Package 'fRagmentomics'

September 23, 2025

Title Extract Fragmentomics Features and Mutational Status

Version 0.2.8

Description A user-friendly R package that enables the characterization of each cfDNA fragment overlapping one or multiple mutations of interest, starting from a sequencing file containing aligned reads (BAM file). fRagmentomics supports multiple mutation input formats (e.g., VCF, TSV, or string "chr:pos:ref:alt" representation), accommodates one-based and zero-based genomic conventions, handles mutation representation ambiguities, and accepts any reference file and species in FASTA format. For each cfDNA fragment, fRagmentomics outputs its size, its 3' and 5' sequences, and its mutational status.

URL https://github.com/ElsaB-Lab/fRagmentomics

BugReports https://github.com/ElsaB-Lab/fRagmentomics/issues

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

biocViews Software, IndelDetection

Suggests covr, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports Biostrings, dplyr, future, future.apply, GenomeInfoDb, GenomicRanges, ggh4x, ggplot2, ggseqlogo, IRanges, magrittr, progressr, purrr, RColorBrewer,

readr,
Rsamtools,
scales,
stringr,
tibble,
tidyr

Contents

`plot_freq_barplot` *Plot Overall Nucleotide Frequency*

Description

Creates a faceted bar plot to compare the overall proportion of each nucleotide across different groups. Includes error bars (95% confidence intervals) and an optional global Chi-squared test for statistical significance.

Usage

```
plot_freq_barplot(
  df_fragments,
  end_motif_5p = "Fragment_Bases_5p",
  end_motif_3p = "Fragment_Bases_3p",
  motif_type = "Both",
  motif_size = 3,
  col_z = "Fragment_Status_Simple",
  vals_z = NULL,
  ...,
  colors_z = "Set2",
  sample_id = NA,
  output_path = NA,
  ggsave_params = list()
)
```

Arguments

`df_fragments`	The input dataframe containing fragment sequence data.
`end_motif_5p`	Character string. Column name for 5' end sequences.
`end_motif_3p`	Character string. Column name for 3' end sequences.
`motif_type`	Character string. Which ends to analyze: 'Start', 'End', or 'Both'.
`motif_size`	A single integer specifying the length of the motif to analyze.

col_z	Character string. Column name for grouping. If NULL, no grouping is applied.
vals_z	A character vector of group names from 'col_z' to include. If NULL, all unique groups in 'col_z' are used.
...	Additional arguments passed on to 'ggplot2::geom_bar()'.
colors_z	A character vector of colors for the groups, or a single string naming an RColorBrewer palette (e.g., "Set2").
sample_id	Sample identifier.
output_path	Character vector for the plot output path.
ggsave_params	A named list of arguments to be passed to 'ggplot2::ggsave()'. For example, 'list(width = 8, height = 6, units = "in", dpi = 300, bg = "white")'. If not provided, sensible defaults will be used.

Value

A 'ggplot' object.

Examples

```
## --- Create a dataset for demonstration ---
# Set a seed for reproducibility
set.seed(42)

# Helper function to generate random DNA sequences with a bias
generate_biased_dna <- function(n_seq, len, prob) {
    bases <- c("A", "C", "G", "T")
   replicate(n_seq, paste(sample(bases, len, replace = TRUE, prob = prob), collapse = ""))
}

# Create 50 "MUT" fragments with a high proportion of 'C'
df_mut <- data.frame(
    Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
    Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
    Fragment_Status_Simple = "MUT"
)

# Create 50 "WT" fragments with a high proportion of 'G'
df_wt <- data.frame(
    Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
    Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
    Fragment_Status_Simple = "WT"
)

# Combine into a single dataframe
example_df <- rbind(df_mut, df_wt)

## --- Function Calls ---

# 1. Default plot: Compares MUT vs. WT groups for 3-mers
#    from both the 5' and 3' ends.
p1 <- plot_freq_barplot(example_df)
print(p1)

# 2. Customized plot: Analyzes only the first nucleotide ('motif_size = 1')
#    of the 5' end ('motif_type = "Start"') using custom colors.
```

```
p2 <- plot_freq_barplot(
    df_fragments = example_df,
    motif_type = "Start",
    motif_size = 1,
    colors_z = c("MUT" = "#d95f02", "WT" = "#1b9e77")
)
print(p2)

# 3. Ungrouped plot: Analyzes the overall nucleotide frequency
#    across all fragments combined.
p3 <- plot_freq_barplot(example_df, col_z = NULL)
print(p3)

# 4. Plot with a subset of groups: If you had more than two groups
#    (e.g., "MUT", "WT", "AMB"), you could select specific ones to plot.
p4 <- plot_freq_barplot(
    df_fragments = example_df,
    vals_z = c("MUT", "WT")
)
print(p4)

# 5. Save the default plot to a temporary folder.
# plot_freq_barplot(
#   df_fragments = example_df,
#   sample_id = "test01_freq",
#   output_path = "test01_freq_nucleotide_frequency.png"
# )

# 6. Save a customized plot with specific dimensions.
# plot_freq_barplot(
#   df_fragments = example_df,
#   motif_type = "Start",
#   motif_size = 1,
#   sample_id = "test02_freq_custom",
#   output_path = "test02_freq_custom_nucleotide_frequency.png"
#   ggsave_params = list(width = 7, height = 5, units = "in")
# )
```

`plot_motif_barplot` *Plot 3-base motif proportions with various representations*

Description

Creates a bar plot to visualize the proportion of 3-base motifs at fragment ends. Supports grouped analysis and three different visual representations: hierarchical faceting by base, log2 fold change, or side-by-side motifs.

Usage

```
plot_motif_barplot(
  df_fragments,
  end_motif_5p = "Fragment_Bases_5p",
  end_motif_3p = "Fragment_Bases_3p",
```

```
  motif_type = "Both",
  motif_start = NULL,
  col_z = "Fragment_Status_Simple",
  vals_z = NULL,
  representation = "split_by_base",
  ...,
  colors_z = c("#FD96A9", "#E88B00", "#0D539E", "#6CAE75"),
  sample_id = NA,
  output_path = NA,
  ggsave_params = list()
)
```

Arguments

df_fragments	The input dataframe containing fragment sequence data.
end_motif_5p	Character string. Column name for 5' end sequences.
end_motif_3p	Character string. Column name for 3' end sequences.
motif_type	Character string. Which ends to analyze: 'Start', 'End', or 'Both'.
motif_start	Optional character vector ('A','C','G','T') to filter motifs by their starting base.
col_z	Character string. Column name for grouping. If NULL, no grouping is applied.
vals_z	A character vector of group names from 'col_z' to include. If NULL, all unique groups in 'col_z' are used.
representation	Character string. The type of plot to generate. • '"split_by_base"' (default): A proportion plot with hierarchical axes, splitting motifs by each base position into facets. • '"differential"': A log2 fold change plot comparing two groups. • '"split_by_motif"': A proportion plot with motifs on the x-axis, with bars for different groups placed side-by-side.
...	Additional arguments passed on to 'ggplot2::geom_bar()'.
colors_z	For the "split_by_base" plot, a character vector of 4 colors for A, C, G, T, or a single string naming an RColorBrewer palette. For other plots, a suitable palette is chosen automatically.
sample_id	Sample identifier.
output_path	Character vector for the plot output path.
ggsave_params	A named list of arguments to be passed to 'ggplot2::ggsave()'. For example, 'list(width = 8, height = 6, units = "in", dpi = 300, bg = "white")'. If not provided, sensible defaults will be used.

Value

A ggplot object.

Examples

```
## --- Create a dataset for demonstration ---
# Set a seed for reproducibility
set.seed(42)

# Helper function to generate random DNA sequences with a bias
```

```
generate_biased_dna <- function(n_seq, len, prob) {
    bases <- c("A", "C", "G", "T")
   replicate(n_seq, paste(sample(bases, len, replace = TRUE, prob = prob), collapse = ""))
}

# Create 50 "MUT" fragments with a high proportion of motifs starting with 'C'
df_mut <- data.frame(
    Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
    Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
    Fragment_Status_Simple = "MUT"
)

# Create 50 "WT" fragments with a high proportion of motifs starting with 'G'
df_wt <- data.frame(
    Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
    Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
    Fragment_Status_Simple = "WT"
)

# Combine into a single dataframe
example_df <- rbind(df_mut, df_wt)

## --- Function Calls for Each Representation ---

# 1. Hierarchical Plot (representation = "split_by_base")
# This is the default. It creates nested facets for each base position.
p1 <- plot_motif_barplot(
    df_fragments = example_df,
    representation = "split_by_base"
)
print(p1)

# You can also filter this plot to show only motifs starting with certain bases.
p1_filtered <- plot_motif_barplot(
    df_fragments = example_df,
    representation = "split_by_base",
    motif_start = c("C", "G")
)
print(p1_filtered)

# 2. Differential Plot (representation = "differential")
# This shows the log2 fold change in motif proportions between two groups.
# It requires exactly two groups specified in 'vals_z'.
p2 <- plot_motif_barplot(
    df_fragments = example_df,
    representation = "differential",
    vals_z = c("MUT", "WT")
)
print(p2)

# 3. Side-by-side Motif Plot (representation = "split_by_motif")
# This creates a more traditional bar plot with motifs on the x-axis and
# bars for each group shown side-by-side.
p3 <- plot_motif_barplot(
    df_fragments = example_df,
    representation = "split_by_motif"
)
```

```
print(p3)

# 4. Save the default hierarchical plot.
# plot_motif_barplot(
#   df_fragments = example_df,
#   sample_id = "test01_hierarchical",
#   output_path = "test01_hierarchical_motif_barplot.png"
# )

# 5. Save the differential plot with custom dimensions.
# plot_motif_barplot(
#   df_fragments = example_df,
#   representation = "differential",
#   vals_z = c("MUT", "WT"),
#   sample_id = "test02_differential",
#   output_path = "test02_differential_motif_barplot.png"
#   ggsave_params = list(width = 12, height = 8, units = "in")
# )
```

`plot_qqseqlogo_meme` *Plot sequence motif composition*

Description

Creates a sequence logo plot showing the proportion of each nucleotide at each specified position, with flexible grouping and faceting.

Usage

```
plot_qqseqlogo_meme(
  df_fragments,
  end_motif_5p = "Fragment_Bases_5p",
  end_motif_3p = "Fragment_Bases_3p",
  motif_type = "Both",
  motif_size = 3,
  col_z = "Fragment_Status_Simple",
  vals_z = NULL,
  colors_z = NULL,
  sample_id = NA,
  output_path = NA,
  ggsave_params = list()
)
```

Arguments

`df_fragments`	The input dataframe containing fragment sequence data.
`end_motif_5p`	Character string. The column name for 5' end sequences.
`end_motif_3p`	Character string. The column name for 3' end sequences.
`motif_type`	Character string. Which ends to analyze: 'Start', 'End', or 'Both'.
`motif_size`	A single integer specifying the length of the motif.

col_z	Character string. The column name for grouping/faceting. If NULL, no grouping is applied.
vals_z	A character vector of group names to include. If NULL, all groups in 'col_z' are used.
colors_z	The color scheme for nucleotides. Can be NULL (default ggseqlogo colors), a character string naming an RColorBrewer palette (e.g., "Dark2"), or a named character vector (e.g., c("A"="blue", "C"="red", ...)).
sample_id	Sample identifier.
output_path	Character vector for the plot output path.
ggsave_params	A named list of arguments to be passed to 'ggplot2::ggsave()'. For example, 'list(width = 8, height = 6, units = "in", dpi = 300, bg = "white")'. If not provided, sensible defaults will be used.

Value

A 'ggplot' object.

Examples

```
## --- Create a dataset for demonstration ---
# Set a seed for reproducibility
set.seed(42)

# Helper function to generate random DNA sequences with a bias
generate_biased_dna <- function(n_seq, len, prob) {
    bases <- c("A", "C", "G", "T")
   replicate(n_seq, paste(sample(bases, len, replace = TRUE, prob = prob), collapse = ""))
}

# Create 50 "MUT" fragments with a high proportion of 'C' at the ends
df_mut <- data.frame(
    Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
    Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.2, 0.5, 0.15, 0.15)),
    Fragment_Status_Simple = "MUT"
)

# Create 50 "WT" fragments with a high proportion of 'G' at the ends
df_wt <- data.frame(
    Fragment_Bases_5p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
    Fragment_Bases_3p = generate_biased_dna(50, 10, prob = c(0.15, 0.15, 0.5, 0.2)),
    Fragment_Status_Simple = "WT"
)

# Combine into a single dataframe
example_df <- rbind(df_mut, df_wt)

## --- Function Calls ---

# 1. Default plot: Shows a 3-base motif from both 5' and 3' ends,
#    separated by a dash, for each group ("MUT" and "WT").
p1 <- plot_qqseqlogo_meme(example_df)
print(p1)

# 2. Analyze a longer, single-end motif: Shows a 5-base motif
```

```
#    from only the 5' end ('motif_type = "Start"').
p2 <- plot_qqseqlogo_meme(
    df_fragments = example_df,
    motif_type = "Start",
    motif_size = 5
)
print(p2)

# 3. Customizing colors: Use a named RColorBrewer palette.
#    Note the separator "-" is not a nucleotide and won't be colored.
p3 <- plot_qqseqlogo_meme(
    df_fragments = example_df,
    colors_z = "Set1"
)
print(p3)

# You can also provide a named vector for full control over colors.
custom_cols <- c("A" = "#1B9E77", "C" = "#D95F02", "G" = "#7570B3", "T" = "#E7298A")
p4 <- plot_qqseqlogo_meme(
    df_fragments = example_df,
    motif_type = "Start",
    colors_z = custom_cols
)
print(p4)

# 4. Ungrouped plot: Analyzes all fragments together as a single group.
p5 <- plot_qqseqlogo_meme(example_df, col_z = NULL)
print(p5)

# 5. Save plot with default settings.
# plot_qqseqlogo_meme(
#   df_fragments = example_df,
#   sample_id = "test01_motif",
#   output_path = "test01_motif_plot.png"
# )

# 6. Save plot with custom dimensions.
# plot_qqseqlogo_meme(
#   df_fragments = example_df,
#   sample_id = "test02_motif_custom",
#   output_path = "test02_motif_custom_plot.png",
#   ggsave_params = list(width = 15, height = 10, units = "cm")
# )
```

`plot_size_distribution`

Plot Fragment Size Distribution

Description

Generates a plot visualizing the distribution of fragment lengths. It allows for grouping by a categorical variable and can represent the distribution as a histogram, a density plot, or an overlay of both. It also displays the sample size (N) for each group in the legend.

Usage

```
plot_size_distribution(
  df_fragments,
  size_col = "Fragment_Size",
  col_z = "Fragment_Status_Simple",
  vals_z = NULL,
  histo_args = list(),
  density_args = list(),
  colors_z = NULL,
  show_histogram = FALSE,
  show_density = TRUE,
  x_limits = c(0, 750),
  histogram_binwidth = 5,
  show_nuc_peaks = TRUE,
  sample_id = NA,
  output_path = NA,
  ggsave_params = list()
)
```

Arguments

df_fragments	The input dataframe containing the fragment data.
size_col	A character string specifying the name of the numeric column that contains the fragment lengths.
col_z	A character string specifying the name of the column to use for grouping the data. If NULL, no grouping is applied.
vals_z	An optional character vector to filter and display only specific groups from 'col_z'. If NULL, all groups are used.
histo_args	A named list of additional arguments to pass to 'ggplot2::geom_histogram()'.
density_args	A named list of additional arguments to pass to 'ggplot2::geom_density()'.
colors_z	A character vector of colors for the groups, or a single string naming an RColorBrewer palette.
show_histogram	A logical value. If TRUE, a histogram layer is added.
show_density	A logical value. If TRUE, a density plot layer is added.
x_limits	An optional numeric vector of length 2 to set the x-axis limits (e.g., c(0, 700)).
histogram_binwidth	A numeric value specifying the bin width for the histogram.
show_nuc_peaks	A logical value. If TRUE, adds vertical lines for nucleosome peaks.
sample_id	Sample identifier.
output_path	Character vector for the plot output path.
ggsave_params	A named list of arguments to be passed to 'ggplot2::ggsave()'. For example, 'list(width = 8, height = 6, units = "in", dpi = 300, bg = "white")'. If not provided, sensible defaults will be used.

Value

A 'ggplot' object representing the size distribution plot.

Examples

```
## --- Create a dataset for demonstration ---
# Set a seed for reproducibility
set.seed(42)

# Generate fragment sizes for two groups with different distributions
# "MUT" group: N=100, shorter fragments
mut_sizes <- rnorm(100, mean = 150, sd = 20)

# "WT" group: N=150, centered around the mononucleosome peak
wt_sizes <- rnorm(150, mean = 170, sd = 25)

# Add some larger, dinucleosomal fragments to both groups
di_nuc_sizes <- rnorm(30, mean = 330, sd = 30)

# Combine into a single dataframe
example_df_size <- data.frame(
    Fragment_Size = c(mut_sizes, wt_sizes, di_nuc_sizes),
    Fragment_Status_Simple = c(
        rep("MUT", 100),
        rep("WT", 150),
        sample(c("MUT", "WT"), 30, replace = TRUE)
    )
)
# Ensure all fragment sizes are positive
example_df_size <- example_df_size[example_df_size$Fragment_Size > 0, ]

## --- Plotting Examples ---

# 1. Default plot: A grouped density plot with nucleosome peaks shown.
p1 <- plot_size_distribution(example_df_size)
print(p1)

# 2. Histogram plot: Show distributions as histograms instead of density curves.
#    We add transparency (alpha) so overlapping bars are visible.
p2 <- plot_size_distribution(
    df_fragments = example_df_size,
    show_histogram = TRUE,
    show_density = FALSE,
    histo_args = list(alpha = 0.6)
)
print(p2)

# 3. Combined plot: Overlay both density curves and histograms.
p3 <- plot_size_distribution(
    df_fragments = example_df_size,
    show_histogram = TRUE,
    show_density = TRUE,
    histo_args = list(alpha = 0.4)
)
print(p3)

# 4. Ungrouped and customized plot: Analyze all fragments together,
#    zoom in on the x-axis, and hide the nucleosome peak lines.
p4 <- plot_size_distribution(
    df_fragments = example_df_size,
```

```
    col_z = NULL,
    x_limits = c(50, 400),
    show_nuc_peaks = FALSE
)
print(p4)

# 5. Save plot with default settings (8x6 inches).
# plot_size_distribution(
#   df_fragments = example_df_size,
#   sample_id = "test01",
#   output_path = "test01_size_distribution.png"
# )

# 6. Save plot with custom dimensions in centimeters.
# plot_size_distribution(
#   df_fragments = example_df_size,
#   sample_id = "test02_custom",
#   output_path = "test02_custom_size_distribution.png",
#   ggsave_params = list(width = 20, height = 10, units = "cm", bg = "white")
# )
```

`run_fRagmentomics` *Analyze fragments*

Description

This is the main function of the package. It provides an end-to-end pipeline for analyzing the allelic state of individual DNA fragments covering specific genomic variants. It takes a list of mutations and an aligned sequencing file (BAM) as input, processes each fragment in parallel, and returns a detailed data frame of results.

Usage

```
run_fRagmentomics(
  mut,
  bam,
  fasta,
  sample_id = NA,
  neg_offset_mate_search = -1000,
  pos_offset_mate_search = 1000,
  one_based = TRUE,
 flag_bam_list = list(isPaired = TRUE, isProperPair = NA, isUnmappedQuery = FALSE,
  hasUnmappedMate = NA, isMinusStrand = NA, isMateMinusStrand = NA, isFirstMateRead =
  NA, isSecondMateRead = NA, isSecondaryAlignment = FALSE, isSupplementaryAlignment =
    FALSE, isNotPassingQualityControls = NA, isDuplicate = FALSE),
  report_tlen = FALSE,
  report_softclip = FALSE,
  report_5p_3p_bases_fragment = 5,
  remove_softclip = FALSE,
  retain_fail_qc = FALSE,
  tmp_folder = tempdir(),
  output_path = NA,
```

```
  n_cores = 8
)
```

Arguments

mut	Path to a .vcf or .tsv file or string representation chr:pos:ref:alt of a mutation.
bam	Path to a BAM file.
fasta	Path to the FASTA file for the reference sequence used for generating the BAM file.
sample_id	Sample identifier.
neg_offset_mate_search	Integer. Use in read_bam. Represents the number of nucleotides to extend upstream (negative direction) from the position of interest when querying the BAM file with Rsamtools. his extension ensures that paired reads are retrieved, even if only one mate overlaps the queried position.
pos_offset_mate_search	Integer. Use in read_bam.
one_based	Boolean. TRUE if fasta is in one based. False if in 0 based.
flag_bam_list	A named list of logicals for filtering reads based on their SAM flag NA = Filter is ignored, TRUE = The read MUST have this flag, FALSE = The read MUST NOT have this flag.
report_tlen	Boolean. Whether to include the TLEN (template length) information in the output.
report_softclip	Boolean. Whether to include the number of soft-clipped bases at the fragment extremities in the output.
report_5p_3p_bases_fragment	Integer. Whether to include N fragment extremity bases in the output.
remove_softclip	Boolean. If set to TRUE, trim soft-clipped bases from the 5' end of Read 5p and from the 3' end of Read 3p.
retain_fail_qc	Boolean. If set to TRUE, retain fragments that failed the various quality checks in the output.
tmp_folder	Character vector for the temporary folder path.
output_path	Character vector for the fragmentomics table output path.
n_cores	Number of cores for parallel computation.

Details

The function executes a multi-step workflow for each variant provided in the 'mut' input:

1. **Input Validation**: All parameters are rigorously checked for correctness (e.g., file existence, data types). Required file indices ('.bai', '.fai') are created automatically if missing.
2. **Variant Normalization**: The input variants are parsed and normalized into a canonical, left-aligned representation using a combination of VCF-style indel padding and the external 'bcftools norm' command.
3. **BAM Read Extraction**: For each normalized variant, the function efficiently queries the BAM file to retrieve all read pairs that cover the genomic locus.

4. **Parallel Fragment Processing**: The core analysis is performed in parallel using the 'future' framework. Each unique DNA fragment is processed by the 'extract_fragment_features' worker function to determine its size, quality metrics, and mutation status (e.g., "MUT", "WT", "AMB", "N/I").
5. **VAF Calculation**: After all fragments for a variant are processed, the Variant Allele Frequency (VAF) is calculated.
6. **Output Generation**: Results from all variants are aggregated into a single data frame. If a value for `output_path` is provided, this data frame is also written to a tab-separated file.

Value

A dataframe containing extracted fragment-level information.

Examples

```
# --- 1. Locate Example Files ---
# The package includes small example files to demonstrate its functionality.
# We locate them using system.file().
mut_file <- system.file(
  "extdata", "mutations_cfdna-test-01_chr1_27433000_27435000.tsv",
  package = "fRagmentomics"
)
bam_file <- system.file(
  "extdata", "cfdna-test-01_chr1_27433000_27435000.bam",
  package = "fRagmentomics"
)
fasta_file <- system.file(
  "extdata", "hg19_chr1_27433000_27435000.fa",
  package = "fRagmentomics"
)

# --- 2. Run the Analysis ---
# This single call runs the full analysis pipeline on the example data.
# The output file is written to a temporary location to avoid cluttering
# the working directory. We use n_cores = 1L for examples.
results <- run_fRagmentomics(
  mut = mut_file,
  bam = bam_file,
  fasta = fasta_file,
  sample_id = "cfdna-test-01",
  output_path = tempdir(),
  n_cores = 1L
)

# --- 3. View the Results ---
# Print the first few rows of the output data frame to see the results.
print(head(results))
```

Index