DNAModAnnot: Protocol full analysis example

Alexis Hardy

2022-01-01

Introduction

DNAModAnnot is a R package providing a comprehensive toolkit for the analysis and annotation of DNA modifications (e.g. 6-methyladenine (6mA)). Its modular architecture allows the analysis of modification detection performed using Pacific Biosciences (PacBio) kineticsTools or Oxford Nanopore Technologies via DeepSignal software.

This documentation provides a guided use of this package using PacBio input data. By following this tutorial, you will learn to:

- 1st step: load and filter the genome and DNA modification data
- 2nd step: analyze the global distribution and motif of DNA modification data
- 3rd step: filter DNA modification data using False Discovery Rate estimations
- 4th step: analyze the DNA modification patterns within genomic annotations

---

Installation

This package is available via GitHub (https://github.com/AlexisHardy/DNAModAnnot) and can be installed using the following code:

```
if (!requireNamespace("BiocManager", quietly = TRUE)) {
  install.packages("BiocManager")
}

BiocManager::install(c('Biostrings', 'BSgenome', 'Gviz', 'seqLogo'))
```

Then you can directly install DNAModAnnot from GitHub using devtools package:

```
install.packages("devtools")
library(devtools)
install_github("AlexisHardy/DNAModAnnot")
```

Or you can install it using the tar.gz file from GitHub repository:

```
setwd("path/to/package/file/")
install.packages("DNAModAnnot_0.0.0.9019.tar.gz", repos = NULL, type = 'source')
```

You should then be able to load the package into your R session with:

```
library(DNAModAnnot)
```

Required data

DNAModAnnot can load data generated by the PacBio platform after using the Pacific Biosciences (PacBio) kineticsTools on PacBio RSI/RSII/Sequel/SequelII data. The following output files will be used by *DNAModAnnot* to perform a stringent analysis of detected modification patterns:

- modifications.csv file (ModCSV): contains kinetics from all bases sequenced.
- modifications.gff file (ModGFF): contains kinetics for all modified bases detected along with methylation level (or fraction) for each of those modifications.

It is also possible for *DNAModAnnot* to load data generated by the Nanopore platform after using the DeepSignal software. Here, the output of the call_modification_frequency.py script (containing modification detection parameters for all target sites sequenced) will then be required (cf Deepsignal documentation).

Files location

First, give the path of each input files required fro this Rmd.

- T. thermophila (June2014) genome assembly (fasta) and annotation (gff3) can be retrieved at http://ciliate.org/index.php/home/downloads.
- modifications.csv and modifications.gff files can be retrieved at https://github.com/AlexisHardy/DNAModAnnot_AdditionalData.
- T. thermophila RNA-seq txt file can be retrieved at https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM692081.
- T. thermophila MNase-seq bed file can be retrieved at https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2534785.
- T. thermophila H2A.Z ChIP-seq bed file can be retrieved at https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2534783.

```
#Change the path of the input files here
organism_genome_path <- "./T_thermophila_June2014_assembly.fasta"
annotations_path <- "./T_thermophila_June2014.gff3"

ModCSV_gpos_path <- "./modifications.csv.gz"
ModGFF_gpos_path <- "./modifications.gff.gz"

expression_file_path <- "./GSM692081_Growth.map.txt"
MNaseSeq_path <- "./GSM2534785_SB210_MNase.120_260.unique.bed"
ChIPSeq_path <- "./GSM2534783_WT_H2A.Z_ChIP.120_260.unique.bed"
```

1st step: load and filter the genome and DNA modification data

Before analyzing modifications distribution, the package *DNAModAnnot* proposes various functions to assess the quality of sequencing and filter out bias (e.g. lack of coverage) coming from some scaffolds/bases.

You can load a genome (fasta file) using `readDNAStringSet` from *Biostrings* package.
The `GetGenomeGRanges` function will return a GRanges object representing the scaffolds which will be used by various functions later.

```
organism_genome <- Biostrings::readDNAStringSet(organism_genome_path)
names(organism_genome) <- gsub(x = names(organism_genome),
                               pattern = " .*",
                               replacement = "")
#We only retrieve the 50 contigs and then we filter the organism_genome object
contigsToKeep <- read.table("./contig_list.txt")[,1]
organism_genome <- organism_genome[
    names(organism_genome) %in% contigsToKeep]

organism_genome_range <- GetGenomeGRanges(organism_genome)
```

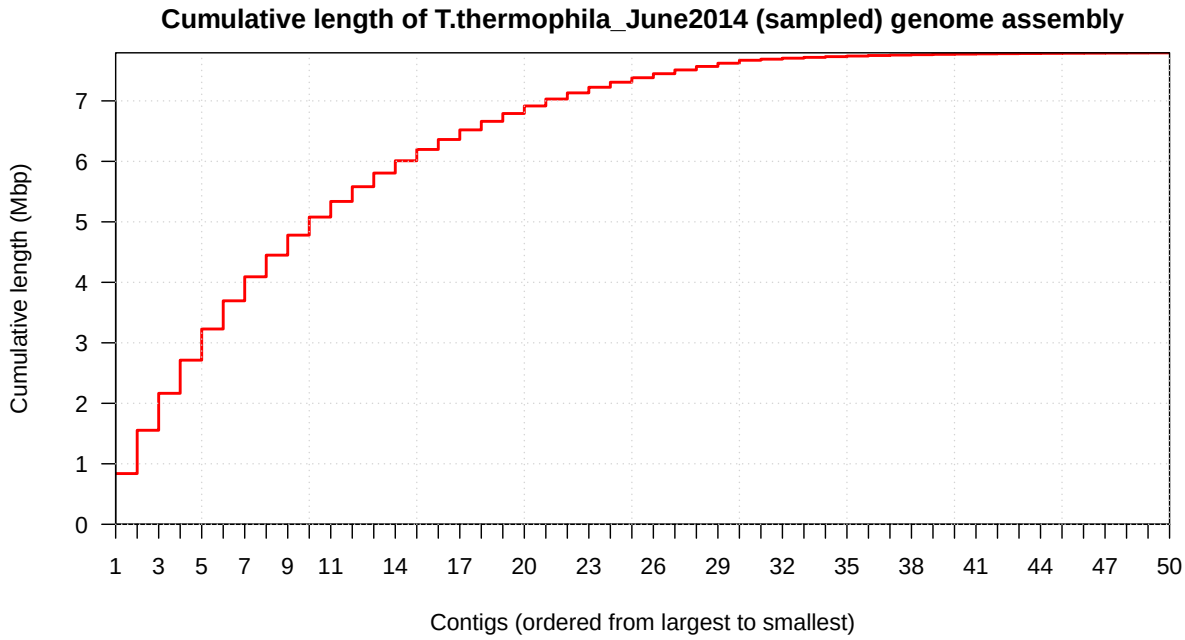
The `GetAssemblyReport` function returns a classic assembly report in a data.frame object.

```
report_assembly <- GetAssemblyReport(
  dnastringsetGenome = organism_genome,
  cOrgAssemblyName = "T.thermophila_June2014 (sampled)"
)
formatC(report_assembly, format = "f")
#>                      T.thermophila_June2014 (sampled)
#> nbcontigs            "50.0000"
#> nbcontigs_ab0        "50.0000"
#> nbcontigs_ab1000     "50.0000"
#> nbcontigs_ab5000     "39.0000"
#> nbcontigs_ab10000    "35.0000"
#> nbcontigs_ab25000    "30.0000"
#> nbcontigs_ab50000    "29.0000"
#> largest_contig_size  "838362.0000"
#> total_length         "7794584.0000"
```

```
#> total_length_ab0     "7794584.0000"
#> total_length_ab1000  "7794584.0000"
#> total_length_ab5000  "7769029.0000"
#> total_length_ab10000 "7741205.0000"
#> total_length_ab25000 "7671504.0000"
#> total_length_ab50000 "7622466.0000"
#> n50                  "397044.0000"
#> n75                  "203516.0000"
#> l50                  "7.0000"
#> l75                  "14.0000"
#> gc_pct               "22.2478"
#> nb_N                 "4700.0000"
```

The `GetContigCumulLength` function returns a data.frame with scaffolds size and cumulated size. This data.frame can be used by `DrawContigCumulLength` function to plot cumulated length by contig.

```
contig_cumul_length <- GetContigCumulLength(organism_genome)
DrawContigCumulLength(
  nContigCumsumLength = contig_cumul_length$cumsum_Mbp_length,
  cOrgAssemblyName = "T.thermophila_June2014 (sampled)",
  lGridInBackground = TRUE
)
```



Cumulative length (Mbp) of 50 contigs from T.thermophila_June2014 genome assembly. Contigs are ordered from largest to smallest contig. Here, only data from 50 random contigs are used.

The modifications.csv (ModCSV) and modifications.gff (ModGFF) files can be loaded using the `ImportPacBioCSV` and `ImportPacBioGFF` functions respectively.

`ImportPacBioCSV` will import PacBioCSV file as an Unstitched GPos object.

> Note that `SortGPos=TRUE` will take a longer time for the importation but will allow to greatly reduce the size of the GPos output.

```
ModCSV_gpos <- ImportPacBioCSV(
  cPacBioCSVPath = "./modifications.csv.gz",
  cSelectColumnsToExtract = c(
    "refName", "tpl", "strand",
    "base", "score",
    "ipdRatio", "coverage"
  ),
  lKeepExtraColumnsInGPos = TRUE,
  lSortGPos = TRUE,
  cContigToBeAnalyzed = names(organism_genome)
)
```

`ImportPacBioGFF` will import PacBioGFF file as a GRanges object.

> Note that `ImportPacBioGFF` will import one modification type at a time that you must precise using `cNameModToExtract` option.

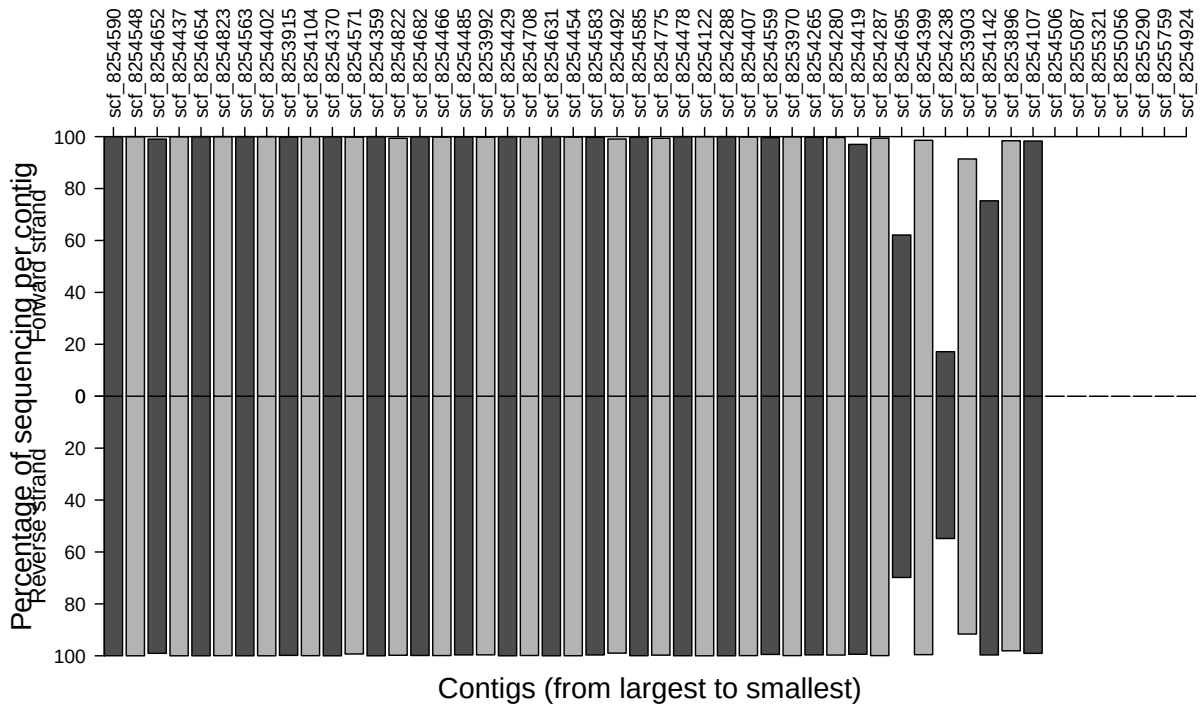
```
ModGFF_granges <- ImportPacBioGFF(
  cPacBioGFFPath = "./modifications.gff.gz",
  cNameModToExtract = "m6A",
  cModNameInOutput = "6mA",
  cContigToBeAnalyzed = names(organism_genome)
)
```

You can use the `GetSeqPctByContig` function with the PacBioCSV GPos object to retrieve, into a list, the percentage of sequenced bases by contig. This list can be used for filtering with the `FiltPacBio` function.

```
contig_pct_seq <- GetSeqPctByContig(ModCSV_gpos,
  grangesGenome = organism_genome_range
)
```

You can also draw a barplot by strand using the `DrawBarplotBothStrands` function with the content of this previous list.

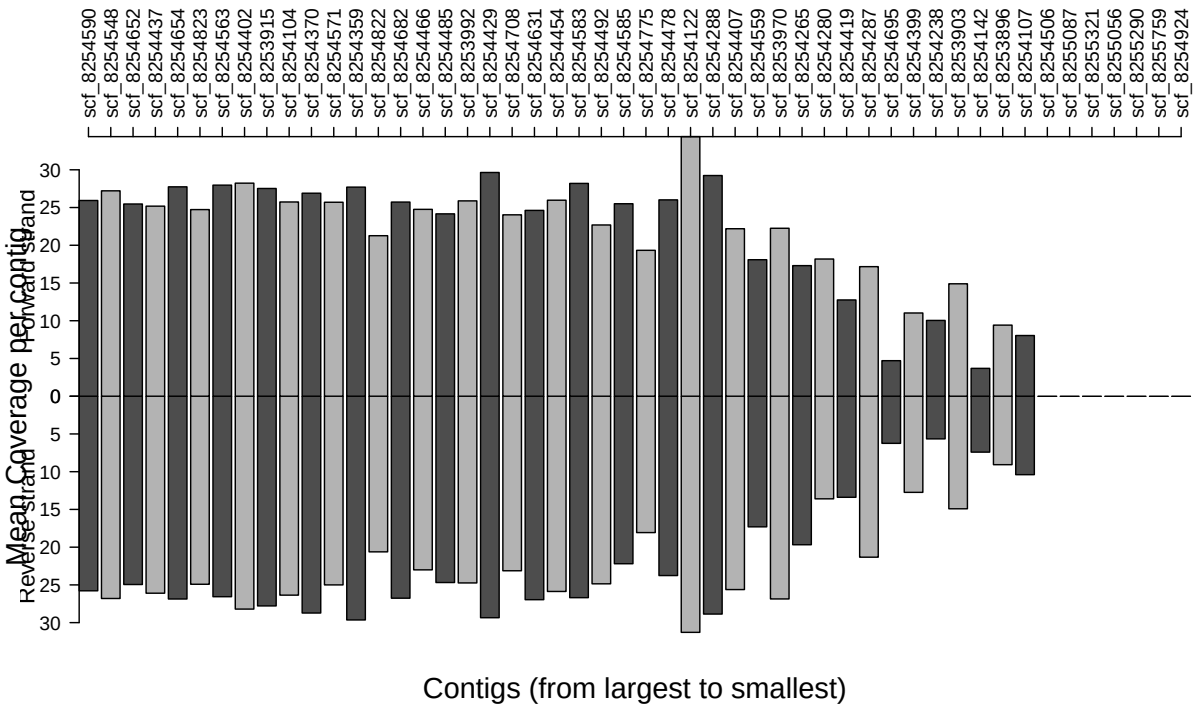
```
DrawBarplotBothStrands(
  nParamByContigForward = contig_pct_seq$f_strand$seqPct,
  nParamByContigReverse = contig_pct_seq$r_strand$seqPct,
  cContigNames = contig_pct_seq$f_strand$refName,
  cGraphName = "Percentage of sequencing per contig"
)
```



Percentage of sequenced bases (percentage of sequencing) per contig and per strand. Contigs are ordered from largest to smallest contig. Upper part: forward strand; lower part: reverse strand. Percentage of sequencing = 100 x Number of sequenced bases/Number of total bases*.
*According to the genome assembly sequence provided.

The same approach can be used for any numeric parameter available in the ModCSV GPos and ModGFF GRanges objects. You can calculate the mean by scaffold for a numeric parameter using the `GetMeanParamByContig` function. Then, you can use the output list to represent barplots by strand using the `DrawBarplotBothStrands` function again.

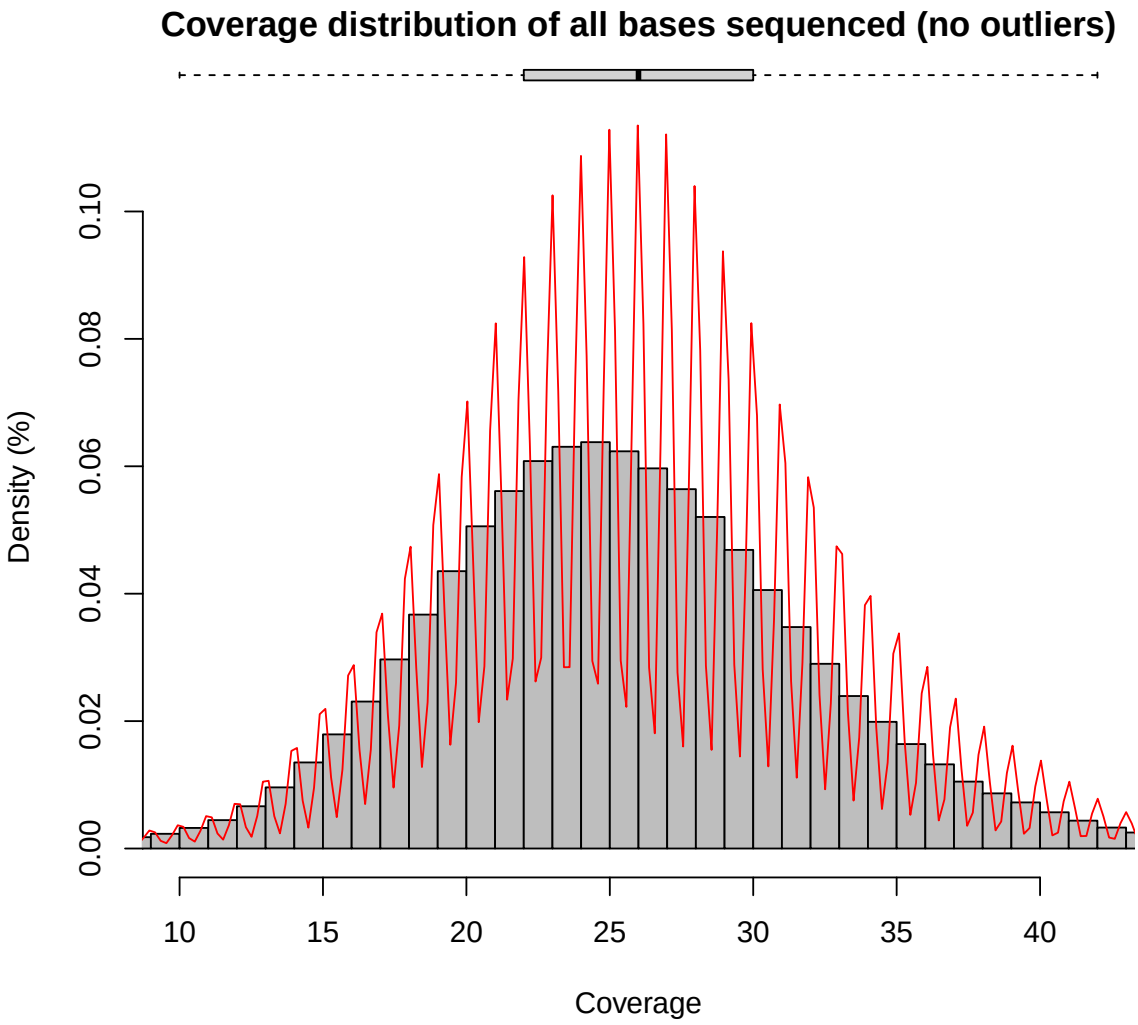
```
contig_mean <- GetMeanParamByContig(
  grangesData = ModCSV_gpos,
  dnastringsetGenome = organism_genome,
  cParamName = "coverage"
)
DrawBarplotBothStrands(
  nParamByContigForward = contig_mean$f_strand$mean_coverage,
  nParamByContigReverse = contig_mean$r_strand$mean_coverage,
  cContigNames = contig_mean$f_strand$refName,
  cGraphName = "Mean Coverage per contig"
)
```



Mean coverage of sequenced bases per contig and per strand. Contigs are ordered from largest to smallest contig. Upper part: forward strand; lower part: reverse strand.

To represent the distribution of a numeric parameter, the `DrawDistriHistBox` function allows you to plot an histogram along a boxplot showing the global range of this parameter.

```
DrawDistriHistBox(ModCSV_gpos$coverage,
  cGraphName = "Coverage distribution of all bases sequenced",
  cParamName = "Coverage",
  lTrimOutliers = TRUE
)
```



Coverage distribution of all sequenced bases. Grey bars: density of coverage values. Red line: smooth representation of the density of coverage values. Boxplot: represents the coverage distribution while highlighting the limit with the outliers values.

You can use the `FiltPacBio` function to filter the ModCSV GPos and ModGFF GRanges objects.
This function will return a list with the new ModCSV GPos as the first element of the list, and the new ModGFF GRanges as the second element of the list.
Some options of this function require ModCSV GPos object or the sequence of the genome (`dnastringsetGenome` object). (see `FiltPacBio` documentation for more information)

```
# First filter: contigs
Mod_filtered_data <- FiltPacBio(
  gposPacBioCSV = ModCSV_gpos,
  grangesPacBioGFF = ModGFF_granges,
  cContigToBeRemoved = NULL,
  dnastringsetGenome = organism_genome,
  nContigMinSize = 1000,
  listPctSeqByContig = contig_pct_seq,
  nContigMinPctOfSeq = 95,
  listMeanCovByContig = contig_mean,
```

```
  nContigMinCoverage = 10
)
ModCSV_gpos <- Mod_filtered_data$csv
ModGFF_granges <- Mod_filtered_data$gff
```

2nd step: analyze the global distribution and motif of DNA modification data

This part will describe some tools which can be used before and/or after filtering based on False Discovery (FDR) estimations.

You can use the `GetModReport` function to obtain a data.frame giving global information about modification distribution (Modification counts, ratio, motifs associated, mean of parameters provided...).

```
report_modifications <- GetModReportPacBio(
  grangesGenome = organism_genome_range,
  grangesPacBioGFF = ModGFF_granges,
  gposPacBioCSV = ModCSV_gpos,
  cOrgAssemblyName = "T.thermophila_June2014 (sampled)",
  dnastringsetGenome = organism_genome,
  cBaseLetterForMod = "A",
  cModNameInOutput = "6mA"
)
```

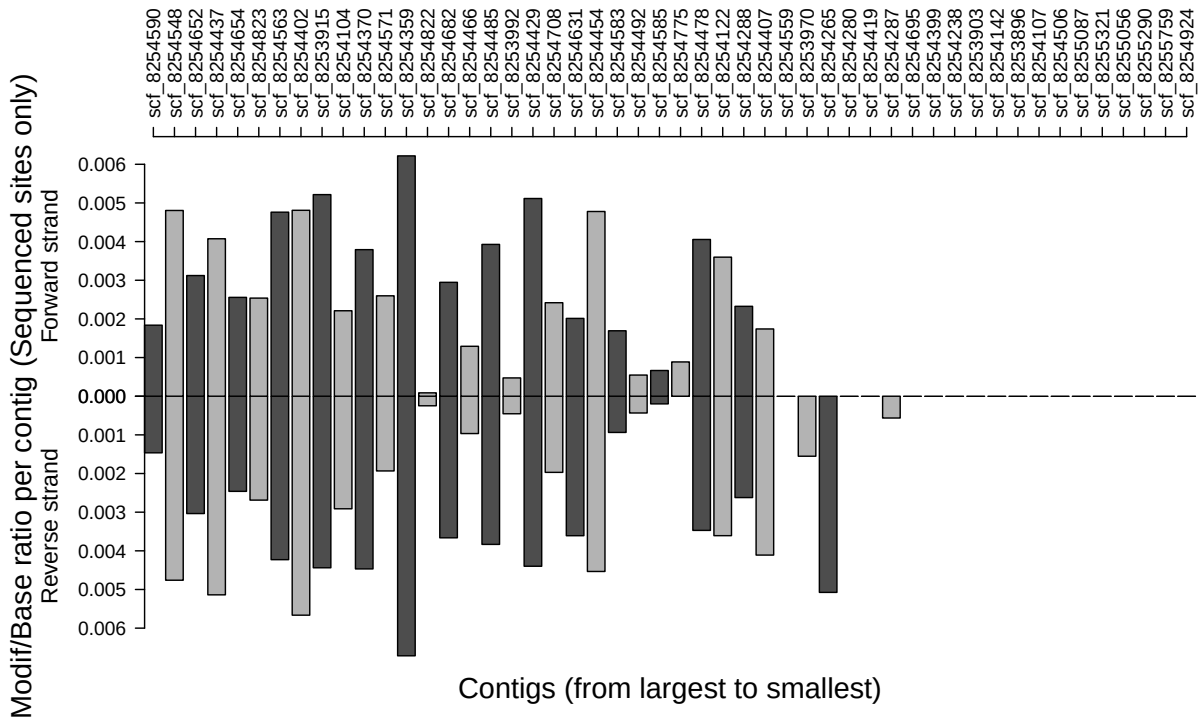
```
formatC(report_modifications, format = "f")
#>                                  T.thermophila_June2014 (sampled)
#> nb6mA                            "19354.0000"
#> nbASequenced                     "15488227.0000"
#> nbAAssembly                      "6056810.0000"
#> ratio6mA                         "0.0012"
#> ratio6mA_corrected               "0.0010"
#> ratio6mAAllAssemblyAs            "0.0032"
#> ratio6mAAllAssemblyAs_corrected "0.0025"
#> mean_frac_6mA                    "0.7693"
#> mean_coverage_6mA                "31.3763"
#> mean_score_6mA                   "43.2575"
#> mean_ipdRatio_6mA                "4.5816"
#> mean_identificationQv_6mA        "24.2551"
#> pct_6mAA                         "0.0067"
#> pct_6mAC                         "0.0019"
#> pct_6mAG                         "0.0072"
#> pct_6mAT                         "0.9842"
#> pct_A6mA                         "0.3562"
#> pct_C6mA                         "0.1997"
#> pct_G6mA                         "0.2239"
#> pct_T6mA                         "0.2203"
#> pct_A6mAA                        "0.0010"
#> pct_A6mAC                        "0.0002"
#> pct_A6mAG                        "0.0013"
```

```
#> pct_A6mAT                    "0.3536"
#> pct_C6mAA                    "0.0005"
#> pct_C6mAC                    "0.0001"
#> pct_C6mAG                    "0.0005"
#> pct_C6mAT                    "0.1986"
#> pct_G6mAA                    "0.0041"
#> pct_G6mAC                    "0.0011"
#> pct_G6mAG                    "0.0035"
#> pct_G6mAT                    "0.2151"
#> pct_T6mAA                    "0.0010"
#> pct_T6mAC                    "0.0005"
#> pct_T6mAG                    "0.0019"
#> pct_T6mAT                    "0.2169"
```

You can use the `GetModRatioByContig` function with the ModGFF GRanges object to retrieve, in a list, the modification ratio (ModRatio) by scaffold.
Again, you can draw a barplot by strand using `DrawBarplotBothStrands` function with the content of this list.

```
contig_mod_ratio <- GetModRatioByContig (ModGFF_granges,
  ModCSV_gpos[ModCSV_gpos$base == "A"],
  dnastringsetGenome = organism_genome,
  cBaseLetterForMod = "A"
)
contig_mod_ratio$f_strand$Mod_ratio[
is.na(contig_mod_ratio$f_strand$Mod_ratio)] <- 0
contig_mod_ratio$r_strand$Mod_ratio[
is.na(contig_mod_ratio$r_strand$Mod_ratio)] <- 0

DrawBarplotBothStrands(
  nParamByContigForward = contig_mod_ratio$f_strand$Mod_ratio,
  nParamByContigReverse = contig_mod_ratio$r_strand$Mod_ratio,
  cContigNames = contig_mod_ratio$f_strand$refName,
  cGraphName = "Modif/Base ratio per contig (Sequenced sites only)"
)
```



Modification ratio (Modified bases / Sequenced bases) per contig and per strand. Contigs are ordered from largest to smallest contig. Upper part: forward strand; lower part: reverse strand. Modification ratio = Number of Modified bases / Number of Sequenced bases. Here, the modification ratio is similar between strands but not between contigs: this must be considered as it could result in a potential bias in the analysis of some annotated features.

You can reveal enrichment (or enrichment + depletion) around modified bases by plotting a logo with the `DrawModLogo` function.

Here, you need to provide the sequence around each modified base. You can get those sequences by using the ModGFF GRanges object with the `GetGRangesWindowSeqandParam` function and retrieving the `sequence` column.
You can decide the length of the sequence to be compared around the modifications using the `nUpstreamBpToAdd` and `nDownstreamBpToAdd` options.

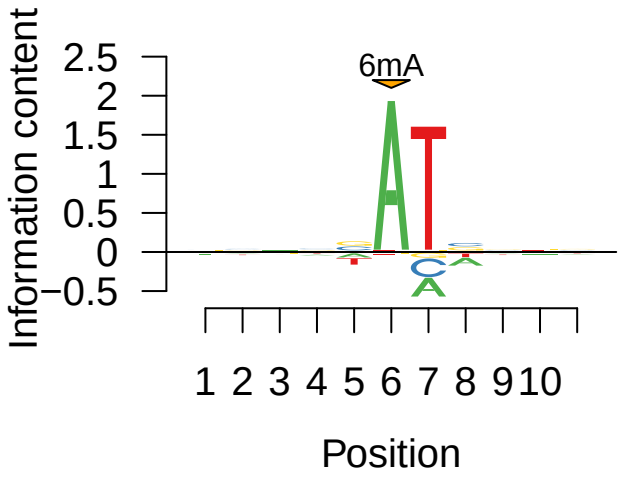
```
ModGFF_granges_seq <- GetGRangesWindowSeqandParam(ModGFF_granges,
  organism_genome_range,
  dnastringsetGenome = organism_genome,
  nUpstreamBpToAdd = 5,
  nDownstreamBpToAdd = 5
)

Seq_ForLogo <- as(ModGFF_granges_seq$sequence, "DNAStringSet")

backgroundACGT = c(
  (100-report_assembly["gc_pct",])/2,
  report_assembly["gc_pct",]/2,
  report_assembly["gc_pct",]/2,
```

```
  (100-report_assembly["gc_pct",])/2
)/100

DrawModLogo(
  dnastringsetSeqAroundMod = Seq_ForLogo,
  nGenomicBgACGT = backgroundACGT, cYunit = "ic_hide_bg",
  nPositionsToAnnotate = c(6), cAnnotationText = c("6mA"), nTagTextFontSize = 12
)
```



Modification sequence logo with enrichment score. Enrichment score is calculated for each position: negative values reflects base depletion while positive values means enrichment.

3rd step: filter the DNA modification data using False Discovery Rate estimations

DNAModAnnot provides tools to estimate False Discovery Rate (FDR) by threshold on parameters associated to modification detection.
These FDR estimations can help deciding filters to be used for filtering DNA modifications to be kept.

First, you can use the `ExtractQuantifByModMotif` function to retrieve a list containing the following elements:

- the names of the motifs over-represented with the modification
- the same motifs + the position of the modification in these motifs
- a table containing the percentage of modifications in each motif tested.
- a PacBioGFF GRangesList with one GRanges object by motif over-represented with the modification. The minimum proportion to define motifs as "over-represented" with the modification can be modified using the `nModMotifMinProp` option.

The PacBioGFF GRangesList object will be used to keep only data from motifs over-represented.

```
listMotif_ModGFF_grangeslist <- ExtractListModPosByModMotif(
  grangesModPos = ModGFF_granges,
  grangesGenome = organism_genome_range,
  dnastringsetGenome = organism_genome,
  nUpstreamBpToAdd = 0, nDownstreamBpToAdd = 1,
  nModMotifMinProp = 0.05,
  cBaseLetterForMod = "A",
  cModNameInOutput = "6mA"
)
```

Before estimating FDR, you must convert the PacBioCSV GPos object into a GRanges object. This will allows you to extract the sequence with the PacBioCSV data (using the `GetGRangesWindowSeqandParam` function) which is required for later steps.

```
SubsetModCSV_granges <- as(ModCSV_gpos[ModCSV_gpos$base == "A"], "GRanges")
ModCSV_granges_seq <- GetGRangesWindowSeqandParam(
  grangesData = SubsetModCSV_granges,
  grangesGenome = organism_genome_range,
  dnastringsetGenome = organism_genome,
  nUpstreamBpToAdd = 0,
  nDownstreamBpToAdd = 1
)
```

Then, `GetFdrEstListByThresh` function can be used to retrieve a list (by motif over-represented) of data.frames. Each data.frame contains the FDR estimated by threshold (for the provided parameter) (and the adjusted FDR ~= cumulative minimum FDR).

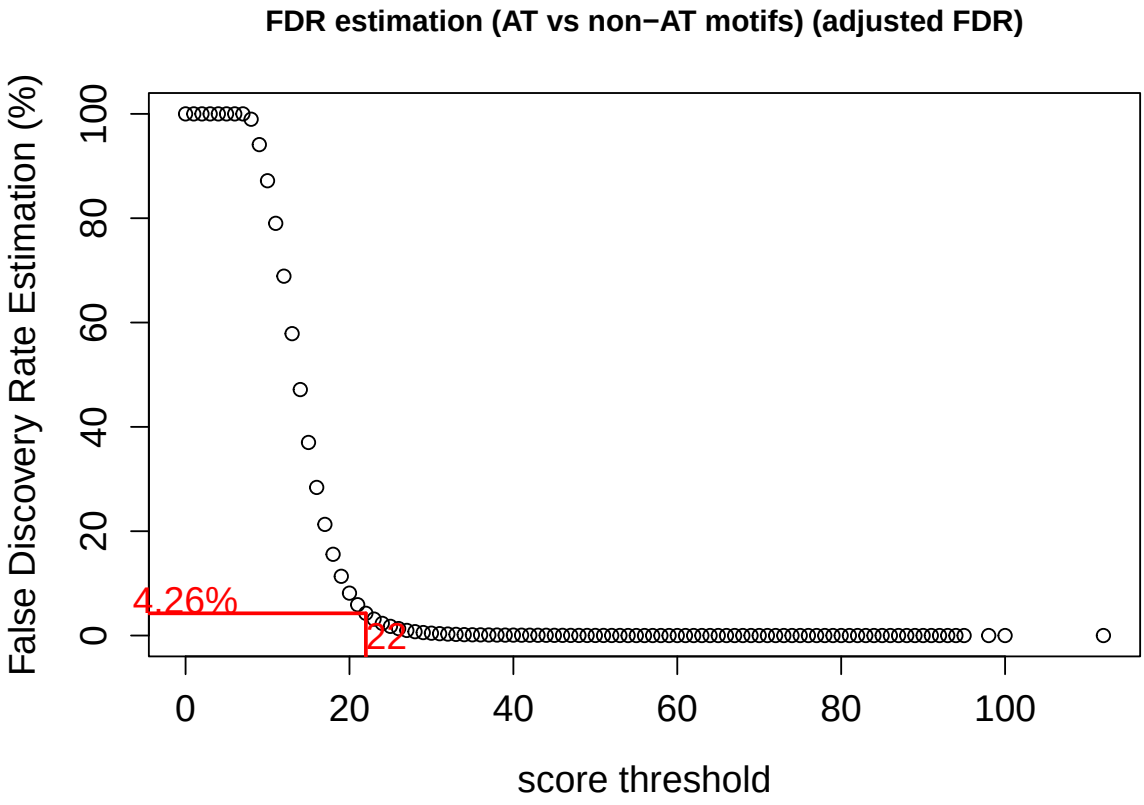
```
score_fdr_by_motif_list <- GetFdrEstListByThresh(
  grangesDataWithSeq = ModCSV_granges_seq,
  grangesDataWithSeqControl = NULL,
  cNameParamToTest = "score",
  nRoundDigits = 1,
  cModMotifsAsForeground = listMotif_ModGFF_grangeslist$motifs_to_analyse
)
```

This FDR estimation list can be used by the `GetFdrBasedThreshLimit` function to retrieve, in a list, the threshold associated to a chosen FDR.
> Note that this chosen FDR is defined by `nFdrPropForFilt` option. The threshold will be defined by the closest value below this level of FDR for each motif (except for some motifs if `lUseBestThrIfNoFdrThr=TRUE` is used).

```
score_fdr_by_motif_limit <- GetFdrBasedThreshLimit(score_fdr_by_motif_list,
  nFdrPropForFilt = 0.05,
  lUseBestThrIfNoFdrThr = TRUE
)
```

This FDR estimation list can also be used by the `DrawFdrEstList` function to represent for each motif the distribution of FDR estimation by threshold.

```
DrawFdrEstList(
  listFdrEstByThr = score_fdr_by_motif_list,
  cNameParamToTest = "score",
  nFdrPropForFilt = 0.05
)
```



False Discovery Rate (FDR) estimation (adjusted) by score threshold and by motif over-represented with the modification. The red lines represent the 5% FDR estimation level (or the closest FDR estimation level below 5%) along its associated threshold on score. Here, for a 5% FDR estimation filter on score, AT motifs reach a FDR estimation below 5% (using a score threshold of 22).

Finally, the output of the `GetFdrBasedThreshLimit` function can be used for filtering by motif using the `FiltPacBio` function.

Note that `grangesPacBioGFF` must be a GRangesList object in this case.

```
ModGFF_grangeslist <- FiltPacBio(
  grangesPacBioGFF = listMotif_ModGFF_grangeslist$GRangesbyMotif,
  listFdrEstByThrIpdRatio = NULL,
  listFdrEstByThrScore = score_fdr_by_motif_limit
)$gff
```

4th step: analyze the DNA modification patterns within genomic annotations

This part will describe the tools that can be used after filtering based on False Discovery (FDR) estimations and can be launched for each motif independently.

You must first subset your data to keep only data for one motif:

```
# Subset data to keep only one motif for the next analyses
ModGFF_granges <- ModGFF_grangeslist[["AT"]]

#Other possibilities
#ModGFF_granges <- listMotif_ModGFF_grangeslist$GRangesbyMotif[["AT"]]
#ModGFF_granges <- ModGFF_granges_seq[ModGFF_granges_seq$sequence == "AT",]

ModCSV_granges <- ModCSV_granges_seq[ModCSV_granges_seq$sequence == "AT",]
```

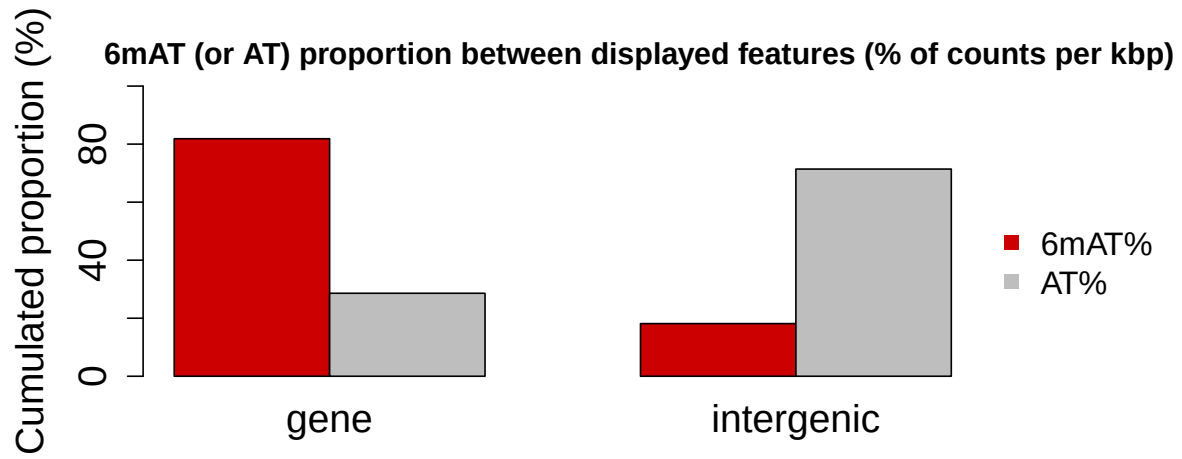
To analyse the distribution of modifications in the genome, you must provide an annotation file as a GRanges object.
The package *rtracklayer* contains various functions for file importation as GRanges objects depending on the format of your annotation file.
For some functions, the feature "intergenic" will be required for comparison between genes and intergenic regions. Here, you can use the `PredictMissingAnnotation` function to complete the annotation with "intergenic" features (and/or introns/exons if your annotation provides exons or introns positions).

```
annot_range <- rtracklayer::readGFFAsGRanges(annotations_path)
annot_range <- PredictMissingAnnotation(
  grangesAnnotations = annot_range,
  grangesGenome = organism_genome_range,
  cFeaturesColName = "type",
  cGeneCategories = c("gene"),
  lAddIntronRangesUsingExon = TRUE
)
#We only retrieve the 50 contigs and then we filter the organism_genome object
contigsToKeep <- read.table("./contig_list.txt")[,1]
annot_range <- annot_range[
  as.character(annot_range@seqnames) %in% contigsToKeep]
```

Counts by feature The `GetModBaseCountsByFeature` function returns the counts of the DNA modification (and its unmodified base) for each feature provided.
The `DrawModBasePropByFeature` function can then calculate (and draw as a barplot) the proportion of modified (or unmodified) base between features to be compared.

```
annot_range_MBcounts <- GetModBaseCountsByFeature(
  grangesAnnotations = annot_range,
  grangesModPos = ModGFF_granges,
  gposModTargetBasePos = ModCSV_granges,
  lIgnoreStrand = FALSE
)
```

```
DrawModBasePropByFeature(
  grangesAnnotationsWithCounts = annot_range_MBcounts,
  cFeaturesToCompare = c("gene", "intergenic"),
  lUseCountsPerkbp = TRUE,
  cBaseMotif = "AT",
  cModMotif = "6mAT"
)
```



6mAT and any AT cumulated proportion (from counts per kbp) between genes and intergenic regions. Here, 6mAT is enriched in genes and its proportion is not explained by the proportion of AT sites (which is more present in intergenic regions).

Quantitative parameter by Feature Modification counts categories You can also compare a quantitative parameter versus Modifications counts in features.

Here you can load an example with RNA-seq data for the annotation provided:

```
expression_dataframe <- read.table(
  file = expression_file_path,
  header = TRUE, sep = "\t"
)
```

You need to load the quantitative parameter to be compared with modification counts as a new column within the annotation (+counts) GRanges object.
Here, we also normalize the counts of mapped reads using the genes size.

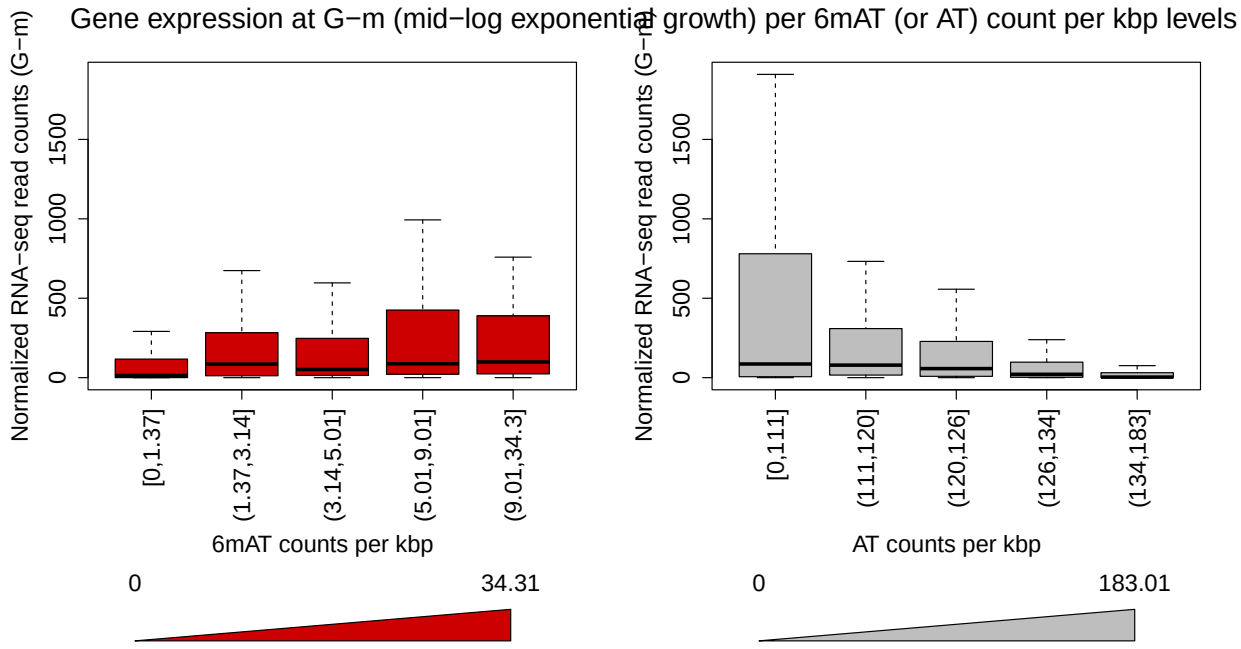
```
genes_range_MBcounts_param <- annot_range_MBcounts[annot_range_MBcounts$type == "gene"]
genes_range_MBcounts_param <- genes_range_MBcounts_param[
  genes_range_MBcounts_param$Name %in% expression_dataframe$Gene_ID
]
```

```r
GenomicRanges::mcols(genes_range_MBcounts_param) <- merge(
  x = GenomicRanges::mcols(genes_range_MBcounts_param),
  by.x = "Name",
  y = expression_dataframe,
  by.y = "Gene_ID"
)

genes_range_MBcounts_param$Number_of_mapped_reads_perkbp <-
  1000*genes_range_MBcounts_param$Number_of_mapped_reads /
  GenomicRanges::width(genes_range_MBcounts_param)
```

Then, you can use the `DrawParamPerModBaseCategories` function. This will plot the distribution of the quantitative parameter provided by category of modified (and unmodified) base counts.

```r
DrawParamPerModBaseCategories(
  grangesAnnotationsWithCounts = genes_range_MBcounts_param,
  cParamColname = "Number_of_mapped_reads_perkbp",
  cParamFullName = "Gene expression at G-m (mid-log exponential growth)",
  cParamYLabel = "Normalized RNA-seq read counts (G-m)",
  cSelectFeature = "gene",
  lUseCountsPerkbp = TRUE,
  cBaseMotif = "AT",
  cModMotif = "6mAT",
  lBoxPropToCount = FALSE, lUseSameYAxis = TRUE
)
```



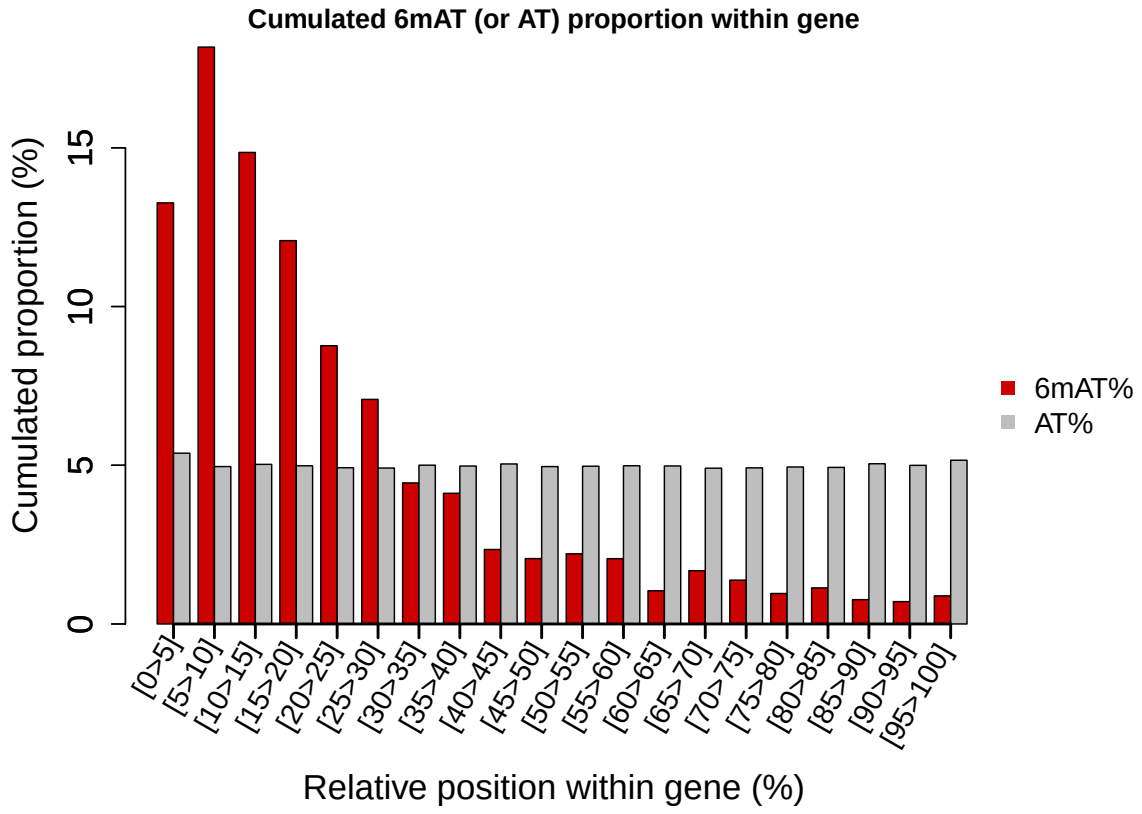
Gene expression at G-m distribution per categories of 6mAT (or AT) counts per kbp. Gene expression at G-m step is estimated using the RNA-seq read counts.

Counts within feature The `GetModBaseCountsWithinFeature` function returns the counts of modified base (and its unmodified target base) within each feature provided.
For this, each feature provided is cut into a specific number of windows (defined by the `nWindowsNb` option) and counts are returned for each window of each feature.

```
genes_range <- annot_range[annot_range$type == "gene", ]
genes_range <- GetModBaseCountsWithinFeature(
  grangesAnnotations = genes_range,
  grangesModPos = ModGFF_granges,
  gposModTargetBasePos = ModCSV_granges,
  lIgnoreStrand = FALSE,
  nWindowsNb = 20
)
```

The `DrawModBaseCountsWithinFeature` function can then represent the distribution within features through a barplot.

```
DrawModBaseCountsWithinFeature(
  grangesAnnotationsWithCountsByWindow = genes_range,
  cFeatureName = "gene",
  cBaseMotif = "AT",
  cModMotif = "6mAT"
)
```



Cumulated 6mAT (or AT) proportion within gene. Genes are cut into windows of relative size and cumulated 6mAT or AT proportion is computed for each window. Here, the cumulated AT proportion remain similar along genes (except a small enrichment at 3' extremity) while 6mAT is enriched at the beginning of genes.

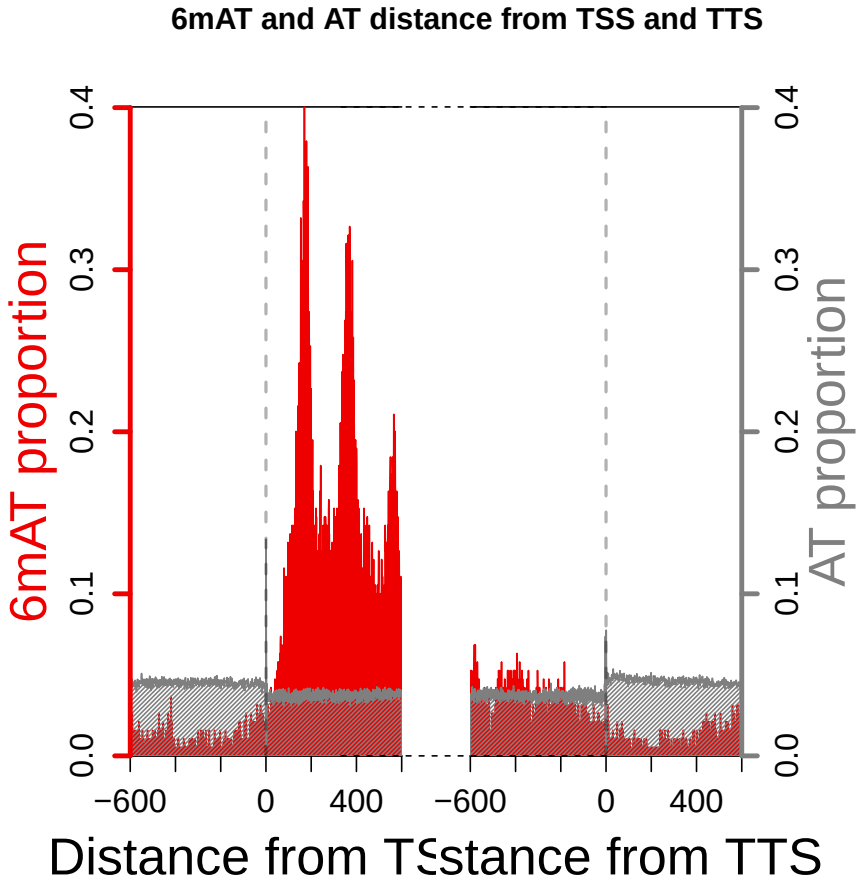
Distance from feature The `GetDistFromFeaturePos` function returns, for each feature provided, its distance, in bp, from a modification (using a window of specific size around each feature).
If the feature provided is larger than 1bp, each feature boundaries will be used instead for computing the distance.

If the `lGetGRangesInsteadOfListCounts` option is set to `FALSE`, the `GetDistFromFeaturePos` function will return instead a list of dataframes giving the counts (or proportion) of modified (or unmodified) bases by distance toward the feature.

```
Mod_distance_feature_countslist <- GetDistFromFeaturePos(
  grangesAnnotations = annot_range,
  cSelectFeature = "gene",
  grangesData = ModGFF_granges,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  cWhichStrandVsFeaturePos = "both",
  nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
Base_distance_feature_countslist <- GetDistFromFeaturePos(
  grangesAnnotations = annot_range,
  cSelectFeature = "gene",
  grangesData = ModCSV_granges,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = TRUE,
  cWhichStrandVsFeaturePos = "both", nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
```

Both Mod and Base lists can be represented on a plot using `DrawModBasePropDistFromFeature` function.

```
DrawModBasePropDistFromFeature(
  listModCountsDistDataframe = Mod_distance_feature_countslist,
  listBaseCountsDistDataframe = Base_distance_feature_countslist,
  cFeaturePosNames = c("TSS", "TTS"),
  cBaseMotif = "AT",
  cModMotif = "6mAT"
)
```



You can also add 1 or 2 additional axis on the plot to compare modification signal with another parameters by using the `AddToModBasePropDistFromFeaturePlot` function.
Here is an example using MNase-seq data to represent nucleosomes:

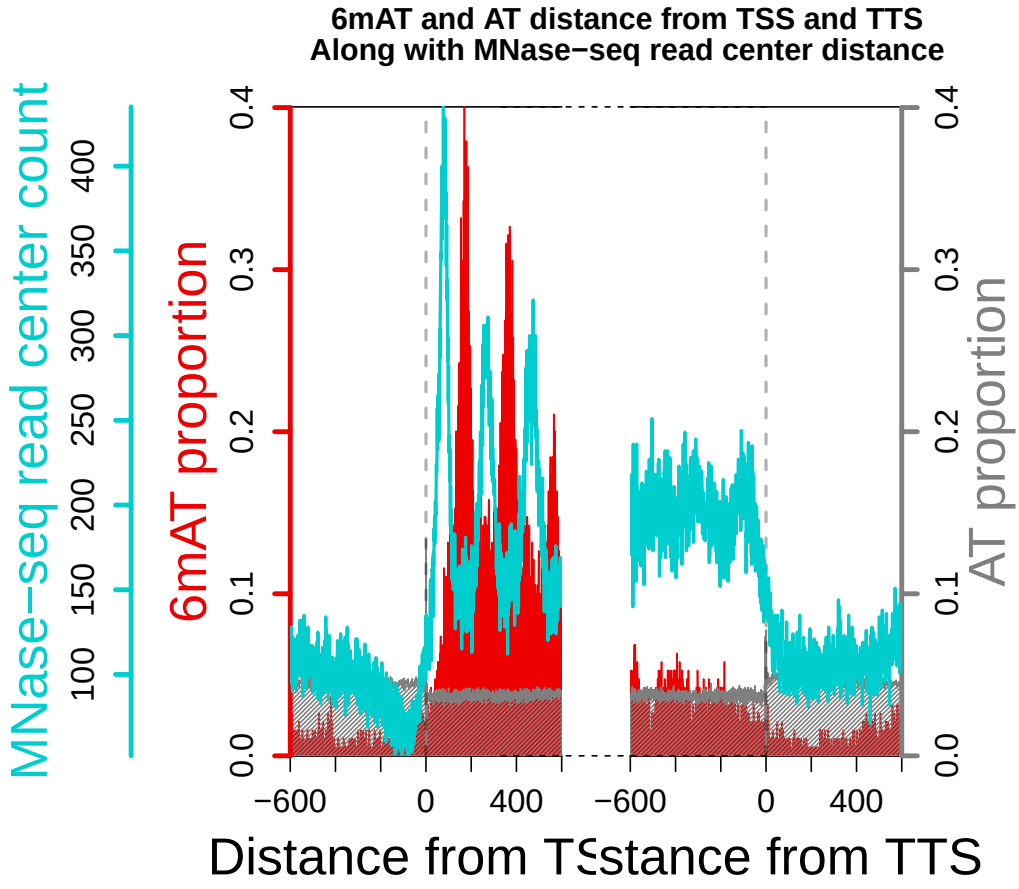
- you can load a bed file using the `import.bed` function from *rtracklayer* package.
- Then, you can retrieve the central position of each read with the `GetGposCenterFromGRanges` function.
- Finally, you can use again the `GetDistFromFeaturePos` function.

```
bedfile_object <- rtracklayer::import.bed(MNaseSeq_path)
bedfile_object <- GetGposCenterFromGRanges(bedfile_object)
bedfile_distance_feature_countslist <- GetDistFromFeaturePos(
  grangesAnnotations = annot_range,
  cSelectFeature = "gene",
  grangesData = bedfile_object,
  lGetGRangesInsteadOfListCounts = FALSE,
  lGetPropInsteadOfCounts = FALSE,
  cWhichStrandVsFeaturePos = "both", nWindowSizeAroundFeaturePos = 600,
  lAddCorrectedDistFrom5pTo3p = TRUE,
  cFeaturePosNames = c("TSS", "TTS")
)
```

Then use this new list with the `AddToModBasePropDistFromFeaturePlot` function to add a new parameter

on the previous graph.

```
DrawModBasePropDistFromFeature(
  listModCountsDistDataframe = Mod_distance_feature_countslist,
  listBaseCountsDistDataframe = Base_distance_feature_countslist,
  cFeaturePosNames = c("TSS", "TTS"),
  cBaseMotif = "AT",
  cModMotif = "6mAT"
)
AddToModBasePropDistFromFeaturePlot(
  dPosCountsDistFeatureStart = bedfile_distance_feature_countslist[[1]],
  dPosCountsDistFeatureEnd = bedfile_distance_feature_countslist[[2]],
  cSubtitleContent = "Along with MNase-seq read center distance",
  cParamYLabel = "MNase-seq read center count",
  cParamColor = "cyan3",
  lAddAxisOnLeftSide = TRUE, cParamLty = 1, cParamLwd = 2
)
```



Distribution of 6mAT sites, AT sites and MNase-seq read centers distance from Transcription Start Sites (TSS; left panel) or Transcription Termination Sites (TTS; right panel). Proportion is computed for each base position around TSS and TTS. Grey dotted vertical line: position of the TSS or TTS. Red: 6mAT proportion; grey: AT proportion; blue: MNase-seq read center counts (represents the positions/accumulation of nucleosomes). No peaks of AT sites can be observed: those are evenly distributed around TSS and TTS. Here, a few peaks of 6mAT signal appears between peaks of the Nucleosome periodic signal downstream

TSS: 6mAT seems slightly more enriched between well-positioned nucleosomes downstream TSS.

Local visualisation You can see locally the distribution of modifications (for any modification or for some motifs) by using the *Gviz* package.
After preparing a set of tracks, you can display them using the `plotTracks` function from *Gviz* package. (see *Gviz* package for more information)

DNAModAnnot provides several functions which can be used alongside the *Gviz* package for local visualization with streaming. The following example is only based on streaming files.
Streaming allows you to plot a large window without being restricted by memory usage via R/RStudio (but using a longer time for plotting).

The `ExportFilesForGViz` function allows the user to export files which can be used for streaming (except for the gff3 format) with the `plotTracks` function from *Gviz* package. Here, this makes possible to use streaming also for genomic annotations using the bam format (and using the `ImportBamExtendedAnnotationTrack` function when making tracks). (see `ExportFilesForGViz` function documentation for more information)

Here, the Rmd do not export files by default and thus needs the following files (available at https://github.com/AlexisHardy/DNAModAnnot_AdditionalData) to generate the next figure.

```
ipdRatio6mABigwig <- "scf_8254548ipdRatio6mA.bw"
GenesBam <- "scf_8254548Genes.bam"
covMNaseSeqBigwig <- "scf_8254548MNaseSeq.bw"
covChIPSeqBigwig <- "scf_8254548ChIPSeq.bw"
```

Here we subset the data for a small dataset example.

```
#Subset annotation to export
SubsetModGFF_granges <- ModGFF_granges[
  as.character(ModGFF_granges@seqnames) == "scf_8254548"
]

#Subset annotation to export
genes_range <- annot_range[annot_range$type == "gene", ]
genes_range <- genes_range[
  as.character(genes_range@seqnames) == "scf_8254548"
]

#Subset MNaseSeq to export and get read coverage
MNaseSeq <- rtracklayer::import.bed(MNaseSeq_path)
MNaseSeq <- MNaseSeq[
  as.character(MNaseSeq@seqnames) == "scf_8254548"
]
covMNaseSeq <- GenomicRanges::GRanges(
  GenomicRanges::coverage(MNaseSeq), seqinfo = GenomicRanges::seqinfo(organism_genome)
  )

#Subset ChIPSeq to export and get read coverage
ChIPSeq <- rtracklayer::import.bed(ChIPSeq_path)
ChIPSeq <- ChIPSeq[
```

```
  as.character(ChIPSeq@seqnames) == "scf_8254548"
]
covChIPSeq <- GenomicRanges::GRanges(
  GenomicRanges::coverage(ChIPSeq), seqinfo = GenomicRanges::seqinfo(organism_genome)
  )
```

Then we use the `ExportFilesForGViz` function to export 4 files.

```
# #Gviz file making for vignette (not run)
ExportFilesForGViz(
  cFileNames = c(
    ipdRatio6mABigwig,
    GenesBam,
    covMNaseSeqBigwig,
    covChIPSeqBigwig
  ),
  listObjects = list(
    SubsetModGFF_granges,
    genes_range,
    covMNaseSeq,
    covChIPSeq
  ),
  cFileFormats = c("bw", "bam", "bw", "bw"),
  lBigwigParametersByStrand = c(TRUE, NA, TRUE, TRUE),
  cBigwigParameters = c("ipdRatio", NA, "score", "score"),
  cBamXaParameters = c(NA, "Name", NA, NA),
  dnastringsetGenome = organism_genome
)
```

To plot an Ideogram, you can use the `IdeogramTrack` function from *Gviz* package. But if you do not have any information on cytobands, you can use the `AdaptedIdeogramTrackWithoutBandsData` function to plot an empty representation of chromosome.

> If the chromosome names that you are using are present in the UCSC database, you can use the command `options(ucscChromosomeNames=TRUE)` before using the *Gviz* package. Otherwise, you must use `options(ucscChromosomeNames=FALSE)` command instead.

```
cContigToViz <- unique(GenomicRanges::seqnames(organism_genome_range))

# Generating Ideogram TRACK--------
options(ucscChromosomeNames = FALSE)
trackIdeogram <- AdaptedIdeogramTrackWithoutBandsData(
  grangesGenome = organism_genome_range,
  cContigToViz = cContigToViz,
  cOrgAssemblyName = "T.thermophila_June2014"
)
```

When making tracks using files generated with the `ExportFilesForGViz` function, you must add `stream = TRUE` to allow streaming when displaying tracks.

```
# Generating classic Gviz tracks

# GenomeAxis TRACK------
trackGenomeAxis <- Gviz::GenomeAxisTrack(cex = 1)

# Sequence TRACK--------
trackSequence <- Gviz::SequenceTrack(organism_genome_path,
  chromosome = cContigToViz,
  add53 = TRUE,
  complement = FALSE, cex = 0.8, stream = TRUE
)

# DATATRACK--------
trackData6mATipdRatio <- Gviz::DataTrack(ipdRatio6mABigwig,
  stream = TRUE, name = "6mAT\nipdRatio", type = "histogram",
  col.histogram = c("red"), fill = "red",
  background.title = "darkred", col = NULL
)

trackDataNuclCoverage <- Gviz::DataTrack(covMNaseSeqBigwig,
                                         stream = TRUE,
                           type="histogram",
                           name = "MNase-seq read \ncoverage",
  col.histogram = c("blue"), fill = "blue",
  background.title = "darkblue", col = NULL)

trackDataH2AZCoverage <- Gviz::DataTrack(covChIPSeqBigwig,
                                         stream = TRUE,
                           type="histogram",
                           name = "H2A.Z ChIP-seq reads \ncoverage",
  col.histogram = c("green"), fill = "green",
  background.title = "darkgreen", col = NULL)
```

To display genomic annotations (in adapted bam file; see `ExportFilesForGViz` function documentation) using streaming, you must use the `ImportBamExtendedAnnotationTrack` function as the import function (via `stream` and `importFunction` options while making the annotation track).

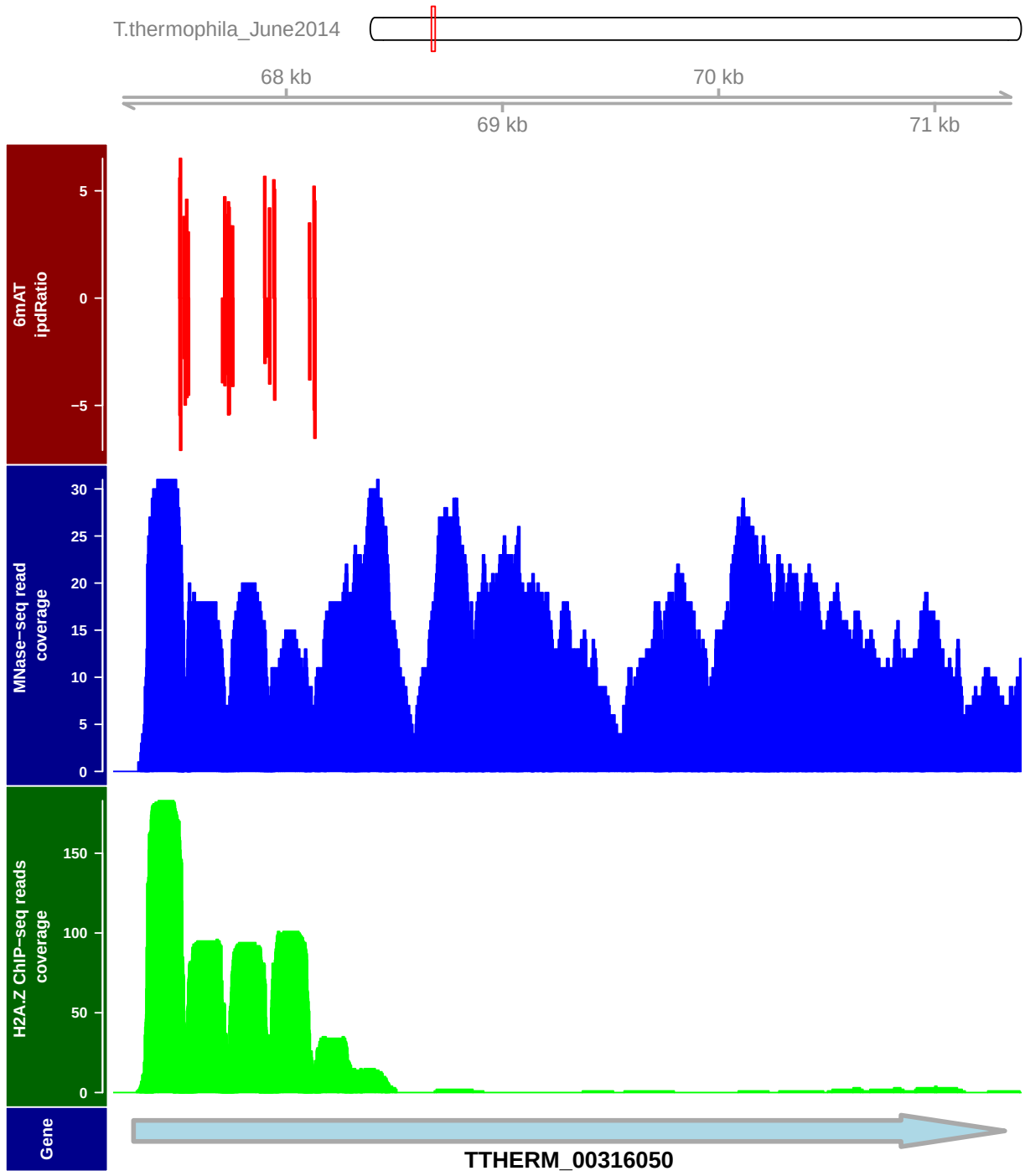
```
# Generating Annotation TRACK with streaming--------
trackAnnotation <- Gviz::AnnotationTrack(GenesBam,
  name = "Gene", stacking = "squish",
  stream = TRUE, importFunction = ImportBamExtendedAnnotationTrack,
  group = "tag", groupAnnotation = "tag",
  just.group = "below",
  fontcolor.group = "black", fontsize.group = 18,
  fill = "lightblue", shape = "fixedArrow",
  arrowHeadWidth = 50, lwd = 3,
  background.title = "darkblue"
)
```

In order to allow the names of the genomic features to be displayed, the "mapping" sub-list of the generated annotation track must be completed with the new "id" and "group" values.

```
trackAnnotation@mapping <- list(id = "tag", group = "tag")
```

Once the tracks are all generated, you can use the `plotTracks` function from *Gviz* package to display them on a chosen location (defined by the `chromosome`,`from` and `to` options).

```
# Plotting Tracks--------
Gviz::plotTracks(
  trackList = list(
    trackIdeogram, trackGenomeAxis,
    trackData6mATipdRatio,
    trackDataNuclCoverage,
    trackDataH2AZCoverage,
    trackAnnotation
  ),
  chromosome = "scf_8254548",
  from = 67200, to = 71400
)
```



Local visualisation of 6mAT sites distribution along their ipdRatio, MNase-seq coverage, and ChIP-seq coverage. Here, clusters of 6mAT signal are more frequently enriched between peaks of MNase-seq/ChIP-seq coverage downstream Transcription Start Sites.

```
sessionInfo()
#> R version 4.1.1 (2021-08-10)
#> Platform: x86_64-apple-darwin17.0 (64-bit)
#> Running under: macOS High Sierra 10.13.1
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
#>
#> locale:
#> [1] fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#> [1] DNAModAnnot_0.0.0.9018
#>
#> loaded via a namespace (and not attached):
#>   [1] colorspace_2.0-2            rjson_0.2.21
#>   [3] ellipsis_0.3.2              biovizBase_1.42.0
#>   [5] htmlTable_2.4.0             XVector_0.34.0
#>   [7] GenomicRanges_1.46.1        base64enc_0.1-3
#>   [9] dichromat_2.0-0             rstudioapi_0.13
#>  [11] bit64_4.0.5                 AnnotationDbi_1.56.2
#>  [13] fansi_1.0.0                 xml2_1.3.3
#>  [15] splines_4.1.1               R.methodsS3_1.8.1
#>  [17] cachem_1.0.6                knitr_1.37
#>  [19] Formula_1.2-4               Rsamtools_2.10.0
#>  [21] seqLogo_1.60.0              cluster_2.1.2
#>  [23] dbplyr_2.1.1                png_0.1-7
#>  [25] R.oo_1.24.0                 compiler_4.1.1
#>  [27] httr_1.4.2                  backports_1.4.1
#>  [29] assertthat_0.2.1            Matrix_1.4-0
#>  [31] fastmap_1.1.0               lazyeval_0.2.2
#>  [33] htmltools_0.5.2             prettyunits_1.1.1
#>  [35] tools_4.1.1                 gtable_0.3.0
#>  [37] glue_1.6.0                  GenomeInfoDbData_1.2.7
#>  [39] dplyr_1.0.7                 rappdirs_0.3.3
#>  [41] Rcpp_1.0.7                  Biobase_2.54.0
#>  [43] vctrs_0.3.8                 Biostrings_2.62.0
#>  [45] rtracklayer_1.54.0          xfun_0.29
#>  [47] stringr_1.4.0               lifecycle_1.0.1
#>  [49] restfulr_0.0.13             ensembldb_2.18.2
#>  [51] XML_3.99-0.8                zlibbioc_1.40.0
#>  [53] scales_1.1.1                BSgenome_1.62.0
#>  [55] VariantAnnotation_1.40.0    hms_1.1.1
#>  [57] MatrixGenerics_1.6.0        ProtGenerics_1.26.0
#>  [59] parallel_4.1.1              SummarizedExperiment_1.24.0
#>  [61] AnnotationFilter_1.18.0     RColorBrewer_1.1-2
#>  [63] yaml_2.2.1                  curl_4.3.2
#>  [65] memoise_2.0.1               gridExtra_2.3
#>  [67] ggplot2_3.3.5               biomaRt_2.50.1
```

```
#>  [69] rpart_4.1-15                latticeExtra_0.6-29
#>  [71] stringi_1.7.6               RSQLite_2.2.9
#>  [73] highr_0.9                   S4Vectors_0.32.3
#>  [75] BiocIO_1.4.0                checkmate_2.0.0
#>  [77] GenomicFeatures_1.46.3      BiocGenerics_0.40.0
#>  [79] filelock_1.0.2              BiocParallel_1.28.3
#>  [81] GenomeInfoDb_1.30.0         rlang_0.4.12
#>  [83] pkgconfig_2.0.3             matrixStats_0.61.0
#>  [85] bitops_1.0-7                evaluate_0.14
#>  [87] lattice_0.20-45             purrr_0.3.4
#>  [89] GenomicAlignments_1.30.0    htmlwidgets_1.5.4
#>  [91] bit_4.0.4                   tidyselect_1.1.1
#>  [93] magrittr_2.0.1              R6_2.5.1
#>  [95] IRanges_2.28.0              generics_0.1.1
#>  [97] Hmisc_4.6-0                 DelayedArray_0.20.0
#>  [99] DBI_1.1.2                   pillar_1.6.4
#> [101] foreign_0.8-82              survival_3.2-13
#> [103] KEGGREST_1.34.0             RCurl_1.98-1.5
#> [105] nnet_7.3-17                 tibble_3.1.6
#> [107] crayon_1.4.2                utf8_1.2.2
#> [109] BiocFileCache_2.2.0         rmarkdown_2.11
#> [111] jpeg_0.1-9                  progress_1.2.2
#> [113] grid_4.1.1                  data.table_1.14.2
#> [115] blob_1.2.2                  digest_0.6.29
#> [117] R.utils_2.11.0              stats4_4.1.1
#> [119] munsell_0.5.0               Gviz_1.38.1
```