
    wget \
    zlib1g-dev \
    libbz2-dev \
    liblzma-dev \
```

```
    libncurses5-dev \
    pandoc \
    git
```

After the installation of all these packages a cleaning step can be useful.

```
RUN rm -rf /var/lib/apt/lists/*
```

R is installed.

```
# Add the CRAN GPG key and repository for R
RUN curl -fsSL https://cloud.r-project.org/bin/linux/ubuntu/marutter_pubkey.asc | gpg
↪  --dearmor -o /usr/share/keyrings/cran.gpg \
    && echo "deb [signed-by=/usr/share/keyrings/cran.gpg]
↪  https://cloud.r-project.org/bin/linux/ubuntu $(lsb_release -cs)-cran40/" \
    | tee /etc/apt/sources.list.d/cran-r.list

# Update again and install R
RUN apt update && apt install -y --no-install-recommends r-base
```

Install Python and Jupyter Lab.

```
# Install JupyterLab
RUN apt update && apt install -y python3 python3-pip python3-venv

# create a virtual environment in which JupyterLab can be installed
RUN python3 -m venv /opt/venv

# Activate virtual environment and install JupyterLab
RUN /opt/venv/bin/pip install --upgrade pip && /opt/venv/bin/pip install jupyterlab

# Set the virtual environment as the default Python path
ENV PATH="/opt/venv/bin:$PATH"

# Make R visible to jupyter
RUN R -e "install.packages('IRkernel')" \
    R -e "IRkernel::installspec(user = FALSE)"
```

Install R packages.

```
# Install R packages
RUN R -e "install.packages(c('BiocManager', 'dplyr', 'ggplot2', 'data.table', 'future',
↪  'cowplot', 'remotes', 'R.utils', 'dplyr', 'rtracklayer', 'tinytex'))"
RUN R -e "BiocManager::install('tidyverse')"

# Install Seurat
RUN R -e "BiocManager::install('Seurat')"

# Install Signac
RUN R -e "remotes::install_github('stuart-lab/signac', ref = 'develop')"
```

I created a custom R package for this project, it needs to be download from GitHub.

```
Install custom R packages
RUN R -e "remotes::install_github('Maiolino-Au/PABMaiolinoPackage')"
```

Create the Scripts folder and download the scripts from GitHub (for visualization purposes a space has been added).

```
RUN mkdir -p /0_Scripts && \
  cd /0_Scripts && \
  curl -O
↪  https://raw.githubusercontent.com/Maiolino-Au/ProgrAppBioinfo_Exam/main/Scripts/
↪  Script_PAB_exam_Maiolino.R && \
  curl -O
↪  https://raw.githubusercontent.com/Maiolino-Au/ProgrAppBioinfo_Exam/main/Scripts/
↪  Maiolino_Au.Rmd
```

Set the default shell as `/bin/bash`.

```
ENV SHELL=/bin/bash
```

Start JupyterLab mapped to port 8888 without requiring a token.

```
CMD jupyter lab --ip=0.0.0.0 --port=8888 --no-browser --allow-root
↪  --ServerApp.allow_origin='*' --ServerApp.token='' #last one disables the token
```

Runinng the Docker container

Command to run the container

for windows

```
@echo off
REM Ottieni il percorso corrente
set "CURRENT_DIR=%cd%"

REM Avvia Docker e monta la cartella come /sharedFolder
docker run -it --rm -p 8888:8888 -v "%CURRENT_DIR%:/sharedFolder" maiolinoaurelio/pab_exam2025
```

for linux

```
docker run -it -rm -p 8888:8888 -v "$(pwd)":/sharedFolder maiolinoaurelio/pab_exam2025
```

Part 2: Data processing and analysis

Step 0: Setup

Load the custom R Packages

```
library(PABMaiolinoPackage)
```

Data Path

It will be assumed that the path to the date is the following:

```
/sharedFolder/
|-- Data/
    |-- matrix/
    |   |-- matrix.mtx.gz
    |   |-- features.tsv.gz
    |   |-- barcodes.tsv.gz
    |-- Homo_sapiens.GRCh38.114.gtf.gz
```

The scripts will be in `/0_Scripts` as shown in the dockerfile.

Step 1: Downlad the data, load the sparse natrix and convert it to a full matrix

Load the data

The three files composing the sparse matrix are loaded and assembled in a full matrix.

This one is then transformed in a `data.frame` in which the first column indicate the `id` of the feature (either a gene or an ATAC-seq peak)

```
s1.make.dt <- function(
    data_path = "/sharedFolder/Data/matrix/"
) {
    # Load the sparse matrix
    matrix <- readMM(file = paste0(data_path, "matrix.mtx.gz"))
    features <- read_tsv(file = paste0(data_path, "features.tsv.gz"),
                         col_names = c("id", "name", "type", "chr", "start", "end"),
                         show_col_types = F)
    barcodes <- read_tsv(file = paste0(data_path, "barcodes.tsv.gz"),
                         col_names = F,
                         show_col_types = F)

    # Assign names to colums and rows and transform the object "matrix" in a matrix
    colnames(matrix) <- barcodes$X1
    rownames(matrix) <- features$id
    matrix <- as.matrix(matrix)

    # Genearate a data.table, a id columns containg the ids of the various genes/peaks is
    ↪  added
```

```
    dt <- as.data.table(matrix, row.names = rownames(matrix))
    dt[, id := rownames(matrix)]
    setcolorder(dt, c("id", setdiff(names(dt), "id"))) # id is put as the first column

    return(dt)
}
```

```
dt <- s1.make.dt(data_path = "/sharedFolder/Data/matrix/")
```

Step 2: Split gene expression and ATAC-seq

The datased we are using contains both expression data and ATAC-seq, we need to separate them. These two different type of data will have a different nomenclature in their id:

- Genes will have an Ensemble id, with begins with "ENSG".
- Peack will begin with the chromosome to which they have been mapped and their id will start with "chr".

With grepl we can generate a logic which will allow us to select only for one or the other.

```
s2.subset.dt <- function(
    string,
    data
) {
  # Use grepl to create a logic that indicates which element of the data.table to take
  sub <- data[grepl(string, data$id)]
  return(sub)
}
```

```
dt_genes <- s2.subset.dt(string = "ENSG", data = dt)
dt_atac <- s2.subset.dt(string = "chr", data = dt)
```

Step 3: Summarize the data

We then create a named vector containing the pseudobulk for the expression of each gene and the intensity of each ATAC peak by computing a column-wise sum.

It is called pseudobulk due to the fact that it is not originated by a biological mixing of the cells, but by a computational aggregation of the data.

```
s3.summarize.data <- function(
    data
) {
  summary <- data[, rowSums(.SD), .SDcols = -"id"]
  names(summary) <- data$id
  return(summary)
}
```

```
genes_summary <- s3.summarize.data(data = dt_genes)
atac_summary <- s3.summarize.data(data = dt_atac)
```

Step 4: Create GenomicRanges objects

The data is then converted in a GenomicRanges object.

In the metadata there will be

- the summarized data
- the gene/peak id

```
s4.create.GenomicRanges <- function(
    type,
    path_f = "/sharedFolder/Data/matrix/features.tsv.gz"
) {
  # Load Features
  features <- as.data.table(
    read_tsv(
      file = path_f,
      col_names = c("id", "name", "type", "chr", "start", "end"),
      show_col_types = F
    )
  )

  # Subset features for genes or peaks
  if (type %in% c("Genes", "genes")) {
    if (type != "Genes") {type = "Genes"}

    features <- features[grepl("Gene", features$type)]
    features$chr[is.na(features$chr)] <- "unknown"

    summary = genes_summary
  } else if (type %in% c("Peaks", "peaks", "ATAC", "Atac", "atac")) {
    if (type != "Peaks") {type = "Peaks"}

    features <- features[grepl("Peaks", features$type) & grepl("chr", features$chr)]
    features$chr[is.na(features$chr)] <- "unknown"

    summary = atac_summary
  } else {
    print("Error, invalid value for type: must be 'Gene' or 'Peaks'")
    return ("")
  }

  # Create GRanges object
  gr <- GRanges(
    seqnames = features$chr,
    ranges = IRanges(
      start = features$start,
      end = features$end
    )
  )
```

```
  # Add metadata
  if (type == "Genes") {
    gr$gene_sum = summary
    gr$gene_id <- features$id
    gr$gene_name <- features$name
  } else if (type == "Peaks") {
    gr$peak_sum = summary
    gr$peak_id <- features$id
  }

  return(gr)
}
```

```
gr_genes <- s4.create.GenomicRanges("genes")
gr_atac <- s4.create.GenomicRanges("atac")
```

Step 5: Gene annotation for ATAC-seq peaks

Creation of a GRanges object for protein coding genes

From the annotation file `Homo_sapiens.GRCh38.114.gtf.gz` a GRanges object is created. This can be used to select wich ATAC peak and which expression data (this is RNA-seq data) correspond with protein coding genes.

```
s5.remap.atac <- function(
    atac_data = gr_atac
) {
  # Find the atac peaks that overlap with coding genes
  overlaps <- findOverlaps(atac_data, h38_coding)

  # Create a list of the genes for which there is an overlap
  gene_list <- split(h38_coding$gene_id[subjectHits(overlaps)], queryHits(overlaps))

  # Assign the genes to the corresponding peaks
  # In the case a peak is overlapping multiple genes they are added as a comma separated
  ↪  list
  atac_data$overlapping_genes <- NA
  atac_data$overlapping_genes[as.integer(names(gene_list))] <- sapply(gene_list, paste,
↪  collapse = ",")

  return(atac_data)
}
```

```
h38_coding <- s5.GR.protein.coding()
```

Remapping the ATAC Peaks

Whith the protein coding GR object we can identyfy which peaks are near to a prtein coding gene. These will be the most likely the have an impact on its expression.

```
s5.remap.atac <- function(
    atac_data = gr_atac
) {
  # Find the atac peaks that overlap with coding genes
  overlaps <- findOverlaps(atac_data, h38_coding)

  # Create a list of the genes for which there is an overlap
  gene_list <- split(h38_coding$gene_id[subjectHits(overlaps)], queryHits(overlaps))

  # Assign the genes to the corresponding peaks
  # In the case a peak is overlapping multiple genes they are added as a comma separated
  ↪  list
  atac_data$overlapping_genes <- NA
  atac_data$overlapping_genes[as.integer(names(gene_list))] <- sapply(gene_list, paste,
↪  collapse = ",")

  return(atac_data)
}
```

```
gr_atac <- s5.remap.atac()
```

Step 6: Finalize expression data

Not all the features in the expression data indicate a protein coding gene. We can subset the relative GRanges object to select only for the features that are identified as coding genes.

```
s6.finalize.exp.data <- function(
    exp_data = gr_genes
) {
  # Add a column to with gene symbols by matching the gene_id to the annotation from the
  ↪  GTF file
  exp_data$gene_symbol <- h38_coding$gene_name[match(exp_data$gene_id,
↪  h38_coding$gene_id)]

  # # Remove rows where gene_symbol is NA
  exp_data <- exp_data[!is.na(exp_data$gene_symbol)]

  return(exp_data)
}
```

```
gr_genes <- s6.finalize.exp.data()
```

Step 7: Data Normalization and Integration

Normalization

The counts are normalized using CPM and then added to the respective GRanges .

```
s7.normalize.cmp <- function(data) {
  log2((data / sum(data) * 1e6) + 1)
}
```

```
genes_cpm <- s7.normalize.cmp(data = genes_summary)
atac_cpm <- s7.normalize.cmp(data = atac_summary)

# The normalized counts are added to the respective GRanges
gr_genes$gene_cpm <- genes_cpm[match(gr_genes$gene_id, names(genes_cpm))]
gr_atac$peak_cpm  <- atac_cpm[match(gr_atac$peak_id, names(atac_cpm))]
```

Merge

The epigenomic and transcriptomic data are merged.

```
s7.merge <- function(
    exp_data = gr_genes,
    atac_data = gr_atac
) {
  merged <- merge(
    as.data.table(exp_data)[, .(gene_id, gene_symbol, chr = seqnames, gene_cpm)],
    as.data.table(atac_data)[, .(overlapping_genes, peak_cpm)],
    by.x = "gene_id", by.y = "overlapping_genes", all.x = TRUE
  )

  merged$chr <- factor(merged$chr, levels = paste0("chr", c(1:22, "X", "Y")))

  return(merged)
}
```

```
merged_data <- s7.merge()
```

Summary

Not every peak was associated with a protein coding gene, of the 81111 peaks present in the data only 52069 where in the vicinity of a protein coding gene.

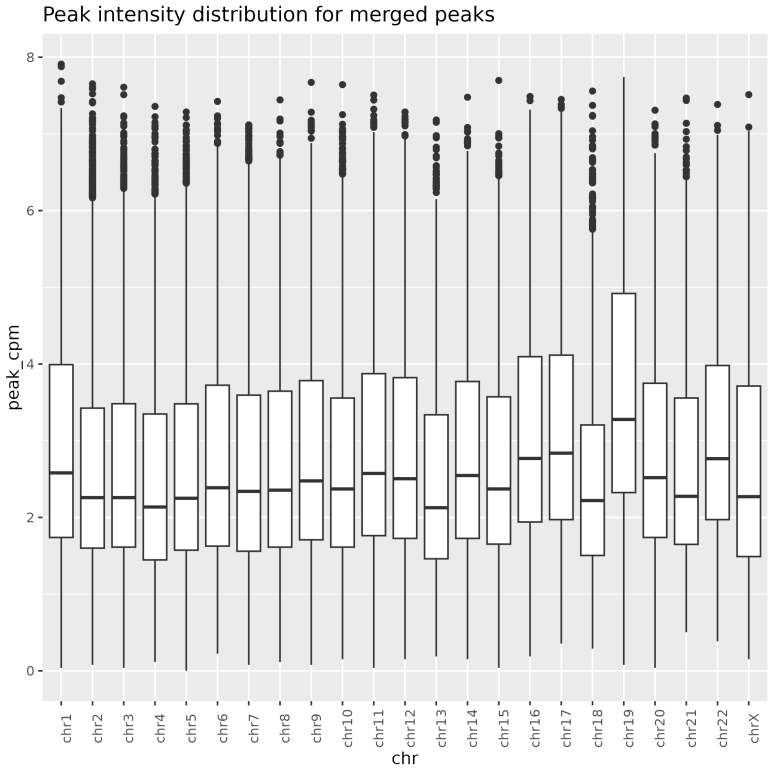
```
s7.summary.table.peaks <- function(
    atac_data = gr_atac
) {
  unmerged <- as.data.table(atac_data)[is.na(overlapping_genes)]
  unmerged<- unmerged[, .(peak_id, seqnames, start, end, peak_cpm)]
  colnames(unmerged)[colnames(unmerged) == "seqnames"] <- "chr"

  return(unmerged)
}
```

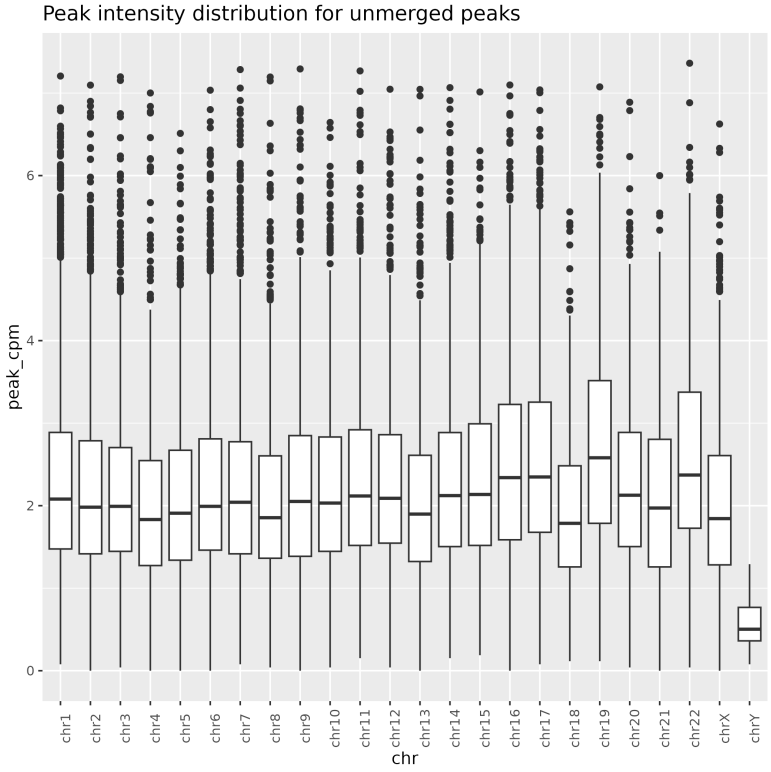
```
unmerged_peaks <- s7.summary.table.peaks()
```

Plot of peak intesnsity distribution for each chromosome

```
s7.summary.plot.peaks <- function(
    data_merged = merged_data,
    data_unmerged = unmerged_peaks
) {
  # Provide "a plot of peak intensity distribution chromosome by chromosome"
  # For peaks that could be merged or for the ones that could not?

  chr_levels <- paste0("chr", c(1:22, "X", "Y"))
  unmerged_peaks$chr <- factor(unmerged_peaks$chr, levels = chr_levels)

  merged <- ggplot(data_merged[!is.na(peak_cpm)], aes(x=chr, y=peak_cpm)) +
    geom_boxplot() + theme(axis.text.x = element_text(angle=90)) +
    labs(title = "Peak intensity distribution for merged peaks")

  unmerged <- ggplot(data_unmerged[!is.na(peak_cpm)], aes(x=chr, y=peak_cpm)) +
    geom_boxplot() + theme(axis.text.x = element_text(angle=90)) +
    labs(title = "Peak intensity distribution for unmerged peaks")

  #Save
  dir_results <- "/sharedFolder/Results"
  if (!dir.exists(dir_results)) {dir.create(dir_results)}

  ggsave("/sharedFolder/Results/s7_peaks_intesnisty_merged.png", plot = merged)
  ggsave("/sharedFolder/Results/s7_peaks_intesnisty_unmerged.png", plot = unmerged)

  return(list(merged, unmerged))
}
```

```
plot_s7_pea <- s7.summary.plot.peaks()
```


Peak intensity distribution for merged peaks
peak_cpm
0
2
4
6
8
chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrX
chr


Peak intensity distribution for unmerged peaks
peak_cpm
0
2
4
6
chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrX
chrY
chr

At the same time not every protein coding gene could be associated with an ATAC peak: 17.3% (6349)

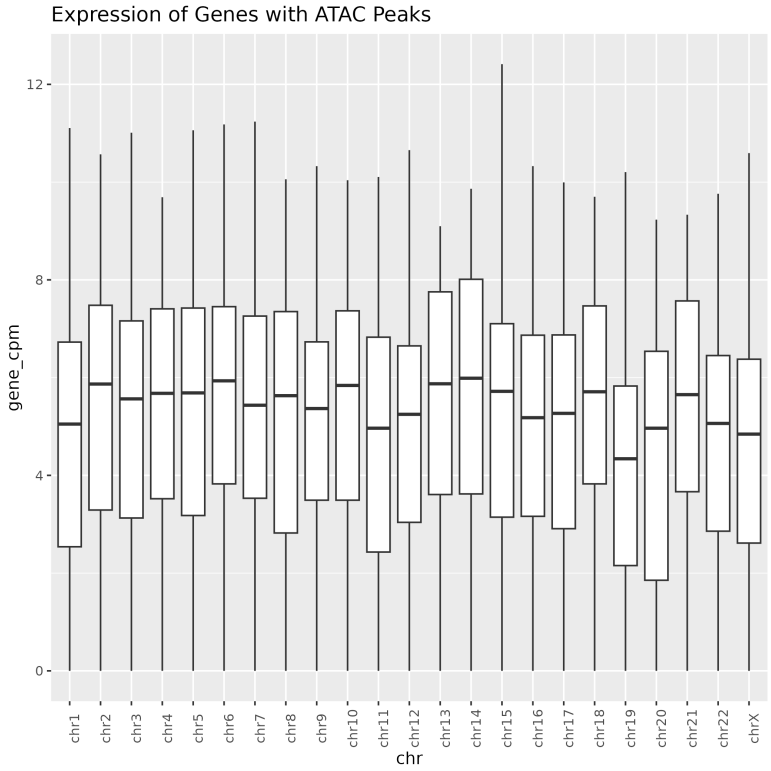
```
s7.summary.table.genes <- function(
    exp_data = gr_genes,
    merged = merged_data
) {
  genes_no_atac <- merged[is.na(peak_cpm) | peak_cpm == 0]
  genes_no_atac <- genes_no_atac[, .(gene_id, gene_symbol, gene_cpm)]

  return(genes_no_atac)
}
```

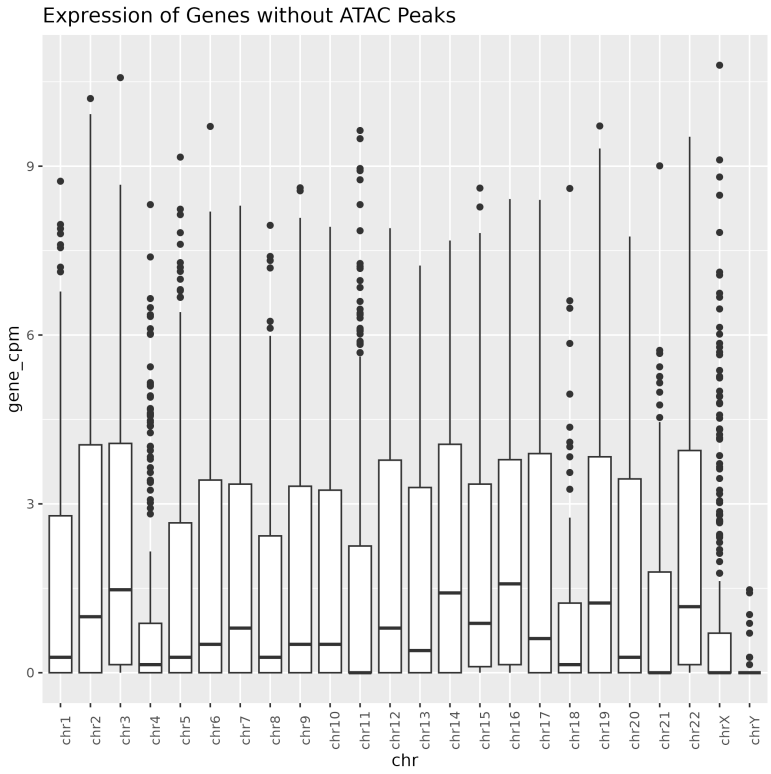
```
genes_no_atac <- s7.summary.table.genes()
```

Plot of the gene expression distribution in the chromosomes

```
s7.summary.plot.genes <- function(
    data = merged_data
) {
  plot <- list()

  # atac
  data_atac <- data[peak_cpm > 0 & grepl("chr", data$chr)]
  merged <- ggplot(data_atac, aes(x=chr, y=gene_cpm)) +
    geom_boxplot() + theme(axis.text.x = element_text(angle=90)) +
    labs(title = "Expression of Genes with ATAC Peaks")

  # no atac
  data_no_atac <- data[(is.na(peak_cpm)| peak_cpm == 0) & grepl("chr", data$chr)]
  unmerged <- ggplot(data_no_atac, aes(x=chr, y=gene_cpm)) +
    geom_boxplot() + theme(axis.text.x = element_text(angle=90)) +
    labs(title = "Expression of Genes without ATAC Peaks")

  #Save
  dir_results <- "/sharedFolder/Results"
  if (!dir.exists(dir_results)) {dir.create(dir_results)}

  ggsave("/sharedFolder/Results/s7_gene_expression_merged.png", plot = merged)
  ggsave("/sharedFolder/Results/s7_gene_expression_unmerged.png", plot = unmerged)

  return(list(merged, unmerged))
}
```

```
plot_s7_genes <- s7.summary.plot.genes()
```


Expression of Genes with ATAC Peaks
gene_cpm
0
4
8
12
chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrX
chr


Expression of Genes without ATAC Peaks
gene_cpm
0
3
6
9
chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrX
chrY
chr

Chromosome-wise Summary of Unmerged Peaks and Genes

Summarizes the count of unmerged ATAC-seq peaks and genes without ATAC signal for each chromosome.

```
s7.summary.chr.wise <- function(

) {
  peak_chr_summary <- unmerged_peaks[, .N, by = chr]
  colnames(peak_chr_summary) <- c("chr", "N_unmerged_peaks")

  gr_genes_dt <- as.data.table(gr_genes)
  genes_no_atac <- merge(genes_no_atac, gr_genes_dt[, .(gene_id, seqnames)], by =
↪  "gene_id", all.x = TRUE)
  gene_chr_summary <- genes_no_atac[, .N, by = seqnames]
  colnames(gene_chr_summary) <- c("chr", "N_unmerged_genes")

  chr_wise_summary <- merge(gene_chr_summary,peak_chr_summary, all = T)

  return(chr_wise_summary)
}
```

```
chr_wise_summary <- s7.summary.chr.wise()
```

chr	N_unmerged_genes	N_unmerged_peaks
chr1	607	3038
chr2	354	2444
chr3	303	1636
chr4	260	1092
chr5	319	1520
chr6	329	2007
chr7	280	1361
chr8	213	1283
chr9	243	1243
chr10	193	1381
chr11	538	1304
chr12	311	1469
chr13	81	740
chr14	202	1098
chr15	168	889
chr16	274	807
chr17	359	1404
chr18	67	533
chr19	443	989
chr20	159	903
chr21	94	369
chr22	113	639
chrX	382	879
chrY	43	14
unknown	13	NA
GL000194.1	1	NA

Step 8: Data Visualization

Scatter plots where generated to show the association between ATAC peaks and gene expression.

```
s8.scatter.plots <- function(chr_num) {
  plot <- ggplot(
    merged_data[!is.na(peak_cpm) & merged_data$chr == paste0("chr", chr_num)],
    aes(x=gene_cpm, y=peak_cpm)
  ) +
    geom_point(alpha=0.5) +
    labs(
      x = "Gene Expression CPM (log2)",
      y = "ATAC Peaks CPM (log2)",
      title = paste("Chromosome", chr_num)
    ) +
    xlim(0, 13) +
    ylim(0, 9) +
    theme_bw() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
  return(plot)
}


s8.save.plots <- function() {
  plot_list <- lapply(c(as.character(1:22), "X", "Y"), s8.scatter.plots)

  dir_results <- "/sharedFolder/Results"
  if (!dir.exists(dir_results)) {dir.create(dir_results)}

  for (i in 1:(length(plot_list)/12)) {
    n <- 12*(i-1)+1
    combined <- cowplot::plot_grid(plotlist = plot_list[(n):(n+11)], ncol = 3)
    ggsave(paste0(
      "/sharedFolder/Results/s8_plot_",
      i,
      ".png"
    ), plot = combined, width = 1920*2, height = 1080*3.6, units = "px")
  }
}
```

```
s8.save.plots()
```

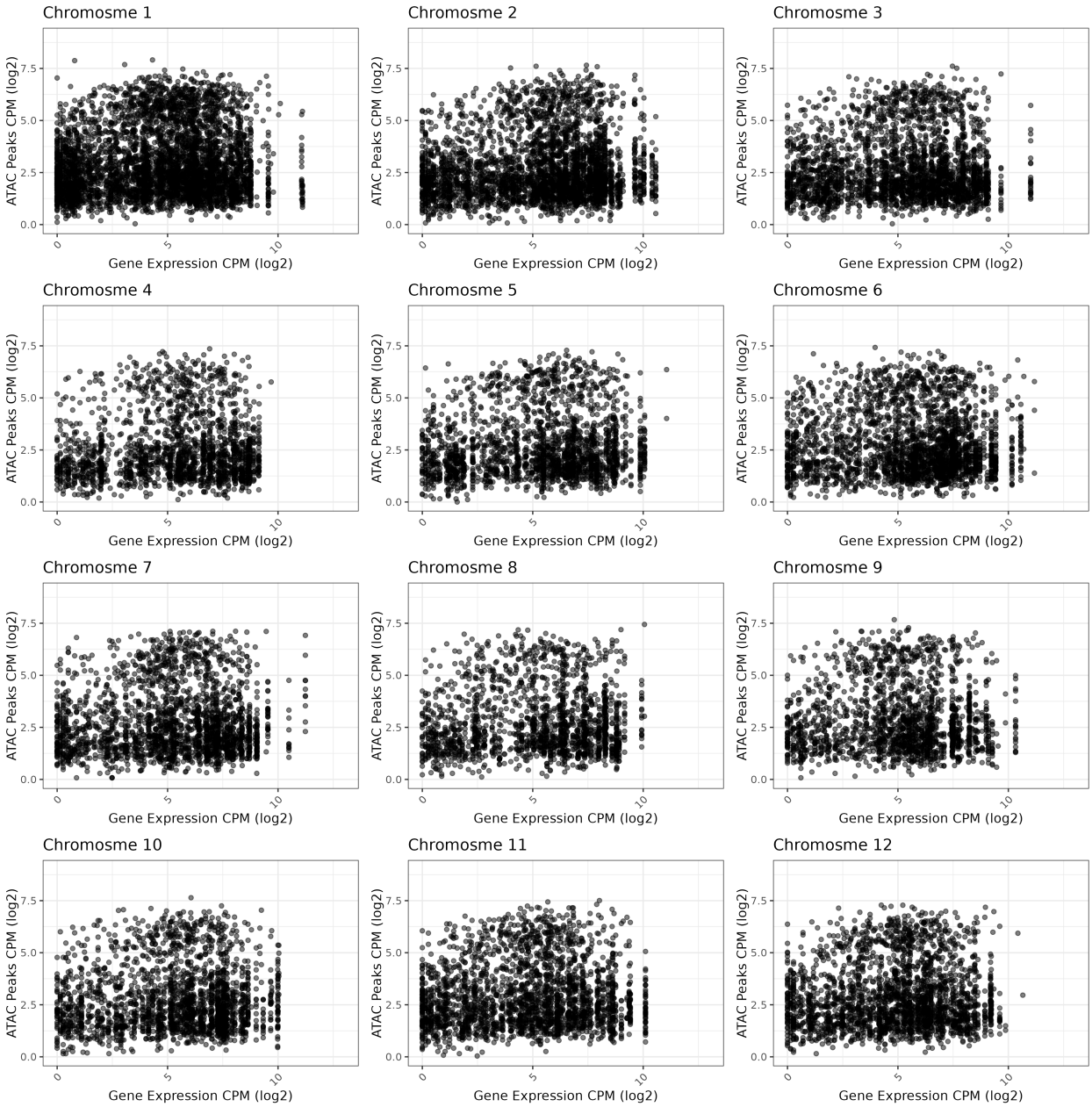


Figure 1: Scatter plot for Chromosomes 1 to 12

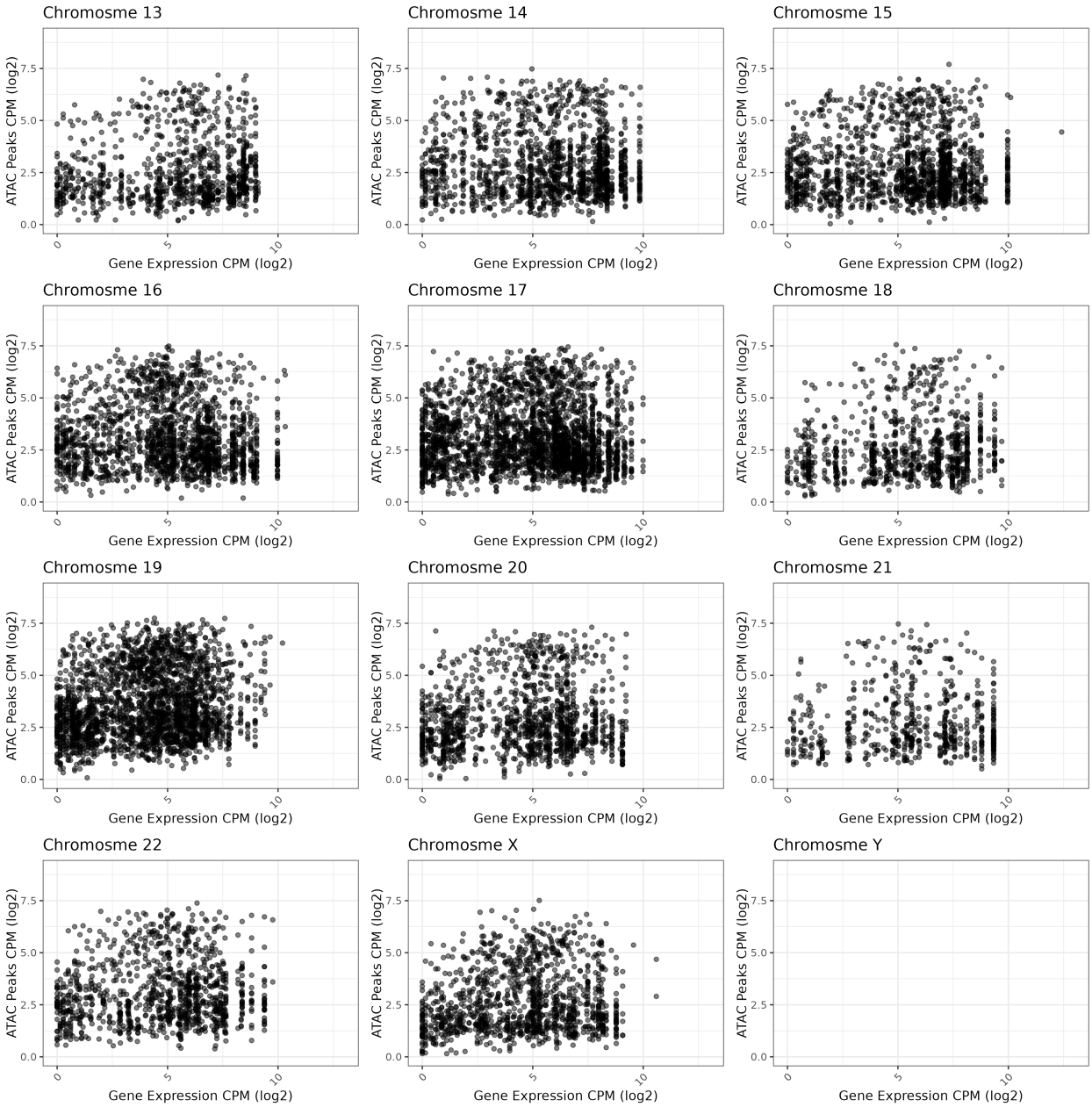


Figure 2: Scatter plot for Chromosomes 13 to 22, X, and Y

Part 3: Report

At last, this report was created using rmarkdown.

```
rmarkdown::render("Maiolino_Au.Rmd", output_dir = "/sharedFolder/Results")
```