In-Class Project: ChIP-Seq Peak Calling with MACS3

Payal Priyadarshini

2025-05-05

Introduction

Welcome! This project will guide you through the process of identifying potential protein binding sites or histone modifications in the genome using **simulated** ChIP-Seq data designed to have a strong signal. We will use a software called MACS3 (Model-based Analysis of ChIP-Seq version 3), a super useful tool for peak calling and several other types of analysis of ChIP-Seq data.

In this exercise, you will: 1. Generate simulated paired-end ChIP-Seq data with stronger enrichment signals. 2. Execute the **MACS3** peak caller on this simulated data from within an R script, attempting default model building. 3. Load the resulting `.narrowPeak` file into R using `rtracklayer` (if peaks were found). 4. Briefly inspect the `GRanges` object containing the peaks and visualize some basic properties (histograms). 5. Visualize the raw read coverage for treatment and control alongside the called peaks in a specific genomic region using `Gviz`.

Dataset: We will generate simulated BED files representing paired-end reads: - Treatment: Simulates reads from a ChIP experiment with strong enrichment at specific locations. - Control: Simulates background reads. - Format: Paired-end reads in BED format.

Paired-End Data: Our simulated data represents paired-end sequencing. MACS3 will attempt to infer the fragment size distribution from the data itself.

Prerequisite: You **must** have MACS3 installed and accessible from your command line *before* proceeding. Test with `macs3 --version` in your terminal.

1. Setup R Environment

Simply run the code chunk below to install to ensure the necessary libraries for the rest of this script :) * `dplyr`: Data manipulation. * `rtracklayer`: Import BED files and MACS3 peak results * `GenomicRanges`: Provides core infrastructure for genomic data * `Gviz`: Required for plotting genomic tracks (coverage, peaks)

```r
if (!requireNamespace("dplyr", quietly = TRUE)) install.packages("dplyr")
if (!requireNamespace("BiocManager", quietly = TRUE)) install.packages("BiocManager")
if (!requireNamespace("rtracklayer", quietly = TRUE)) BiocManager::install("rtracklayer")
# Gviz has many dependencies, installation might take a while
if (!requireNamespace("Gviz", quietly = TRUE)) BiocManager::install("Gviz")
# GenomicRanges is usually installed as a dependency, but good to be explicit
if (!requireNamespace("GenomicRanges", quietly = TRUE)) BiocManager::install("GenomicRanges")
```

```r
# Load the libraries for this session
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(rtracklayer)
```

```
## Loading required package: GenomicRanges

## Loading required package: stats4

## Loading required package: BiocGenerics

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:dplyr':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:dplyr':
##
##     first, rename

## The following object is masked from 'package:utils':
##
##     findMatches
## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname

## Loading required package: IRanges

##
## Attaching package: 'IRanges'

## The following objects are masked from 'package:dplyr':
##
##     collapse, desc, slice

## Loading required package: GenomeInfoDb

## Warning: package 'GenomeInfoDb' was built under R version 4.3.3
```

```r
library(GenomicRanges)
library(Gviz)
```

```
## Loading required package: grid
```

2. Simulate Paired-End ChIP-Seq Data

Normally, the callpeaks function of MACS3 would be used on .bam or .bed files. However, these files are quite large, so in order to save our computers from the torture of a gigantic download and prevent the need for dealing with the FU servers, we decided to simulate paired-end ChIP-Seq Data. If you are interested, there is information in the "Simulation_Algo.txt" file in this project's directory that explains the overall logic behind the simulation.

Run the code chunk below to generate the simulated ChIP-Seq data and answer the following questions:

1. If we were to use real ChIP-Seq data in this analysis, where could we normally find such data (e.g., which website/database)?
2. Which file formats are passable to the callpeaks function of MACS3?
3. What are the important data quality control measures one should take before attempting to call peaks?

```r
# Sim params
set.seed(456)
n_pairs_treatment <- 8000
n_pairs_control <- 1500
read_length <- 50
fragment_mean_size <- 200
fragment_sd_size <- 25
genome_size <- 1e6
chr_name <- "chrSim"

# Helper function to generate reads
generate_read_pair <- function(pair_id, chr, center, frag_len, read_len, strand_choice = c("+", "-")) {
  frag_start <- max(0, round(center - frag_len / 2))
  frag_end <- frag_start + frag_len
  if (sample(strand_choice, 1) == "+") {
    read1_start <- frag_start
    read1_end <- frag_start + read_len
    read2_start <- frag_end - read_len
    read2_end <- frag_end
    strand1 <- "+"
    strand2 <- "-"
  } else {
    read1_start <- frag_end - read_len
    read1_end <- frag_end
    read2_start <- frag_start
    read2_end <- frag_start + read_len
    strand1 <- "-"; strand2 <- "+"
  }
  read1_start <- max(0, read1_start)
  read1_end <- min(genome_size, read1_end)
  read2_start <- max(0, read2_start)
  read2_end <- min(genome_size, read2_end)
  if (read1_start >= read1_end) read1_end <- read1_start + 1
  if (read2_start >= read2_end) read2_end <- read2_start + 1
  read1 <- data.frame(chr = chr, start = read1_start, end = read1_end,
                      name = paste0("sim_read_", pair_id, "/1"), score = 0, strand = strand1)
  read2 <- data.frame(chr = chr, start = read2_start, end = read2_end,
                      name = paste0("sim_read_", pair_id, "/2"), score = 0, strand = strand2)
  return(rbind(read1, read2))
}

# Generate treatment reads
hotspot_centers <- round(genome_size * c(0.15, 0.35, 0.65, 0.85))
n_hotspots <- length(hotspot_centers)
n_per_hotspot <- round(n_pairs_treatment * 0.90 / n_hotspots)
n_background <- n_pairs_treatment - (n_per_hotspot * n_hotspots)
hotspot_sd <- fragment_mean_size * 0.5

centers_treatment <- c(
  unlist(lapply(hotspot_centers, function(center) {
    rnorm(n_per_hotspot, mean = center, sd = hotspot_sd)
  })),
  runif(n_background, min = 0, max = genome_size)
)
centers_treatment <- pmax(0, pmin(genome_size, round(centers_treatment)))

frag_lengths_treatment <- round(rnorm(n_pairs_treatment, mean = fragment_mean_size, sd = fragment_sd_si
frag_lengths_treatment[frag_lengths_treatment < read_length * 2] <- read_length * 2

cat("Generating treatment reads...\n")
```

```
## Generating treatment reads...
```

```r
treatment_reads_list <- lapply(1:n_pairs_treatment, function(i) {
  generate_read_pair(pair_id = i, chr = chr_name, center = centers_treatment[i],
                     frag_len = frag_lengths_treatment[i], read_len = read_length)
})
```

```r
treatment_bed <- bind_rows(treatment_reads_list) %>% arrange(chr, start)

# Generate control reads
cat("Generating control reads...\n")
```

```
## Generating control reads...
```

```r
centers_control <- round(runif(n_pairs_control, min = 0, max = genome_size))
centers_control <- pmax(0, pmin(genome_size, centers_control))

frag_lengths_control <- round(rnorm(n_pairs_control, mean = fragment_mean_size, sd = fragment_sd_size))
frag_lengths_control[frag_lengths_control < read_length * 2] <- read_length * 2

control_reads_list <- lapply(1:n_pairs_control, function(i) {
  generate_read_pair(pair_id = i + n_pairs_treatment, chr = chr_name, center = centers_control[i],
                     frag_len = frag_lengths_control[i], read_len = read_length)
})
control_bed <- bind_rows(control_reads_list) %>% arrange(chr, start)

# Write BED files
sim_input_dir <- "macs_simulated_input"
if (!dir.exists(sim_input_dir)) dir.create(sim_input_dir)
treatment_bed_file <- file.path(sim_input_dir, "treatment_strong.bed")
control_bed_file <- file.path(sim_input_dir, "control_strong.bed")

cat("Writing BED files...\n")
```

```
## Writing BED files...
```

```r
write.table(treatment_bed, file = treatment_bed_file, sep = "\t", row.names = FALSE, col.names = FALSE,
write.table(control_bed, file = control_bed_file, sep = "\t", row.names = FALSE, col.names = FALSE, quo

cat("Simulated BED files created:\n")
```

```
## Simulated BED files created:
```

```r
cat("- Treatment:", treatment_bed_file, "\n")
```

```
## - Treatment: macs_simulated_input/treatment_strong.bed
```

```r
cat("- Control:", control_bed_file, "\n")
```

```
## - Control: macs_simulated_input/control_strong.bed
```

```r
if (!file.exists(treatment_bed_file) || file.info(treatment_bed_file)$size == 0) stop("Failed to create
if (!file.exists(control_bed_file) || file.info(control_bed_file)$size == 0) stop("Failed to create or
cat("BED files written successfully.\n")
```

```
## BED files written successfully.
```

3. Run MACS3 on Simulated Data

Now we construct and execute the `macs3 callpeak` command using the simulated BED files. We will let MACS3 attempt to build its model. Edit the code below where indicated.

```
# Define output directory for MACS3 results
macs3_output_dir <- "macs3_results_simulated"
if (!dir.exists(macs3_output_dir)) dir.create(macs3_output_dir)

# Define the output name prefix
output_name <- "simulated_run"

# TASK 3.1
# ------------------------------------------------------------------------
# Construct the MACS3 command arguments for input into the system2() function
# for command line operations from R.
# Hints:
# 1. Appropriate genome size is "1e6".
# 2. Files containing simulated data are in .bed format.
# 3. The output directory for the MACS3 peak calling is specified in the variable macs3_output_dir.
# 4. The output name for the MACS3 peak calling output file is specified in the variable output_name.
# 5. A good default q-value is 0.05.
# 6. In our instance, we would like to bypass building a shifting model for
#    determining the shift of the reads towards the center.
   macs3_args <- c(
  "callpeak",
  "-t", treatment_bed_file,
  "-c", control_bed_file,
  "-f", "BED",
  "-g", "1e6",
  "-n", chr_name,
  "--outdir", "/Users/payalpriyadarshini/Documents/Data_Science_2nd_sem/macs3_output_dir",
  "-q", "0.05",
  "--nomodel"
)

# TASK 3.2
# ------------------------------------------------------------------------
# Execute MACS3:
# Edit the value of macs3_executable (below) to be the full file path to your MACS3 install.
# This can be found by calling "which macs3" in your Terminal.
macs3_executable <- "/opt/anaconda3/bin/macs3" # <-- EDIT THIS LINE with full path if needed
# ------------------------------------------------------------------------

cat("Running MACS3 on simulated data...\n")
```

```
## Running MACS3 on simulated data...
```

```
full_command_string <- paste(macs3_executable, paste(macs3_args, collapse = " "))

# TASK 3.3
# ------------------------------------------------------------------------
# Evoke a command line operation (using system2()) to call MACS3's callpeaks
# function.
# Hint: The macs3_executable variable is used as the first term in a MACS3 command line call
# Example: In "macs3 -f arg1 -n arg2 ..." macs3 (or its full path) is stored in macs3_executable above.

start_time <- Sys.time()
macs3_output <- system2(macs3_executable, args = macs3_args)      # YOUR CODE HERE
# ----------------------------------------------------------------------
end_time <- Sys.time()
duration <- end_time - start_time


cat("\n---------- MACS3 Output Log Start ----------\n")
```

```
##
## ---------- MACS3 Output Log Start ----------
```

```
#cat(paste(macs3_output, collapse = "\n"), "\n")
cat("---------- MACS3 Output Log End ----------\n\n")
```

```
## ---------- MACS3 Output Log End ----------
```

```
cat("MACS3 run duration:", duration, "seconds\n\n")
```

```
## MACS3 run duration: 0.5642381 seconds
```

```
# Check if output files were created
expected_peak_file <- file.path(macs3_output_dir, paste0(output_name, "_peaks.narrowPeak"))
expected_xls_file <- file.path(macs3_output_dir, paste0(output_name, "_peaks.xls"))
```

4. Load and Explore MACS3 Peaks

Assuming MACS3 completed successfully and generated a `.narrowPeak` file, we load it into R. Please also answer the following questions:

1.

```
# Path to the narrowPeak file generated by MACS3
narrowPeak_file <- file.path("/Users/payalpriyadarshini/Documents/Data_Science_2nd_sem/macs3_output_dir/
peaks <- NULL # Initialize peaks object

# TASK 4
# ----------------------------------------------------------------------
# Import the narrowPeak file (narrowPeak_file) generated by MACS3 using
# the rtracklayer library.

if (file.exists(narrowPeak_file)) {
  cat("Importing peaks from:", narrowPeak_file, "...\n")
  peaks <- import(narrowPeak_file) # YOUR CODE HERE
  if (!is.null(peaks)) {
      cat("\nSuccessfully imported", length(peaks), "peaks.\n")
```

```
    cat("\nPeaks object summary (first 6):\n")
    show(head(peaks))
  } else {
    cat("\nPeak import failed despite file existing.\n")
  }
} else {
  cat("\nCould not find the MACS3 narrowPeak file to import:", narrowPeak_file, "\n")
  cat("Check the MACS3 log and the warning messages from the previous chunk.\n")
}
```

```
## Importing peaks from: /Users/payalpriyadarshini/Documents/Data_Science_2nd_sem/macs3_output_dir/chrS
##
## Successfully imported 4 peaks.
##
## Peaks object summary (first 6):
## GRanges object with 4 ranges and 6 metadata columns:
##       seqnames        ranges strand |        name     score signalValue
##          <Rle>     <IRanges>  <Rle> | <character> <numeric>   <numeric>
##   [1]   chrSim 149663-150387      * | chrSim_peak_1      5135     149.658
##   [2]   chrSim 349623-350420      * | chrSim_peak_2      4762     130.676
##   [3]   chrSim 649638-650357      * | chrSim_peak_3      4537     117.911
##   [4]   chrSim 849609-850378      * | chrSim_peak_4      5098     148.523
##          pValue    qValue      peak
##       <numeric> <numeric> <integer>
##   [1]   519.510   513.511       348
##   [2]   479.846   476.218       380
##   [3]   457.159   453.798       363
##   [4]   514.312   509.868       402
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

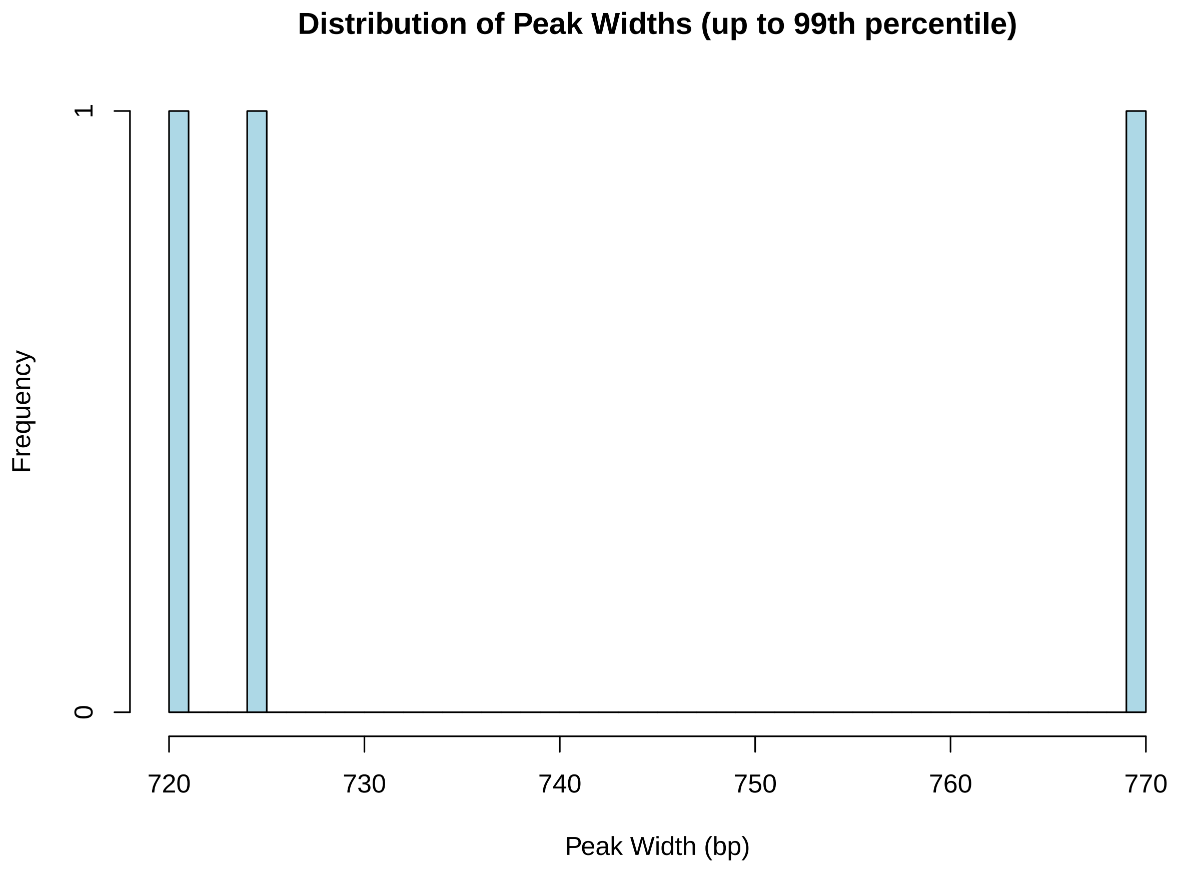
5. Basic Peak Visualization (Optional Histograms)

This code visualizes the distribution of peak widths and scores. Simply run the code below to get the output, no coding required for you :)

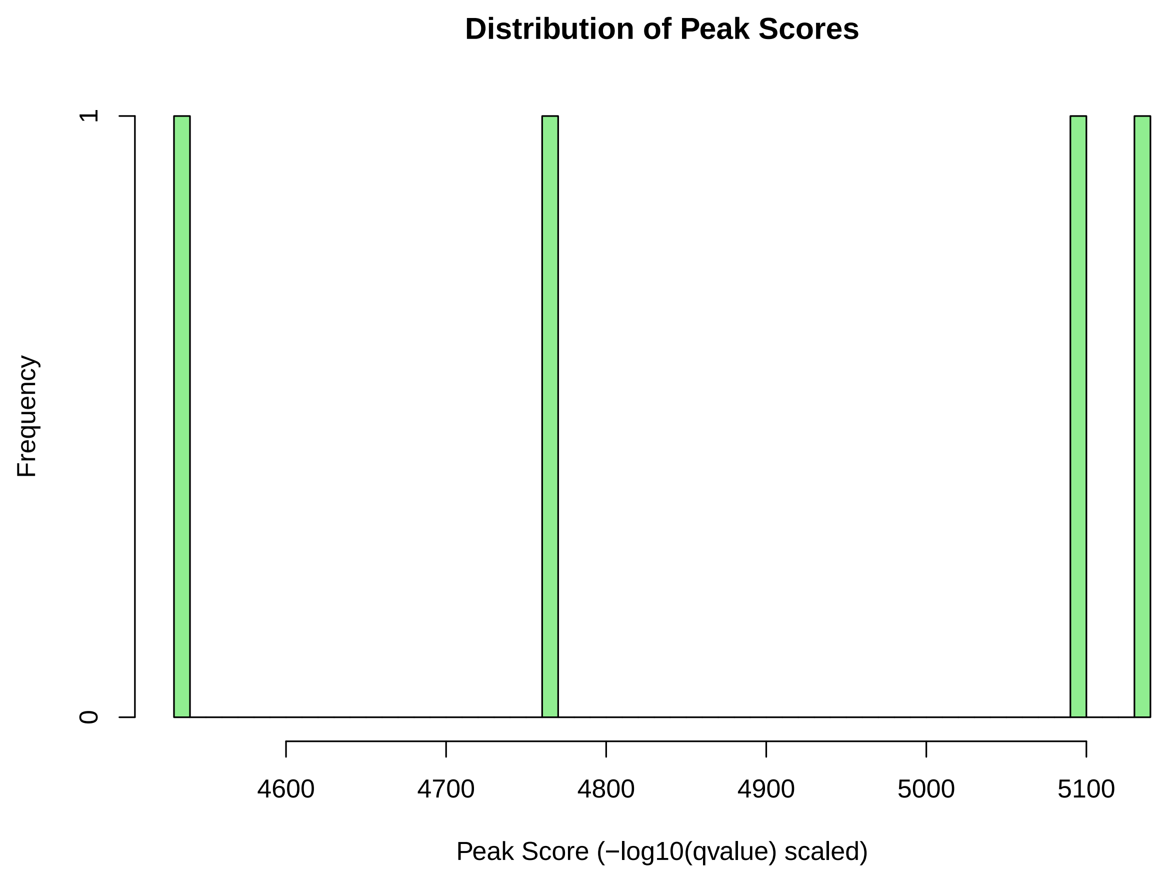
```
if (!is.null(peaks) && length(peaks) > 0) {
  peak_widths <- width(peaks)
  reasonable_widths <- peak_widths[peak_widths < quantile(peak_widths, 0.99, na.rm = TRUE)]
  hist(reasonable_widths, breaks = 50, main = "Distribution of Peak Widths (up to 99th percentile)", xla

  peak_scores <- mcols(peaks)$score
  hist(peak_scores, breaks = 50, main = "Distribution of Peak Scores", xlab = "Peak Score (-log10(qvalu

  cat("\nSummary statistics for loaded peaks:\n")
  cat("Widths:\n"); print(summary(peak_widths))
  cat("\nScores:\n"); print(summary(peak_scores))
} else {
  cat("\nNo peaks loaded, skipping histogram visualization.\n")
}
```



Distribution of Peak Widths (up to 99th percentile)
Frequency
0
1
720
730
740
750
760
770
Peak Width (bp)



```
##
## Summary statistics for loaded peaks:
## Widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   720.0   723.8   747.5   753.2   777.0   798.0
##
## Scores:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4537    4706    4930    4883    5107    5135
```

6. Visualize Read Coverage and Peaks

The code below generates a nice plot of where the peaks are located throughout the genome. Again, no coding needed as a reward for your hard work in the earlier tasks :) However, please answer the following questions:

1. Explain the difference in patterns between the control and treatment samples.
2. What would a control sample with significant peaks suggest about your prior ChIP-Seq experiment?

```
# Check if we have the necessary inputs for plotting
if (exists("treatment_bed_file") && file.exists(treatment_bed_file) &&
    exists("control_bed_file") && file.exists(control_bed_file) &&
```

```r
  exists("peaks") && !is.null(peaks) && length(peaks) > 0) {

cat("\nGenerating coverage plot...\n")

# Define region of interest
plot_center <- hotspot_centers[1]
plot_window <- 15000
plot_start <- max(0, plot_center - plot_window / 2)
plot_end <- min(genome_size, plot_center + plot_window / 2)
plot_chr <- chr_name

cat("Plotting region:", plot_chr, ":", plot_start, "-", plot_end, "\n")

# Import BED files as GRanges
cat("Importing BED files for coverage calculation...\n")
gr_treatment <- tryCatch(import(treatment_bed_file, format = "bed"), error=function(e) {message("Erro
gr_control <- tryCatch(import(control_bed_file, format = "bed"), error=function(e) {message("Error im

if(is.null(gr_treatment) || is.null(gr_control)) {
    cat("Could not import BED files, skipping coverage plot.\n")
} else {
    # Calculate coverage
    cat("Calculating coverage...\n")
    cov_treatment <- coverage(gr_treatment)[[plot_chr]]
    cov_control <- coverage(gr_control)[[plot_chr]]

    # Get data for ylim calculation
    # Ensure region indices are valid before subsetting coverage vectors
    max_coord_treat <- length(cov_treatment)
    max_coord_ctrl <- length(cov_control)
    valid_indices_treat <- seq(max(1, plot_start), min(max_coord_treat, plot_end))
    valid_indices_ctrl <- seq(max(1, plot_start), min(max_coord_ctrl, plot_end))

    # Handle cases where the plot region might be outside the actual data range
    if(length(valid_indices_treat) == 0 || length(valid_indices_ctrl) == 0) {
        cat("Plotting region is outside the range of the coverage data. Skipping plot.\n")
    } else {
        # Extract numeric coverage data for the plot region
        plot_data_treat <- as.numeric(cov_treatment[valid_indices_treat])
        plot_data_ctrl <- as.numeric(cov_control[valid_indices_ctrl])

        # Calculate max coverage for ylim *from the numeric vectors*
        max_cov <- max(max(plot_data_treat), max(plot_data_ctrl), 1) # Ensure ylim >= 1

        # --- Create Gviz Tracks ---
        gtrack <- GenomeAxisTrack()

        dtrack_treatment <- DataTrack(start = valid_indices_treat, width = 1,
                                      chromosome = plot_chr, genome = "simG",
                                      data = plot_data_treat, # Use pre-calculated vector
                                      type = "histogram", name = "Treatment Cov",
                                      background.title = "darkblue", col.histogram = "darkblue", fill
```

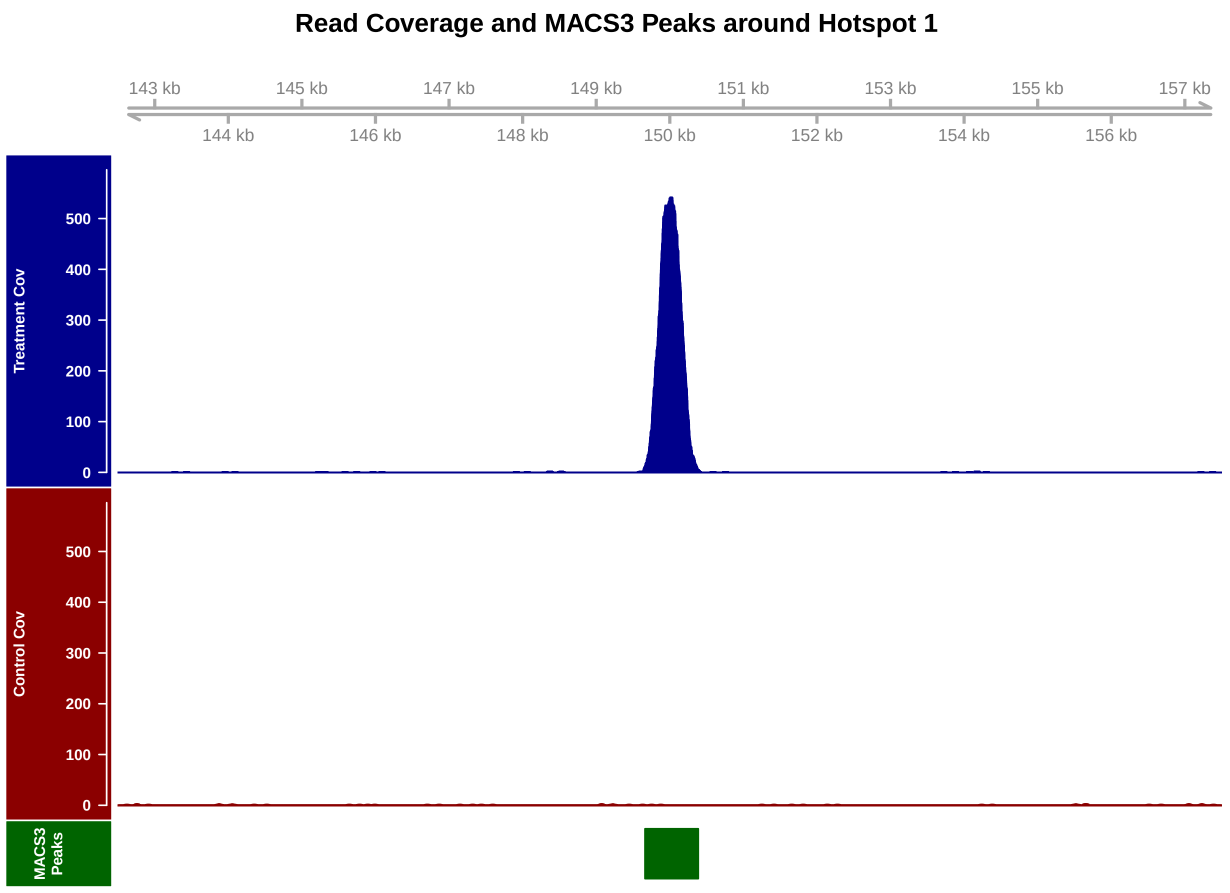
```
          dtrack_control <- DataTrack(start = valid_indices_ctrl, width = 1,
                                      chromosome = plot_chr, genome = "simG",
                                      data = plot_data_ctrl, # Use pre-calculated vector
                                      type = "histogram", name = "Control Cov",
                                      background.title = "darkred", col.histogram = "darkred", fill.his

          atrack_peaks <- AnnotationTrack(peaks, name = "MACS3 Peaks",
                                          chromosome = plot_chr,
                                          background.title = "darkgreen", fill = "darkgreen", col="dark
                                          shape="box")

          # --- Plot Tracks ---
          cat("Plotting tracks...\n")
          plotTracks(list(gtrack, dtrack_treatment, dtrack_control, atrack_peaks),
                     from = plot_start, to = plot_end, chromosome = plot_chr,
                     main = paste("Read Coverage and MACS3 Peaks around Hotspot 1"),
                     ylim = c(0, max_cov * 1.1), # Use pre-calculated max_cov
                     cex.main = 1
                     )
          cat("Coverage plot generated.\n")
      } # End check for valid indices
  } # End if BED files imported

} else {
  cat("\nSkipping coverage plot because peaks were not loaded or BED files are missing.\n")
  if (!exists("peaks")) cat(" -> Reason: 'peaks' object does not exist.\n")
  else if (is.null(peaks)) cat(" -> Reason: 'peaks' object is NULL.\n")
  else if (length(peaks) == 0) cat(" -> Reason: 'peaks' object is empty (length is 0).\n")
  if (!exists("treatment_bed_file") || !file.exists(treatment_bed_file)) cat(" -> Reason: Treatment BED
  if (!exists("control_bed_file") || !file.exists(control_bed_file)) cat(" -> Reason: Control BED file
}
```

```
##
## Generating coverage plot...
## Plotting region: chrSim : 142500 - 157500
## Importing BED files for coverage calculation...
## Calculating coverage...
## Plotting tracks...
```



```
## Coverage plot generated.
```

Conclusion

In this exercise, you have: 1. Generated simulated paired-end ChIP-Seq data with stronger enrichment signals. 2. Executed the **MACS3** peak caller on this simulated data from within an R script, attempting default model building. 3. Loaded the resulting `.narrowPeak` file into R using `rtracklayer` (if peaks were found). 4. Briefly inspected the `GRanges` object containing the peaks and visualized some basic properties (histograms). 5. Visualized the raw read coverage for treatment and control alongside the called peaks in a specific genomic region using `Gviz`. ““