ChIP-seq with Bioconductor in R
class by Peter Humburg and DataCamp

ChIP-seq attempts to answer how do cells know what to do?

Regulating gene expression
inside each cell a complex machinery of proteins is responsible for ensuring that the right genes are translated into proteins
inhibitors are proteins that bind to DNA to deactivate specific genes
inhibitors have to be removed through the interaction with other proteins before genes can be expressed
then a complex of activating transcription factors can then bind to the DNA, allowing gene expression to proceed
this regulatory machinery ensures that a cell correctly performs its role
disruption of the process can lead to proliferation of cell growth (cancer) or a varitey of other diseases

ChIP stands for chromatin immunoprecipitation
a technique that can be used to extract specific proteins and the parts of the genome they were bound to
we can then use the DNA sequences attached to the proteins to infer the sites across the genome that they interact with
do this by identifying regions of the genome that are overrepresented in the sequencing data
by then comparing these sites between healthy volunteers and say cancer patients we can potentially uncover the mechanisms that are responsible for the differences between them

Accessing ChIP-seq data in R
functions used in R to interrogate sequencing data
mapped sequence reads are typically stored in BAM files
load data from BAM files with readGAlignments() function
library(GenomicAlignments)
reads <- readGAlignments(‘file_name’)
info about the chromosome reads are available via seqnames()
location on the chromosome can be accessed via start() and end()
how many reads cover any given position in the genome?
coverage(reads)

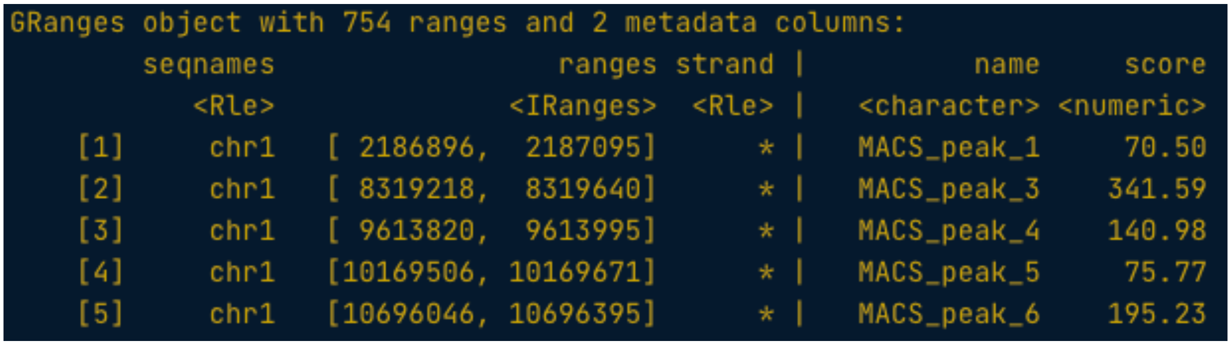
Accessing peak calls
peak calls are main units of interest in the analysis of a ChIP-seg experiment

they highlight regions of the genome with a high concentration of reads
peak calls are typically stored in BED files
each peak is associated with a score
the score quantifies the strength of this particular peak
peak calls are loaded with import.bed() function
obtain coordinates of peaks by calling the chrom() and ranges() functions
score() function provides access to peak scores

Example

```
# Print the 'reads' object to obtain a summary of the data
print(reads)

# Get the *start* position of the first read
start_first <- start(reads)[1]

# Get the *end* position of the last read
end_last <- end(reads)[length(reads)]

# Compute the number of reads covering each position in the selected region
cvg <- coverage(reads)

# Print a summary of the 'peaks' object
print(peaks)

# Use the score function to find the index of the highest scoring peak
max_idx <- which.max(score(peaks))

# Extract the genomic coordinates of the highest scoring peak using the `chrom` and `ranges` functions
max_peak_chrom <- chrom(peaks)[max_idx]
max_peak_range <- ranges(peaks)[max_idx]
```

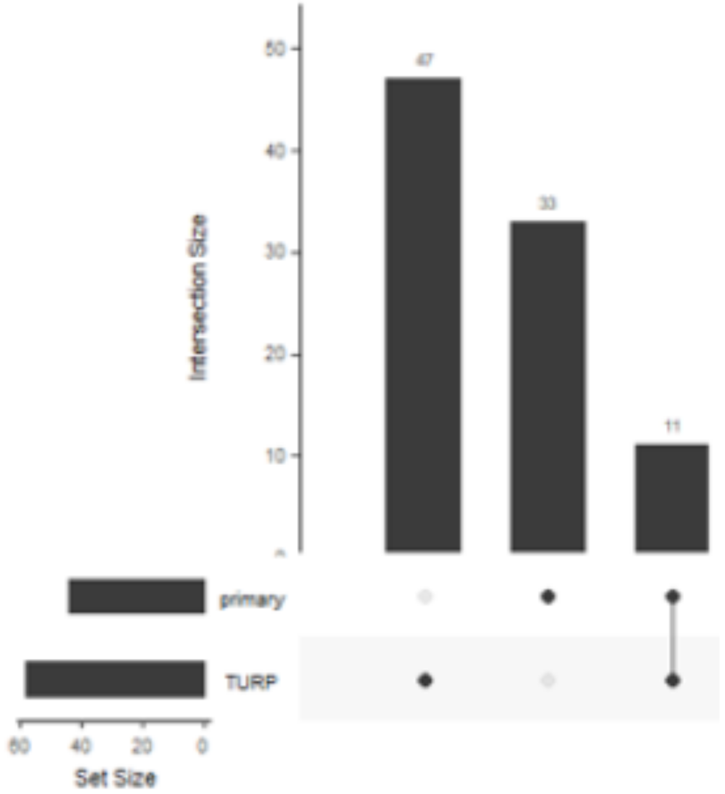
output>



```
GRanges object with 754 ranges and 2 metadata columns:
        seqnames                ranges strand |        name     score
           <Rle>             <IRanges>  <Rle> | <character> <numeric>
    [1]     chr1 [ 2186896,  2187095]      * | MACS_peak_1     70.50
    [2]     chr1 [ 8319218,  8319640]      * | MACS_peak_3    341.59
    [3]     chr1 [ 9613820,  9613995]      * | MACS_peak_4    140.98
    [4]     chr1 [10169506, 10169671]      * | MACS_peak_5     75.77
    [5]     chr1 [10696046, 10696395]      * | MACS_peak_6    195.23
```

ChIP-seq Workflow
first step in the analysis is to take the collection of reads obtained from the sequencing machine
and to locate their position in the genome
known as ‘read mapping’
involves identifying the best match for each read sequence in a standardised version of the genome (reference genome)

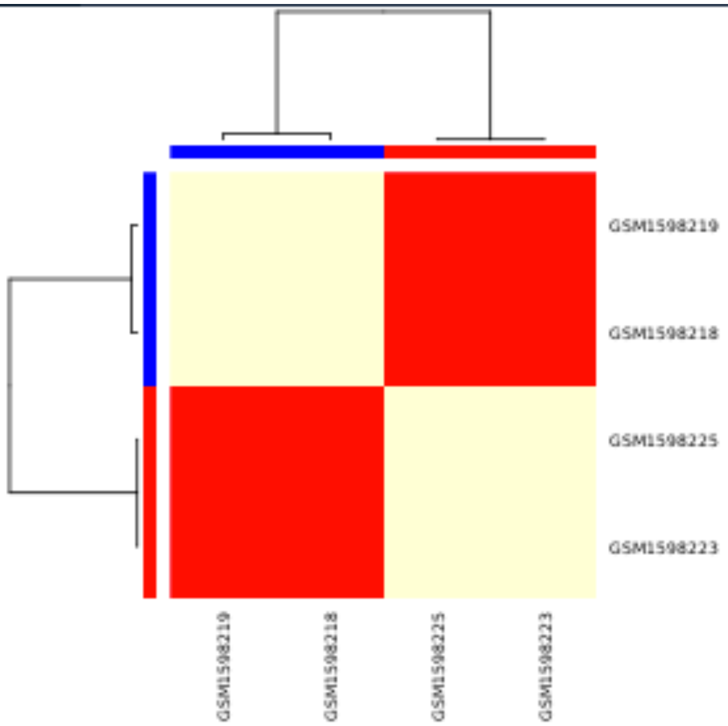
reads are mapped to the reference (coverage profile)
total number of reads overlapping with that position is determined
then used to identify peaks in this coverage profile
these correspond to the likely location of binding sites for the protein of interest
next step is data import
then quality control
then analysis by comparing samples
goal is to identify interesting peaks
a peak to be of interest it needs to play a direct role in the difference between samples
our example identifies AR binding sites that are preferentially used in either primary or treatment resistant tumors
summarize results
start by creating a heatmap > helps highlight similarities and differences between samples
also special UpSetR package provides useful plots to assess the degree of similarity between samples
example of plot with UpSetR



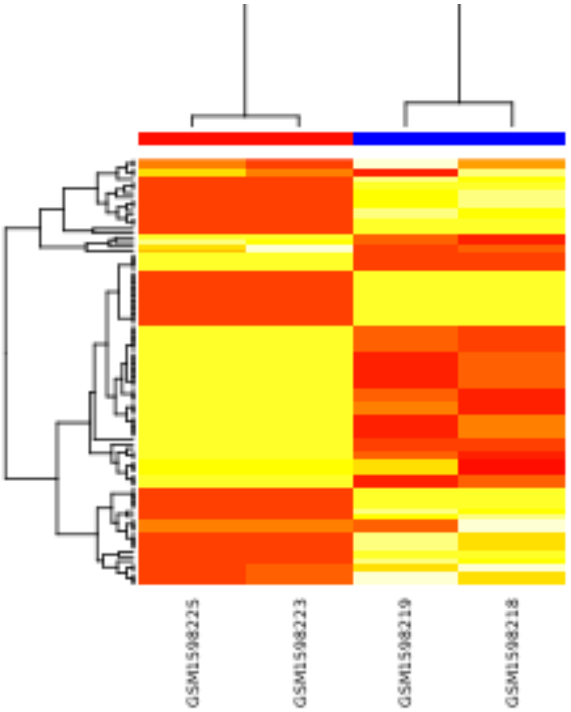
Example

```
# Create a vector of colors to label groups (there are 2 samples per group)
#rep() used to replicate elements in a vector
#rep(x, times) basic syntax where x is the vector to be repeated and times equates to the number of times to repeat each element in the vector
group <- c(primary = rep("blue", 2), TURP = rep("red", 2))

# Plot the sample correlation matrix `sample_cor` as a heat map
# Use the group colors to label the rows and columns of the heat map
heatmap(sample_cor, ColSideColors = group, RowSideColors = group,
        cexCol = 0.75, cexRow = 0.75, symm = TRUE)

# Create a heat map of peak read counts
# Use the group colors to label the columns of the heat map
heatmap(read_counts, ColSideColors = group, labRow = "", cexCol = 0.75)
```

output>



#shows that samples form blocks according to their group
#these plots help us to assess sample quality

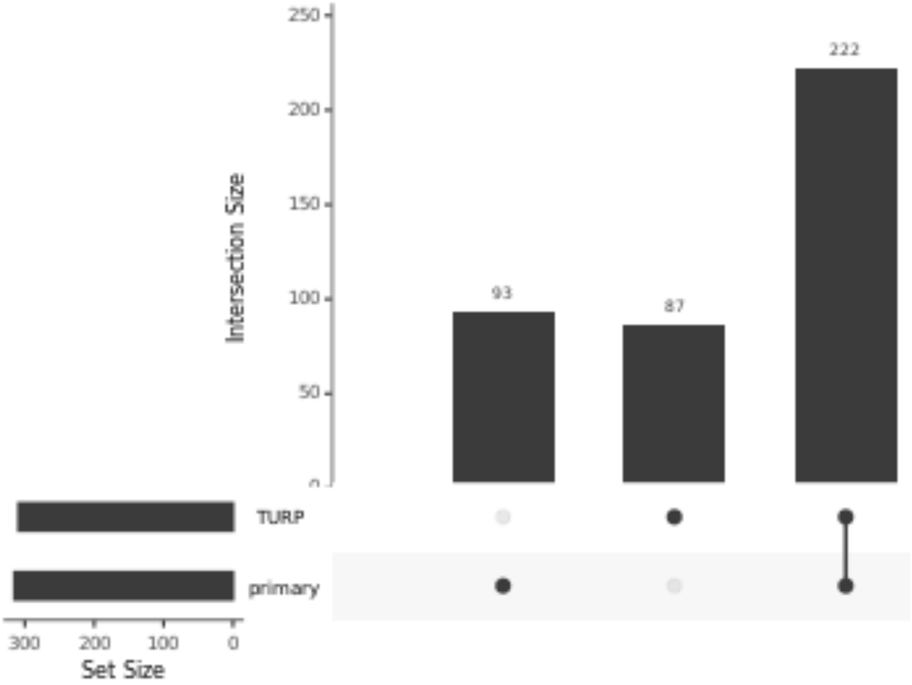


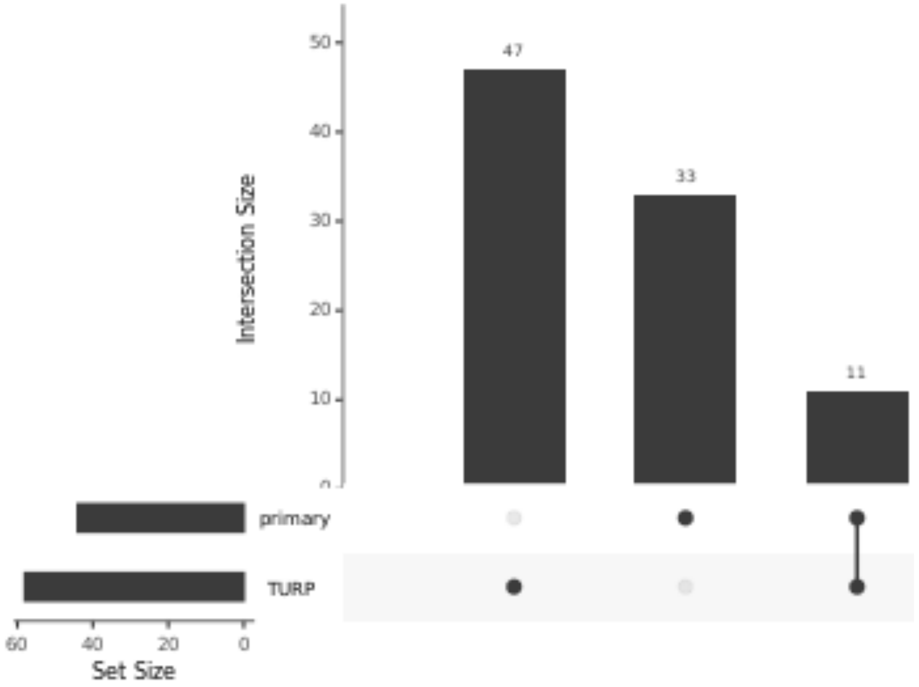
#different samples are represented as columns
#different peaks as rows
#cell color corresponds to the height of that peak

Example - looking at the full gene sets and the differentially bound peaks

```
# Take a look at the full gene sets
print(ar_sets)

# Visualise the overlap between the two groups using the `upset` function
upset(fromList(ar_sets))

# Print the genes with differential binding
print(db_sets)

# Visualise the overlap of differentially bound peaks between the two groups using
the `upset` function
upset(fromList(db_sets))
```

output>



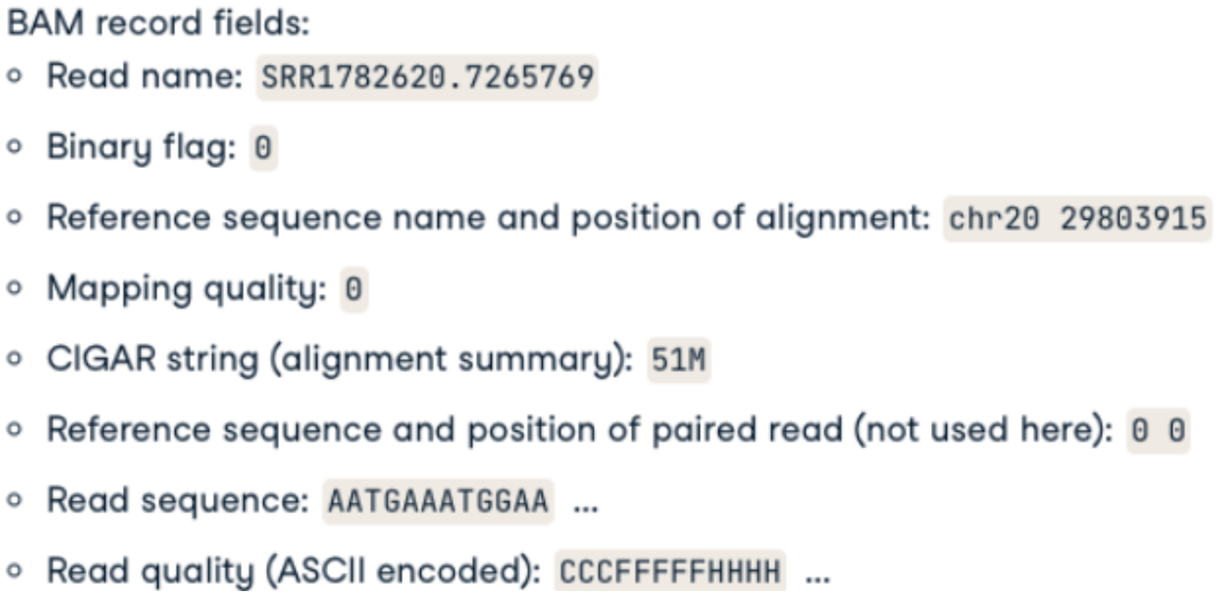


#to understand what the observed differences in protein binding actually mean it is helpful to associate observed peaks with genes
#above visualises the overlap in genes associated with peak calls in the two groups of samples
#the vertical bars corresponds to the size of one subset
#the dots below the vertical bars indicate which groups these genes were observed in
#above shows that 47 genes associated with peaks in the TURP condition and not in the Primary condition
#these genes act as a starting point of the investigation

ChIP-seq results summary
is there evidence for a systematic difference between groups?

Importing data
map the reads that come off the sequencing machine to the reference genome
identify the peaks (coverage profile)
*this is usually done outside of R with specialised tools
example tools for mapping (BWA, Bowtie 2, Stampy)
example tools for calling peaks (MACS2, PeakSeq, SISSRs)

mapped sequence reads are stored in a binary file format > Binary Sequence Alignment/Map (BAM) file
This is what the BAM holds:



BAM record fields:

- Read name: `SRR1782620.7265769`
- Binary flag: `0`
- Reference sequence name and position of alignment: `chr20 29803915`
- Mapping quality: `0`
- CIGAR string (alignment summary): `51M`
- Reference sequence and position of paired read (not used here): `0 0`
- Read sequence: `AATGAAATGGAA` ...
- Read quality (ASCII encoded): `CCCFFFFFHHHH` ...

tells you how it was mapped to the reference genome
tells you which part of the reference genome is most similar to this read
how they differ
how reliable the alignment is
*much easier to use these tools than to deal with this data directly

R reads BAM files by using the Rsamtools package
provides functions to index, read, filter, and write BAM files
we import mapped reads with readGAlignments

```
library(GenomicAlignments)
reads <- readGAlignments(bam_file)
#this returns a GAlignments object
```

gives us information for each read

with BamViews we do not need to load all reads from a BAM file
saving space and increasing efficiency
example

```
library(GenomicRanges)
library(Rsamtools)
ranges <- GRanges(...)
views <- BamViews(bam_file, bamRanges=ranges)
#then import reads as before
reads <- readGAlignments(views)
```

this is nice because now we can look at a specific gene or regions of interest (like peak calls)

Importing peak calls
import.bed loads peak calls from a BED file

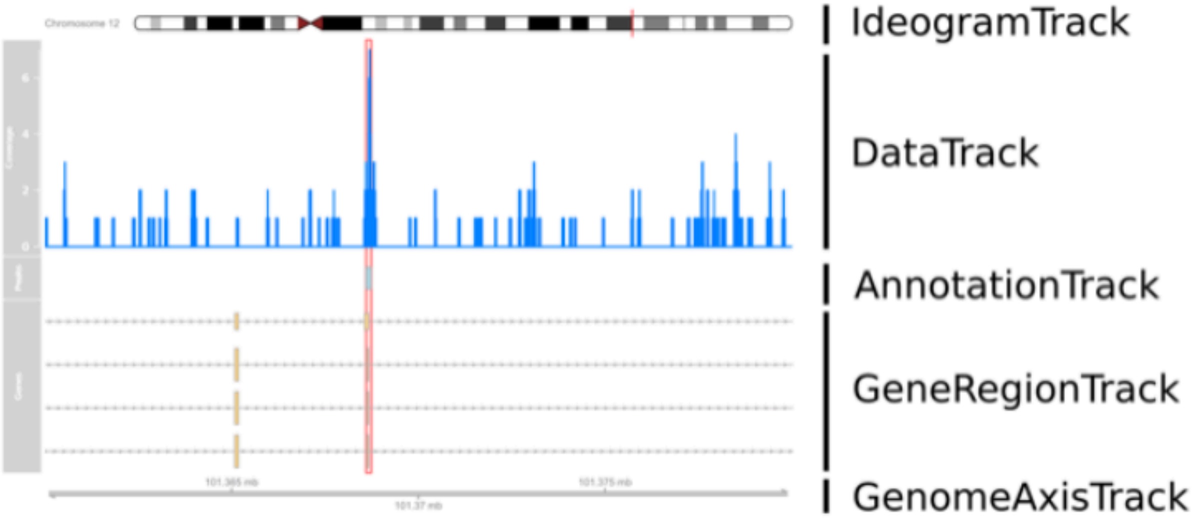
example

```
library(rtracklayer)
peaks <- import.bed(peak_bed, genome=‘hg19’)
#adding ‘genome’ identifier allows additional info to be added to the ouput automatically
#then we use ‘peaks’ to define views into the BAM files
bams <- BamViews(bam_file, bamRanges=peaks)
reads <- readGAlignments(bams)
```

Example

```
# Load reads form chr20_bam file
reads <- readGAlignments(chr20_bam)

# Create a `BamViews` object for the range 29805000 - 29820000 on chromosome 20
bam_views <- BamViews(chr20_bam, bamRanges=GRanges("chr20", IRanges(start=29805000, end=29820000)))

# Load only the reads in that view
reads_sub <- readGAlignments(bam_views)

# Inspect the `reads_sub` object
str(reads_sub)

# Load peak calls from chr20_peaks
peaks <- import.bed(chr20_peaks)

# Create a BamViews object
bam_views <- BamViews(chr20_bam, bamRanges=peaks)

# Load the reads
reads <- readGAlignments(bam_views)
```

Visualising individual peaks in their genomic context using Gviz package
organizes data in tracks
each aligned to the same genomic coordinates
this makes it easy to combine data from different sources into a single plot
example Gviz plot



ideogram > tells us what chromosome we are looking at and roughly where our data is located on that chromosome
data track (or coverage track) > read coverage is computed as the number of reads overlapping a given position in the genome
example - a coverage of 5 means that 5 reads (potentially starting at different positions) include this location in their alignment
annotation (peaks/gene regions) track > highlight the location of certain features relative to the read coverage

- above shows one track showing peak calls
- and one visualizing transcript annotations for genes located in this part of the genome

genome axis track > provides more detailed information about the location on the chromosome

Setting-up coordinates

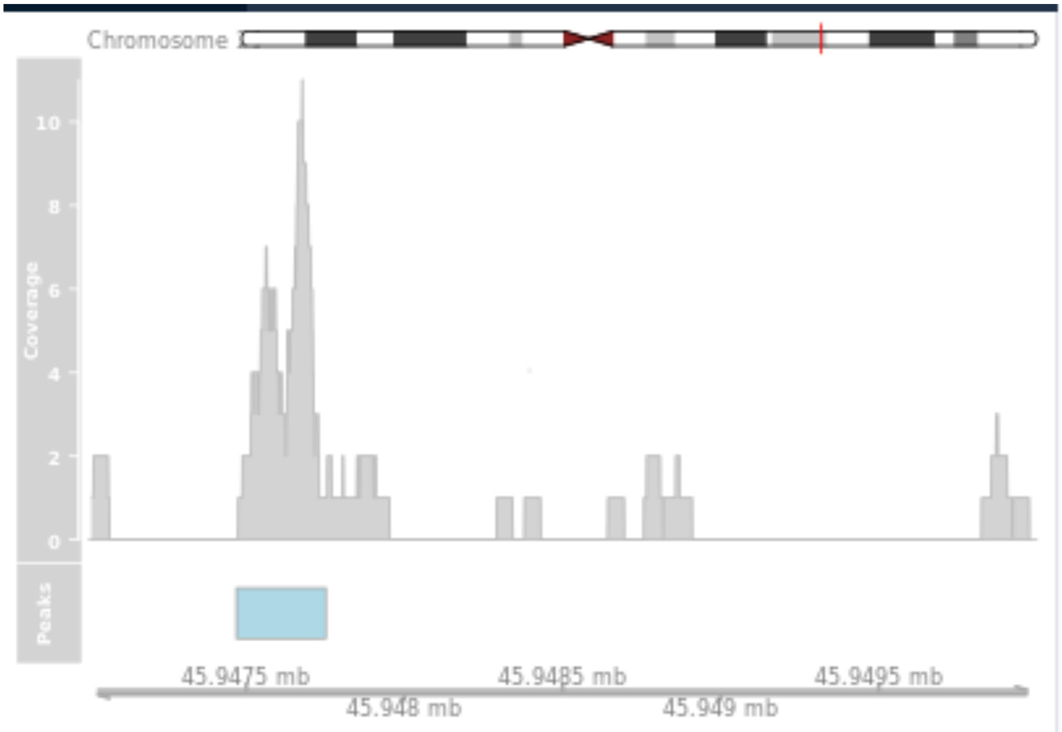
```
library(Gviz)
library(TxDb.Hsapiens.UCSC.hg19.knownGene) #this package allows us to display existing genomic annotations such as gene location
ideogram <- IdeogramTrack(“chr12”, “hg19”) #shows location on the chromosome
cover_track <- DataTrack(cover_ranges, window-100000, type=‘h’, name=“Coverage”)
#cover_ranges needs to be a GRanges object
#’window’ helps us adjust the display (makes it high resolution)
#type ‘h’ creates histogram display
peak_track <- AnnotationTrack(peaks, name=“Peaks”)
#this allows us to display the peak calls
tx <= GeneRegionTrack(TxDb.Hsapiens.UCSC.hg19.knownGene, chromosome=‘chr12’, start=101360000, end=101380000, name=“Genes”)
axis <- GenomeAxisTrack() #shows the coordinates of the plotted region
```

```
plotTracks(list(ideogram, cover_track, peak_track, tx, axis), from=101360000,
to=101380000)
```

Example

```
# Create annotation track
peak_track <- AnnotationTrack(peak_calls, name="Peaks")

# Create data track
cover_track <- DataTrack(cover_ranges, window=10500, type="polygon",
name="Coverage",
                    fill.mountain=c("lighgrey", "lightgrey"), col.mountain="grey")

# Produce plot
plotTracks(list(ideogram, cover_track, peak_track, GenomeAxisTrack()),
chromosome="chr20", from=start_pos, to=end_pos)
```

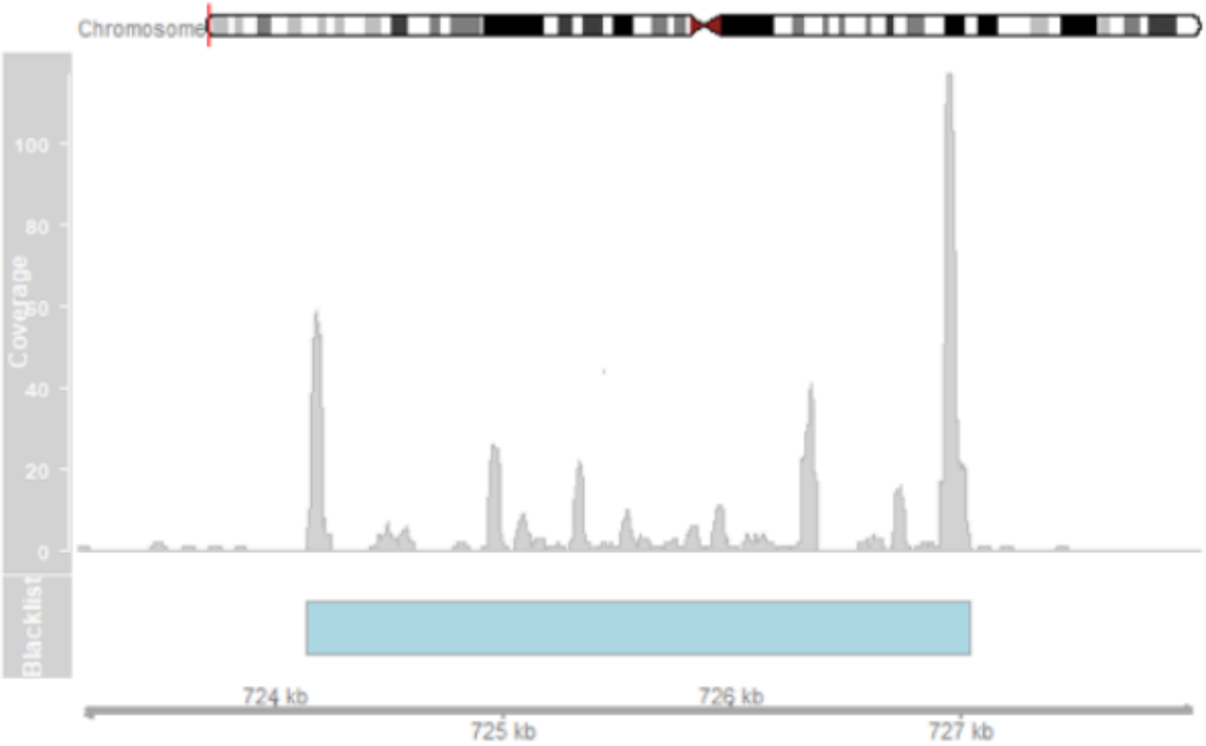
output>



Cleaning ChIP-seq data
need to remove artifacts within your data to reduce noise

Common problems
incorrect mapped reads may produce false peaks
genomic repeats - sequences that occur over and over again
*problematic if the repeats in the sample and the reference do not match up

low complexity regions like the ends of the arms of a chromosome (quality in reference sequences tends to be low in these regions)
*because there is a lot of sequence similarity over extended regions, the origin of reads is difficult to determine
*many regions that tend to accumulate incorrectly mapped reads are known
this is nice because now we can exclude them
example



the red line at the start of the arm of the chromosome that we are looking at marks the region we are looking at
*we have multiple large peaks > we have to be wary > likely chance these are artifacts rather than actual protein binding sites

Amplification bias
arises because of the way DNA extracted from cells is processed prior to sequencing
DNA fragments extracted from cells are copied multiple times prior to sequencing
needed in order to obtain enough DNA for sequencing
however some fragments will produce more copies than others
this means some fragments will produce multiple reads
which can pile up to give the qppearance of a peak in coverage

Quality Control Reports
usefult to obtain summaries of all these potential problems in a systematic way across all samples in a study

ChiPQC is an R package that produces an HTML report in your working directory with standard quality metrics for all samples in your study presented as a series of tables and plots
maps the BAM and BED files to a .csv file with sampleID and other desired columns (ie. condition, tissue, treatment, …)
example

```
library(ChIPQC)
qc_report <- ChIPQC(experiment=“sample_info.csv”, annotatin=“hg19”)
ChIPQCreport(qc_report)
```

How to clean the data
standard practice to group all reads that share the same mapping coordinates and retain only one read alignment per group
this guards against amplification bias
reads that map to more than one location in the genome > may imply incorrect alignment > remove prior to peak calling
reads with low mapping quality > the same, may imply incorrect alignment > remove prior to peak calling
lastly remove peaks in blacklisted regions

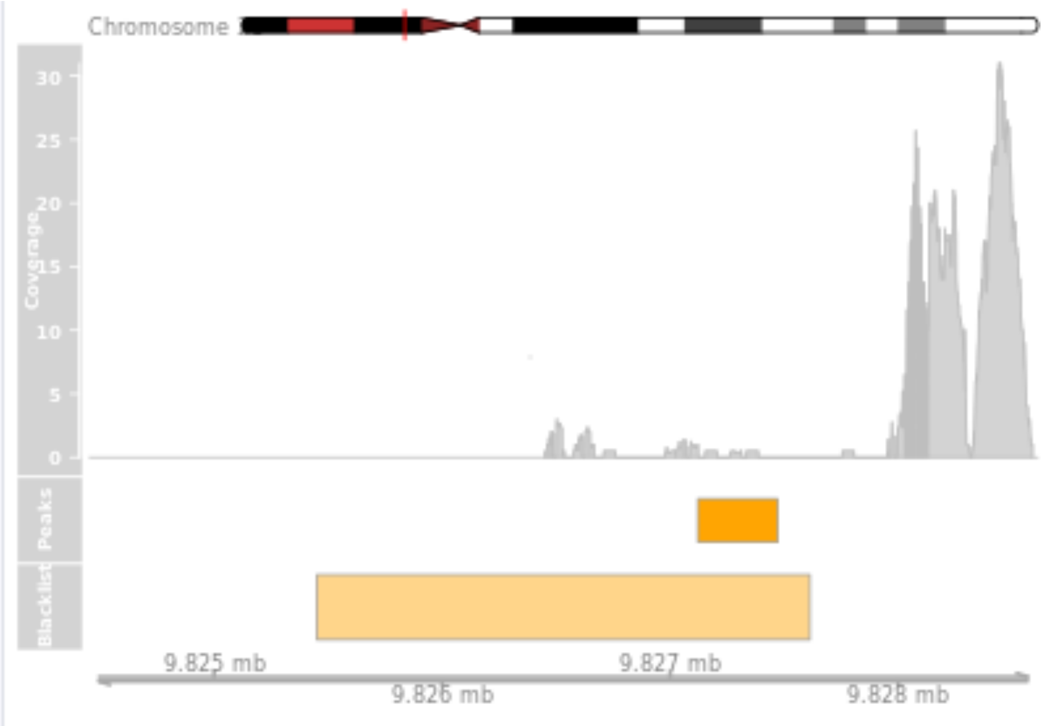
- some peak callers have the ability to do this for you
- can also find a list of these regions created by the ENCODE project

a side > ENCODE aims to create a catologue of functional elements in the human genome

Example

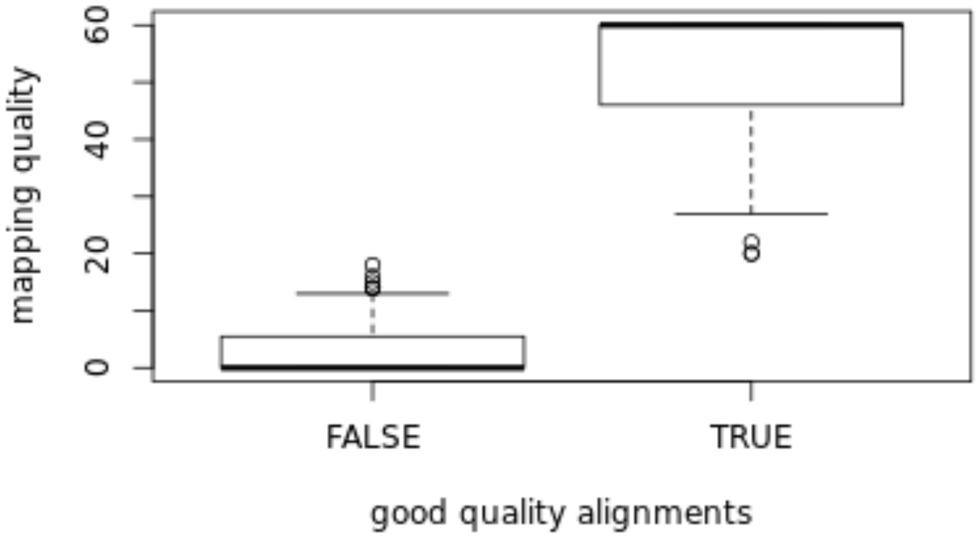
```
# Find all overlaps between peaks and blacklisted regions
blacklisted <- findOverlaps(peaks, blacklist.hg19, type="within")

# Create a plot to display read coverage together with peak calls and blacklisted regions in the selected region
cover_track <- DataTrack(cover, window=10500, type="polygon", name="Coverage",
                    fill.mountain=c("lighgrey", "lightgrey"), col.mountain="grey")

# Calculate peak_track and region_track, plot plotTracks
peak_track <- AnnotationTrack(peaks, name="Peaks", fill="orange")
region_track <- GeneRegionTrack(region, name="Blacklist")
plotTracks(list(ideogram, cover_track, peak_track, region_track, GenomeAxisTrack()),
        chromosome="chr21", from=start(region)-1000, to=end(region)+1000)

# Remove all blacklisted peaks
clean_peaks <- peaks[-from(blacklisted)]
```

output>

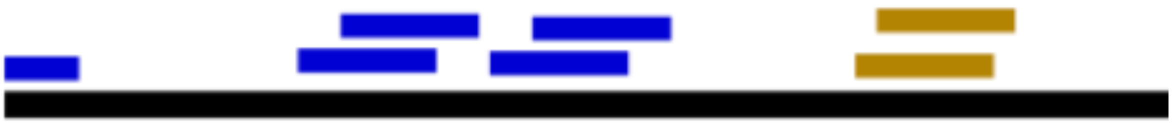


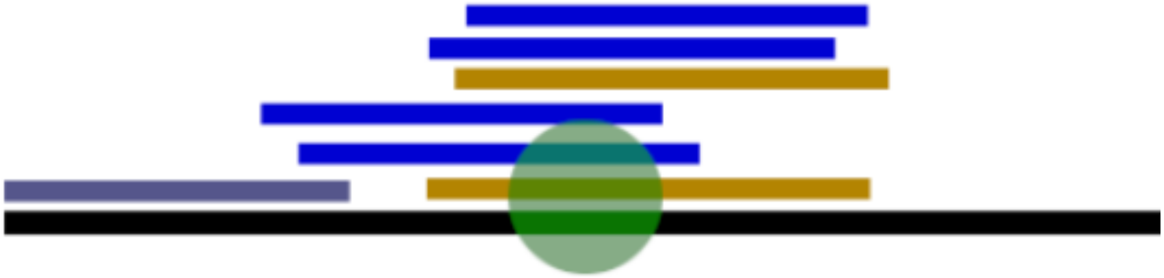
```
# Load reads with mapping qualities by requesting the "mapq" entries
reads <- readGAlignments(bam_file, param=ScanBamParam(what="mapq"))

# Identify good quality alignments
high_mapq <- mcols(reads)$mapq >= 20

boxplot(mcols(reads)$mapq ~ high_mapq, xlab="good quality alignments",
ylab="mapping quality")

# Remove low quality alignments
reads_good <- subset(reads, high_mapq)
```

output>

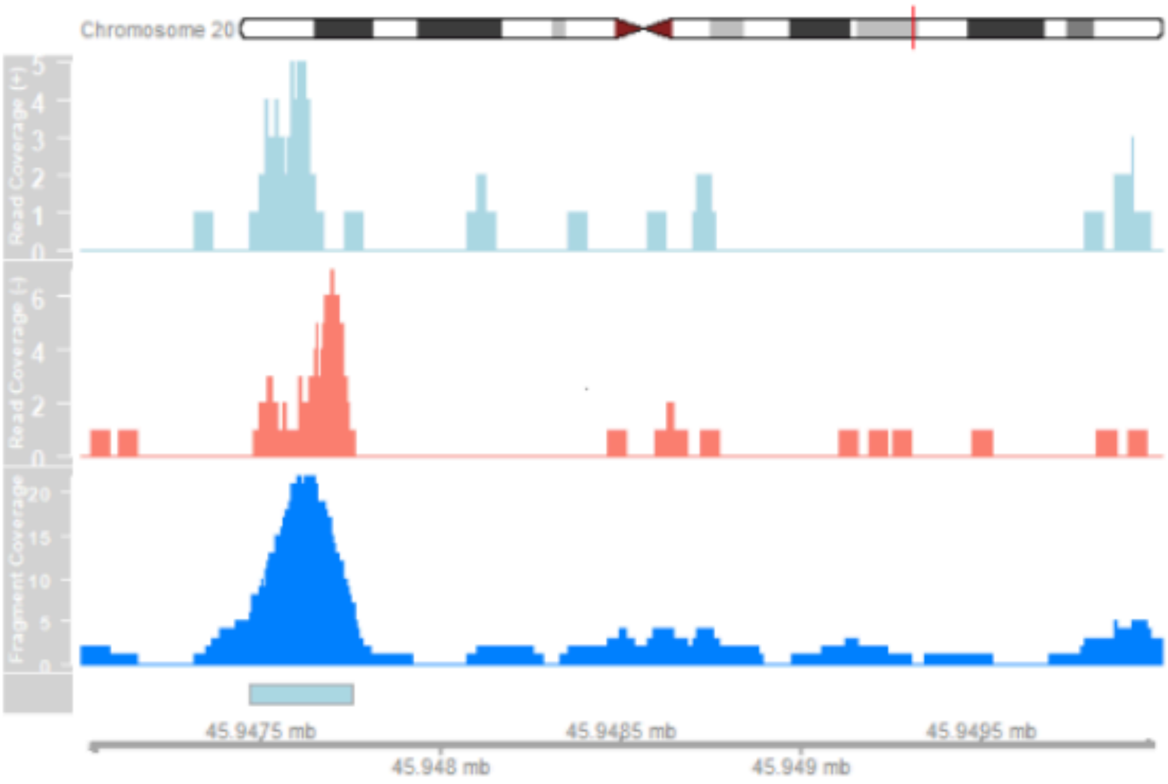


Assessing enrichment
need to extend reads
remember sequencing is generated by DNA fragments that the protein of interest is bound to
*this results in several reads from both ends of the fragment mapping to similar locations in the genome
clustering either sided of the protein binding site
*signal becomes a lot clearer once reads are extended to the length of the full fragment
reads from both ends will now overlap and form a more pronounced peak
visual example - prior to aggregation



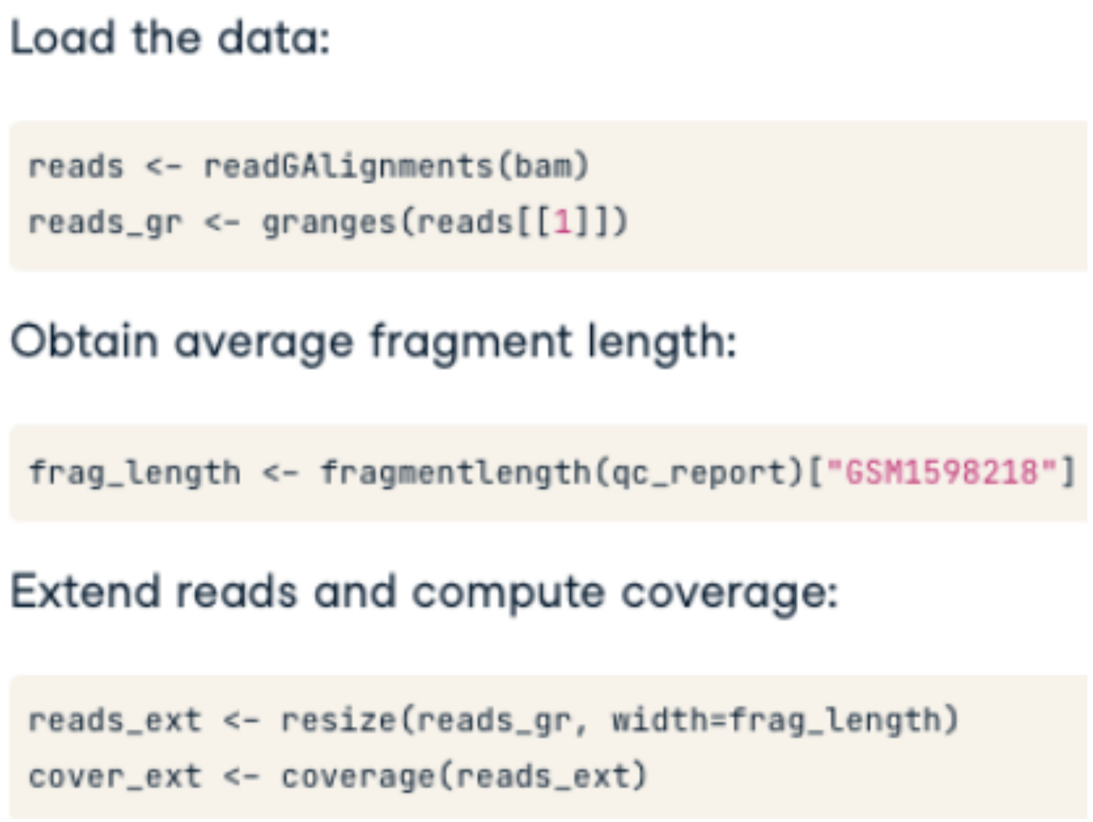


green dot representing new more pronounced peak



further example
top two coverage tracks read coverage for the forward and reverse strand respectively
the third represents total coverage after reads have been extended to the mean fragment length

Extending reads



Load the data:

```
reads <- readGAlignments(bam)
reads_gr <- granges(reads[[1]])
```

Obtain average fragment length:

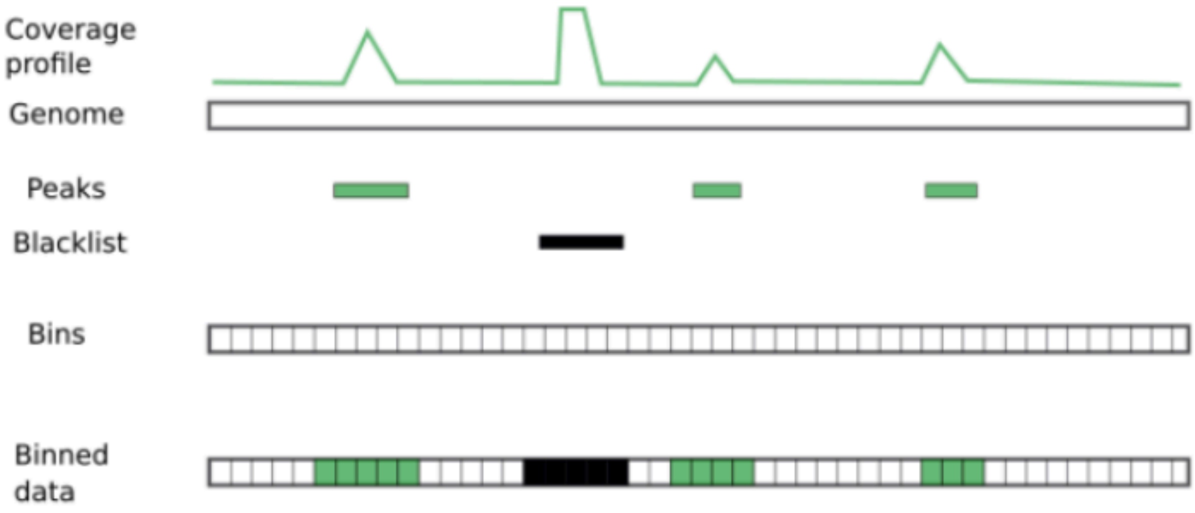
```
frag_length <- fragmentlength(qc_report)["GSM1598218"]
```

Extend reads and compute coverage:

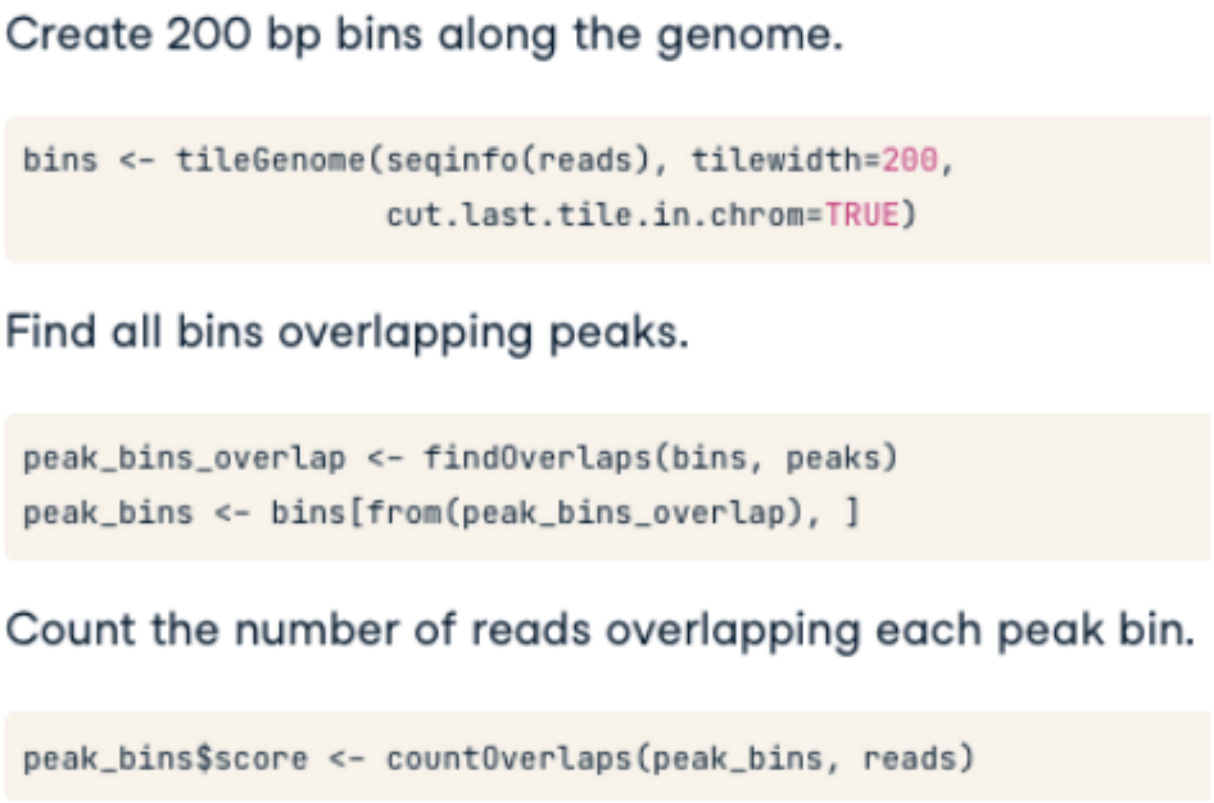
```
reads_ext <- resize(reads_gr, width=frag_length)
cover_ext <- coverage(reads_ext)
```

resize() is from the GenomicRanges package
allows you to specify the desired width of a fragment via the 'width' argument

Next enrichment
how does coverage in peaks compare to coverage in other parts of the genome?
need to partition the genome into short intervals
example into 200 base pair long intervals
then assign each bin either to a peak, a blacklisted region, or background



with R:



Create 200 bp bins along the genome.

```
bins <- tileGenome(seqinfo(reads), tilewidth=200,
                   cut.last.tile.in.chrom=TRUE)
```

Find all bins overlapping peaks.

```
peak_bins_overlap <- findOverlaps(bins, peaks)
peak_bins <- bins[from(peak_bins_overlap), ]
```

Count the number of reads overlapping each peak bin.

```
peak_bins$score <- countOverlaps(peak_bins, reads)
```

countOverlaps() counts the number of fragments in each of the selected bins

*wrapping the above code for convenience

```
count_bins <- function(reads, target, bins){
        overlap <- from(findOverlaps(bins, targe))
        target_bins <- bins[overlap, ]
        target_bins$score <- contOverlaps(target_bins, reads)
        target_bins
}
```

Coverage for the blacklisted regions

```
peak_bins <- count_bins(reads_ext, peaks, bins)
bl_bins <- count_bins(reads_ext, blacklist.hg19, bins)
```

Background coverage

measure background coverage is to consider the coverage for all remaining bins after peaks and blacklists are removed

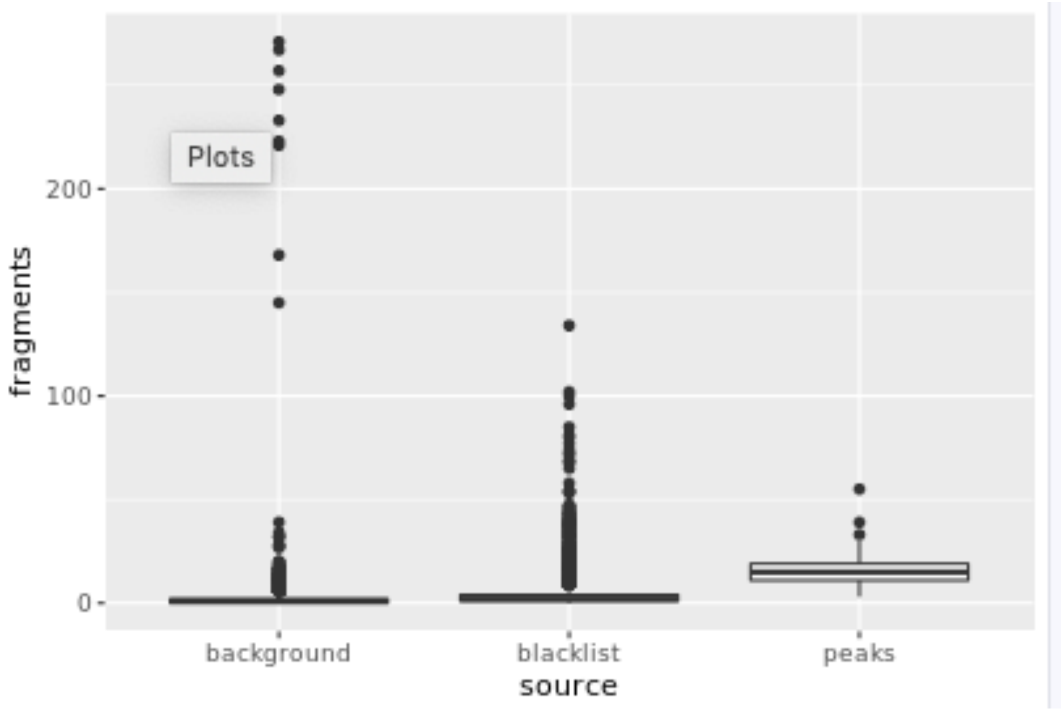
we can do this by subsetting

```
#remove all bins already accounted for
bkg_bins <- subset(bins, !bins %in% peak_bins & !bins %in% bl_bins)
#count number of reads overlapping with each remaining bin
bkg_bins$score <- countOverlaps(bkg_bins, reads_ext)
```

Example

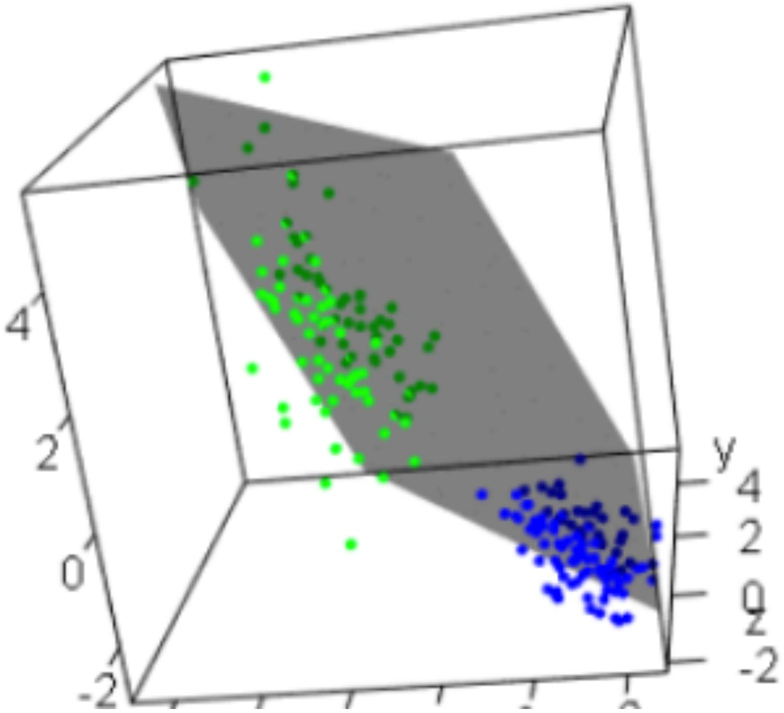
```
# Extend reads to the average fragment length of 183 bp
reads_ext <- resize(reads_gr, width=183)

# Compute coverage
cover <- coverage(reads_ext)

# Prepare read counts for plotting by organising them in data frames
peak_scores <- data.frame(source="peaks", fragments=peak_bins$score)
bl_scores <- data.frame(source="blacklist", fragments=bl_bins$score)
bkg_scores <- data.frame(source="background", fragments=bkg_bins$score)
scores <- rbind(peak_scores, bl_scores, bkg_scores)

# Create a boxplot of the read counts by bin type
ggplot(scores, aes(y=fragments, x=source)) + geom_boxplot()
```

output>

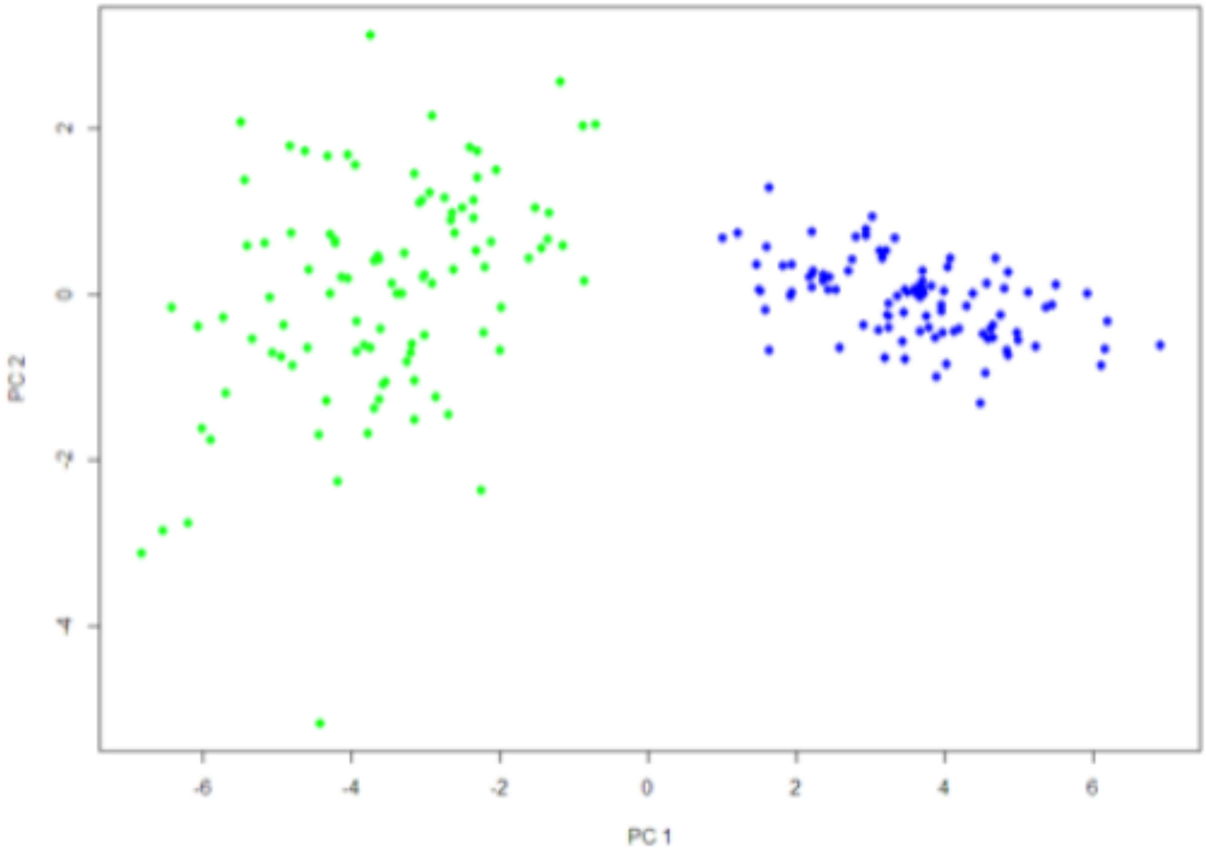


Intro to differential binding
our example > examiing primary vs treatment resistant prostate tumors
goal is to identify molecular mechanisms that cause this difference in response
are samples from the same group generally similar?
are samples from different groups different?
what are the differences?

PCA is one method we can use to answer these questions
PCA is a method used to uncover some of the underlying structure within a dataset
PCA identifies the directions (or principle components) with the most variation between data points
using the first two principle components we can define a plane that passes through the cloud of data points
goal is to minimize the overall distance between points and the plane as much as possible
visualizing:



by rotating the data we can get a view that highlights the main differences between data points
we project this view onto a 2D scatter plot (PCA plot)



we can create this with the ChIPQC package

```
qc_result <- ChIPQC(“sample.csv”, “hg19”)
```

*need to create a consistent set of peaks across all samples for this to work

```
#from the DiffBind package we use dba.count() which will provide us with a suitable set of concensus peaks
#’summits’ argument determines the width of the resulting peaks
counts <- dba.count(qc_results, summits=250)
plotPrincomp(counts)
```

Another option - hierarchical clustering

clustering is based on the observed read counts for each peak

uses the pairwise distances between samples to build a tree

compute this way:

```
distance <- dist(t(coverage))
#computes the distance between the rows of a matrix
#t is for transpose and this will give you the distance between samples
#hclust will create a dendrogram
dendro <- hclust(distance)
#plot dendrogram
plot(dendro)
```

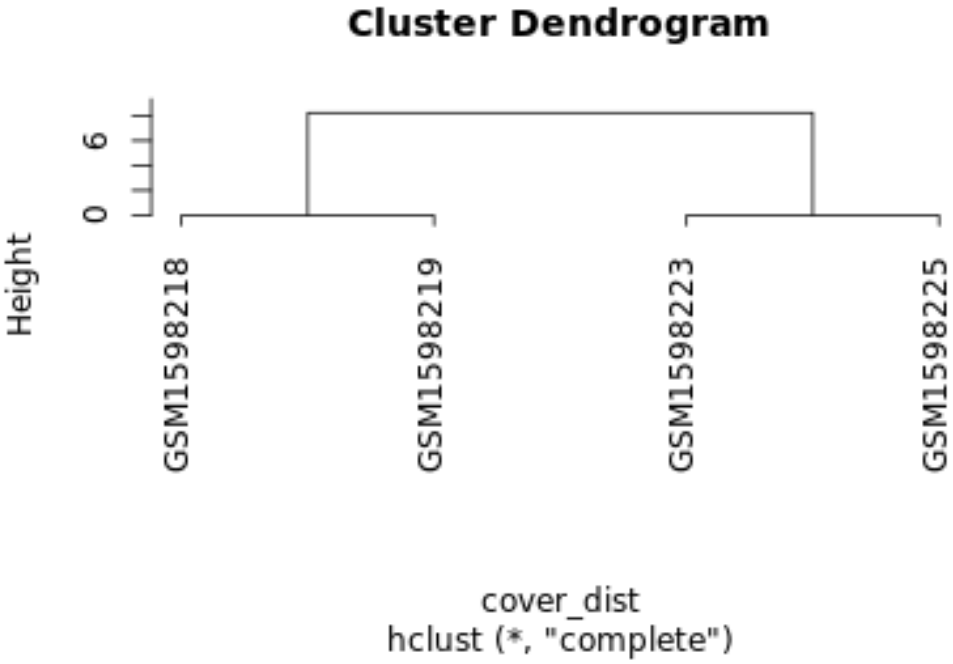
Employ this clustering with a heatmap
DiffBind package allows us to facilitate this

```
dba.plotHeatmap(peaks, maxSites = peak_count, correlations = FALSE)
#arguments ensure all peaks are plotted instead of correlations between samples
```

Example

```
# Compute the pairwise distances between samples using `dist`
cover_dist <- dist(t(cover))

# Use `hclust()` to create a dendrogram from the distance matrix
cover_dendro <- hclust(cover_dist)

# Plot the dendrogram
plot(cover_dendro)
```

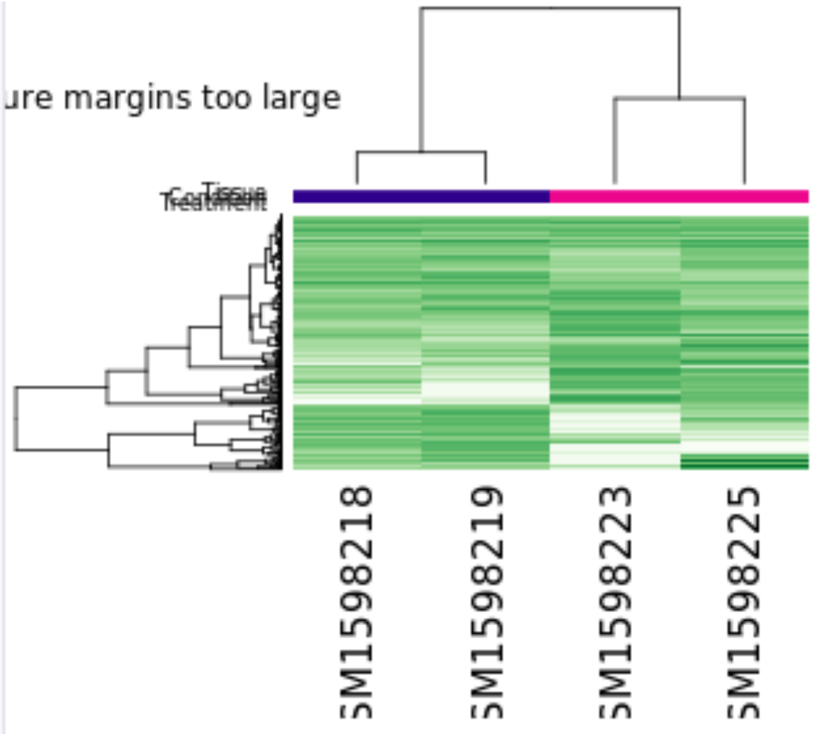
output>



```
# Print the `peaks` object
print(peaks)

# Obtain the coordinates of the merged peaks
merged_peaks <- peaks$merged

# Extract the number of peaks present in the data
peak_count <- nrow(merged_peaks)
```

```
# Create a heatmap using the `dba.plotHeatmap()` function
dba.plotHeatmap(peaks, maxSites = peak_count, correlations = FALSE)
```

ouput>



Testing for differential binding
do statistical analysis with DiffBind package
which interfaces to use other tools such as DESeq2 or edgeR

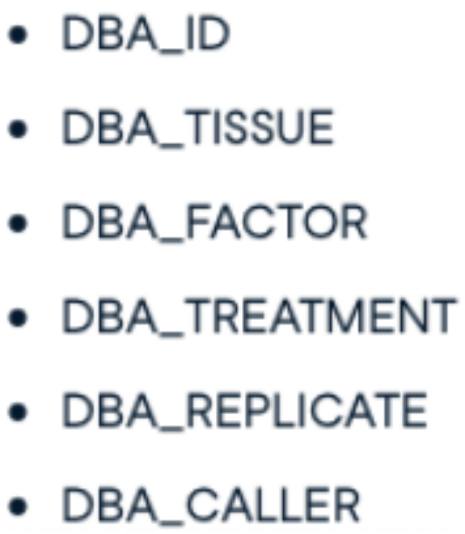
Start with creating a consistent shared peak set
and then counting reads in peak set
can use ChIPQC output as input to the analysis
with R:

```
peak_counts <- dba.count(qc_output, summits=250)
#summits argument signals that peaks should be re-centered around the consensus peak
#250 represents the width of the resulting peak on either side
#this will give us 500 base pair wide peaks
```

to run the analysis we need to tell DiffBind how the samples should be split into groups

```
peak_counts <- dba.contrast(peak_counts, categories = DBA_CONDITION)
```

addition supported categories:



- DBA_ID
- DBA_TISSUE
- DBA_FACTOR
- DBA_TREATMENT
- DBA_REPLICATE
- DBA_CALLER

*Background coverage can easily be mistaken for peaks
*control samples can be used to assess background coverage in the absence of ChIP signal to filter out this noise

Running the analysis

```
bind_diff <- dba.analyze(peak_counts)
```

Looking at the results

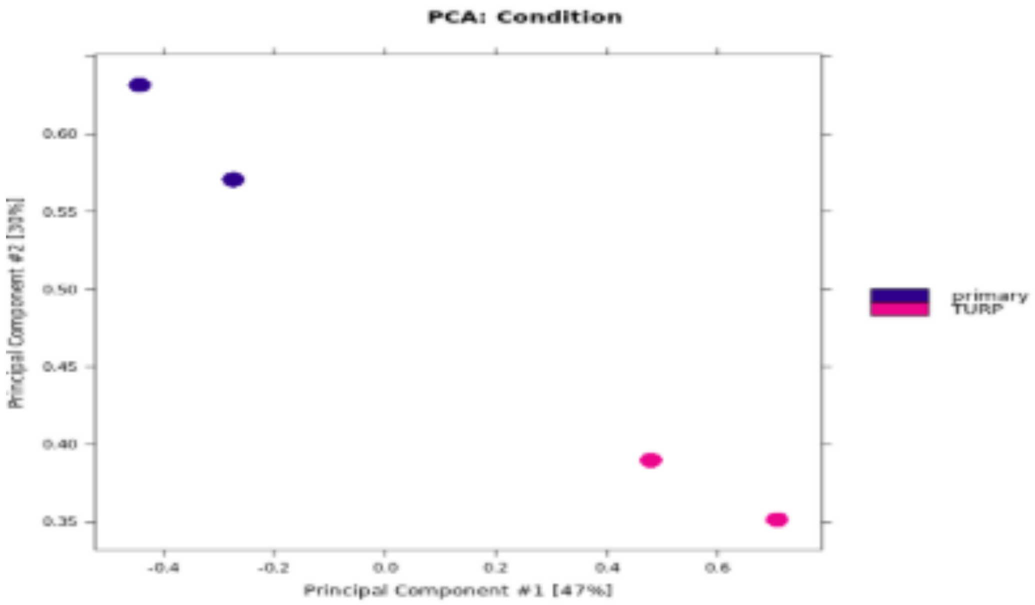
```
dba.plotPCA(bind_diff, DBA_Condition, contrast=1)
dba.plotHeatmap(bind_diff, DBA_Condition, contrast=1)
```

these will give you some sense of the extent to which the two groups differ in their binding patterns

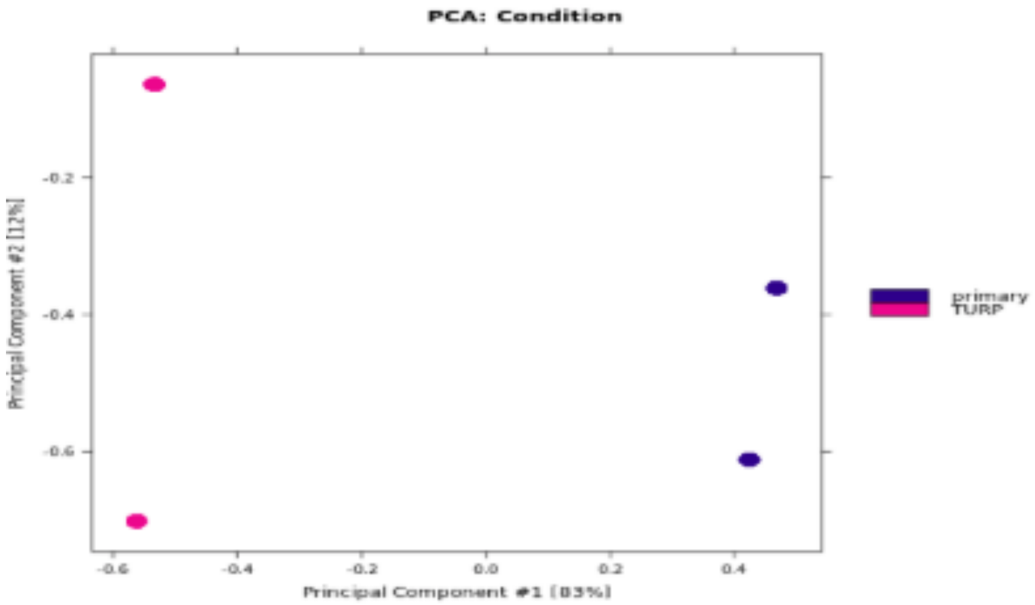
Example

```
# Examine the ar_binding object
print(ar_binding)

# Identify the category corresponding to the tumor type contrast
contrast <- DBA_CONDITION

# Establish the contrast to compare the two tumor types
dba_peaks <- dba.contrast(ar_binding, categories=contrast, minMembers=2)

# Examine the dba_peaks object to confirm that the contrast has been added
print(dba_peaks)

# Examine the `ar_binding` object to confirm that it contains the required contrast
print(ar_binding)
```

```
# Run the differential binding analysis
ar_diff <- dba.analyze(ar_binding)

# Examine the result
print(ar_diff)

# Create a PCA plot using all peaks
dba.plotPCA(ar_diff, DBA_CONDITION)

output>
```

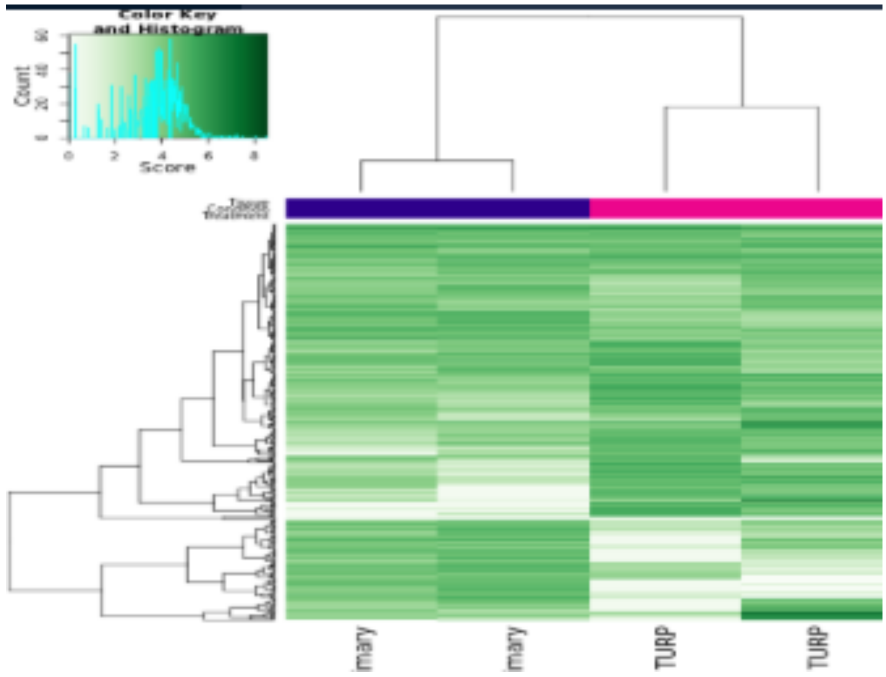


```
# Create a PCA plot using only differentially bound peaks
dba.plotPCA(ar_diff, DBA_CONDITION, contrast = 1)

output>
```



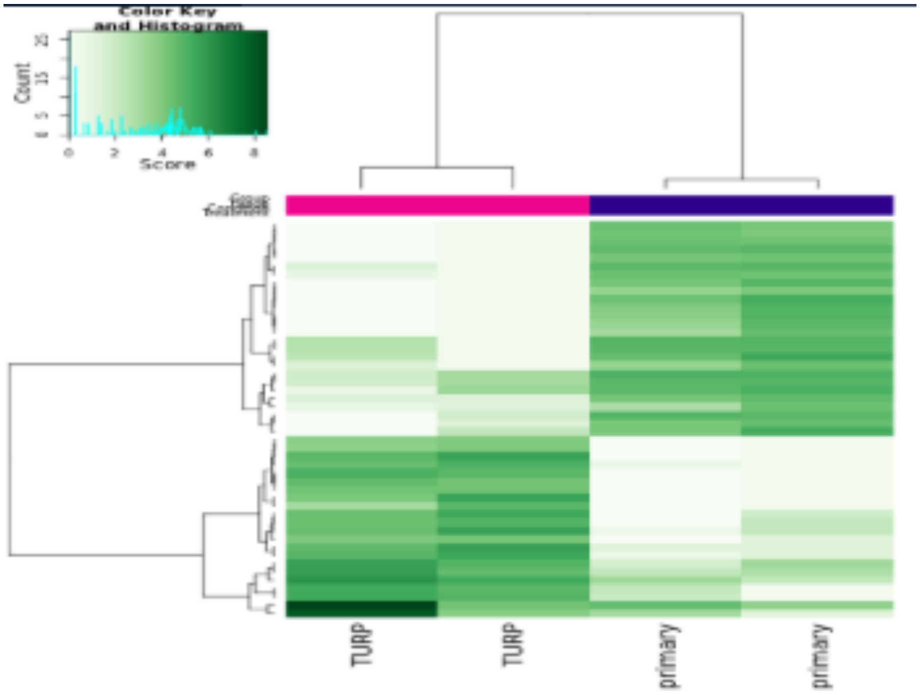
# Create a PCA plot using only differentially bound peaks
dba.plotPCA(ar_diff, DBA_CONDITION, contrast = 1)

output>



# Create a heatmap using only differentially bound peaks
dba.plotHeatmap(ar_diff, DBA_CONDITION, contrast=1, correlations = FALSE)

output>



Further visualizing the results

first we’ll look at MA plots

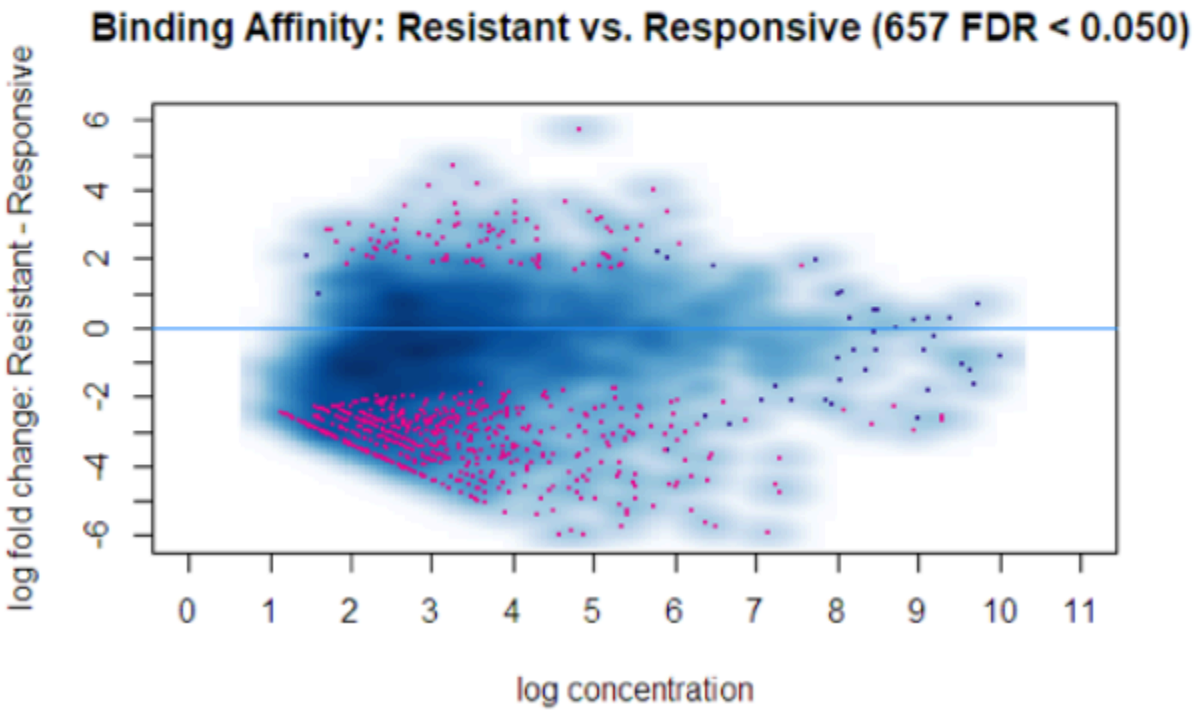
visualises the relationship between change in peak intensity between conditions and average peak intensity

can do this with the DiffBind package as well

example

```
dba.plotMA(dba_object)
```

output>

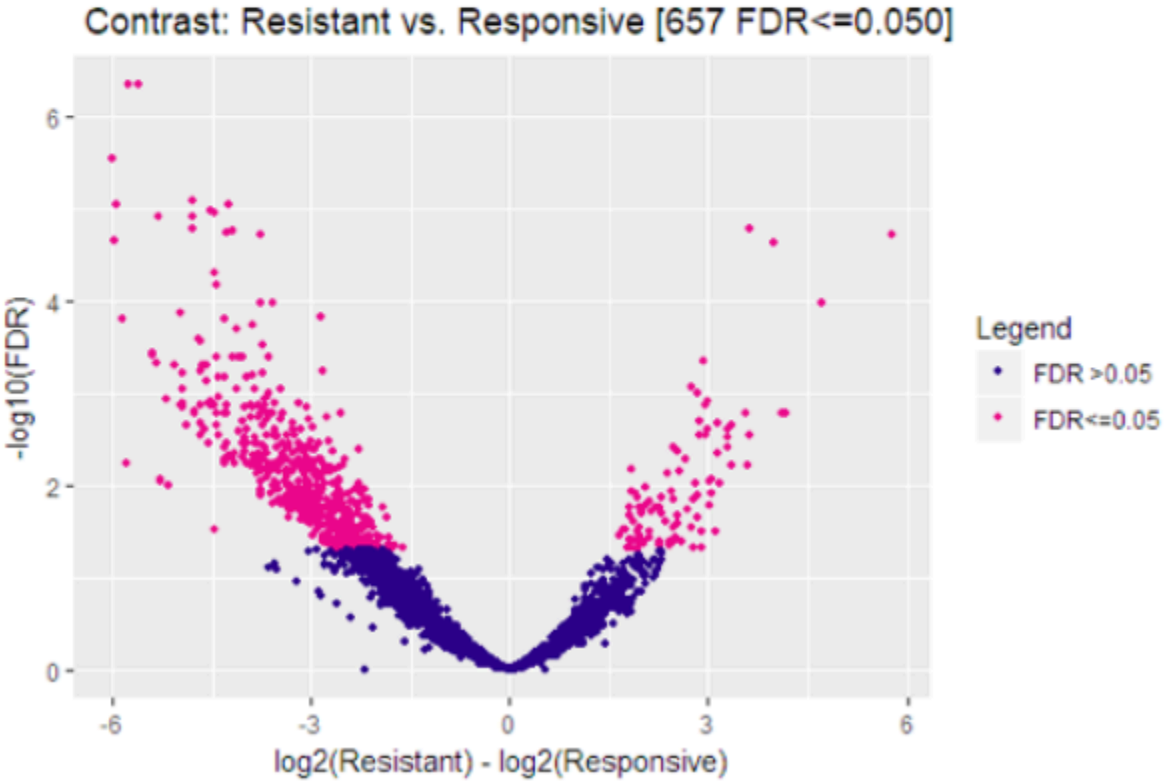


shows the log peak intensity on the x-axis
log fold change on the y-axis
above - most data points have been smoothed into a density cloud (this is done to make it easier to see concentration)
points corresponding to differentially bound peaks are highlighted in pink
always make sure that proper normalization has been performed

recap on differential binding
refers to the identification and comparison of differenecs in the binding patterns of a protein (often a transcription factor) between two or more conditions or experimental groups
recap on the process again

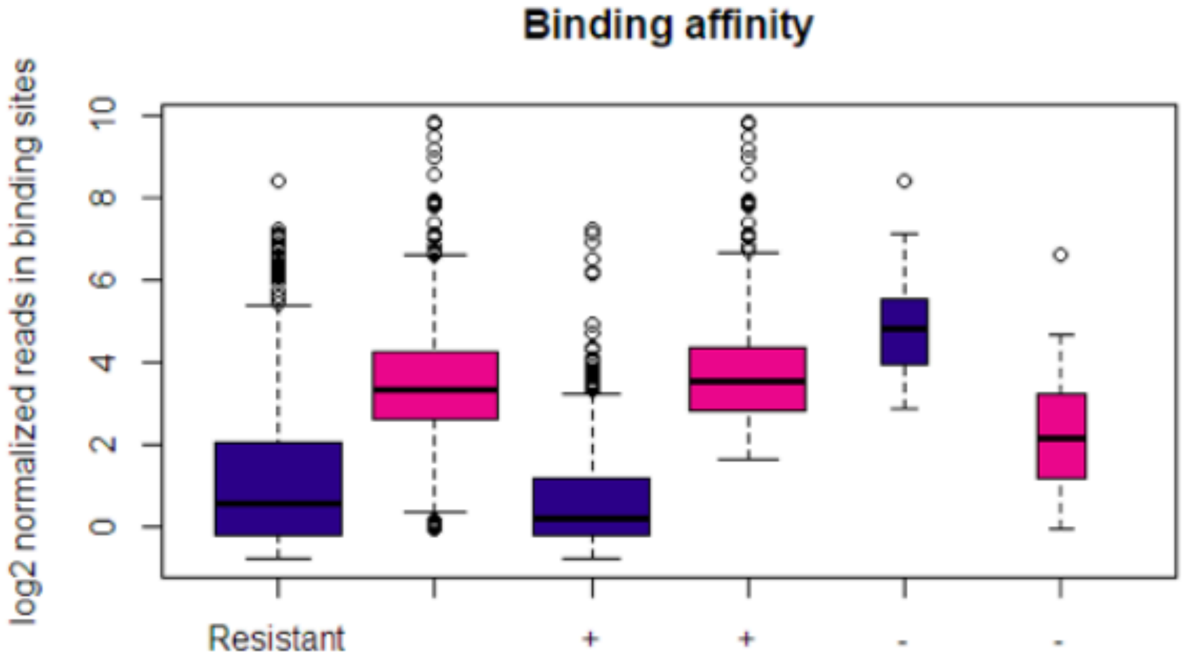
1. ChIP-seq experiemnt
2. sequencing
3. read alignment to a reference genome
4. peak calling - regions of the genome with a high density of aligned reads
5. differential binding analysis - identify genomic regions where the binding of the protein significantly differs between conditions
6. statistical testing - assess significance of peaks vs random chance
7. visualization

Additional plots
dba.plotVolcano(dba_object)
output>



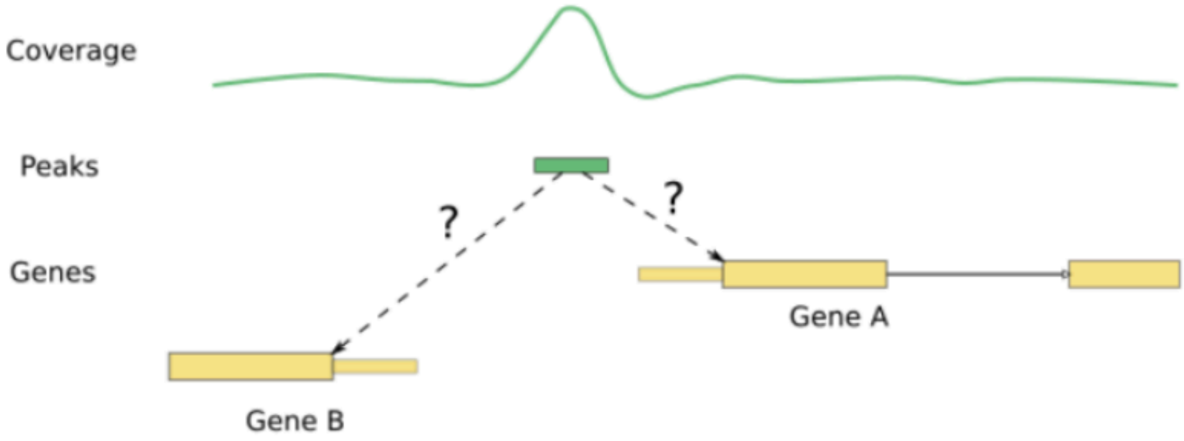
FDR stands for false discovery rate
volcano plot plots negative log p-values (or false discovery rates) as a function of log fold change
peaks with significant evidence for differential binding are highlighted in pink
volcanos are useful because they display the significance of the change in peak intensity together with the magnitude of the change

```
dba.plotBox(dba_object)
output>
```



+ indicates sites with increased affinity in Responsive
- indicates sites with increased affinity in Resistant

Interpreting peaks
we are really interested in gene regulation
we are attempting to identify genes that are regulated by the binding of certain transcription factors
commonly we assign the peak to the closest gene
to visualize:



to do this we need to obtain info about gene locations
then we can assign peaks to the closest genes
then we can create lists of genes with changes in protein binding between the two groups

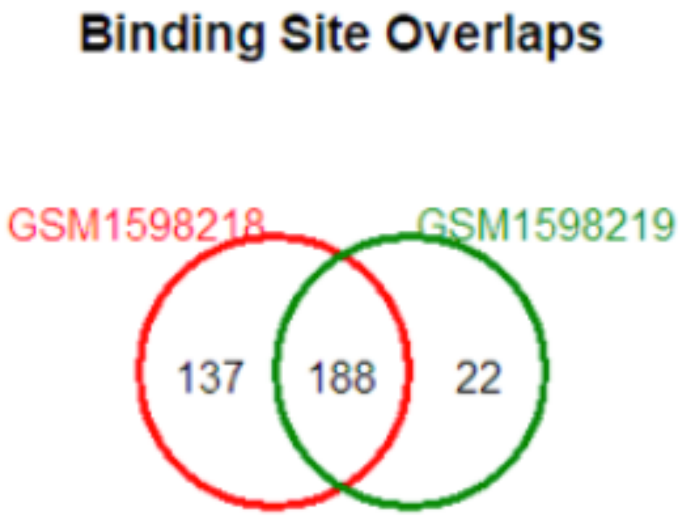
Transcript annotations
’TxDb’ packages provide info about the location of all known transcripts and genes in a given genome
remember Entrez IDs > unique gene identifiers

Annotating peaks
annotates peaks with their closest gene
example

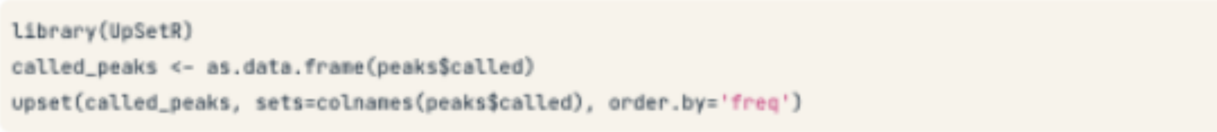
```
library(ChIPpeakAnno)
annoPeaks(peaks, human_genes, bindingType=‘startSite’,
bindingRegion=c(-5000,5000))
#this requires two GRanges objects (peaks and human_genes for this example)
#one has peak coordinates
#the other has annotations
#bindingType gives instructions for how to match these two GRange objects
#bindingRegion for this example requires peaks to be within 5 kilo bases of the
transcription start site
```

Visualize similarites and differences
can start with a Venn

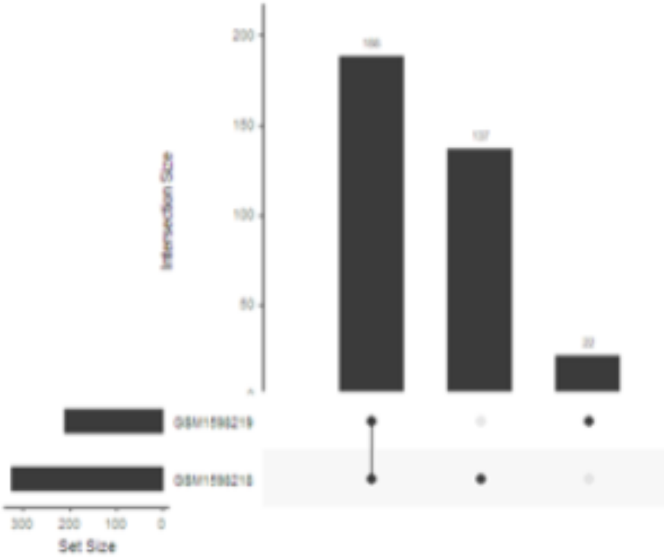
```
dba.plotVenn(peaks, mask=1:2)
```



for larger samples UpSet plots are better



```
library(UpSetR)
called_peaks <- as.data.frame(peaks$called)
upset(called_peaks, sets=colnames(peaks$called), order.by='freq')
```



Example
# Extract peaks from ChIPQCexperiment object
peak_calls <- peaks(ar_calls)

# Only keep samples that passed QC
peak_passed <- peak_calls[qc_pass]

# Find overlaps between peak sets
peaks_combined <- findOverlapsOfPeaks(peak_passed[[1]], peak_passed[[2]], peak_passed[[3]], peak_passed[[4]], maxgap=50)

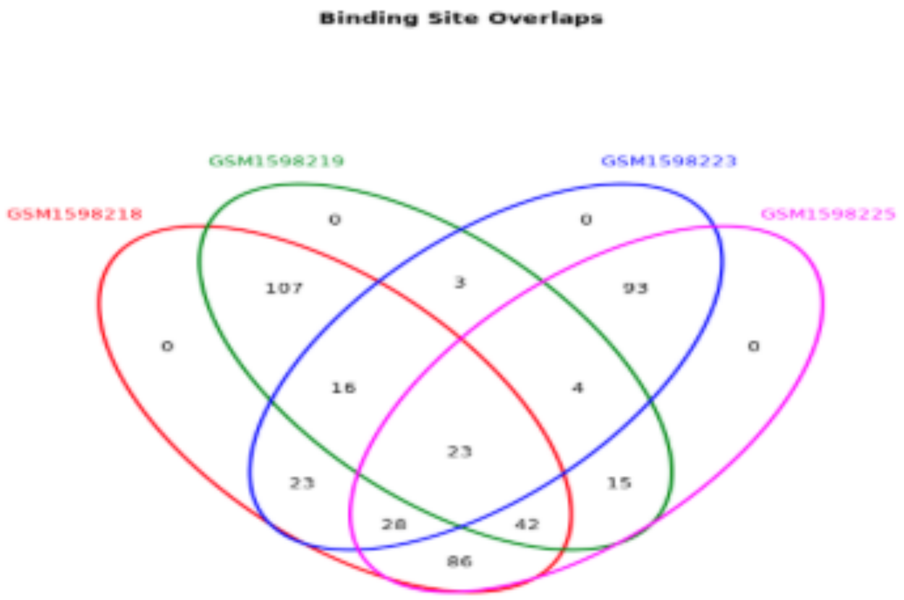
# Examine merged peak set
print(peaks_combined)

# Obtain gene symbols
gene_symbol <- select(org.Hs.eg.db, keys=human_genes$gene_id, columns="SYMBOL", keytype="ENTREZID")

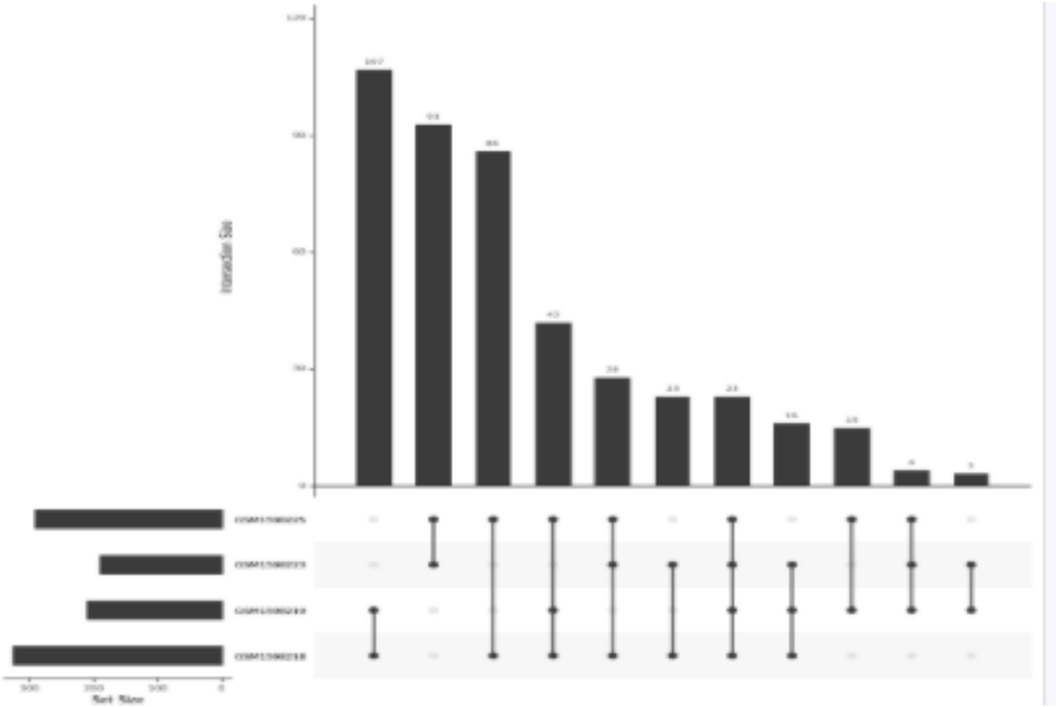
# Examine the structure of the returned annotations
str(gene_symbol)

# Add gene symbols to gene coordinates
human_genes$symbol <- gene_symbol$SYMBOL

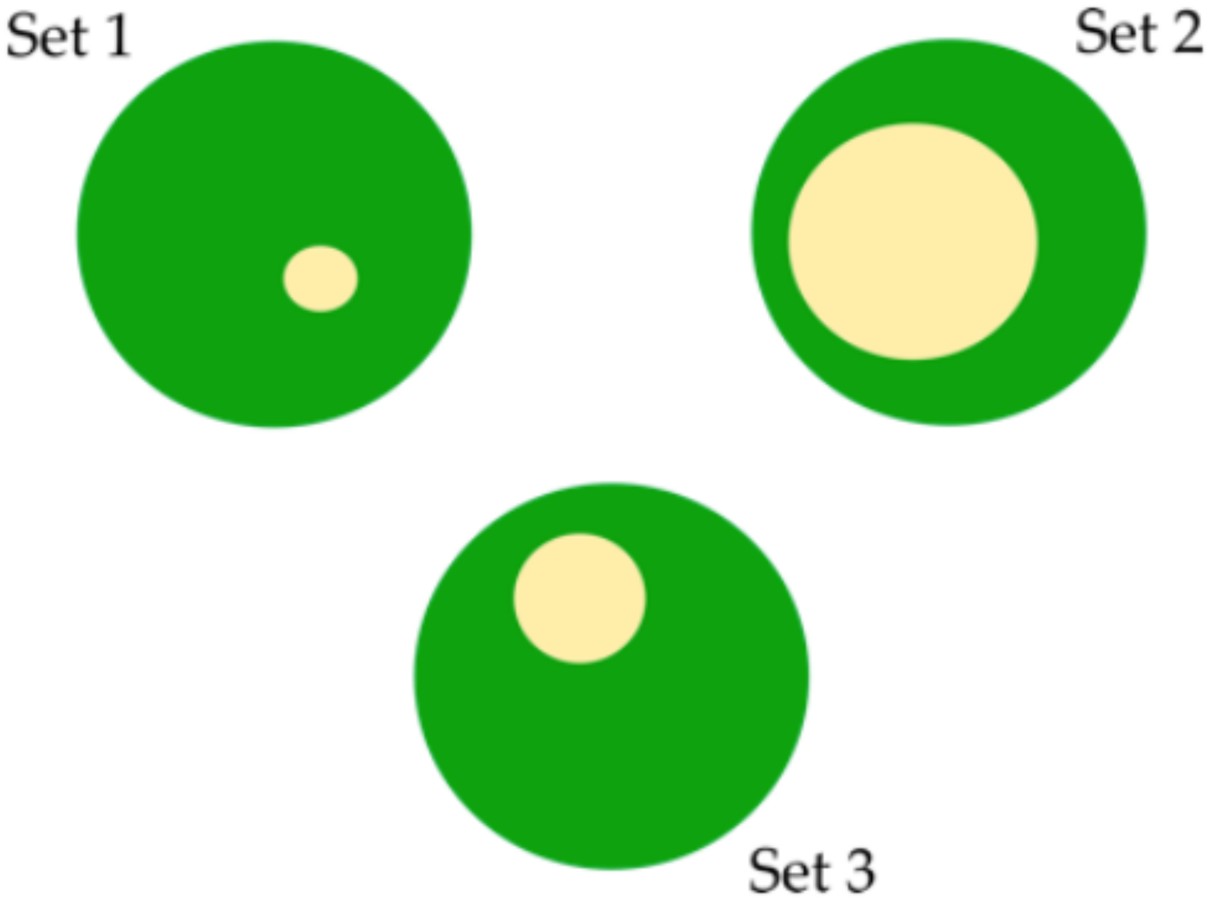
```
# Examine output
print(human_genes)

# Annotate peaks with closest gene
peak_anno <- annoPeaks(peaks_merged, human_genes, bindingType="startSite",
bindingRegion=c(-5000,5000))

# How many peaks were found close to genes?
length(peak_anno)

# Where are peaks located relative to genes?
table(peak_anno$insideFeature)

# Create Venn diagram
dba.plotVenn(ar_diff, mask=1:4)

output>
```



```
# Convert the matrix of called peaks into a data frame
called_peaks <- as.data.frame(ar_diff$called)

# Create UpSet plot
upset(called_peaks, keep.order = TRUE, sets=colnames(ar_diff$called),
order.by="freq")

ouput>
```



Interpreting affected gene lists
what are these genes doing?
gene set enrichment approach
this involves defining groups of genes that are related by their function in some way
visualize:



large proportion of peak associated genes like set 2 are likely to be of relevance

finding enriched gene sets

```
library(chipenrich)
chipenrich(peaks, genome=’hg19’, genesets=‘hallmark’, locusdef=‘nearest_tss’)
```

this package allows you to provide peak locations directly without having to annotate first

‘genome’ argument indicates reference genome

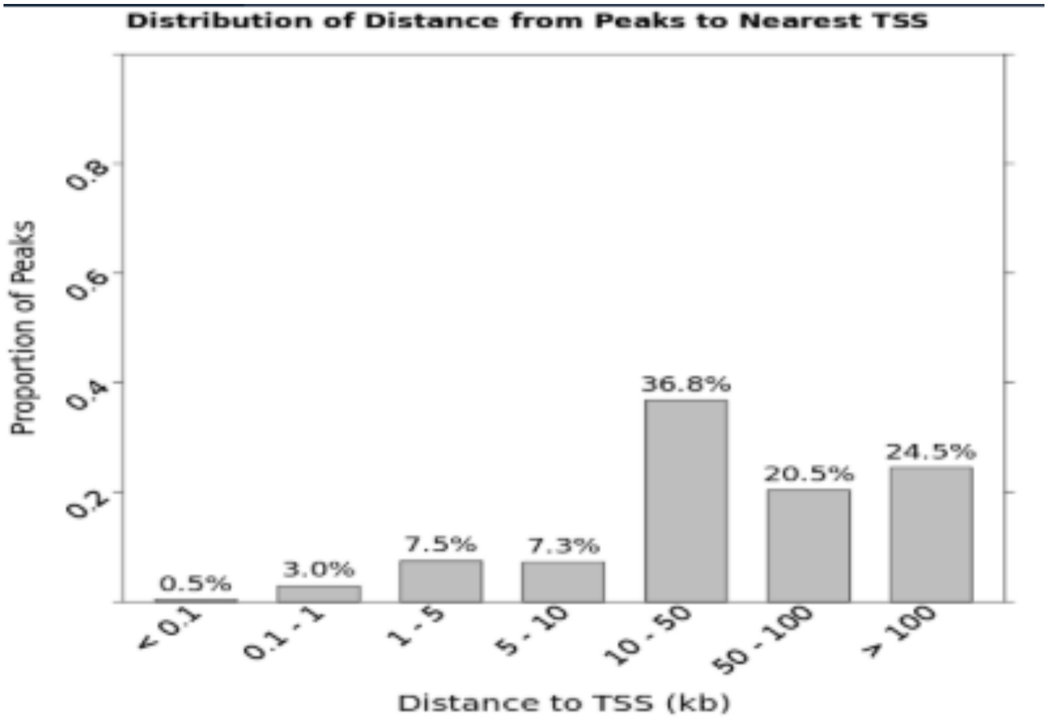
‘genesets’ selects one of several supported genesets

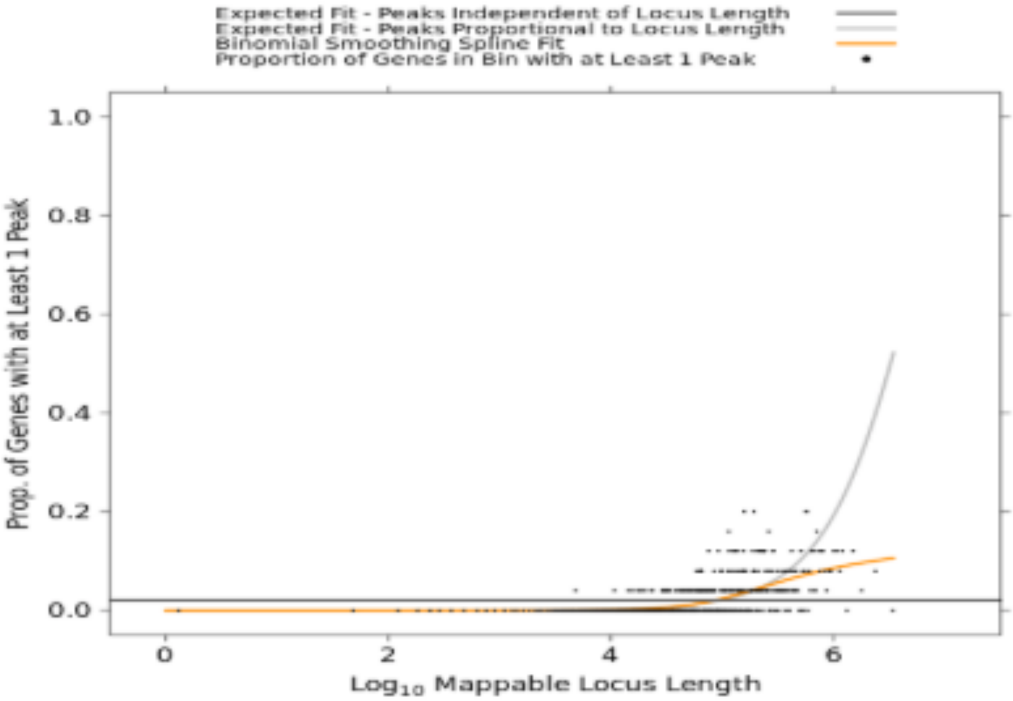
‘locusdef’ determines how peaks hould be associated with genes #here we use closest transcription start site

Example

```
# Plot distribution of distances between peaks and transcription start sites
plot_dist_to_tss(peaks, genome = "hg19")

# Plot relationship between gene length and presence of peaks
plot_chipenrich_spline(peaks, genome = "hg19", mappability=50)
```

output>





Break down of each plot:

1. **Plot distribution of distances between peaks and transcription start sites:**
   - This plot visualizes the distribution of distances between ChIP-seq peaks (likely regions of interest in the genome, such as regions where a protein binds) and transcription start sites (TSS) of genes. The x-axis represents the distances

between the peaks and TSS, while the y-axis shows the frequency of occurrences for each distance range. This plot provides insights into the genomic locations of the identified peaks relative to gene transcription start sites.

2. **Plot relationship between gene length and presence of peaks:**
   - This plot explores the relationship between the length of genes and the presence of ChIP-seq peaks. It likely uses a spline curve to depict the trend. The x-axis represents gene lengths, and the y-axis shows the presence or absence of ChIP-seq peaks. The curve's shape indicates whether there's a correlation between gene length and the likelihood of having peaks. This type of analysis helps identify potential associations between gene characteristics (like length) and the binding patterns observed in ChIP-seq experiments.

Example cont'd

```
# Select all peaks with higher intensity in treatment resistant samples
turp_peaks <- peaks_binding[, "GSM1598218"] + peaks_binding[, "GSM1598219"]
< peaks_binding[, "GSM1598223"] + peaks_binding[, "GSM1598225"]

# Run enrichment analysis
enrich_turp <- chipenrich(peaks_comb[turp_peaks, ], genome="hg19",
               genesets = "hallmark", out_name = NULL,
               locusdef = "nearest_tss", qc_plots=FALSE)

# Print the results of the analysis
print(enrich_turp$results)
```

output>

```
      Geneset.Type Geneset.ID                      Description
1  Hallmark (MSigDB)      M5957             HALLMARK_PANCREAS_BETA_CELLS
2  Hallmark (MSigDB)      M5916                 HALLMARK_APICAL_SURFACE
3  Hallmark (MSigDB)      M5950             HALLMARK_ALLOGRAFT_REJECTION
4  Hallmark (MSigDB)      M5921                     HALLMARK_COMPLEMENT
5  Hallmark (MSigDB)      M5915                 HALLMARK_APICAL_JUNCTION
6  Hallmark (MSigDB)      M5908               HALLMARK_ANDROGEN_RESPONSE
7  Hallmark (MSigDB)      M5946                    HALLMARK_COAGULATION
8  Hallmark (MSigDB)      M5944                    HALLMARK_ANGIOGENESIS
9  Hallmark (MSigDB)      M5902                      HALLMARK_APOPTOSIS
10 Hallmark (MSigDB)      M5934            HALLMARK_XENOBIOTIC_METABOLISM
11 Hallmark (MSigDB)      M5892
HALLMARK_CHOLESTEROL_HOMEOSTASIS
12 Hallmark (MSigDB)      M5924                HALLMARK_MTORC1_SIGNALING
13 Hallmark (MSigDB)      M5913
```

			HALLMARK_INTERFERON_GAMMA_RESPONSE
14	Hallmark (MSigDB)	M5907	HALLMARK_ESTROGEN_RESPONSE_LATE
15	Hallmark (MSigDB)	M5911	HALLMARK_INTERFERON_ALPHA_RESPONSE
16	Hallmark (MSigDB)	M5901	HALLMARK_G2M_CHECKPOINT
17	Hallmark (MSigDB)	M5937	HALLMARK_GLYCOLYSIS
18	Hallmark (MSigDB)	M5906	HALLMARK_ESTROGEN_RESPONSE_EARLY
19	Hallmark (MSigDB)	M5891	HALLMARK_HYPOXIA
20	Hallmark (MSigDB)	M5923	HALLMARK_PI3K_AKT_MTOR_SIGNALING
21	Hallmark (MSigDB)	M5926	HALLMARK_MYC_TARGETS_V1
22	Hallmark (MSigDB)	M5935	HALLMARK_FATTY_ACID_METABOLISM
23	Hallmark (MSigDB)	M5941	HALLMARK_UV_RESPONSE_UP
24	Hallmark (MSigDB)	M5925	HALLMARK_E2F_TARGETS
25	Hallmark (MSigDB)	M5939	HALLMARK_P53_PATHWAY
26	Hallmark (MSigDB)	M5930	HALLMARK_EPITHELIAL_MESENCHYMAL_TRANSITION
27	Hallmark (MSigDB)	M5953	HALLMARK_KRAS_SIGNALING_UP
28	Hallmark (MSigDB)	M5893	HALLMARK_MITOTIC_SPINDLE
29	Hallmark (MSigDB)	M5956	HALLMARK_KRAS_SIGNALING_DN
30	Hallmark (MSigDB)	M5890	HALLMARK_TNFA_SIGNALING_VIA_NFKB
31	Hallmark (MSigDB)	M5947	HALLMARK_IL2_STAT5_SIGNALING
32	Hallmark (MSigDB)	M5932	HALLMARK_INFLAMMATORY_RESPONSE
33	Hallmark (MSigDB)	M5909	HALLMARK_MYOGENESIS
34	Hallmark (MSigDB)	M5942	HALLMARK_UV_RESPONSE_DN
35	Hallmark (MSigDB)	M5951	HALLMARK_SPERMATOGENESIS
36	Hallmark (MSigDB)	M5898	HALLMARK_DNA_REPAIR
37	Hallmark (MSigDB)	M5896	HALLMARK_TGF_BETA_SIGNALING
38	Hallmark (MSigDB)	M5948	HALLMARK_BILE_ACID_METABOLISM
39	Hallmark (MSigDB)	M5938	HALLMARK_REACTIVE_OXIGEN_SPECIES_PATHWAY
40	Hallmark (MSigDB)	M5922	HALLMARK_UNFOLDED_PROTEIN_RESPONSE
41	Hallmark (MSigDB)	M5895	HALLMARK_WNT_BETA_CATENIN_SIGNALING
42	Hallmark (MSigDB)	M5949	HALLMARK_PEROXISOME
43	Hallmark (MSigDB)	M5910	HALLMARK_PROTEIN_SECRETION
44	Hallmark (MSigDB)	M5905	HALLMARK_ADIPOGENESIS
45	Hallmark (MSigDB)	M5919	HALLMARK_HEDGEHOG_SIGNALING
46	Hallmark (MSigDB)	M5936	HALLMARK_OXIDATIVE_PHOSPHORYLATION
47	Hallmark (MSigDB)	M5897	HALLMARK_IL6_JAK_STAT3_SIGNALING
48	Hallmark (MSigDB)	M5903	HALLMARK_NOTCH_SIGNALING
49	Hallmark (MSigDB)	M5945	HALLMARK_HEME_METABOLISM

```
50 Hallmark (MSigDB)      M5928                 HALLMARK_MYC_TARGETS_V2
      P.value       FDR      Effect   Odds.Ratio   Status N.Geneset.Genes
1  0.007210798 0.3366696   2.0650386 7.885602e+00 enriched             39
2  0.013466783 0.3366696   1.9070347 6.733094e+00 enriched             44
3  0.055226772 0.9204462   1.2000853 3.320400e+00 enriched            200
4  0.099016124 0.9433876   1.0309644 2.803768e+00 enriched            200
5  0.123477705 0.9433876   0.9641464 2.622548e+00 enriched            200
6  0.131277605 0.9433876   1.1296276 3.094504e+00 enriched            101
7  0.132074264 0.9433876   1.1301876 3.096237e+00 enriched            138
8  0.153006954 0.9562935   1.4901855 4.437919e+00 enriched             36
9  0.261645178 0.9832372   0.8382321 2.312275e+00 enriched            161
10 0.270128932 0.9832372   0.8252106 2.282361e+00 enriched            200
11 0.307096452 0.9832372   1.0572811 2.878534e+00 enriched             74
12 0.308608297 0.9832372   0.7603828 2.139095e+00 enriched            200
13 0.333942626 0.9832372   0.7215693 2.057660e+00 enriched            199
14 0.369629032 0.9832372   0.6680817 1.950492e+00 enriched            200
15 0.370567850 0.9832372   0.9268041 2.526422e+00 enriched             97
16 0.414857491 0.9832372   0.6078045 1.836395e+00 enriched            200
17 0.424828541 0.9832372   0.5940336 1.811280e+00 enriched            200
18 0.519424905 0.9832372   0.4789070 1.614309e+00 enriched            200
19 0.565013411 0.9832372   0.4292091 1.536042e+00 enriched            199
20 0.636750814 0.9832372   0.4865777 1.626740e+00 enriched            105
21 0.762665708 0.9832372   0.3113319 1.365242e+00 enriched            198
22 0.766003568 0.9832372   0.3061772 1.358223e+00 enriched            158
23 0.801835270 0.9832372   0.2585896 1.295102e+00 enriched            158
24 0.848185660 0.9832372   0.1971499 1.217927e+00 enriched            200
25 0.941704828 0.9832372   0.0751372 1.078032e+00 enriched            200
26 0.596772045 0.9832372  -0.5446876 5.800230e-01 depleted            199
27 0.632976060 0.9832372  -0.4913727 6.117860e-01 depleted            200
28 0.761266682 0.9832372  -0.3114038 7.324181e-01 depleted            200
29 0.787106058 0.9832372  -0.2781636 7.571730e-01 depleted            199
30 0.791398025 0.9832372  -0.2711955 7.624674e-01 depleted            200
31 0.811615352 0.9832372  -0.2443141 7.832416e-01 depleted            200
32 0.822419900 0.9832372  -0.2304902 7.941442e-01 depleted            200
33 0.842358130 0.9832372  -0.2043783 8.151539e-01 depleted            200
34 0.972238830 0.9832372 -13.4793606 1.399549e-06 depleted            144
35 0.973404957 0.9832372 -12.7673218 2.852480e-06 depleted            135
36 0.973687327 0.9832372 -12.1263283 5.415051e-06 depleted            149
37 0.975154999 0.9832372 -11.9548439 6.428021e-06 depleted             54
38 0.975942720 0.9832372 -12.5187551 3.657411e-06 depleted            111
39 0.976698181 0.9832372 -11.5268656 9.861566e-06 depleted             49
40 0.976752526 0.9832372 -12.3566713 4.300964e-06 depleted            112
41 0.977378938 0.9832372 -11.9380109 6.537139e-06 depleted             42
```

```
42 0.977618249 0.9832372 -12.5373207 3.590135e-06 depleted            104
43 0.978424277 0.9832372 -12.8760081 2.558708e-06 depleted             96
44 0.978983596 0.9832372 -13.5316571 1.328238e-06 depleted            200
45 0.979401399 0.9832372 -12.4368600 3.969541e-06 depleted             36
46 0.979612012 0.9832372 -13.1903447 1.868556e-06 depleted            197
47 0.979696172 0.9832372 -12.5388526 3.584640e-06 depleted             87
48 0.980112685 0.9832372 -11.8219938 7.341306e-06 depleted             32
49 0.980135570 0.9832372 -13.6090416 1.229330e-06 depleted            199
50 0.983237225 0.9832372 -11.9429655 6.504830e-06 depleted             58
   N.Geneset.Peak.Genes Geneset.Avg.Gene.Length Geneset.Peak.Genes
1                     2               187356.28          5126, 6726
2                     2               209852.25           351, 1946
3                     3               113673.32   5788, 7042, 10225
4                     3               141499.61     716, 2153, 4324
5                     3               157525.62   3675, 5788, 25945
6                     2               161568.85         9510, 84159
7                     2               116728.85           716, 4324
8                     1               155037.22                 351
9                     2               134149.50           351, 7042
10                    2               102425.97          4128, 7042
11                    1               107865.30                6282
12                    2               110946.77         5033, 10097
13                    2               121400.47          716, 84159
14                    2               118319.26            374, 799
15                    1                89686.42                 716
16                    2               135063.36          1946, 7514
17                    2               133232.24          2584, 5033
18                    2               149021.69            374, 799
19                    2               169486.20          2584, 5033
20                    1               142428.66               10097
21                    1                77524.61                7514
22                    1               102593.39                4128
23                    1               117646.30                4128
24                    1                86673.06                7514
25                    1               100589.85                 351
26                    1               219793.90                 374
27                    1               206398.79               22903
28                    1               155311.85                 613
29                    1               171727.17                7042
30                    1               151486.12                 374
31                    1               143893.89                5033
32                    1               152045.26                5099
33                    1               151066.60                 351
```

```
34          0       296799.65
35          0       151024.85
36          0        63996.66
37          0       162524.91
38          0       115350.19
39          0       101098.65
40          0        83518.00
41          0       178339.31
42          0       102425.82
43          0       146945.33
44          0       112868.34
45          0       281805.72
46          0        71107.53
47          0       100468.59
48          0       161979.34
49          0       100340.71
50          0        61808.02
```

```
# Examine the top gene sets
head(enrich_primary$results)

# Extract the gene IDs for the top ranking set
genes <- enrich_primary$results$Geneset.Peak.Genes[1]

# Split gene IDs into a vector
gene_ids <- strsplit(genes, ', ')[[1]]

# Convert gene IDs to gene symbols
gene_symbol <- select(org.Hs.eg.db, keys=gene_ids, columns="SYMBOL",
keytype="ENTREZID")

# Print the result
print(gene_symbol)
```

```
output>
head(enrich_primary$results)
  Geneset.Type Geneset.ID                                Description
1 KEGG Pathways   hsa04110                                 Cell cycle
2 KEGG Pathways   hsa00533 Glycosaminoglycan biosynthesis - keratan sulfate
3 KEGG Pathways   hsa04115                     p53 signaling pathway
4 KEGG Pathways   hsa00052                      Galactose metabolism
5 KEGG Pathways   hsa00480                     Glutathione metabolism
```

```
6 KEGG Pathways   hsa04977               Vitamin digestion and absorption
      P.value       FDR   Effect Odds.Ratio   Status N.Geneset.Genes
1 0.001626919 0.3335183 2.009188   7.457261 enriched             123
2 0.009044566 0.6339066 2.837527  17.073498 enriched              15
3 0.009276682 0.6339066 1.978029   7.228479 enriched              68
4 0.014550627 0.6935708 2.609276  13.589215 enriched              27
5 0.019359318 0.6935708 2.499356  12.174651 enriched              50
6 0.020299634 0.6935708 2.468228  11.801512 enriched              24
  N.Geneset.Peak.Genes Geneset.Avg.Gene.Length Geneset.Peak.Genes
1                    3                88319.47  4616, 8555, 10912
2                    1               149392.57               2683
3                    2               114026.68        4616, 10912
4                    1                67549.17               2683
5                    1                45632.53               4257
6                    1                96953.92               4363

# Extract the gene IDs for the top ranking set
genes <- enrich_primary$results$Geneset.Peak.Genes[1]

# Split gene IDs into a vector
gene_ids <- strsplit(genes, ', ')[[1]]

# Convert gene IDs to gene symbols
gene_symbol <- select(org.Hs.eg.db, keys=gene_ids, columns="SYMBOL",
keytype="ENTREZID")
'select()' returned 1:1 mapping between keys and columns

# Print the result
print(gene_symbol)
  ENTREZID  SYMBOL
1     4616 GADD45B
2     8555  CDC14B
3    10912 GADD45G
```

General format of KEGG URLs:
https://www.kegg.jp/pathway/pathway/<pathway_id>+<gene_id>+...+<gene_id>

```
# This is the base URL for all KEGG pathways
base_url <- "https://www.kegg.jp/pathway/"

# Add pathway ID to URL
path_url <- paste0(base_url, top_path)
```

```
# Collapse gene IDs into selection string
gene_select <- paste(genes, collapse="+")

# Add gene IDs to URL
path_url <- paste(path_url, gene_select, sep="+")

ouput>
https://www.kegg.jp/pathway/hsa04110+4616+8555+10912
```