Package 'GencoDymo2'

May 15, 2025

Type Package

Title Comprehensive Analysis of 'GENCODE' Annotations and Splice Site Motifs

Version 1.0.1

Maintainer Monah Abou Alezz <aboualezz.monah@hsr.it>

Description A comprehensive suite of helper functions designed to facilitate the analysis of genomic annotations from the 'GENCODE' database <https://www.gencodegenes.org/>, supporting both human and mouse genomes. This toolkit enables users to extract, filter, and analyze a wide range of annotation features including genes, transcripts, exons, and introns across different 'GENCODE' releases. It provides functionality for cross-version comparisons, allowing researchers to systematically track annotation updates, structural changes, and feature-level differences between releases. In addition, the package can generate high-quality FASTA files containing donor and acceptor splice site motifs, which are formatted for direct input into the 'MaxEntScan' tool (Yeo and Burge, 2004 <doi:10.1089/1066527041410418>), enabling accurate calculation of splice site strength scores.

License GPL (>= 3)

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

URL https://github.com/monahton/GencoDymo2,
https://monahton.github.io/GencoDymo2/

BugReports https://github.com/monahton/GencoDymo2/issues

Imports Biostrings (>= 2.72.1), BSgenome (>= 1.66.3), BSgenome.Hsapiens.UCSC.hg38 (>= 1.4.5), data.table (>= 1.17.0), dplyr (>= 1.1.4), GenomicRanges (>= 1.56.2), methods (>= 4.4.0), plotrix (>= 3.8.4), progress (>= 1.2.3), RCurl (>= 1.98.1.17), rtracklayer (>= 1.64.0), tidyr (>= 1.3.1), tools (>= 4.4.0), utils (>= 4.4.0)

Suggests devtools (>= 2.4.5), IRanges (>= 2.38.1), knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Monah Abou Alezz [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-2006-4250>),
Lorenzo Salviati [ctb],
Roberta Alfieri [ctb],
Silvia Bione [ctb]

Repository CRAN

Date/Publication 2025-05-15 14:00:10 UTC

Contents

`assign_splice_sites` *Assign intron donor and acceptor splice sites consensus*

Description

This function takes a data frame of intron coordinates and a genome sequence (ideally human or mouse) and returns a data frame with two additional columns for the donor and acceptor splice site consensus sequences. It prepares the donor and acceptor sequences based on the provided intron coordinates and the specified genome (e.g., human hg38), making it useful for downstream analysis of splicing events.

Usage

```
assign_splice_sites(input, genome = BSgenome.Hsapiens.UCSC.hg38, verbose = TRUE)
```

Arguments

`input`	A data frame containing intron coordinates with the following columns: • `seqnames`: The chromosome name. • `intron_start`: The start position of the intron. • `intron_end`: The end position of the intron. • `strand`: The strand on which the intron is located (`+` or `-`). • `transcript_id`: The ID of the transcript to which the intron belongs. • `intron_number`: The number of the intron within the transcript. • `gene_name`: The name of the gene. • `gene_id`: The gene ID.
`genome`	The genome sequence (BSgenome object) for the species. Default is the human genome (hg38). This object is required for extracting the consensus sequences from the genome at the specified intron positions.
`verbose`	Logical. If TRUE, the function prints progress messages while preparing the splice site data. Default is TRUE.

Details

This function performs the following steps:

- First, it prepares the splice site sequences for both donor and acceptor sites by calculating their positions based on the strand orientation and intron coordinates. The donor splice site is typically located at the 5' end of the intron, while the acceptor splice site is at the 3' end.
- The function utilizes the `getSeq` function from the `BSgenome` package to extract the nucleotide sequences for both donor and acceptor sites from the specified genome (default is hg38 for humans).
- The resulting sequences are added as new columns (`donor_ss` and `acceptor_ss`) to the original input data frame.
- The final data frame includes the splice site sequences for each intron, allowing for analysis of splicing efficiency or identification of consensus motifs.

Value

A data frame containing the original intron data, with two additional columns:

- `donor_ss`: The donor splice site consensus sequence for each intron.
- `acceptor_ss`: The acceptor splice site consensus sequence for each intron.

See Also

`extract_introns`, `find_cryptic_splice_sites`

Examples

```
suppressPackageStartupMessages(library(BSgenome.Hsapiens.UCSC.hg38))
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
introns_df <- extract_introns(gtf_v1)
result <- assign_splice_sites(introns_df, genome = BSgenome.Hsapiens.UCSC.hg38)
```

---

calculate_gc_content *Calculate GC Content of Genomic Features*

Description

Computes the GC content percentage for exons or introns using genome sequence data from a BSgenome object.

Usage

```
calculate_gc_content(input, genome, verbose = TRUE)
```

Arguments

input	A data frame containing genomic features (exons/introns) with seqnames, start, end, and strand columns.
genome	A BSgenome object representing the reference genome (e.g., BSgenome.Hsapiens.UCSC.hg38).
verbose	A logical indicating whether to print progress messages. Defaults to TRUE.

Details

This function extracts the DNA sequence for each feature using genomic coordinates, then calculates the proportion of G and C nucleotides. Requires a BSgenome package for the relevant genome.

Value

The input data frame with an additional gc_content column containing GC percentages for each feature.

Examples

```
library(BSgenome.Hsapiens.UCSC.hg38)
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
gtf_with_gc <- calculate_gc_content(gtf_v1, genome = BSgenome.Hsapiens.UCSC.hg38, verbose = FALSE)
```

`classify_exons` *Classify Exons by Their Relative Transcript Position*

Description

Categorizes each exon in a transcript as single, first, inner, or last based on its position. This classification helps in analyzing transcript structure and splicing patterns.

Usage

```
classify_exons(input, verbose = TRUE)
```

Arguments

`input`	A character string specifying the path to a GTF/GFF3 file or a data frame containing GTF-formatted data. The file should follow the GENCODE format conventions.
`verbose`	A logical indicating whether to display progress messages and exon counts. Defaults to `TRUE`.

Details

The function processes the input GTF data to:

1. Load the GTF file (if a path is provided) or use the provided data frame.
2. Filter entries to include only exons.
3. For each transcript, count the total exons and determine each exon's position.
4. Classify exons based on their position and the total count.

If `verbose = TRUE`, the function prints counts of each exon type.

Value

A data frame identical to the input but with an additional column `EXON_CLASSIFICATION`. This column contains one of the following values for each exon:

- `"single_exon"`: The transcript contains only one exon.
- `"first_exon"`: The first exon in a multi-exon transcript.
- `"last_exon"`: The last exon in a multi-exon transcript.
- `"inner_exon"`: Exons between the first and last in multi-exon transcripts.

Examples

```
# Example 1: Using the provided example GTF files
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- classify_exons(file_v1)

# Example 2: Using a pre-loaded data frame
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
classified_data_v1 <- classify_exons(gtf_v1)
```

`compare_release` *Compare Annotation Counts Between Two GENCODE Releases*

Description

Compares the number of annotated genomic elements (genes, transcripts, exons, introns) between two specified GENCODE releases. The function accepts input data as either file paths (in GTF/GFF format) or pre-loaded data frames. It computes the absolute difference (delta), the percentage change relative to a chosen baseline, and determines the direction of change (increase or decrease).

Usage

```
compare_release(input1, input2, type, gene_type = NULL, baseline = "count2")
```

Arguments

`input1`	A character string specifying the file path to a GTF/GFF file from the first GENCODE release, or a data frame containing the annotation data.
`input2`	A character string specifying the file path to a GTF/GFF file from the second GENCODE release, or a data frame containing the annotation data.
`type`	A character string indicating the type of genomic element to compare. Valid options are `"gene"`, `"transcript"`, `"exon"`, or `"intron"`.
`gene_type`	An optional character string specifying a particular gene biotype to filter comparisons (e.g., `"protein_coding"`, `"lncRNA"`). If `NULL` (default), all gene types are included.
`baseline`	A character string defining the baseline for calculating percentage change. Options include: • `"count1"`: Uses the count from the first input (release) as the baseline. • `"count2"`: Uses the count from the second input (release) as the baseline (default). • `"average"`: Uses the average of the counts from both inputs as the baseline.

Details

This function processes two GENCODE releases to compare annotation counts for a specified genomic element type. Key steps include:

1. Input Handling: If inputs are file paths, they are loaded into data frames using the `load_file` function. Data frames are used directly.
2. Element Filtering: If `gene_type` is specified, annotations are filtered to include only that gene biotype.
3. Count Calculation: The number of elements (genes, transcripts, etc.) of the specified type is counted in each release.
4. Delta and Percentage: The absolute difference (delta) and percentage change are calculated based on the chosen baseline.
5. Direction Determination: The direction of change is determined by comparing counts between the two releases.

The function provides both numerical results and a formatted console output highlighting key metrics.

Value

A list with the following elements:

- `delta`: The absolute difference in the number of annotations.
- `percentage`: The percentage change relative to the selected baseline.
- `direction`: A string indicating the direction of the change ("increase", "decrease", or "no change").

See Also

`load_file`, `get_gtf`, `get_gff3`.

Examples

```
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
file_v2 <- system.file("extdata", "gencode.v2.example.gtf.gz", package = "GencoDymo2")
# Example 1: Using data frames with the provided example GTF files
gtf_v1 <- load_file(file_v1)
gtf_v2 <- load_file(file_v2)
comparison <- compare_release(gtf_v1, gtf_v2, type = "gene")
```

df_to_fasta *Convert Data Frame to FASTA File*

Description

Converts a data frame containing sequence IDs and sequences into a FASTA-formatted file, optionally compressed as gzip.

Usage

```
df_to_fasta(df, id_col, seq_col, output_file = NULL, gzip = TRUE, verbose = TRUE)
```

Arguments

`df`	A data frame with at least two columns: one for sequence IDs and one for sequences.
`id_col`	A character string specifying the column name containing sequence IDs.
`seq_col`	A character string specifying the column name containing sequence data.
`output_file`	A character string specifying the output file path. If NULL, the function will stop with an informative message.
`gzip`	A logical indicating whether to compress the output as a gzip file. Defaults to `TRUE`.
`verbose`	A logical indicating whether to print progress messages. Defaults to `TRUE`.

Details

This function efficiently writes large sequence datasets to FASTA format, handling compression and progress reporting. It validates input columns and manages memory by processing data in chunks.

Value

No return value. Writes a FASTA file to the specified path.

Examples

```
temp_dir <- tempdir()
temp_output <- file.path(temp_dir, "output.fa.gz")
seq_data <- data.frame(
  transcript_id = c("ENST0001", "ENST0002"),
  sequence = c("ATGCTAGCTAG", "GCTAGCTAGCT")
)
df_to_fasta(seq_data, "transcript_id", "sequence", temp_output)
```

`eliminate_redundant_elements` *Eliminate Redundant Genomic Elements*

Description

Removes redundant genomic elements (exons or introns) from a data frame, ensuring each element is uniquely represented. Redundancy is determined by genomic coordinates and gene ID.

Usage

```
eliminate_redundant_elements(input, element_type = "exon")
```

Arguments

`input`	A data frame containing genomic element coordinates (exons or introns) with columns `seqnames`, `start`, `end`, and `gene_id`.
`element_type`	The type of genomic element to process. Valid options are `"exon"` (default) or `"intron"`.

Details

This function uses genomic coordinates (chromosome, start, end) and gene ID to identify and remove duplicate entries. For exons, coordinates are directly compared. For introns, coordinates are derived from `intron_start` and `intron_end` columns (check `extract_introns` function for more details)

Value

A data frame with redundant elements removed, retaining only unique entries based on coordinates and gene ID.

Examples

```
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
# Eliminate redundant exons
nonredundant_exons <- eliminate_redundant_elements(gtf_v1, element_type = "exon")
```

extract_cds_sequences *Extract Coding Sequences (CDS) from GTF Annotations*

Description

Extracts CDS regions from a GTF annotation file or data frame using genomic coordinates and retrieves corresponding DNA sequences from a BSgenome reference.

Usage

```
extract_cds_sequences(input, genome, save_fasta, output_file, verbose)
```

Arguments

`input`	A character string (GTF file path) or data frame containing CDS annotations.
`genome`	A BSgenome object for the relevant genome. Defaults to human (hg38).
`save_fasta`	A logical indicating whether to save sequences to a FASTA file. Defaults to `FALSE`.
`output_file`	A character string specifying the FASTA output path. If `NULL`, uses "CDS.fa".
`verbose`	A logical indicating whether to print progress messages. Defaults to `TRUE`.

Details

This function processes CDS entries from the input GTF, extracts their sequences from the reference genome, and optionally saves them in FASTA format. Useful for downstream analyses like protein translation.

Value

A data frame containing CDS annotations with corresponding sequences. If `save_fasta = TRUE`, also writes a FASTA file.

Examples

```
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
# Human CDS extraction
suppressPackageStartupMessages(library(BSgenome.Hsapiens.UCSC.hg38))
suppressPackageStartupMessages(library(GenomicRanges))
gtf_granges <- GRanges(gtf_v1)
cds_seqs <- extract_cds_sequences(gtf_granges, BSgenome.Hsapiens.UCSC.hg38, save_fasta = FALSE)
```

extract_element_by_strand
Extract Genomic Elements by Strand

Description

Filters a data frame or GTF/GFF3 file to extract specific genomic elements (genes, transcripts, exons, or introns) located on a specified DNA strand (+ or -).

Usage

```
extract_element_by_strand(input, type, strand, verbose = TRUE)
```

Arguments

`input`	A data frame derived from a GTF/GFF3 file or containing genomic annotations.
`type`	A character string specifying the type of element to extract. Valid options: `"gene"`, `"transcript"`, `"exon"`, `"intron"`.
`strand`	A character string indicating the DNA strand to filter. Valid options: `"+"` (forward) or `"-"` (reverse).
`verbose`	A logical indicating whether to print the count of extracted elements. Defaults to `TRUE`.

Details

This function filters the input data based on the `type` and `strand` columns. It is useful for strand-specific analyses, such as studying antisense transcripts or strand-biased genomic features.

Value

A data frame containing only the elements of the specified type located on the chosen strand.

Examples

```
# Example 1: Extract genes on the forward strand using gencode.v1
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
forward_genes_v1 <- extract_element_by_strand(gtf_v1, type = "gene", strand = "+")

# Example 2: Extract exons on the reverse strand using gencode.v1
reverse_exons_v1 <- extract_element_by_strand(gtf_v1, type = "exon", strand = "-")
```

extract_introns *Extract Intron Coordinates from GENCODE Annotations*

Description

Processes a GTF file or data frame to extract intron coordinates, including their genomic positions, transcript associations, and metadata. The function handles both positive and negative strands, ensuring correct orientation of intron boundaries. It is designed to work with GENCODE-formatted annotations.

Usage

```
extract_introns(input, verbose = TRUE)
```

Arguments

`input`	A character string specifying the file path to a GTF/GFF3 file, or a data frame containing GTF data. The input must include columns: `seqnames`, `start`, `end`, `strand`, `type`, and `transcript_id`.
`verbose`	Logical. If `TRUE` (default), progress messages are printed during execution.

Details

1. Input Handling: Loads the GTF file if a path is provided; uses the data frame directly if supplied.
2. Exon Filtering: Extracts exon records and classifies them (e.g., first/last exon) if needed.
3. Multi-Exon Transcripts: Removes single-exon transcripts to focus on spliced transcripts.
4. Intron Calculation: Determines intron coordinates by identifying gaps between consecutive exons, adjusting for strand orientation.
5. Output: Returns a data frame with intron coordinates and metadata, sorted by gene and position.

Value

A data frame with the following columns:

- `seqnames`: Chromosome or scaffold name.
- `intron_start`, `intron_end`: Genomic start/end positions of the intron.
- `width`: Length of the intron (intron_end - intron_start + 1).
- `gene_id`, `transcript_id`, `intron_number`: Gene/transcript identifiers and intron position within the transcript.
- Additional metadata columns from the original GTF (e.g., `gene_name`).

See Also

`load_file`, `classify_exons`

Examples

```
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")

# From a pre-loaded data frame
gtf_v1 <- load_file(file_v1)
introns <- extract_introns(gtf_v1)
```

`extract_single_exon` *Identify Single-Exon Genes/Transcripts in GENCODE Data*

Description

This function identifies single-exon genes or transcripts from a genomic annotation dataset, typically obtained from the GENCODE database. It processes input data, either from a file or an already loaded data frame, and returns either single-exon genes or single-exon transcripts based on the user's selection. The function works seamlessly with GTF/GFF files or data frames that include the standard GENCODE annotation fields. This is especially useful for identifying genes or transcripts that do not undergo splicing.

Usage

```
extract_single_exon(input, level = "gene", output_file = NULL)
```

Arguments

`input`	Either a file path to a GTF/GFF3 file or a data frame representing genomic annotations. The input data should contain at least the columns `type`, `gene_id`, `transcript_id`, and `exon_number`. If a file path is provided, it is assumed to be a GTF/GFF file.
`level`	A character string specifying the level of analysis. It should be either `"gene"` or `"transcript"`. The default is "gene". The selection allows users to specify whether they want to identify single-exon genes or single-exon transcripts in the annotation.
`output_file`	Optional. A file path to save the results as a tab-separated file. If not provided, the results will not be saved. The file will include the IDs of the single-exon genes or transcripts along with their respective exon IDs. This feature is useful for exporting results for further analysis or reporting purposes.

Details

1. Input Validation: Checks for required columns and valid `level` specification.
2. Exon Counting: Groups exons by gene/transcript and counts occurrences.
3. Single-Exon Filtering: Retains only entities with exactly one exon.
4. Output: Returns filtered results; optionally writes to file.

Value

A data frame containing the IDs of single-exon genes or transcripts based on the selected level. The returned data frame contains the following columns:

- `gene_id` or `transcript_id`: The IDs of the single-exon genes or transcripts.
- `exon_count`: The count of exons for each entity (always 1 for single-exon entities).
- `exon_id`: The ID of the exon associated with the single-exon gene or transcript.

The data frame can then be used for downstream analysis, such as identifying unique single-exon genes or transcripts, or for exporting to a file.

See Also

`extract_introns`, `load_file`

Examples

```
# Example input data frame
input <- data.frame(
  type = c("exon", "exon", "exon", "exon", "exon"),
  gene_id = c("gene1", "gene1", "gene2", "gene3", "gene4"),
  transcript_id = c("tx1", "tx1", "tx2", "tx3", "tx4"),
  exon_number = c(1, 2, 1, 1, 1),
  exon_id = c("e1", "e2", "e1", "e1", "e1")
)

# Identify single-exon genes
single_exon_genes <- extract_single_exon(input, level = "gene")
print(single_exon_genes)

# Identify single-exon transcripts
single_exon_transcripts <- extract_single_exon(input, level = "transcript")
print(single_exon_transcripts)
```

`extract_ss_motif` *Extract Splice Site Motifs for MaxEntScan Analysis (5' or 3')*

Description

This function extracts splice site motifs (5' splice site (5ss) or 3' splice site (3ss)) from a genomic dataset. It retrieves the donor or acceptor splice site motifs for each intron, based on the strand orientation, and compiles them into a FASTA file, which can be used for further analysis (e.g., MaxEntScan).

Usage

```
extract_ss_motif(input, genome, type, verbose, save_fasta, output_file)
```

Arguments

`input`	A data frame containing genomic information with the following required columns: • `seqnames`: Chromosome or scaffold names. • `strand`: Strand orientation, either '+' or '-'. • `intron_start`: Start position of the intron. • `intron_end`: End position of the intron. • `transcript_id`: Identifier for the transcript. • `intron_number`: Identifier for the intron within the transcript.
`genome`	A genome object from the BSgenome package (default is `BSgenome.Hsapiens.UCSC.hg38`).
`type`	A string indicating which splice site motif to extract. One of `"5ss"` (donor splice site) or `"3ss"` (acceptor splice site).
`verbose`	Logical; if `TRUE`, progress messages will be printed. Default is `TRUE`.
`save_fasta`	Logical; if `TRUE`, a FASTA file will be saved containing the extracted motifs. Default is `FALSE`.
`output_file`	A string specifying the output file path and name for the FASTA file. If `NULL`, a default name will be used (either "5ss_motif_fasta.fa" or "3ss_motif_fasta.fa").

Details

This function performs the following steps:

- Based on the `type` argument, the function prepares coordinates for extracting either donor (5ss) or acceptor (3ss) splice site motifs, adjusting the motif start and end positions depending on the strand orientation.
- The motif sequences are then extracted from the specified genome using the `getSeq` function from the BSgenome package.
- If `save_fasta` is `TRUE`, a FASTA file is generated containing the extracted motifs, with transcript IDs and intron numbers used as FASTA headers.

Value

A data frame with:

- `donor_ss_motif` or `acceptor_ss_motif`: 9bp (5' ss) or 23bp (3' ss) sequence.
- Genomic coordinates and transcript metadata.

See Also

`assign_splice_sites`, `df_to_fasta`

Examples

```
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
introns <- extract_introns(gtf_v1)
suppressPackageStartupMessages(library(BSgenome.Hsapiens.UCSC.hg38))
```

```
# Extract donor splice site motifs
motifs_df <- extract_ss_motif(introns, BSgenome.Hsapiens.UCSC.hg38, "5ss", verbose = FALSE)

# Extract acceptor splice site motifs without saving the FASTA file
motifs_df <- extract_ss_motif(introns, BSgenome.Hsapiens.UCSC.hg38, "3ss", verbose = FALSE)
```

---

find_cryptic_splice_sites

Identify Potential Cryptic Splice Sites.

Description

This function identifies potential cryptic splice sites by comparing sequence motifs in introns to canonical splice site motifs (donor and acceptor). Cryptic splice sites are those that do not match the canonical donor (GT) or acceptor motifs (AG). It compares the identified splice sites with the provided canonical motifs and flags the sites that differ from the canonical patterns, making it useful for studying aberrant splicing events.

Usage

```
find_cryptic_splice_sites(input, genome, canonical_donor, canonical_acceptor, verbose)
```

Arguments

`input`	A data frame containing intron coordinates, ideally generated by `extract_introns()` and `assign_splice_sites()`. Must contain columns: `seqnames`, `intron_start`, `intron_end`, `strand`, `transcript_id`, `intron_number`, `gene_name`, `gene_id`, `donor_ss` and `acceptor_ss`.
`genome`	A BSgenome object representing the genome sequence. This is used to extract the sequence for each intron to identify splice sites.
`canonical_donor`	A character vector of canonical donor splice site motifs. Default is `c("GT")`.
`canonical_acceptor`	A character vector of canonical acceptor splice site motifs. Default is `c("AG")`.
`verbose`	Logical; if `TRUE`, progress messages are printed. Default is `TRUE`.

Details

This function performs the following steps:

- It assigns donor and acceptor splice sites to each intron using the `assign_splice_sites` function.
- It compares the identified donor and acceptor splice sites against the provided canonical motifs (`GT` for donor and `AG` for acceptor by default). If the splice site sequences do not match the canonical motifs, they are flagged as cryptic.

- The function returns a data frame with the same intron information, including additional columns `cryptic_donor` and `cryptic_acceptor` indicating whether the splice sites are cryptic.
- The progress of the function is printed if the `verbose` argument is set to `TRUE`, showing also the total number of cryptic donor and acceptor sites and their respective percentages.

Value

The input data frame with two logical columns:

- `cryptic_donor`: `TRUE` if donor site is non-canonical.
- `cryptic_acceptor`: `TRUE` if acceptor site is non-canonical.

See Also

`assign_splice_sites`, `extract_ss_motif`

Examples

```
suppressPackageStartupMessages(library(BSgenome.Hsapiens.UCSC.hg38))
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
introns_df <- extract_introns(gtf_v1)
introns_ss <- assign_splice_sites(introns_df, genome = BSgenome.Hsapiens.UCSC.hg38)
cryptic_sites <- find_cryptic_splice_sites(introns_ss, BSgenome.Hsapiens.UCSC.hg38)
head(cryptic_sites)
```

`get_gff3` *Download GFF3 File from the GENCODE Database*

Description

Downloads a GFF3 file for a specified species, release version, and annotation type from the GENCODE database. The file is saved to a user-specified directory or the current working directory by default.

Usage

```
get_gff3(species, release_version, annotation_type, dest_folder)
```

Arguments

`species` A character string indicating the species. Supported values are:
- "human"
- "mouse"

`release_version`
A character string specifying the release version. Options include:

- "latest_release": Fetches the latest release for the species.
- "release_X": Specific release version for human (e.g., "release_42").
- "release_MX": Specific release version for mouse (e.g., "release_M36").

`annotation_type`
A character string specifying the annotation type. Supported values include:

- "annotation.gff3.gz"
- "basic.annotation.gff3.gz"
- "chr_patch_hapl_scaff.annotation.gff3.gz"
- "chr_patch_hapl_scaff.basic.annotation.gff3.gz"
- "long_noncoding_RNAs.gff3.gz"
- "primary_assembly.annotation.gff3.gz"
- "primary_assembly.basic.annotation.gff3.gz"
- "tRNAs.gff3.gz"
- "polyAs.gff3.gz"

`dest_folder` A character string specifying the destination folder. Defaults to the current working directory.

Details

The function dynamically determines the correct file URL based on the provided parameters and downloads the GFF3 file to the desired location. If "latest_release" is specified for `release_version`, the function will first determine the latest available release using `get_latest_release()`.

Value

A character string specifying the full path of the downloaded GFF3 file.

Examples

```
# Download the latest human GTF file with primary assembly annotations into a temp directory
temp_dir <- tempdir()
gff3_file <- get_gff3(
  species = "human",
  release_version = "latest_release",
  annotation_type = "primary_assembly.basic.annotation.gff3.gz",
  dest_folder = temp_dir
)
print(gff3_file)

# Download a specific mouse release with long noncoding RNA annotations into a temp directory
temp_dir <- tempdir()
gff3_file <- get_gff3(
  species = "mouse",
  release_version = "release_M36",
  annotation_type = "long_noncoding_RNAs.gff3.gz",
  dest_folder = temp_dir
)
print(gff3_file)
```

`get_gtf` *Download GTF File from the GENCODE Database*

Description

Downloads a GTF file for a specified species, release version, and annotation type from the GENCODE database. The file is saved to a user-specified directory or the current working directory by default.

Usage

```
get_gtf(species, release_version, annotation_type, dest_folder)
```

Arguments

`species` A character string indicating the species. Supported values are:

- "human"
- "mouse"

`release_version` A character string specifying the release version. Options include:

- "latest_release": Automatically fetches the latest release for the specified species.
- "release_X": Specific human release (e.g., "release_47").
- "release_MX": Specific mouse release (e.g., "release_M36").

`annotation_type` A character string specifying the annotation type. Valid options are:

- "annotation.gtf.gz"
- "basic.annotation.gtf.gz"
- "chr_patch_hapl_scaff.annotation.gtf.gz"
- "chr_patch_hapl_scaff.basic.annotation.gtf.gz"
- "long_noncoding_RNAs.gtf.gz"
- "primary_assembly.annotation.gtf.gz"
- "primary_assembly.basic.annotation.gtf.gz"
- "tRNAs.gtf.gz"
- "polyAs.gtf.gz"

`dest_folder` A character string specifying the destination folder where the file will be downloaded. Defaults to the current working directory.

Details

The function dynamically determines the correct file URL based on the provided parameters and downloads the GTF file to the desired location. If "latest_release" is specified for `release_version`, the function will first determine the latest available release using `get_latest_release()`.

Value

A character string specifying the path to the downloaded GTF file.

Examples

```
# Download the latest human GTF file with primary assembly annotations into a temp directory
temp_dir <- tempdir()
gtf_file <- get_gtf(
  species = "human",
  release_version = "latest_release",
  annotation_type = "primary_assembly.basic.annotation.gtf.gz",
  dest_folder = temp_dir
)
print(gtf_file)

# Download a specific mouse release with long noncoding RNA annotations into a temp directory
temp_dir <- tempdir()
gtf_file <- get_gtf(
  species = "mouse",
  release_version = "release_M36",
  annotation_type = "long_noncoding_RNAs.gtf.gz",
  dest_folder = temp_dir
)
print(gtf_file)
```

`get_latest_release` *Get the Latest Gencode Release Dynamically*

Description

This function retrieves the latest available release of the Gencode database for a given species (human or mouse) by querying the relevant FTP directory. It automates the process of identifying the latest release of annotations for human or mouse.

Usage

```
get_latest_release(species, verbose = TRUE)
```

Arguments

`species`	A character string indicating the species. Supported values are: • "human" • "mouse"
`verbose`	Logical. If TRUE (default), the function prints the latest release.

Details

The function accesses the GENCODE FTP directory and parses the available folders to determine the latest release. It is particularly useful for bioinformatics workflows that require up-to-date annotation files without manual checks.

Value

A character string representing the latest release version for the specified species (e.g., "release_42" for human or "release_36" for mouse).

Examples

```
# Retrieve the latest release version for human
human_release <- get_latest_release(species = "human", verbose = FALSE)
cat("Latest human GENCODE release: release_47")

# Get the latest release for mouse
mouse_release <- get_latest_release("mouse", verbose = TRUE)
```

`load_file` *Load a GTF or GFF3 file from GENCODE as a data frame.*

Description

This function imports a GTF or GFF3 file (commonly from the GENCODE website) and converts it into a data frame. The function provides flexibility for users to work with genomic feature files easily in the R environment.

Usage

```
load_file(filename)
```

Arguments

`filename`	A character string representing the path to the GTF or GFF3 file (e.g., "gencode.vM36.annotation.gtf.gz"). The file could be in GTF or GFF3 format and must be downloaded from a reliable source like GENCODE.

Details

The function uses the `rtracklayer` package to import the GTF or GFF3 file and returns it as a data frame. The user should ensure that the input file is properly formatted and accessible from the specified path. Files larger than a few hundred MBs may take longer to load and process.

Value

A data frame containing the parsed content of the GTF or GFF3 file. The data frame includes standard columns such as 'seqnames', 'start', 'end', 'strand', 'feature', and 'gene_id', among others.

Examples

```
# Load example GTF files from the package
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
head(gtf_v1)
```

`spliced_trans_length` *Calculate Spliced Transcript Lengths*

Description

Computes the spliced length of transcripts by summing the lengths of their constituent exons. This represents the mature RNA length after intron removal.

Usage

```
spliced_trans_length(input)
```

Arguments

`input`	A data frame containing GENCODE annotations in GTF format, including `type`, `transcript_id`, and `width` columns.

Details

This function processes the input data to:

1. Filter entries to include only exons.
2. Group exons by transcript ID.
3. Sum the widths of all exons per transcript.

The result provides the total length of the mature spliced RNA for each transcript.

Value

A data frame with two columns: `transcript_id` and `transcript_length`, where the latter is the sum of exon widths for each transcript.

Examples

```
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
spliced_lengths <- spliced_trans_length(gtf_v1)
```

`stat_summary` *Generate Summary Statistics for Genomic Elements*

Description

Calculates descriptive summary statistics (mean, median, standard deviation, etc.) for the lengths of exons or introns, grouped by classification (e.g., first exon, inner intron).

Usage

```
stat_summary(input, type, verbose = TRUE)
```

Arguments

`input`	A data frame containing classified exons or introns. For exons, must include `EXON_CLASSIFICATION`. For introns, requires classification columns.
`type`	A character string specifying the element type. Valid options: `"exon"` or `"intron"`.
`verbose`	A logical indicating whether to print progress messages. Defaults to `TRUE`.

Details

For exons, statistics are grouped by `EXON_CLASSIFICATION`. For introns, groups include `first_intron`, `inner_intron`, and splice site types (e.g., `gc_intron`).

Value

A data frame with summary statistics for each element group, including mean, median, standard deviation, standard error, quartiles, and sample size.

Examples

```
file_v1 <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
gtf_v1 <- load_file(file_v1)
# Exon statistics
exon_stats <- stat_summary(gtf_v1, type = "exon")
```

tiny_example_gtf_files

Tiny example GTF files

Description

These are minimal GTF files used for examples and testing within the package.

- **gencode.v1.example.gtf.gz** contains two genes:
  - GeneA: a single-exon, unspliced gene.
  - GeneB: a spliced gene with two transcripts and multiple exons.
- **gencode.v2.example.gtf.gz** contains the same two genes as in `gencode.v1.example.gtf.gz` plus:
  - GeneC: a new spliced gene with multiple transcripts and many exons.

These files are stored in the `inst/extdata/` directory and can be accessed using `system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")` or `system.file("extdata", "gencode.v2.example.gtf.` `package = "GencoDymo2")`.

Format

Two external GTF files.

Source

Generated manually for testing purposes.

See Also

`load_file`

Examples

```
tiny_v1_path <- system.file("extdata", "gencode.v1.example.gtf.gz", package = "GencoDymo2")
tiny_v2_path <- system.file("extdata", "gencode.v2.example.gtf.gz", package = "GencoDymo2")

gtf1 <- load_file(tiny_v1_path)
head(gtf1)

gtf2 <- load_file(tiny_v2_path)
head(gtf2)
```

Index