# Peak Calling Lymphoblastoid Cell Line

Dr. Nishat Mohammad

2025-01-06

## Load Libraries

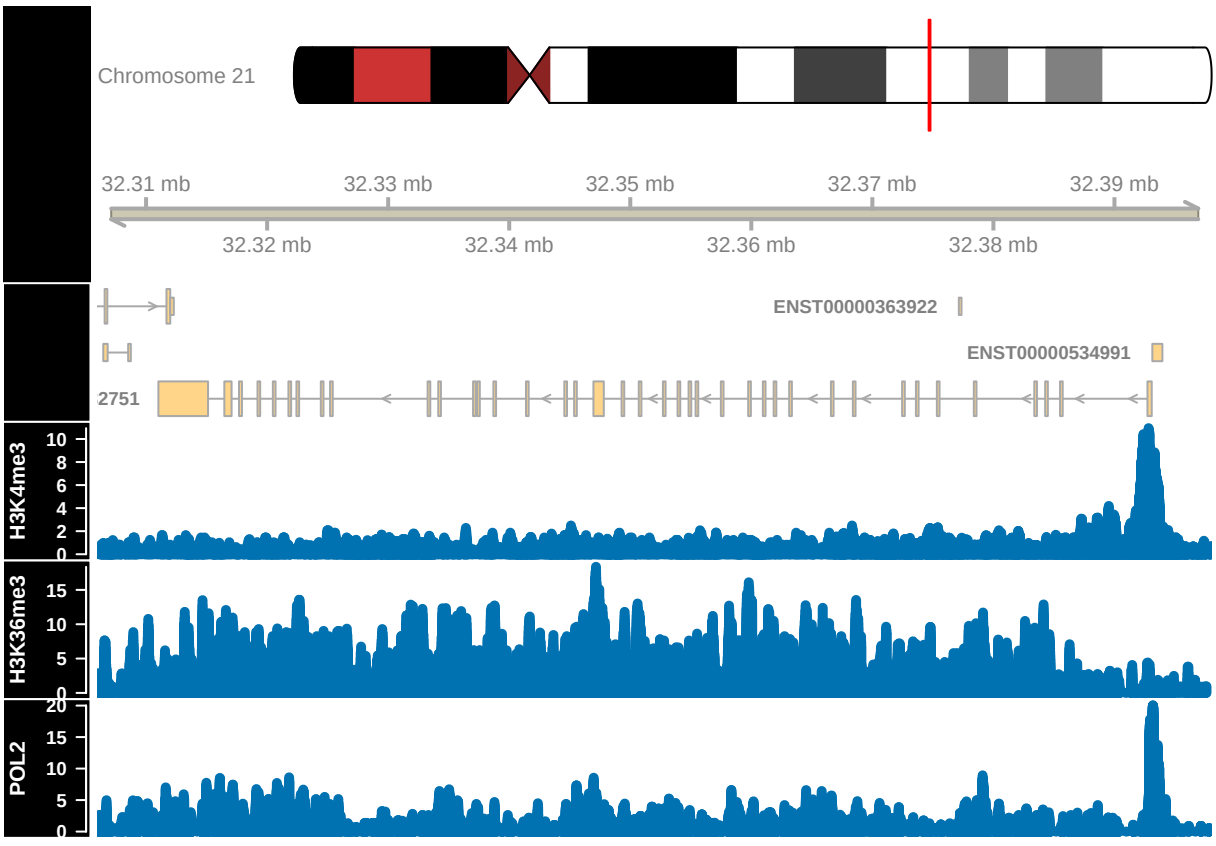
```
library(compGenomRData)
library(GenomeInfoDb)
library(GenomicRanges)
library(GenomicAlignments)

library(pheatmap)
library(rtracklayer)
library(Gviz)
library(ggplot2)

library(BSgenome.Hsapiens.UCSC.hg38)
library(AnnotationHub)
library(tidyr)
library(dplyr)

library(GenomicFeatures)
library(normr)
library(txdbmaker)
library(MotifDb)
library(TFBSTools)
```

## 1. Signal Profile Visualization

```
# set names for chip-seq bigWig files
chip_files = list(
  H3K4me3 = 'GM12878_hg38_H3K4me3.chr21.bw',
  H3K36me3 = 'GM12878_hg38_H3K36me3.chr21.bw',
  POL2 = 'GM12878_hg38_POLR2A.chr21.bw')

# Get data
data_path = system.file('extdata/chip-seq',package='compGenomRData')

# get full paths
chip_files = lapply(chip_files, function(x){
    file.path(data_path, x)})
```

```
# import ChIP bigWig files
chip_profiles = lapply(chip_files, rtracklayer::import.bw)

hub = AnnotationHub()
gtf = hub[['AH61126']]

# select chr21
seqlevels(gtf, pruning.mode='coarse') = '21'

# Get chromosome names
ensembl_seqlevels = seqlevels(gtf)

# Add chr prefix to chromosome names
ucsc_seqlevels = paste0('chr', ensembl_seqlevels)

# replace ensembl with ucsc chromosome names
seqlevels(gtf, pruning.mode='coarse') = ucsc_seqlevels


# convert  gtf annotation into a data base
txdb = makeTxDbFromGRanges(gtf)

# define gene track object
gene_track  = GeneRegionTrack(txdb, chr='chr21', genome='hg38')


# Get chromosome length information
hg_chrs = getChromInfoFromUCSC('hg38')
hg_chrs = subset(hg_chrs, (grepl('chr21$',chrom)))

# convert dataframe to  vector
seqlengths = with(hg_chrs, setNames(size, chrom))

# construct ideogram track
chr_track = IdeogramTrack(
    chromosome = 'chr21',
    genome     = 'hg38')

# constructs coordinate system
axis = GenomeAxisTrack(
    range = GRanges('chr21', IRanges(1, width=seqlengths)))

data_tracks = lapply(names(chip_profiles), function(exp_name){
    # select experiment using   exp_name
    DataTrack(
        range = chip_profiles[[exp_name]],
        name = exp_name,
        # type of track
        type = 'h',
        # line width parameter
        lwd = 5)})

# select   start coordinate for   URB1 gene
```

```r
start = min(start(subset(gtf, gene_name == 'URB1')))

# select   end coordinate for   URB1 gene
end   = max(end(subset(gtf, gene_name == 'URB1')))

# plot signal profiles around   URB1 gene
plotTracks(
    trackList = c(chr_track, axis, gene_track, data_tracks),
    # relative track sizes
    sizes = c(1,1,1,1,1,1),
    # background color
    background.title = "black",
    # controls visualization of gene sets
    collapseTranscripts = "longest",
    transcriptAnnotation = "symbol",
    # coordinates to visualize
    from = start - 5000,
    to = end + 5000)
```




This chunk generates a visualization of ChIP-Seq signal profiles around the **URB1** gene on chromosome 21.

- **ChIP-Seq Data Import**: The code sets file paths for ChIP-Seq bigWig files (H3K4me3, H3K36me3, POL2) and imports the data using `rtracklayer::import.bw`.
- **Chromosome and Gene Annotation**: Loads gene annotation data from UCSC Genome Browser, adjusts chromosome naming (from Ensembl to UCSC format), and creates a transcript database (`txdb`).

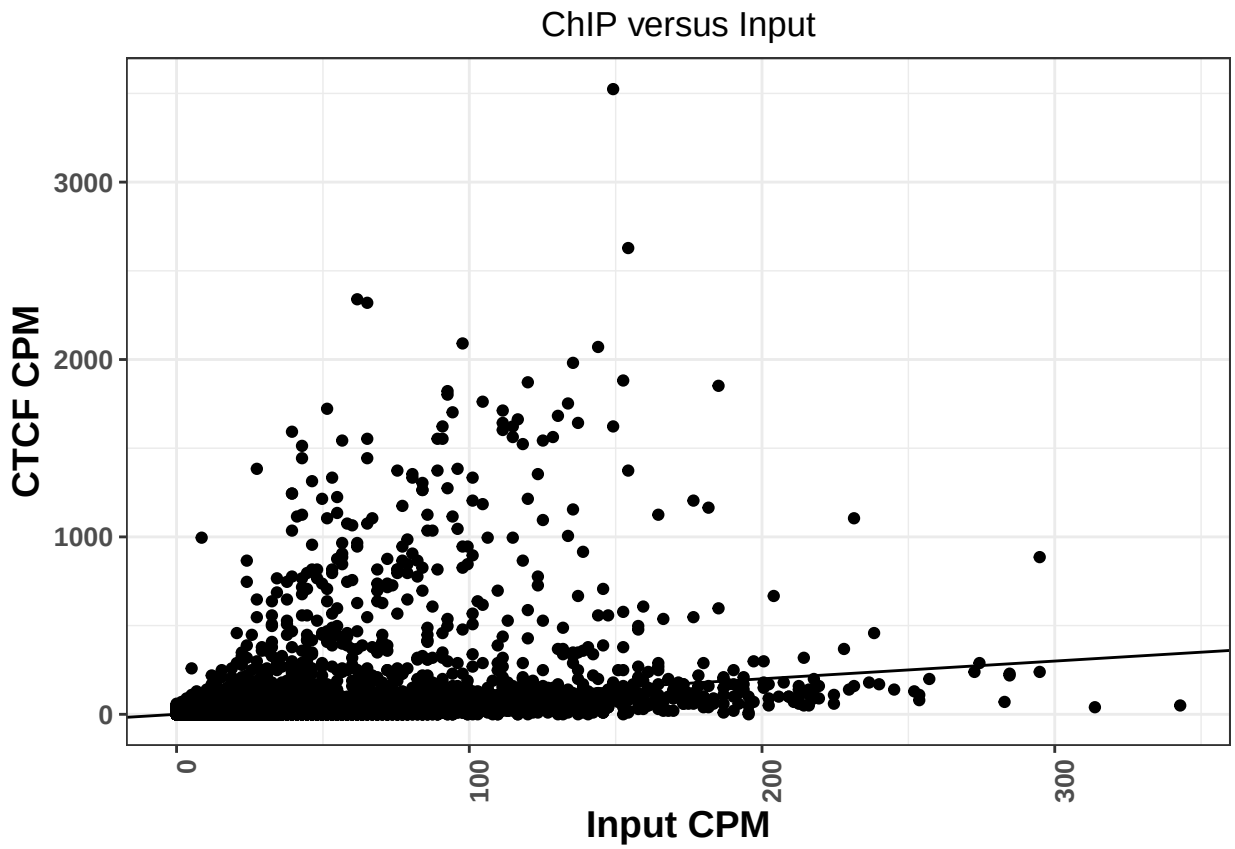
- **Track Creation**:
  - **Gene Region Track**: A `GeneRegionTrack` is created to visualize gene annotations.
  - **Ideogram and Axis Tracks**: `IdeogramTrack` and `GenomeAxisTrack` are constructed for chromosome visualization and coordinate scaling.
  - **Data Tracks**: `DataTrack` objects are created for each ChIP-Seq profile to display signal intensity across the genome.
- **Plotting**: The `plotTracks` function is used to plot the tracks, focusing on a 10kb region around the **URB1** gene, with customized visual settings (track size, background color).

This process allows for a detailed exploration of ChIP-Seq signals in the context of gene annotations for **URB1** on chromosome 21.

## 2.1 Sharp peaks

```
# full path to ChIP data file
chip_file = file.path(data_path, 'GM12878_hg38_CTCF_r1.chr21.bam')

# Get full path to Control data file
control_file = file.path(data_path, 'GM12878_hg38_Input_r5.chr21.bam')

# Get cpm for each experiment
# select chromosome
hg_chrs = getChromInfoFromUCSC('hg38')
hg_chrs = subset(hg_chrs, grepl('chr21$',chrom))

seqlengths = with(hg_chrs, setNames(size, chrom))

# define windows
tilling_window = unlist(tileGenome(seqlengths, tilewidth=1000))

# count reads
counts = summarizeOverlaps(
    features = tilling_window,
    reads    = c(chip_file, control_file))

# normalize read counts
counts = assays(counts)[[1]]
cpm = t(t(counts)*(1000000/colSums(counts)))

# convert matrix into a data frame
cpm = data.frame(cpm)
ggplot(data = cpm, aes(
  x = GM12878_hg38_Input_r5.chr21.bam,
  y = GM12878_hg38_CTCF_r1.chr21.bam)) +
  geom_point() +
  geom_abline(slope = 1) +
   theme_bw() +
```

```
 theme_bw() +
scale_fill_brewer(palette='Set2') +
 theme(axis.text   = element_text(size=10, face='bold'),
       axis.title  = element_text(size=14,face="bold"),
       plot.title  = element_text(hjust = 0.5),
       axis.text.x = element_text(angle = 90, hjust = 1)) +
xlab('Input CPM') +
ylab('CTCF CPM') +
ggtitle('ChIP versus Input')
```




This code performs ChIP-Seq data analysis by normalizing read counts and visualizing the relationship between ChIP (CTCF) and control (Input) samples.

- **ChIP and Control Data**: The paths to ChIP-Seq (CTCF) and control (Input) BAM files are specified.
- **Chromosome Selection**: Chromosome 21 data is selected from UCSC, and chromosome lengths are extracted.
- **Tiling and Read Counting**: The genome is tiled into 1000bp windows, and read counts are summarized using `summarizeOverlaps` for both the ChIP and control data.
- **Normalization**: Counts are normalized to reads per million (CPM) for each sample.
- **Visualization**: A scatter plot is created with normalized CTCF vs. Input counts using `ggplot2`, with a reference line (`geom_abline`) and customized aesthetics (text size, axis labels, etc.).

This analysis provides insights into the correlation between ChIP-Seq (CTCF) and control (Input) signals across chromosome 21.

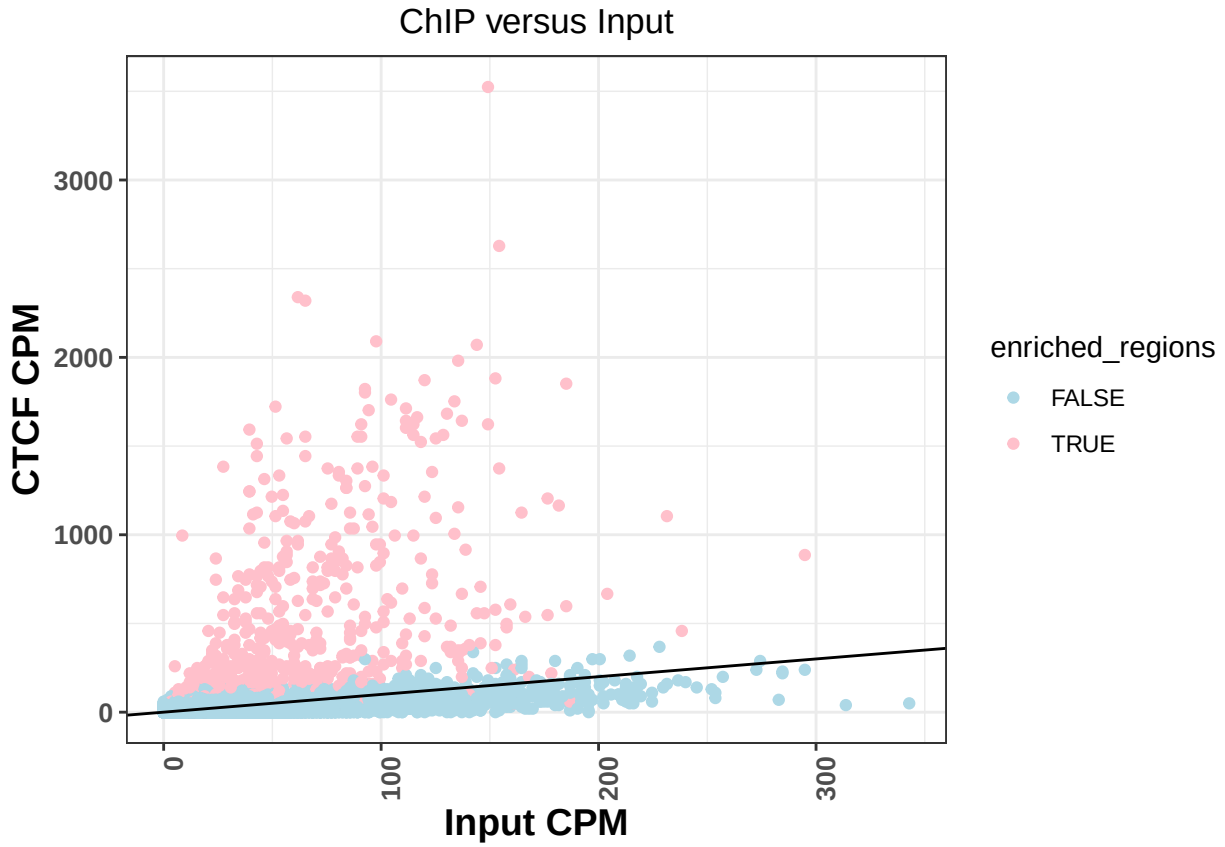
### 2.2 Peak Enrichment

```
# Peak calling using chip and control
ctcf_fit = enrichR(
            # ChIP file
            treatment = chip_file,
            # control file
            control   = control_file,
            # genome version
            genome    = "hg38",
            # print intermediary steps during   analysis
            verbose   = FALSE)

# Get Summary
summary(ctcf_fit)
```

```
## NormRFit-class object
##
## Type:                  'enrichR'
## Number of Regions:     12353090
## Number of Components:  2
## Theta* (naive bg):     0.137
## Background component B: 1
##
## +++ Results of fit +++
## Mixture Proportions:
## Background       Class 1
##     97.72%         2.28%
## Theta:
## Background       Class 1
##      0.103         0.695
##
## Bayesian Information Criterion:  539882
##
## +++ Results of binomial test +++
## T-Filter threshold: 4
## Number of Regions filtered out: 12267164
## Significantly different from background B based on q-values:
## TOTAL:
##            ***        **         *         .                n.s.
## Bins         0       627       120       195        87     84897
## %        0.000     0.711     0.847     1.068     1.166    96.209
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '  ' 1 'n.s.'
```

```
# Get ranges
ctcf_peaks = getRanges(ctcf_fit)

# annotate ranges with supporting p value
```

```r
ctcf_peaks$qvalue = getQvalues(ctcf_fit)

# annotate ranges with calculated enrichment
ctcf_peaks$enrichment = getEnrichment(ctcf_fit)

# selects ranges which correspond to enriched class
ctcf_peaks = subset(ctcf_peaks, !is.na(component))

# filter by a stringent q value threshold
ctcf_peaks = subset(ctcf_peaks, qvalue < 0.01)

# order peaks based on q value
ctcf_peaks = ctcf_peaks[order(ctcf_peaks$qvalue)]

# write   peaks loacations into a txt table
#write.table(ctcf_peaks, file.path(data_path, 'CTCF_peaks.txt'),
#            row.names=F, col.names=T, quote=F, sep='\t')
write.csv(ctcf_peaks, "../results/ctcf_peaks.csv", row.names = FALSE)

# find enriched tilling windows
enriched_regions = countOverlaps(tilling_window, ctcf_peaks) > 0

# Visualize
cpm$enriched_regions = enriched_regions

ggplot(data = cpm, aes(
  x = GM12878_hg38_Input_r5.chr21.bam,
  y = GM12878_hg38_CTCF_r1.chr21.bam,
  color = enriched_regions)) +
  geom_point() +
  geom_abline(slope = 1) +
   theme_bw() +
  scale_fill_brewer(palette='Set2') +
   theme(
    axis.text   = element_text(size=10, face='bold'),
    axis.title  = element_text(size=14,face="bold"),
    plot.title  = element_text(hjust = 0.5),
    axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab('Input CPM') +
  ylab('CTCF CPM') +
  ggtitle('ChIP versus Input') +
  scale_color_manual(values=c('lightblue','pink'))
```



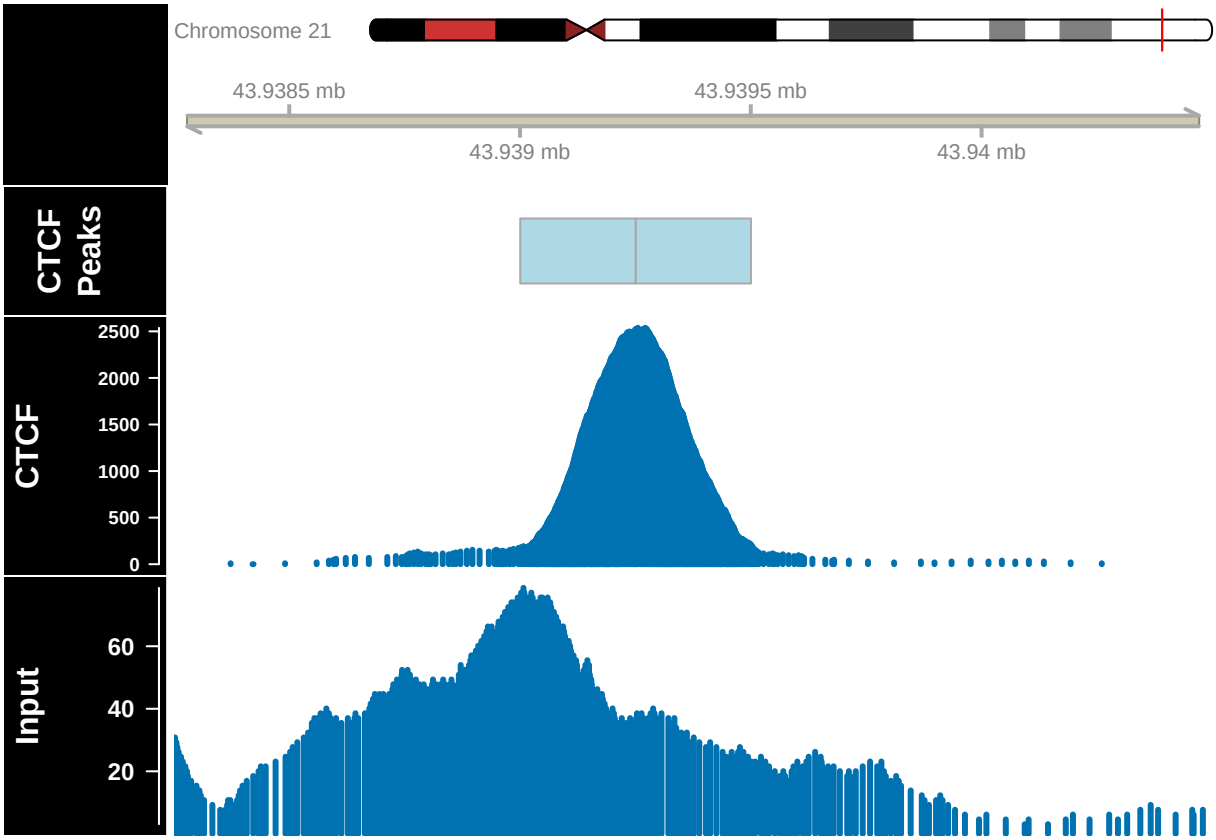

This code performs peak calling and enrichment analysis on ChIP-Seq data to identify and visualize enriched regions.

- **Peak Calling**: The `enrichR` function is used to call peaks from ChIP (CTCF) and control (Input) BAM files for the hg38 genome, with intermediary steps suppressed.
- **Peak Annotation**: The resulting peaks are annotated with q-values and enrichment scores.
- **Peak Filtering**: Peaks are filtered based on a stringent q-value threshold ($< 0.01$) and ordered by q-value.
- **Peak Output**: The peak locations are saved in a text file (`CTCF_peaks.txt`).
- **Enriched Regions**: Enriched regions are identified by counting overlaps between tiling windows and ChIP peaks.
- **Visualization**: A scatter plot is generated to compare normalized ChIP (CTCF) and control (Input) counts, with points colored based on whether they correspond to enriched regions.

This analysis identifies significant ChIP-Seq peaks and visualizes their enrichment relative to the control.

### 2.3 Signal Around Most Enriched Peak

```
# calculate   coverage for one bam file
calculateCoverage = function(
  bam_file,
  extend = 200){
  # load reads into R
  reads = readGAlignments(bam_file)
  # convert reads into a GRanges object
  reads = granges(reads)
  # resize   reads to 200bp
  reads = resize(reads, width=extend, fix='start')
  # get   coverage vector
  cov   = coverage(reads)
  # normalize   coverage vector to   sequencing depth
  cov = round(cov * (1000000/length(reads)),2)
  # convert   coverage go a GRanges object
  cov   = as(cov, 'GRanges')
  # keep only chromosome 21
  seqlevels(cov, pruning.mode='coarse') = 'chr21'
  return(cov)}

# Get coverage for  ChIP file
ctcf_cov = calculateCoverage(chip_file)

# Get coverage for  control file
cont_cov = calculateCoverage(control_file)

#Get chromosome coordinates
chr_track  = IdeogramTrack('chr21', 'hg38')
axis = GenomeAxisTrack(range = GRanges(
  'chr21', IRanges(1, width=seqlengths)))

# Peaks track
peaks_track = AnnotationTrack(ctcf_peaks, name = "CTCF Peaks")

# Signal files
chip_track  = DataTrack(
    range = ctcf_cov,
    name  = "CTCF",
    type  = 'h',
    lwd   = 3)

cont_track  = DataTrack(
    range = cont_cov,
    name  = "Input",
    type  = 'h',
    lwd=3)

plotTracks(
    trackList = list(chr_track, axis, peaks_track, chip_track, cont_track),
    sizes     = c(.2,.5,.5,1,1),
    background.title = "black",
    from = start(ctcf_peaks)[1] - 1000,
    to   = end(ctcf_peaks)[1]   + 1000)
```




This code calculates and visualizes coverage for ChIP-Seq (CTCF) and control (Input) BAM files.

- **Coverage Calculation**:
  - The `calculateCoverage` function loads BAM files, resizes the reads to 200bp, computes the coverage, and normalizes it based on sequencing depth. It then converts the coverage into a GRanges object and keeps only data for chromosome 21.
- **Coverage for ChIP and Control Files**:
  - The function is applied to both the ChIP (CTCF) and control (Input) BAM files to obtain their coverage.
- **Visualization**:
  - Various tracks are created:
    - **Chromosome Track**: Visualizes chromosome 21.
    - **Axis Track**: Displays the genomic axis.
    - **Peaks Track**: Shows annotated CTCF peaks.
    - **ChIP and Control Tracks**: Visualize the coverage for ChIP (CTCF) and control (Input) files.
- The tracks are combined and plotted using `plotTracks`, with a specified range around the first peak to highlight the signal and peaks.

This analysis visualizes the ChIP-Seq coverage and peaks, helping to assess the signal enrichment and compare it to the control.

### 3.1 Broad Regions

```
# fetch   ChIP-file for H3K36me3
chip_file    = file.path(data_path, 'GM12878_hg38_H3K36me3.chr21.bam')

# fetch   corresponding input file
control_file = file.path(data_path, 'GM12878_hg38_Input_r5.chr21.bam')

# define   window width for   counting
countConfiguration = countConfigSingleEnd(binsize = 5000)

# find broad peaks using enrichR
h3k36_fit = enrichR(
            # ChIP file
            treatment   = chip_file,
            # control file
            control     = control_file,
            # genome version
            genome      = "hg38",
            verbose     = FALSE,
            # window size for counting
            countConfig = countConfiguration)

summary(h3k36_fit)
```

```
## NormRFit-class object
##
## Type:                  'enrichR'
## Number of Regions:     617665
## Number of Components:  2
## Theta* (naive bg):     0.197
## Background component B: 1
##
## +++ Results of fit +++
## Mixture Proportions:
## Background       Class 1
##      85.4%         14.6%
## Theta:
## Background       Class 1
##      0.138         0.442
##
## Bayesian Information Criterion:  741525
##
## +++ Results of binomial test +++
## T-Filter threshold: 5
## Number of Regions filtered out: 610736
## Significantly different from background B based on q-values:
## TOTAL:
##           ***       **        *        .              n.s.
## Bins        0     1005      314      381      237     4992
## %        0.00     9.18    12.04    15.52    17.68    45.58
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '  ' 1 'n.s.'
```

This code identifies broad peaks in the H3K36me3 ChIP-Seq data and compares it to an input control file.

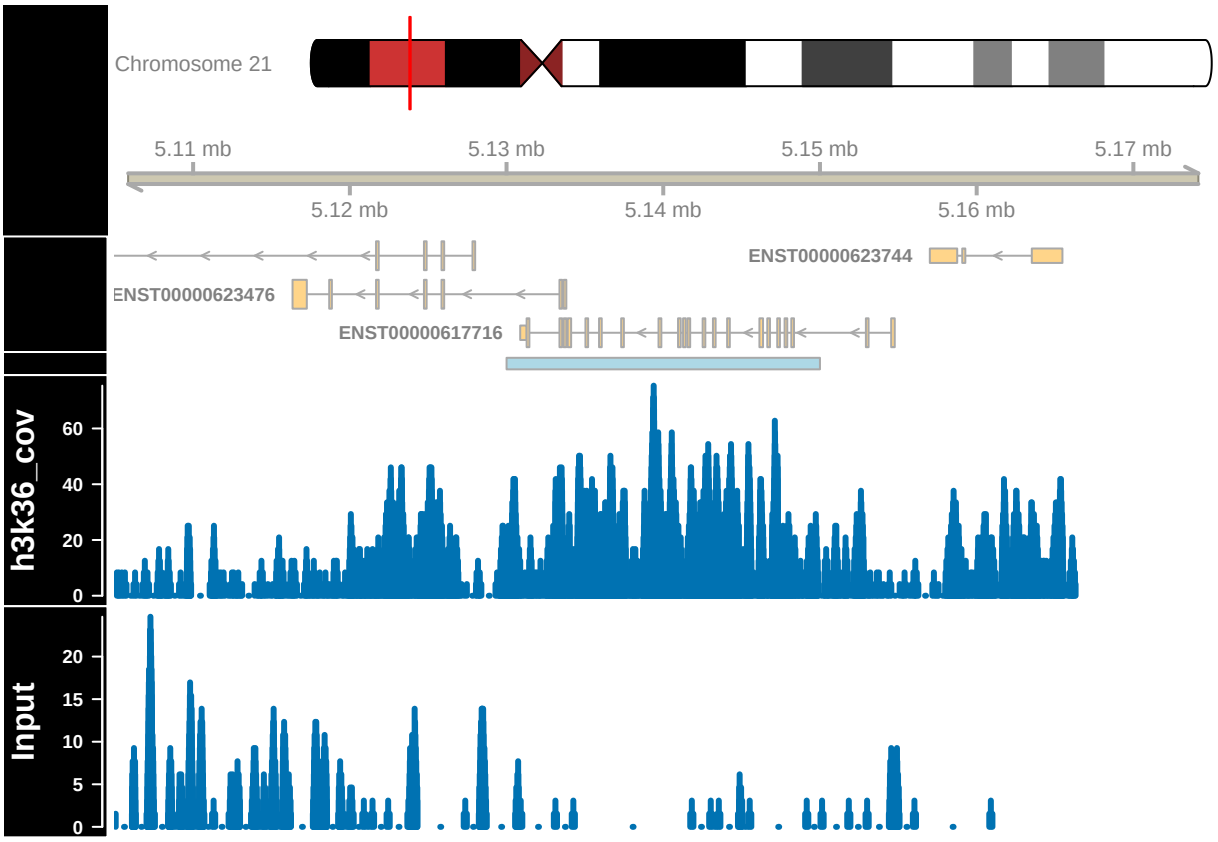
- **File Fetching**:
  - The ChIP-Seq file for H3K36me3 and its corresponding control file (Input) are retrieved.
- **Counting Configuration**:
  - A window size of 5000 bases is defined for counting using `countConfigSingleEnd`.
- **Broad Peak Identification**:
  - The `enrichR` function is used to identify broad peaks in the ChIP-Seq data by comparing the ChIP file (H3K36me3) against the control (Input), using the specified window size for counting.
- **Results Summary**:
  - The results of the broad peak calling are summarized with `summary(h3k36_fit)`.

This analysis helps identify broad enriched regions in H3K36me3 ChIP-Seq data, which can be important for understanding histone modifications and transcriptional regulation.

### 3.2 Enriched Regions

```
# get locations of broad peaks
h3k36_peaks = getRanges(h3k36_fit)

# extract qvalue and enrichment
h3k36_peaks$qvalue = getQvalues(h3k36_fit)
h3k36_peaks$enrichment = getEnrichment(h3k36_fit)

# select proper peaks
h3k36_peaks = subset(h3k36_peaks, !is.na(component))
h3k36_peaks = subset(h3k36_peaks, qvalue < 0.01)
h3k36_peaks = h3k36_peaks[order(h3k36_peaks$qvalue)]

# collapse nearby enriched regions
h3k36_peaks = reduce(h3k36_peaks)
# construct  data tracks for  H3K36me3 and Input files
h3k36_cov = calculateCoverage(chip_file)
data_tracks = list(
    h3k36 = DataTrack(h3k36_cov,  name = 'h3k36_cov',  type='h', lwd=3),
    input = DataTrack(cont_cov,   name = 'Input',      type='h', lwd=3))

# define  window for   visualization
start = min(start(h3k36_peaks[2])) - 25000
end   = max(end(h3k36_peaks[2])) + 25000

# create peak track
peak_track = AnnotationTrack(reduce(h3k36_peaks), name='H3K36me3')

# plots  enriched region
plotTracks(
    trackList = c(chr_track, axis, gene_track, peak_track, data_tracks),
    sizes     = c(.5,.5,.5,.1,1,1),
```

```
    background.title     = "black",
    collapseTranscripts  = "longest",
    transcriptAnnotation = "symbol",
    from = start,
    to   = end)
```




This code identifies and visualizes enriched regions for H3K36me3 broad peaks.

- **Peak Identification**:
  - Broad peaks are extracted from the `h3k36_fit` object using `getRanges()`.
  - Q-values and enrichment scores are computed for each peak, and only significant peaks (q-value $< 0.01$) are retained.
- **Peak Filtering and Collapse**:
  - The significant peaks are ordered by q-value and nearby enriched regions are collapsed using `reduce()`.
- **Coverage Calculation**:
  - Coverage for the H3K36me3 ChIP-Seq and control (Input) files is calculated using the `calculateCoverage()` function.
- **Data Tracks Construction**:
  - Data tracks for the H3K36me3 ChIP-Seq and control (Input) files are created using `DataTrack()`.
- **Visualization**:

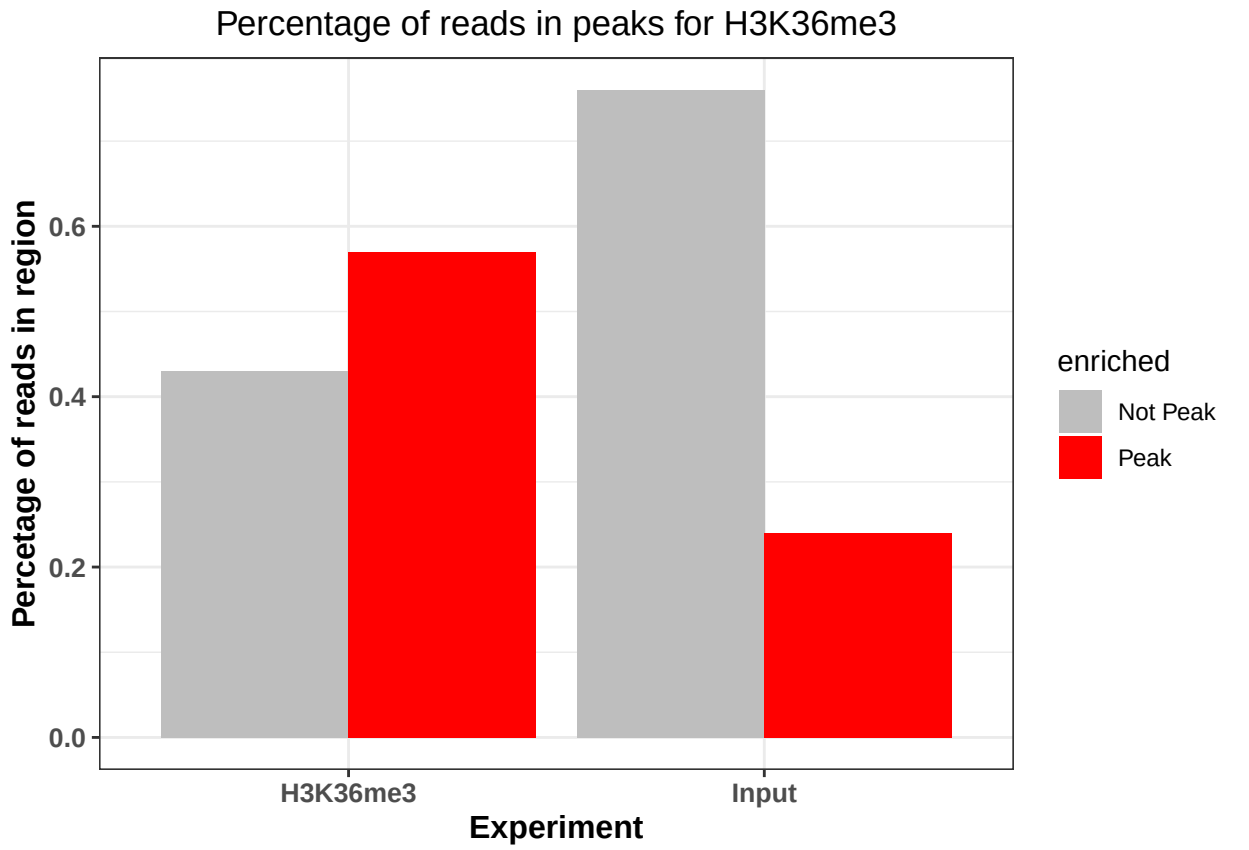
- A genomic region containing the enriched peaks is visualized with `plotTracks()`, including chromosome track, gene track, peak track, and coverage tracks for both H3K36me3 and Input.

This process provides a comprehensive visualization of H3K36me3 enriched regions across the genome.

## 4. Peak Quality Control

```
# extract, per tilling window, counts from   fit object
h3k36_counts = data.frame(getCounts(h3k36_fit))

# change   column names of   data.frame
colnames(h3k36_counts) = c('Input','H3K36me3')

# extract   q-value corresponding to each bin
h3k36_counts$qvalue = getQvalues(h3k36_fit)

# define which regions are peaks using a q value cutoff
h3k36_counts$enriched[is.na(h3k36_counts$qvalue)]  = 'Not Peak'
h3k36_counts$enriched[h3k36_counts$qvalue > 0.05]  = 'Not Peak'
h3k36_counts$enriched[h3k36_counts$qvalue <= 0.05] = 'Peak'

# remove   q value column
h3k36_counts$qvalue = NULL

# reshape   data.frame into a long format
h3k36_counts_df = tidyr::pivot_longer(
    data = h3k36_counts,
    cols = -enriched,
    names_to = 'experiment',
    values_to = 'counts')

# sum   number of reads in   Peak and Not Peak regions
h3k36_counts_df = group_by(.data = h3k36_counts_df, experiment, enriched)
h3k36_counts_df = summarize(.data = h3k36_counts_df, num_of_reads = sum(counts))

# calculate   percentage of reads.
h3k36_counts_df       = group_by(.data = h3k36_counts_df, experiment)
h3k36_counts_df       = mutate(.data = h3k36_counts_df, total=sum(num_of_reads))
h3k36_counts_df$percentage = with(h3k36_counts_df, round(num_of_reads/total,2))


ggplot(data = h3k36_counts_df, aes(
        x = experiment,
        y = percentage,
        fill = enriched)) +
  geom_bar(stat='identity', position='dodge') +
  theme_bw() +
  theme(axis.text = element_text(size=10, face='bold'),
     axis.title = element_text(size=12,face="bold"),
     plot.title = element_text(hjust = 0.5)) +
  xlab('Experiment') +
  ylab('Percetage of reads in region') +
```

```
ggtitle('Percentage of reads in peaks for H3K36me3') +
scale_fill_manual(values=c('gray','red'))
```




This code performs quality control on H3K36me3 peaks by assessing the counts within tiling windows and calculating the percentage of reads in peak versus non-peak regions.

- **Data Extraction and Preprocessing**:
  - Counts are extracted from the `h3k36_fit` object using `getCounts()` and stored in a data frame.
  - Column names are updated to reflect the treatment (`Input`, `H3K36me3`).
- **Peak Classification**:
  - Based on q-values, regions are classified as "Peak" (q-value  0.05) or "Not Peak" (q-value $>$ 0.05).
  - Rows with missing q-values are also classified as "Not Peak."
- **Data Reshaping**:
  - The data is reshaped into a long format using `pivot_longer()` for easier plotting.
- **Read Summarization**:
  - The total number of reads in peak and non-peak regions is calculated by grouping and summarizing the data.
- **Percentage Calculation**:
  - The percentage of reads in peak regions is calculated by dividing the number of reads in each region by the total number of reads for each experiment.
- **Visualization**:

- A bar plot is created to visualize the percentage of reads in peak versus non-peak regions, with separate bars for each experiment and color-coded for enriched (peak) and non-enriched (not peak) regions.

This analysis provides an overview of the distribution of reads across peak and non-peak regions for H3K36me3.

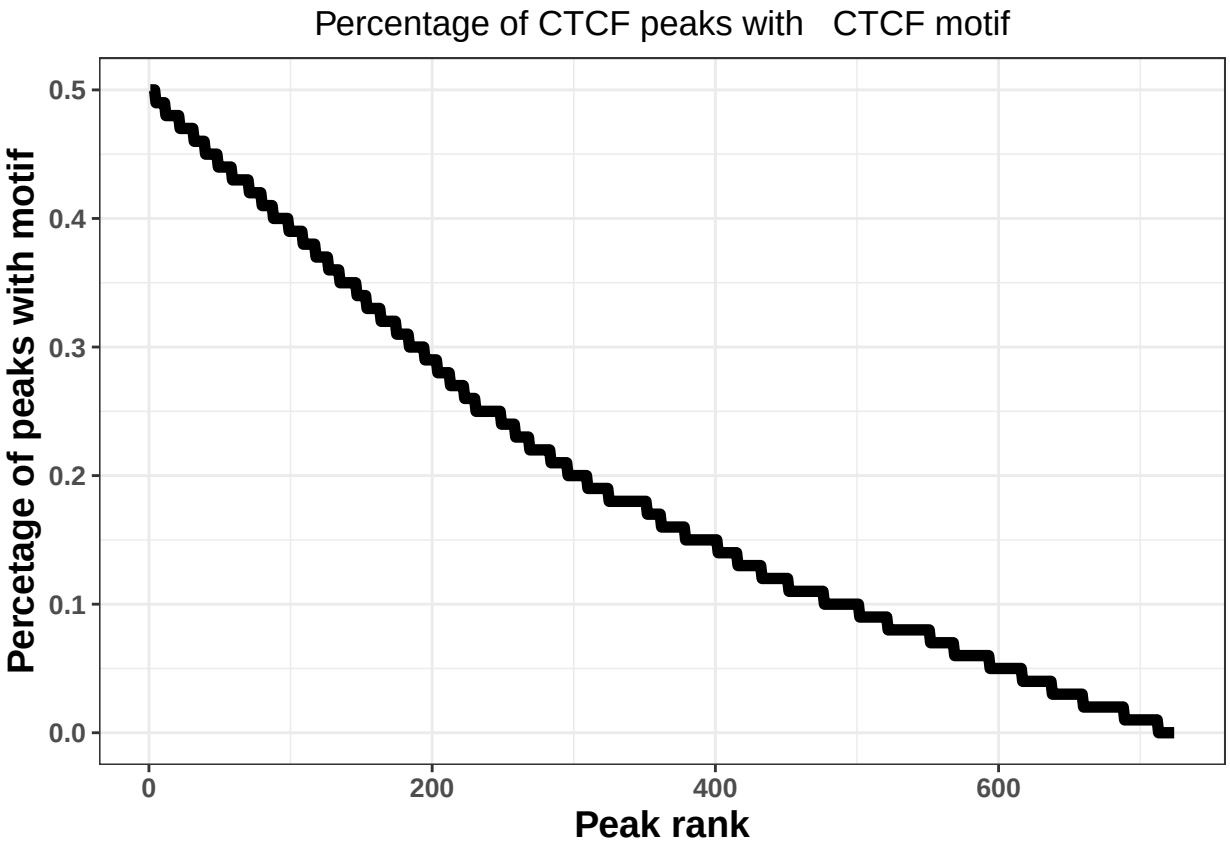
## 5. DNA Motifs on Peaks

```
# fetch   CTCF motif from   data base
motifs = query(query(MotifDb, 'Hsapiens'), 'CTCF')
motifs
```

```
## MotifDb object of length 16
## | Created from downloaded public sources, last update: 2022-Mar-04
## | 16 position frequency matrices from 9 sources:
## |        HOCOMOCOv10:    2
## | HOCOMOCOv11-core-A:    2
## |        JASPAR_2014:    1
## |        JASPAR_CORE:    1
## |         jaspar2016:    1
## |         jaspar2018:    2
## |         jaspar2022:    4
## |          jolma2013:    1
## |       SwissRegulon:    2
## | 1 organism/s
## |           Hsapiens:   16
## Hsapiens-HOCOMOCOv10-CTCFL_HUMAN.H10MO.A
## Hsapiens-HOCOMOCOv10-CTCF_HUMAN.H10MO.A
## Hsapiens-HOCOMOCOv11-core-A-CTCFL_HUMAN.H11MO.0.A
## Hsapiens-HOCOMOCOv11-core-A-CTCF_HUMAN.H11MO.0.A
## Hsapiens-JASPAR_CORE-CTCF-MA0139.1
## ...
## Hsapiens-jaspar2022-CTCF-MA1929.1
## Hsapiens-jaspar2022-CTCF-MA1930.1
## Hsapiens-jolma2013-CTCF
## Hsapiens-SwissRegulon-CTCFL.SwissRegulon
## Hsapiens-SwissRegulon-CTCF.SwissRegulon
```

```
# Subset motifs
ctcf_motif  = motifs[[1]]

# extend peak regions
ctcf_peaks_resized = resize(ctcf_peaks, width = 400, fix = 'center')

# extract   sequences around   peaks
seq = getSeq(BSgenome.Hsapiens.UCSC.hg38, ctcf_peaks_resized)


# convert   motif matrix to PWM, and scan   peaks
ctcf_pwm    = PWMatrix(ID = 'CTCF', profileMatrix = ctcf_motif)
```

```
hits = searchSeq(ctcf_pwm, seq, min.score="80%", strand="*")
hits = as.data.frame(hits)


# label which peaks contain CTCF motifs
motif_hits_df = data.frame(peak_order = 1:length(ctcf_peaks))

motif_hits_df$contains_motif = motif_hits_df$peak_order %in% hits$seqnames

motif_hits_df = motif_hits_df[order(-motif_hits_df$peak_order),]

# calculate  percentage of peaks with motif for peaks of descending strength
motif_hits_df$perc_peaks = with(
  motif_hits_df,
  cumsum(contains_motif)/max(peak_order))
motif_hits_df$perc_peaks = round(motif_hits_df$perc_peaks, 2)


# plot  cumulative distribution of motif hit percentages
ggplot(motif_hits_df, aes(x = peak_order, y = perc_peaks)) +
  geom_line(size=2) +
  theme_bw() +
  theme(
    axis.text = element_text(size=10, face='bold'),
    axis.title = element_text(size=14,face="bold"),
    plot.title = element_text(hjust = 0.5)) +
  xlab('Peak rank') +
  ylab('Percetage of peaks with motif') +
  ggtitle('Percentage of CTCF peaks with  CTCF motif')
```



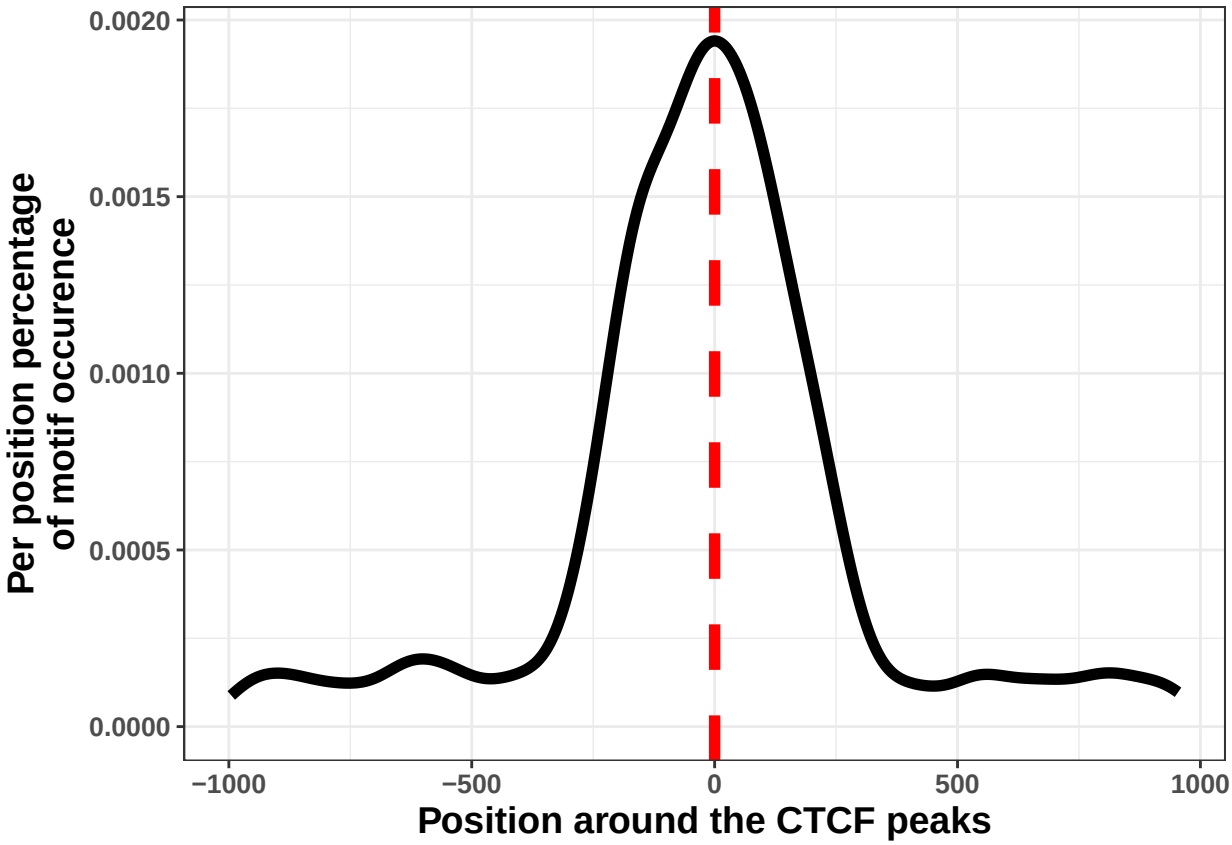

This code analyzes the presence of the CTCF motif in peak regions:

- **Motif Retrieval**:
  - The CTCF motif is fetched from the `MotifDb` database for human (Hsapiens).
- **Peak Region Resizing**:
  - Peak regions (`ctcf_peaks`) are extended by resizing them to 400 base pairs centered on the peak.
- **Sequence Extraction**:
  - Sequences around the resized peaks are extracted from the `BSgenome.Hsapiens.UCSC.hg38` reference genome.
- **Motif Scanning**:
  - The CTCF motif matrix is converted to a position weight matrix (PWM), which is then used to scan the peak sequences for motif hits with a minimum score of 80%.
- **Motif Hit Classification**:
  - Peaks are labeled as containing the CTCF motif if a hit is found. A cumulative percentage of peaks containing the motif is calculated in order of peak strength.
- **Visualization**:
  - A line plot is generated showing the cumulative percentage of peaks with the CTCF motif as a function of peak rank, indicating how motif presence correlates with peak strength.

This analysis helps identify and quantify the CTCF motif distribution in peak regions.

## 6. Motif Localization

```
# resize region around peaks to +/- 1kb
ctcf_peaks_resized = resize(ctcf_peaks, width = 2000, fix='center')

# fetch sequence
seq = getSeq(BSgenome.Hsapiens.UCSC.hg38,ctcf_peaks_resized)

# convert the motif matrix to PWM, and scan the peaks
ctcf_pwm = PWMatrix(ID = 'CTCF', profileMatrix = ctcf_motif)
hits1 = searchSeq(ctcf_pwm, seq, min.score="80%", strand="*")
hits1 = as.data.frame(hits1)

# set the position relative to the start
hits1$position = hits1$start - 1000

# plot the motif hits around peaks
ggplot(data=hits1, aes(position)) +
  geom_density(size=2) +
  theme_bw() +
  geom_vline(xintercept = 0, linetype=2, color='red', size=2) +
  xlab('Position around the CTCF peaks') +
  ylab('Per position percentage\nof motif occurence') +
  theme(
    axis.text = element_text(size=10, face='bold'),
    axis.title = element_text(size=14,face="bold"),
    plot.title = element_text(hjust = 0.5))
```




This code analyzes the distribution of CTCF motifs around peak regions:

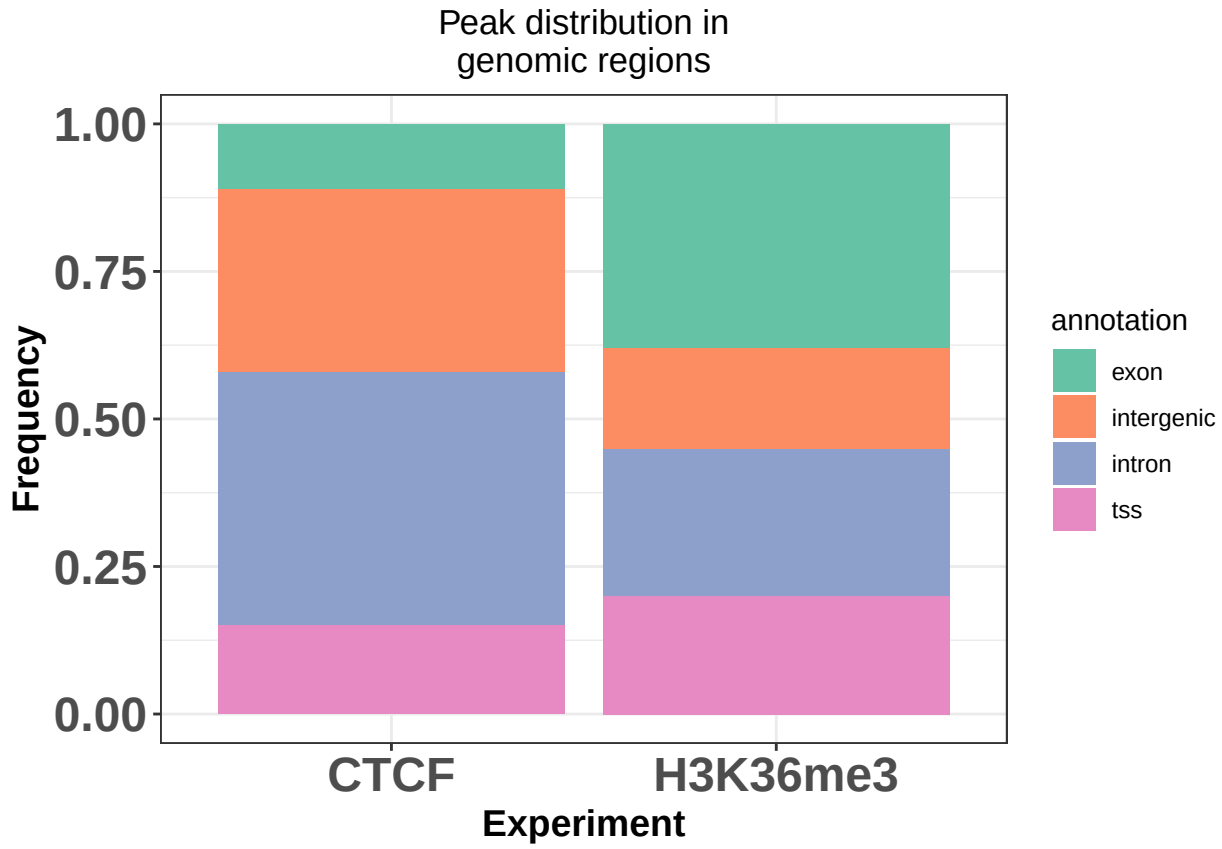
- **Peak Region Resizing**:
  - The region around CTCF peaks is resized to a 2kb window centered on each peak.
- **Sequence Extraction**:
  - Sequences from the resized peak regions are retrieved using the `BSgenome.Hsapiens.UCSC.hg38` reference genome.
- **Position Adjustment**:
  - The position of motif hits is adjusted relative to the center of the peak, with the start of the region set as -1000.
- **Motif Hit Distribution**:
  - A density plot is generated to show the distribution of motif hits across the resized peak regions, with a red vertical line marking the peak center (position = 0).

This plot helps visualize the localization of the CTCF motif relative to CTCF peak regions.

## 7. Peak Annotation

```r
# Get annotation
seqlevels(gtf, pruning.mode='coarse') = '21'
seqlevels(gtf, pruning.mode='coarse') = paste0('chr', seqlevels(gtf))

# create   annotation hierarchy
annotation_list = GRangesList(
  tss    = promoters(subset(gtf, type=='gene'), 1000, 1000),
  exon   = subset(gtf, type=='exon'),
  intron = subset(gtf, type=='gene'))


# function which annotates    location of each peak
annotatePeaks = function(peaks, annotation_list, name){
  # getting disjoint regions
  # collapse touching enriched regions
  peaks = reduce(peaks)

  # overlapping peaks and annotation
  # find overlaps between    peaks and annotation_list
  result = as.data.frame(findOverlaps(peaks, annotation_list))

  # annotating peaks and fetch annotation names
  result$annotation = names(annotation_list)[result$subjectHits]

  # rank by annotation precedence
  result = result[order(result$subjectHits),]

  # remove overlapping annotations
  result = subset(result, !duplicated(queryHits))


  # Get statistics and count number of peaks in each annotation category
  result = group_by(.data = result, annotation)
  result = summarise(.data = result, counts = length(annotation))

  # Get number of intergenic peaks
  result = rbind(result,
                 data.frame(annotation = 'intergenic',
                            counts     = length(peaks) - sum(result$counts)))

  result$frequency  = with(result, round(counts/sum(counts),2))
  result$experiment = name

  return(result)
}


# Get list of CTCF and H3K36me3 peaks
peak_list = list(
    CTCF     = ctcf_peaks,
    H3K36me3 = h3k36_peaks)

# calculate    distribution of peaks in annotation for each experiment
```

```r
annot_peaks_list = lapply(names(peak_list), function(peak_name){
  annotatePeaks(peak_list[[peak_name]], annotation_list, peak_name)})

# combine a list of data.frames into one data.frame
annot_peaks_df = dplyr::bind_rows(annot_peaks_list)


# plot   distribution of peaks in genomic features
ggplot(data = annot_peaks_df,
       aes(x    = experiment,
           y    = frequency,
           fill = annotation)) +
  geom_bar(stat='identity') +
  scale_fill_brewer(palette='Set2') +
  theme_bw()+
  theme(
    axis.text = element_text(size=18, face='bold'),
    axis.title = element_text(size=14,face="bold"),
    plot.title = element_text(hjust = 0.5))  +
  ggtitle('Peak distribution in\ngenomic regions') +
  xlab('Experiment') +
  ylab('Frequency')
```




This code annotates peaks and analyzes their distribution across genomic features:

- **Downloading Annotation**:

  - Annotations for chromosome 21 are fetched from `AnnotationHub` and refined to include only relevant sequences.

- **Annotation Hierarchy**:
  - A list of annotations is created, including:
    * **TSS (Transcription Start Sites)**: Promoters of genes with a 1000 bp window.
    * **Exons**: All exons.
    * **Introns**: Gene-based introns.

- **Peak Annotation Function**:
  - The function `annotatePeaks()` is used to find overlaps between peaks and the defined annotations. It calculates the number of peaks in each annotation category and ranks them by precedence (e.g., TSS > exon > intron).
  - Intergenic peaks (peaks not overlapping any gene feature) are also counted.

- **Peak List**:
  - A list of CTCF and H3K36me3 peaks is defined, and the `annotatePeaks()` function is applied to each peak set.

- **Plotting**:
  - A bar plot is generated showing the distribution of peaks across different genomic regions, such as TSS, exons, and introns, for both CTCF and H3K36me3 experiments.

The plot visualizes the frequency of peaks located in different genomic features for each experiment.

## 8. Conclusion

In this peak calling exercise, identification and annotated genomic regions enriched for specific histone modifications, such as H3K36me3 and CTCF binding sites was carried out. Through peak detection and motif analysis, identification of key peaks that are significantly associated with CTCF motifs was done. By resizing the peak regions and extracting sequences, we were able to determine the relative position of these motifs within the peaks.

classification of peaks based on their genomic location, categorizing them into features such as promoters (TSS), exons, and introns was done and quantification as well of the distribution of these peaks across different regions of the genome. The results indicated distinct peak distributions for each experiment, with significant differences in the frequency of peaks overlapping specific genomic features. This analysis provides valuable insights into the localization of key regulatory elements and their potential role in gene expression regulation. It shows the importance of peak annotation in understanding the functional relevance of enriched genomic regions in the context of chromatin modifications and transcription factor binding.