Package: ViralEntropR (via r-universe)

May 30, 2026

Title A Computational Pipeline for Entropy-Informed Detection of Emerging Viral Variants

Version 0.6.2

Description Implements an entropy-informed pipeline for detecting emerging variants in viral amino acid sequence data, extending prior clustering-based approaches including hemagglutinin clustering methods (Li et al., 2015) <doi:10.1142/9789814667944_0018>. Provides a fully vectorized FASTA preprocessing toolkit covering header parsing, two-pass date and country extraction, ambiguous-residue filtering, and integer encoding under a 25-symbol amino acid alphabet. Computes per-site Shannon entropy across user-defined cumulative, sliding, or disjoint temporal partitions and clusters per-site entropy values using Gaussian mixture models via 'mclust' (Scrucca et al., 2016) <doi:10.32614/RJ-2016-021>. Quantifies temporal distributional shifts between partitions using the Hellinger distance (van der Vaart, 1998) <doi:10.1017/CBO9780511802256>, and detects temporal change points non-parametrically using energy statistics (Matteson and James, 2014) <doi:10.1080/01621459.2013.849605> via 'ecp' or wild binary segmentation (Fryzlewicz, 2014) <doi:10.1214/14-AOS1245> via 'HDcpDetect'. Per-site amino-acid frequency tables and entropy trajectory plots characterize sequence composition and evolutionary dynamics across time. A configurable multi-variant simulation engine generates synthetic sequence time series with known ground truth for benchmarking detection pipelines. A curated dataset of SARS-CoV-2 Variants of Concern and Variants of Interest with associated lineage and surveillance metadata is included, along with a bundled National Center for Biotechnology Information (NCBI) Spike protein sample and vignettes demonstrating the full workflow.

License MIT + file LICENSE

Language en-GB

Date 2026-05-07

URL https://github.com/vadimtyuryaev/ViralEntropR, https://doi.org/10.5281/zenodo.19040165, https://vadimtyuryaev.github.io/ViralEntropR/

BugReports https://github.com/vadimtyuryaev/ViralEntropR/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Imports ggplot2 (>= 3.4.0), grDevices, HDcpDetect, ecp, kableExtra, lubridate, magrittr, mclust, rlang, stats, stringr, utils, zoo

Suggests Biostrings, DT, dplyr, here, knitr, readxl, rmarkdown, R.rsp, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr, R.rsp

NeedsCompilation no

Author Vadim Tyuryaev [aut, cre] (ORCID: <https://orcid.org/0009-0008-1361-6265>), Jane Heffernan [aut], Hanna Jankowski [aut]

Maintainer Vadim Tyuryaev <vadim.tyuryaev@gmail.com>

Depends R (>= 3.5.0)

Repository https://cran.r-universe.dev

Date/Publication 2026-05-30 13:40:21 UTC

RemoteUrl https://github.com/cran/ViralEntropR

RemoteRef HEAD

RemoteSha a9b293057a1d5b20f074216fc899da6ebdd0389b

Contents

`calculate_entropy` *Calculate Shannon Entropy*

Description

Computes the Shannon entropy of a categorical vector.

Usage

```
calculate_entropy(vctr, base = 2, precision = 6)
```

Arguments

`vctr`	A vector (character, factor, or integer) representing categorical data.
`base`	A numeric scalar. The base of the logarithm. Default is 2.
`precision`	Integer. The number of decimal places to round the result to. Default is 6.

Details

Entropy is calculated as $H(X) = -\sum p(x) \log_b p(x)$, where $p(x)$ is the proportion of observations belonging to category x.

Value

A numeric scalar representing the entropy. Returns 0 if the vector contains only one unique value or has length 0.

Examples

```
seq_vec = c("A", "A", "T", "G", "C", "A")
calculate_entropy(seq_vec)

# Pure homogeneity
calculate_entropy(rep("A", 10))
```

`calculate_hellinger_matrix` *Calculate Hellinger Distance Matrix*

Description

Computes the Hellinger distance between the amino acid distribution of a reference time point (first partition) and all subsequent time points, for each requested site.

Usage

```
calculate_hellinger_matrix(
  partitions,
  sites = seq_len(ncol(partitions[[1]])),
  aa_levels = 25L,
  normalized = FALSE,
  labels = paste0("T", seq_along(partitions)),
  include_freq_tables = FALSE
)
```

Arguments

`partitions`	A list of data frames, one per time window. Each data frame must have numeric-encoded amino acid sequences as columns (integers 1 to `aa_levels`). This is typically the `$Partitions` element returned by `partition_time_windows`.
`sites`	Integer vector. Indices of the sites to analyse. Defaults to all sites (`seq_len(ncol(partitions[[1]]))`).
`aa_levels`	Integer. Alphabet size. Must match the encoding used when the partitions were created: `encode_aa_sequence` produces values 1–25 by default (20 standard residues, three ambiguous codes, `*`, `-`). Default is `25L`.
`normalized`	Logical. If `TRUE`, scales distances by $1/\sqrt{2}$ to bound the result in $[0, 1]$. Default is `FALSE`.
`labels`	Character vector of partition labels. Length must equal `length(partitions)`. The first label names the reference partition; subsequent labels become column names of the returned matrix. Defaults to `"T1"`, `"T2"`, ...
`include_freq_tables`	Logical. If `TRUE`, the return value also includes raw count tables and proportion tables for each site. Default is `FALSE`.

Details

The Hellinger distance between two discrete distributions P and Q is:

$$H(P, Q) = \sqrt{\sum_{i=1}^{k} (\sqrt{p_i} - \sqrt{q_i})^2}$$

When `normalized = TRUE` the result is scaled by $1/\sqrt{2}$, bounding the distance to $[0, 1]$. Otherwise the range is $[0, \sqrt{2}]$.

Internally, amino acid counts per site per partition are tabulated using `get_site_counts` (built on `tabulate`). Per-partition proportions and Hellinger distances are then computed by fully vectorised matrix operations — no inner loop over partitions.

Value

When `include_freq_tables = FALSE` (default): a numeric matrix with rows corresponding to `sites` and columns corresponding to `labels[-1]`, each entry being the Hellinger distance from the reference partition (`labels[1]`) at that site. Row names are the site indices; column names are taken from `labels[-1]`. With default labels, the reference partition is `"T1"` and the matrix has columns `"T2"`, `"T3"`, ...

When `include_freq_tables = TRUE`: a named list with elements `Sites`, `Hellinger_Distances`, `Frequency_Tables`, and `Proportions_Tables`.

See Also

`partition_time_windows` for producing temporally partitioned data, `encode_aa_sequence` for integer encoding consistent with `aa_levels`, and `detect_changepoints_ecp` or `detect_changepoints_hdcp` for downstream change-point detection on the returned matrix.

Examples

```
# Toy 3-partition data: site 1 starts homogeneous (all Alanine, 1),
# acquires Valine (20) across two later partitions; site 2 stays
# constant (all Valine throughout).
p1 = data.frame(s1 = c(1, 1, 1, 1, 1),  s2 = c(20, 20, 20, 20, 20))
p2 = data.frame(s1 = c(20, 20, 20, 20, 20), s2 = c(20, 20, 20, 20, 20))
p3 = data.frame(s1 = c(1, 1, 20, 20, 20), s2 = c(20, 20, 20, 20, 20))
parts = list(T1 = p1, T2 = p2, T3 = p3)

result = calculate_hellinger_matrix(parts, sites = 1:2)
print(result)
# Site 1 has nonzero distance in T2 and T3 (composition shift).
# Site 2 has zero distance throughout (constant).

# With raw frequency tables for inspection.
result2 = calculate_hellinger_matrix(parts, sites = 1:2,
                                     include_freq_tables = TRUE)
print(result2$Frequency_Tables[[1]])
```

`cluster_sites_by_entropy`

Cluster a Univariate Numeric Vector by Gaussian Mixture Model

---

Description

Wraps Mclust for unsupervised clustering of a univariate numeric vector, with preprocessing rules and edge-case handling tailored to per-site Shannon entropy values from viral sequence data, which is the package's primary use case, but applicable to any univariate data the user wishes to cluster by GMM.

Usage

```
cluster_sites_by_entropy(
  entropies,
  nr,
  nsites = length(entropies),
  precision = 6L,
  removez = TRUE,
  removesngl = TRUE,
  transfr = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

entropies	Numeric vector to cluster. In the package's primary use case these are per-site Shannon entropy values, but any univariate numeric vector is accepted.
nr	Integer. Total number of sequences from which the entropies were computed. Required only when removesngl = TRUE.
nsites	Integer. Expected number of sites. If it mismatches length(entropies), the actual length is used with a warning. Default is length(entropies).
precision	Integer. Decimal places for rounding during singleton threshold comparison and the all-identical uniqueness check. Default is 6.
removez	Logical. If TRUE, removes sites with entropy = 0 (invariant sites), using a small tolerance 1e-9 to absorb floating-point near-zeros. Default is TRUE.
removesngl	Logical. If TRUE, removes sites whose entropy equals the singleton value (one differing sequence out of nr). Uses tolerance-based comparison. Default is TRUE.
transfr	A function, or an object of class transform with a $transform() method, applied to entropies before clustering. Default is NULL (no transformation).
verbose	Logical. If TRUE, emits diagnostic warnings for non-fatal events (empty partitions, Mclust failures, etc.). Default is FALSE.
...	Additional arguments passed to Mclust.

Details

In the package's typical use, sites are clustered by their Shannon entropy to identify groups of residue positions with similar variability across a sequence collection. Two preprocessing rules apply when clustering entropies: removez = TRUE drops invariant sites (entropy = 0), and removesngl

= TRUE drops singleton sites whose entropy corresponds to exactly one differing sequence across nr rows.

Class assignment rules (applied in priority order):

- **No rows remaining after filtering**: empty DataFrame returned with a zero-length class column (consistent schema).
- **Single row remaining**: class 1 assigned directly; Mclust is not called (undefined on 1 observation).
- **All entropies identical**: class 999 for all sites (sentinel — one undifferentiated group).
- **Normal Mclust result**: raw class labels 1, 2, ..., G. These are Mclust's own integer labels, ordered by increasing component mean (univariate Mclust orders components by mean) — call relabel_entropy_classes() on the returned data frame to obtain application-friendly class labels (highest-entropy class = 1, lowest-entropy class = G). relabel_entropy_classes on the returned DataFrame to standardise so that class 1 = highest-entropy group.
- **Mclust failure**: empty DataFrame returned (same schema as the no-rows case), treating the partition as uninformative.

Value

A named list with two elements:

FitObject	The raw Mclust result, or a minimal list(classification = integer(0L)) when clustering was bypassed or failed.
DataFrame	A data frame with columns sites (original site indices), entropies (values after any transformation), and class (GMM cluster label). The class column is **always** present in every return path, including zero-row DataFrames. Downstream consumers need only guard on nrow(df) > 0 before accessing class values. Raw Mclust labels are returned as-is; call relabel_entropy_classes to standardise label ordering.

See Also

calculate_entropy for computing per-site entropy values, relabel_entropy_classes for standardising the returned class labels, and partition_time_windows, which calls this function on each temporal partition.

Examples

```
# Clear bimodal structure: 5 low-entropy + 5 high-entropy sites.
set.seed(42)
entropies <- c(rnorm(5, mean = 0.1, sd = 0.01),
               rnorm(5, mean = 1.5, sd = 0.1))
result <- cluster_sites_by_entropy(entropies, removez = FALSE,
                                   removesngl = FALSE)
print(result$DataFrame)

# Single-row edge case: class = 1 assigned directly.
res1 <- cluster_sites_by_entropy(0.35, removesngl = FALSE)
print(res1$DataFrame)
```

```
# All-identical edge case: class = 999 (sentinel, one undifferentiated group).
res2 <- cluster_sites_by_entropy(c(0.35, 0.35, 0.35), removesngl = FALSE)
print(res2$DataFrame)
```

---

decode_aa_sequence *Decode Amino Acid Sequences*

Description

Converts an integer-encoded matrix of amino acids back to its character representation under the package's 25-symbol alphabet. Inverse of encode_aa_sequence.

Usage

```
decode_aa_sequence(matrix_input)
```

Arguments

matrix_input	Numeric matrix of integer-encoded amino acids, typically the output of encode_aa_sequence. Values outside 1:25 (including 0 and NA) decode to the sentinel string "0". A non-matrix input is coerced via as.matrix.

Details

Decoding is a single vectorised lookup against the fixed alphabet (A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V for the twenty standard residues, then B, Z, X, *, - for ambiguous codes, stop codons, and gaps). The input matrix is flattened, indexed against the alphabet vector in one operation, and reshaped — there is no per-row or per-element loop.

Values outside the valid range 1:25 (including 0, which encode_aa_sequence produces for unrecognised characters) are returned as the sentinel string "0". This preserves round-trip consistency with the encoder: encoding then decoding any character originally outside the alphabet yields "0" rather than throwing an error. NA values are also mapped to "0".

Row and column names of the input matrix are preserved on the output.

Value

A character matrix of the same dimensions as matrix_input, with the same dimnames. Each cell is either a one-character amino acid code from the 25-symbol alphabet, or the sentinel "0" for out-of-range and missing values.

See Also

encode_aa_sequence for the inverse operation; fasta_to_char_matrix for the FASTA-to-character-matrix step that typically precedes encoding.

Examples

```
# 1. Decode a numeric matrix.
num_mat = matrix(c(1, 2, 25, 10), nrow = 2, byrow = TRUE)
decoded = decode_aa_sequence(num_mat)
print(decoded)
# 1 -> "A", 2 -> "R", 25 -> "-", 10 -> "I"

# 2. Round-trip consistency check (excluding unknowns).
orig = matrix(c("A", "C", "W", "G"), nrow = 2)
enc  = encode_aa_sequence(orig)
dec  = decode_aa_sequence(enc)
all.equal(orig, dec)

# 3. Out-of-range and NA values both decode to the sentinel "0".
decode_aa_sequence(matrix(c(0, NA, 30, 5), nrow = 2))
```

---

detect_changepoints_ecp

Detect Temporal Change Points (ECP)

Description

Runs Energy Change Point detection on time-series matrices (typically Hellinger distance matrices) over one or more time windows.

Usage

```
detect_changepoints_ecp(
  data_matrix,
  min_window,
  max_window,
  n_timesteps,
  rolling_window = FALSE,
  dynamic_k = FALSE,
  ...
)
```

Arguments

`data_matrix`	A numeric matrix (Time x Features). Rows are time points, columns are features (e.g. Hellinger distances per site).
`min_window`	Integer. Starting row index of the initial window.
`max_window`	Integer. Ending row index of the initial window. Must not exceed `nrow(data_matrix)`.
`n_timesteps`	Integer >= 0. Number of *additional* detections to run after the initial window. The function performs `n_timesteps + 1` detections in total. Set to `0` for a single detection on the initial window.

`rolling_window` Logical. If `TRUE`, both window endpoints advance per step (sliding window of constant length). If `FALSE` (default), only the end advances (expanding window).

`dynamic_k` Logical. If `TRUE`, sets `K` to `nrow(subset) - 2L` at each step, so the algorithm considers the maximum number of change points permitted by the window length (subject to `minsize`). Useful for exploratory analysis where the true number of change points is unknown a priori. Most idiomatic when combined with `n_timesteps > 0`, where the maximum admissible K legitimately scales with window size, but also valid for single-window exploratory use. If `FALSE` (default), the value of `K` supplied via `...` is honoured at every step; if no `K` is supplied, the `ks.cp3o` default of 1 is used.

`...` Additional arguments passed to `ks.cp3o`. Common choices include `K` (maximum number of change points, default 1) and `minsize` (minimum segment size, default 30). Note that `K` is overridden when `dynamic_k = TRUE`; a warning is emitted in that case.

Details

The function applies `ks.cp3o` repeatedly to slices of `data_matrix`, advancing the slice forward by one row at each step. A total of `n_timesteps + 1` detections are performed: one on the initial window `[min_window, max_window]`, then `n_timesteps` additional detections on subsequent windows.

Two window-advancement modes are supported:

- **Expanding** (`rolling_window = FALSE`, default): the start index is fixed at `min_window`; the end index grows by 1 each step. Each iteration uses one more row than the previous. Natural for online surveillance, where change-point detection is re-run as new data accumulates.
- **Rolling** (`rolling_window = TRUE`): both start and end indices advance by 1 each step, keeping the window length constant. Natural for retrospective windowed analysis, where local change-point structure is examined in fixed-size segments.

Value

A named list:

`Points_List` List of integer vectors `c(start, end)` giving the row index window used at each step.

`ECP_list` List of full `ks.cp3o` result objects, one per step. Inspect `$cpLoc` for the optimal change-point locations at each candidate count, and `$gofM` for the goodness-of-fit curve, to apply post-hoc filtering.

`ECP_est_list` List of change-point estimate vectors (the algorithm's own selection, equivalent to `$estimates`).

Warnings and errors

`stop()` Triggered if `max_window` exceeds `nrow(data_matrix)`: the initial window cannot be constructed and no detection can run.

`warning()` Triggered (and execution continues) if (a) `max_window + n_timesteps` exceeds `nrow(data_matrix)`, in which case the offending later iterations are skipped and `NULL` entries appear in the result lists; (b) `dynamic_k = TRUE` and `K` is also supplied via `...`, in which case the user-supplied `K` is silently overridden by `nrow(subset) - 2L`.

Examples

```
set.seed(123)
baseline = matrix(rnorm(50, mean = 0, sd = 0.1), nrow = 10, ncol = 5)
variant  = matrix(rnorm(50, mean = 3, sd = 0.1), nrow = 10, ncol = 5)
data_mat = rbind(baseline, variant)

# Single-window detection: one true change point at row 11.
res = detect_changepoints_ecp(
  data_matrix = data_mat,
  min_window  = 1,
  max_window  = 20,
  n_timesteps = 0,
  minsize     = 5
)
print(res$ECP_est_list[[1]])
```

---

detect_changepoints_hdcp

Detect Temporal Change Points (HDcpDetect)

Description

Runs high-dimensional change point detection on time-series matrices (typically Hellinger distance matrices) over one or more time windows, using either binary segmentation or wild binary segmentation.

Usage

```
detect_changepoints_hdcp(
  data_matrix,
  min_window,
  max_window,
  n_timesteps,
  rolling_window = FALSE,
  wild = FALSE,
  ...
)
```

Arguments

`data_matrix`	A numeric matrix (Time x Features). Rows are time points, columns are features (e.g. Hellinger distances per site).

`min_window`	Integer. Starting row index of the initial window.
`max_window`	Integer. Ending row index of the initial window. Must not exceed `nrow(data_matrix)`.
`n_timesteps`	Integer >= 0. Number of *additional* detections to run after the initial window. The function performs `n_timesteps + 1` detections in total. Set to `0` for a single detection on the initial window.
`rolling_window`	Logical. If `TRUE`, both window endpoints advance per step (sliding window of constant length). If `FALSE` (default), only the end advances (expanding window).
`wild`	Logical. If `TRUE`, uses `wild.binary.segmentation`; if `FALSE` (default), uses `binary.segmentation`. See Details for the methodological distinction.
`...`	Additional arguments passed to the chosen segmentation function. Note that `M` (number of random intervals) is only accepted by `wild.binary.segmentation`; passing it with `wild = FALSE` produces an "unused argument" error.

Details

The function applies `binary.segmentation` or `wild.binary.segmentation` repeatedly to slices of `data_matrix`, advancing the slice forward by one row at each step. A total of `n_timesteps + 1` detections are performed: one on the initial window `[min_window, max_window]`, then `n_timesteps` additional detections on subsequent windows.

Two window-advancement modes are supported, mirroring `detect_changepoints_ecp`:

- **Expanding** (`rolling_window = FALSE`, default): the start index is fixed at `min_window`; the end index grows by 1 each step. Each iteration uses one more row than the previous. Natural for online surveillance, where change-point detection is re-run as new data accumulates.
- **Rolling** (`rolling_window = TRUE`): both start and end indices advance by 1 each step, keeping the window length constant. Natural for retrospective windowed analysis.

Choice of segmentation method. Binary segmentation (`wild = FALSE`, default) is the classical recursive method: it finds the most likely change point, splits the series, and recurses. Wild binary segmentation (`wild = TRUE`) is the variant of Fryzlewicz (2014) that draws random subintervals to improve detection of multiple closely-spaced change points; it accepts an additional `M` argument controlling the number of random intervals.

Value

A named list:

`Points_List`	List of integer vectors `c(start, end)` giving the row index window used at each step.
`HDcp_list`	List of full segmentation result objects, one per step. Steps where `end_idx` exceeds the matrix rows return `NULL` with a warning.

Warnings and errors

`stop()` Triggered if `max_window` exceeds `nrow(data_matrix)`: the initial window cannot be constructed and no detection can run.

`warning()` Triggered (and execution continues) if `max_window + n_timesteps` exceeds `nrow(data_matrix)`, in which case the offending later iterations are skipped and `NULL` entries appear in `HDcp_list`.

See Also

detect_changepoints_ecp for the energy-statistic alternative; calculate_hellinger_matrix for the typical upstream input.

Examples

```
set.seed(42)
baseline = matrix(rnorm(50, mean = 0, sd = 0.1), nrow = 10, ncol = 5)
variant  = matrix(rnorm(50, mean = 3, sd = 0.1), nrow = 10, ncol = 5)
data_mat = rbind(baseline, variant)

# Single-window detection with the default binary segmentation.
res = detect_changepoints_hdcp(
  data_matrix = data_mat,
  min_window  = 1,
  max_window  = 20,
  n_timesteps = 0
)
print(res$HDcp_list[[1]])

# Wild binary segmentation with a custom number of random intervals.
res_wild = detect_changepoints_hdcp(
  data_matrix = data_mat,
  min_window  = 1,
  max_window  = 20,
  n_timesteps = 0,
  wild        = TRUE,
  M           = 100
)
print(res_wild$HDcp_list[[1]])
```

---

encode_aa_sequence *Encode Amino Acid Sequences*

Description

Converts a character matrix of amino acid codes to its integer-encoded representation under the package's 25-symbol alphabet. Inverse of decode_aa_sequence.

Usage

```
encode_aa_sequence(matrix_input)
```

Arguments

matrix_input	A character matrix of amino acid codes, typically produced by fasta_to_char_matrix. A non-matrix input is coerced via as.matrix. Lowercase input is accepted; characters are uppercased before lookup.

Details

Encoding is a single vectorised lookup against a fixed named map. The input matrix is flattened, normalised to uppercase, indexed against the alphabet's name vector in one operation, and reshaped — there is no per-row or per-element loop.

The 25-symbol alphabet covers the twenty standard residues (`A=1, R=2, N=3, D=4, C=5, Q=6, E=7, G=8, H=9, I=10, L=11, K=12,M=13, F=14, P=15, S=16, T=17, W=18, Y=19, V=20`), the three IUPAC ambiguous codes `B=21, Z=22, X=23`, the stop codon `*=24`, and the alignment gap `-=25`.

Any character outside this alphabet — including `NA`, the empty string, lowercase letters not in the standard set after uppercasing (e.g. `J` for leucine/isoleucine), and non-letter symbols — maps to the sentinel `0`. This sentinel is recognised by downstream functions: `filter_ambiguous_sequences` treats `0` as ambiguous (alongside `B`, `X`, `Z`) and removes affected sequences, and `decode_aa_sequence` round-trips it back to the string `"0"`.

Row and column names of the input matrix are preserved on the output.

Value

A numeric matrix of the same dimensions as `matrix_input`, with the same `dimnames`. Each cell is either an integer in `1:25` from the alphabet table above, or `0` for any input not in the alphabet (including `NA` and the empty string).

See Also

`decode_aa_sequence` for the inverse operation; `fasta_to_char_matrix` for the FASTA-to-character-matrix step that typically precedes encoding; `filter_ambiguous_sequences` for downstream removal of sequences containing ambiguous or unrecognised residues.

Examples

```
# 1. Encode a simple matrix of sequences.
seq_mat = matrix(c("A", "R", "N", "D"), nrow = 2, byrow = TRUE)
encoded = encode_aa_sequence(seq_mat)
print(encoded)

# 2. Gaps and unknown characters.
# '-' maps to 25; '?' is not in the alphabet and maps to the sentinel 0.
gapped_mat = matrix(c("A", "-", "G", "?"), nrow = 1)
encode_aa_sequence(gapped_mat)

# 3. Lowercase input is accepted (uppercased before lookup).
encode_aa_sequence(matrix(c("a", "r", "n", "d"), nrow = 2))

# 4. NA and empty string both map to 0.
encode_aa_sequence(matrix(c("A", NA, "G", ""), nrow = 2))
```

extract_fasta_countries

Extract Countries from FASTA Sequence Names

Description

Extracts country names from the sequence name strings of an AAStringSet object loaded via readAAStringSet. Handles single-word (e.g. UK), hyphenated (e.g. Timor-Leste), and multi-word (e.g. United States of America) country names.

Usage

```
extract_fasta_countries(sequence, position, problematic_characters = FALSE)
```

Arguments

sequence	An AAStringSet object.
position	Integer (1–4). Location of the country field within the sequence name string: • **1** — text before the first | (e.g. SouthKorea|...). • **2** — text between the first and second |. • **3** — text between the first and second /. • **4** — text after the last |.
problematic_characters	Logical. If TRUE, sequence names are re-encoded to UTF-8, replacing non-representable bytes with their escaped form. Useful for FASTA files with non-ASCII characters in headers. Default is FALSE.

Details

The function selects one of four regex patterns based on position and applies it to each sequence name via str_extract. **Only the first match per header is returned.** If a header contains multiple delimited fields, the country must be in the first such field for the corresponding position value to extract it correctly. For example, with a GISAID-style header Spike|hCoV-19/USA/OH/.../2021|2021-05-15|EPI_ISL_..., position = 3 (between slashes) returns USA, but position = 2 (between pipes) returns hCoV-19/USA/OH/..., not USA. Inspect representative headers with names(sequence)[1] before choosing position.

Encoding. FASTA files with non-ASCII characters in headers (accented characters, byte-order marks, etc.) can break regex extraction. Setting problematic_characters = TRUE re-encodes headers to UTF-8 with non-representable bytes escaped, allowing the regex to proceed.

Value

A named list with three elements:

countries	Character vector of extracted country strings, one per sequence. NA where extraction failed (no match against the chosen pattern).
message	A single character string summarising extraction success.

missing_id	Integer vector of indices where extraction failed, or NA if all extractions succeeded.

See Also

extract_fasta_dates for the date-extraction companion; readAAStringSet for loading the input AAStringSet.

Examples

```
path_sample  <- system.file("extdata", "sarscov2_sample.fasta.gz",
                            package = "ViralEntropR")
fasta_sample <- Biostrings::readAAStringSet(path_sample)

# Inspect header structure to confirm field positions before extraction.
sample(names(fasta_sample), 1)

# Extract countries (position 2 = between first and second pipe).
result <- extract_fasta_countries(fasta_sample, position = 2)
result$message
sort(table(result$countries), decreasing = TRUE)
```

---

extract_fasta_dates *Extract Dates from FASTA Sequence Names*

Description

Extracts date strings from the sequence name strings of an AAStringSet object loaded via readAAStringSet. Several built-in date patterns are provided for the common date conventions used in NCBI and GISAID exports; a fully custom regex can also be supplied.

Usage

```
extract_fasta_dates(
  sequence,
  option = 1,
  date_format = "%Y-%m-%d",
  custom_pattern = NULL
)
```

Arguments

sequence	An AAStringSet object.
option	Integer (1, 2, 3, or 4). Selects the built-in pattern when custom_pattern is not supplied. See Details.
date_format	Character. strptime-style format string used to coerce extracted strings to Date. Default is "%Y-%m-%d". Use "%Y-%m" together with option = 3.
custom_pattern	Character or NULL. A custom regex passed directly to str_extract. When supplied, option is ignored. Default is NULL.

Details

Date strings of the form `yyyy-mm-dd` are matched between pipe characters (`|...|`) by default. Day value `00` (a common GISAID convention indicating unknown collection day) is accepted in the raw string and corrected to `01` before coercion to `Date`. Both raw and corrected versions are returned, so the caller can decide how to treat unknown-day records downstream.

Choosing a built-in pattern. The four options correspond to the four most common date conventions in viral sequence repositories:

- `option = 1`: `yyyy-mm-dd` between pipes — GISAID export format, where the date is followed by additional pipe-delimited fields.
- `option = 2`: `yyyy-dd-mm` between pipes — some European data sources reverse day and month.
- `option = 3`: `yyyy-mm` between pipes — month-level resolution, useful when the source omits or hides the day. Pair with `date_format = "%Y-%m"`.
- `option = 4`: `yyyy-mm-dd` at end of header — NCBI Virus export format, where the collection date is the final field with no trailing delimiter. This is the format of the bundled `sarscov2_sample`.

For datasets where the date does not lie between pipes, supply a `custom_pattern` matching whatever surrounding context the headers provide.

Coercion to Date. When `date_format = "%Y-%m"` the function uses `as.yearmon` for coercion so that year-month strings are handled correctly (base `as.Date` cannot parse `"2021-05"` alone). For all other formats, base `as.Date` with the supplied `date_format` is used.

Output alignment. All six elements of the return list are the same length as the input `sequence`. Where extraction fails for a record, the corresponding entries are `NA`; `missing_id` lists the affected indices.

Value

A named list of six elements, each aligned with the input `sequence`:

`raw_date_strings`
: Character vector of extracted date strings before any correction. `NA` where extraction failed.

`corrected_date_strings`
: Character vector with `-00` substrings replaced by `-01`. `NA` where extraction failed.

`raw_dates`
: `Date` vector coerced from `raw_date_strings`. `NA` for unparseable or missing strings (including any record where day = `00`).

`corrected_dates`
: `Date` vector coerced from `corrected_date_strings`. `NA` for unparseable or missing strings.

`message`
: Character string summarising extraction success.

`missing_id`
: Integer vector of indices where extraction failed, or `NA` if all extractions succeeded.

See Also

extract_fasta_countries for the country-extraction companion; readAAStringSet for loading the input AAStringSet; as.yearmon for the year-month coercion path.

Examples

```
path_sample  <- system.file("extdata", "sarscov2_sample.fasta.gz",
                            package = "ViralEntropR")
fasta_sample <- Biostrings::readAAStringSet(path_sample)

# Inspect header structure to confirm date field position.
sample(names(fasta_sample), 1)
# The bundled sample uses NCBI Virus format: date is at end of header.

# Default usage on bundled sample: option = 4 for end-of-header dates.
dates <- extract_fasta_dates(fasta_sample, option = 4)
dates$message
head(dates$corrected_dates)
range(dates$corrected_dates, na.rm = TRUE)

# Custom regex for non-standard headers:
dates_custom <- extract_fasta_dates(
  fasta_sample,
  custom_pattern = "[0-9]{4}-(0?[1-9]|1[0-2])-(0?[1-9]|[12][0-9]|3[01]|00)"
)
```

---

fasta_to_char_matrix *Convert FASTA Object to Character Matrix*

Description

Converts an AAStringSet object loaded via readAAStringSet into a character matrix where rows are sequences and columns are residue positions (sites). Inverse structural transformation of encode_aa_sequence's expected input shape.

Usage

```
fasta_to_char_matrix(fsta)
```

Arguments

fsta	An AAStringSet object, typically the output of readAAStringSet. May be aligned or unaligned (see Details).

Details

Alignment. The function expects an aligned `AAStringSet` — all sequences of equal width. Unaligned input is accepted and shorter sequences are right-padded with the gap character `"-"` to match the longest sequence, but downstream entropy-based analysis assumes positional homology across rows; if sequences in the input are not biologically aligned, results from per-site computations will not be meaningful. For unaligned input, run a multiple-sequence alignment (e.g. `msa::msa()` or `DECIPHER::AlignSeqs()`) before calling this function.

Performance. Conversion is fully vectorised: all sequences are coerced to a single character string vector, split simultaneously, and reshaped into a matrix in one operation. No per-row loop, no intermediate list of split sequences kept alive — substantially faster than per-row `strsplit` on large inputs (100k+ sequences).

Value

A character matrix with `length(fsta)` rows and `max(nchar(as.character(fsta)))` columns. Each cell contains a single-character amino acid code from the input sequences (or the gap character `"-"` for padded positions in unaligned input). The matrix has no row or column names; sequence names from the `AAStringSet` are not carried over. An empty input (`length(fsta) == 0`) returns an empty 0-by-0 character matrix.

See Also

`encode_aa_sequence` for converting the resulting character matrix to an integer-encoded matrix; `filter_ambiguous_sequences` for removing rows containing ambiguous residues; `readAAStringSet` for loading FASTA files into the input format.

Examples

```
# Convert the bundled sample to a character matrix.
path  <- system.file("extdata", "sarscov2_sample.fasta.gz",
                     package = "ViralEntropR")
fasta <- Biostrings::readAAStringSet(path)
mat   <- fasta_to_char_matrix(fasta)
dim(mat)
mat[1:3, 1:10]
```

`filter_ambiguous_sequences`

Remove Sequences Containing Ambiguous Residues

Description

Removes rows (sequences) that contain at least one ambiguous amino acid residue (B, J, X, or Z) — and, under integer-encoded input, any unrecognised character — from a sequence matrix. Accepts both integer-encoded matrices and character matrices.

Usage

```
filter_ambiguous_sequences(NumMatrix, option = 1)
```

Arguments

NumMatrix	A matrix. Rows are sequences, columns are sites. Either integer-encoded (option = 1) or character (option = 2). Despite the name, character matrices are also accepted under option = 2.
option	Integer. 1 (default) for integer-encoded matrices produced by encode_aa_sequence; 2 for character matrices produced by fasta_to_char_matrix.

Details

What is removed. Sequences are flagged for removal if any of their residue positions contain one of the four IUPAC ambiguous codes:

- B — Aspartate / Asparagine.
- J — Leucine / Isoleucine.
- X — any residue.
- Z — Glutamate / Glutamine.

What is NOT removed. Standard alignment gaps (-, integer code 25) are retained — gaps represent known absences rather than uncertain identities and are typically positionally meaningful in aligned data. Sequences containing only canonical 20 amino acids and gaps are kept.

How input mode is handled.

- option = 1 (integer-encoded input from encode_aa_sequence): rows are removed if any cell equals 0 (unrecognised — including J, NA, empty, lowercase mismatches, byte-order marks, and other characters that fell outside the encoding alphabet), 21 (B), 22 (Z), or 23 (X). The 0 sentinel acts as a catch-all for anything not in the 25-symbol alphabet.
- option = 2 (character input from fasta_to_char_matrix): rows are removed if any cell is exactly "B", "J", "X", or "Z". Unrecognised characters in character input (e.g. lowercase letters, NA, empty strings) are NOT caught at this stage; encode first if you need that catch-all behaviour.

Performance. Detection is fully vectorised: a single logical matrix comparison followed by rowSums counts ambiguous residues per sequence in one C-level call, replacing the original row-by-row loop for a substantial speed improvement on large matrices (100k+ rows).

Value

A named list:

OriginalDim	Character string reporting the number of input sequences.
NewDim	Character string reporting the number of sequences remaining after filtering.
NumberAmbiguous	Character string reporting the number of sequences that contained at least one ambiguous residue.

RangeAmbiguous	Character string reporting the min and max count of ambiguous residues per removed sequence, or "No ambiguous sequences found" when none were removed.
DeletedSeqId	Integer vector of row indices that were removed. Empty integer vector if nothing was removed.
FilteredMatrix	The filtered matrix with ambiguous rows removed, preserving the original column structure and storage mode.

See Also

encode_aa_sequence and fasta_to_char_matrix for producing the typical input; decode_aa_sequence for inspecting the surviving sequences in character form.

Examples

```
# Synthetic example: 50 sequences, 10 sites, drawn from canonical residues.
set.seed(1)
m <- matrix(sample(1:20, 500, replace = TRUE), nrow = 50, ncol = 10)
# Inject ambiguous codes into 3 specific rows: 21 (B), 23 (X), 0 (unrecognised).
m[c(3, 17, 42), sample(1:10, 3)] <- c(21, 23, 0)
result <- filter_ambiguous_sequences(m, option = 1)
cat(result$NumberAmbiguous, "\n")
cat(result$RangeAmbiguous, "\n")
dim(result$FilteredMatrix)

# Character-mode example.
chr <- matrix(c("M", "K", "T", "I", "I", "X", "K", "T", "I", "I"),
              nrow = 2, byrow = TRUE)
filter_ambiguous_sequences(chr, option = 2)$DeletedSeqId
```

---

partition_time_windows

Partition Data into Time Windows

Description

Splits a data frame of time-stamped sequences into discrete time windows, computes per-site entropy and Mclust clustering for each window.

Usage

```
partition_time_windows(
  data,
  n_sites,
  window_length = 1,
  window_type = 3,
  start_date = NULL,
  end_date = NULL,
```

```
  date_format = "%b-%Y",
  verbose = FALSE,
  ...
)
```

Arguments

data	Data frame. Must contain a column named Date coercible to Date. Columns 1 through n_sites must be the numeric site columns; any additional columns (such as Country or other per-sequence metadata) may follow.
n_sites	Integer. Number of site columns. Site columns must occupy positions 1 through n_sites of data; any other columns (Date, optionally Country, etc.) must come after.
window_length	Integer. Window duration in months. Default is 1.
window_type	Integer (1, 2, or 3). Windowing strategy: 1 = Cumulative, 2 = Sliding, 3 = Disjoint. Default is 3.
start_date	Date or character. Window start. Defaults to earliest date in data.
end_date	Date or character. Window end (cutoff). Defaults to latest date in data.
date_format	Character. Format string for date labels. Default is "%b-%Y".
verbose	Logical. If TRUE, prints a progress bar and completion summary. Default is FALSE.
...	Additional arguments passed to cluster_sites_by_entropy.

Details

Three windowing strategies are supported via window_type. Throughout the descriptions below, T denotes the number of whole months between start_date and end_date, and w denotes window_length:

- **1 — Cumulative**: Start is fixed; end expands by w months each step. Each window includes all prior data plus one more period. Produces floor(T / w) chunks.
- **2 — Sliding**: A window of w months slides one month at a time. Produces T - w + 1 chunks.
- **3 — Disjoint**: Consecutive non-overlapping windows of w months. Produces floor(T / w) chunks. Default.

For example, with T = 12 months and w = 2: cumulative produces 6 chunks (each progressively larger); sliding produces 11 chunks (overlapping, each 2 months wide); disjoint produces 6 chunks (each exactly 2 months, non-overlapping).

Empty windows. When a window contains no observations, the corresponding entries in the returned lists carry placeholder values: Entropies is a zero-vector of length n_sites; Clusters is a schema-consistent empty result matching cluster_sites_by_entropy's empty-input return; Max_Entropy is NA_integer_. Downstream consumers can guard on nrow(Partitions[[i]]) > 0 or !is.na(Max_Entropy[i]).

Column layout requirement. The function extracts site columns as data[, seq_len(n_sites), drop = FALSE]. The site columns must therefore occupy positions 1 through n_sites. Date (and any other metadata columns such as Country) must come after the site columns. A common error is to place Date first; this will produce incorrect entropy values.

Value

A named list:

`Partitions`	List of data frame chunks, one per window.
`Entropies`	List of numeric entropy vectors, one per window.
`Clusters`	List of clustering results from `cluster_sites_by_entropy`, one per window.
`Max_Entropy`	Numeric vector. Maximum cluster class label per window (equals number of clusters found; `NA` if window was empty).
`Dates_Labels`	Character vector of window label strings.
`N_partitions`	Integer. Total number of windows.

See Also

`calculate_entropy` for the per-site entropy computation; `cluster_sites_by_entropy` for the GMM clustering applied per window; `relabel_entropy_classes` for standardizing the cluster labels in downstream consumers; and `calculate_hellinger_matrix` for typical downstream use of the `Partitions` output.

Examples

```
dates <- seq(as.Date("2020-01-01"), as.Date("2020-06-01"), by = "month")
df <- data.frame(
  s1   = 1L,
  s2   = c(rep(1L, 15), rep(2L, 15)),
  Date = rep(dates, each = 5)
)
res <- partition_time_windows(df, n_sites = 2, window_length = 2,
                              window_type = 3, verbose = TRUE)
print(res$Dates_Labels)
print(res$Max_Entropy)
```

`plot_entropy_trajectories` *Plot Shannon Entropy Trajectories*

Description

Plots per-site Shannon entropy as continuous trajectories across time partitions for a selected set of sequence sites, using the output of `partition_time_windows`.

Usage

```
plot_entropy_trajectories(
  part_data,
  sites = NULL,
  labels = NULL,
  site_colors = NULL,
  by_group = FALSE,
  groups_list = NULL,
  line_type_groups = NULL,
  line_size_groups = NULL,
  transformation = NULL,
  line_size = 1.5,
  legend = TRUE,
  legend_text_size = 12,
  x_angle = 45,
  grayscale = FALSE,
  plot_title = "Shannon Entropy Trajectories"
)
```

Arguments

part_data	Named list. Output of partition_time_windows, optionally with per-partition Clusters[[i]]$DataFrame already relabeled by the user via relabel_entropy_classes. Must contain elements Clusters, Max_Entropy, Dates_Labels, and N_partitions.
sites	Integer vector. Site indices to include. Defaults to the union of all sites observed across all partitions (i.e. every site that has non-zero, non-singleton entropy in at least one partition window).
labels	Character vector of length N_partitions. Partition labels used on the x-axis. Defaults to part_data$Dates_Labels.
site_colors	Named character vector. Names are site indices as character strings (e.g. "681"); values are colour strings (e.g. "#FB8072"). Sites absent from site_colors receive automatically assigned colours. Default is NULL (all colours auto-assigned).
by_group	Logical. If TRUE, maps line type and line width to site groups defined by groups_list. Default is FALSE.
groups_list	List of integer vectors. Each element specifies the site indices belonging to one explicit group. Sites in sites not covered by any explicit group are automatically assigned to a remainder group appended as the final element. Total group count (explicit plus remainder) must not exceed 6. Required when by_group = TRUE.
line_type_groups	Character vector. One line-type string per group (in order, including the automatic remainder group). Must have length equal to the total number of groups. Defaults to "solid" for the first group and "dashed" for all remaining groups.
line_size_groups	Numeric vector. One line-width value per group (in order, including the automatic remainder group). Must have length equal to the total number of groups. Defaults to 2 for the first group and 1 for all remaining groups.

`transformation` Object of class `"transform"` or `"trans"` as returned by `trans_new`, or `NULL` (identity, no transformation). Applied to the y-axis via `scale_y_continuous`. Default is `NULL`.

`line_size` Numeric. Line width used when `by_group = FALSE`. Default is `1.5`.

`legend` Logical. If `TRUE` (default), the site colour legend is displayed.

`legend_text_size` Numeric. Font size of legend text in points. Default is `12`.

`x_angle` Numeric. Rotation angle of x-axis tick labels in degrees. Default is `45`.

`grayscale` Logical. If `TRUE`, overrides `site_colors` and renders all trajectories in greyscale. Default is `FALSE`.

`plot_title` Character. Plot title string. Default is `"Shannon Entropy Trajectories"`.

Details

For each partition the function extracts the GMM clustering result from `part_data$Clusters` and assembles a long-format data frame spanning all selected sites across all partitions. Sites absent from a given partition (removed by zero-entropy or singleton filtering, or because the partition window was empty) are silently omitted from that partition's trajectory and do not interrupt adjacent observations.

Class relabeling. This function does not perform any relabeling of GMM class labels. If class 1 must denote the highest-entropy group throughout the returned `$Data_Frame` (e.g. before passing it to `plot_site_class_trajectory`), the user should call `relabel_entropy_classes` on each partition's `Clusters[[i]]$DataFrame` and update `Max_Entropy[i]` to `1L` prior to calling this function.

Colour scheme. Site colours are specified through `site_colors`, a named character vector whose names are site indices (as character strings) and whose values are valid R colour strings. Any site not listed in `site_colors` receives an automatically assigned colour from the HCL `"Dark 2"` qualitative palette. The final colour mapping is returned as `$Colors` so the same scheme can be passed to subsequent calls for cross-plot consistency.

Group-stratified trajectories (`by_group = TRUE`)**.** When biological groupings must be distinguished visually (e.g. defining SNP sites vs. other mutation sites), `groups_list` partitions `sites` into explicitly named groups. Any site not assigned to an explicit group is automatically collected into a remainder group appended as the final element of `groups_list`. Line type and line width are mapped to group membership via `line_type_groups` and `line_size_groups`, both of which must have length equal to the total number of groups (explicit plus the automatic remainder). At most six groups are supported.

`max_class` **column.** The returned `$Data_Frame` carries a `max_class` column recording the label of the highest-entropy GMM component for each partition, taken directly from `part_data$Max_Entropy[i]`. This column is consumed by `plot_site_class_trajectory` (red labels) and by downstream class-assignment tables.

Value

A named list with five elements:

Data_Frame	Long-format data frame with columns sites (factor), entropies (numeric), class (factor), max_class (integer), period (integer), and coverage (character, partition label). Suitable for direct input to plot_site_class_trajectory.
Plot	A ggplot object. Augment with additional layers (e.g. geom_vline for VOC emergence events) before printing or saving with ggsave.
Colors	Named character vector mapping each plotted site index (character) to its assigned colour string. Pass as site_colors to subsequent calls to plot_entropy_trajectories for a consistent colour scheme across figures.
XBreaks	Integer vector of partition period indices. Pass as xbreaks to plot_site_class_trajectory.
XLabels	Character vector of partition labels aligned with XBreaks. Pass as xlabels to plot_site_class_trajectory.

See Also

partition_time_windows, relabel_entropy_classes, plot_site_class_trajectory

Examples

```
# Three-period synthetic dataset with distinct trajectory shapes:
# Site 1: up-then-down (peak in P2).
# Site 2: monotone up.
# Per-partition Shannon entropy (bits):
#   P1 (Jan)  s1: 0.469  s2: 0.469
#   P2 (Feb)  s1: 1.522  s2: 0.881
#   P3 (Mar)  s1: 0.722  s2: 1.522
df <- data.frame(
  s1 = c(
    c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L),         # P1: 9:1
    c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 3L, 3L),         # P2: 4:4:2
    c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L, 2L)          # P3: 8:2
  ),
  s2 = c(
    c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L),         # P1: 9:1
    c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L, 2L, 2L),         # P2: 7:3
    c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 3L, 3L)          # P3: 4:4:2
  ),
  Date = rep(
    seq(as.Date("2020-01-01"), by = "month", length.out = 3L),
    each = 10L
  )
)

part_data <- partition_time_windows(
  data          = df,
  n_sites       = 2L,
  window_length = 1L,
  window_type   = 3L,
  start_date    = "2020-01-01",
  end_date      = "2020-04-01",
  removez       = FALSE,
```

```
  removesngl    = FALSE
)

result <- plot_entropy_trajectories(part_data = part_data)
print(result$Plot)
```

`plot_site_class_trajectory` *Plot GMM Entropy Class Trajectory for a Single Site*

Description

Plots the Shannon entropy trajectory for a single sequence site across time partitions, with geom-label overlays at each partition recording the site's current GMM class (above the line, in green) and the partition's highest-entropy class label (below the line, in red). The visual contrast between the two labels tracks the moment a site enters the highest-entropy cluster — the entropy-based analogue of variant emergence detection.

Usage

```
plot_site_class_trajectory(
  data_frame,
  site,
  site_color = "steelblue",
  xbreaks,
  xlabels,
  col_current = "springgreen2",
  col_max_class = "red2",
  label_size = 3,
  x_angle = 45,
  line_size = 1.5,
  plot_title = NULL,
  save = FALSE,
  save_path = NULL,
  save_extension = ".png",
  width = 20,
  height = 15,
  dpi = 300
)
```

Arguments

`data_frame`	Data frame. The `$Data_Frame` element from `plot_entropy_trajectories`. Must contain columns `sites`, `entropies`, `class`, `max_class`, `period`, and `coverage`.
`site`	Integer (length 1). The site index to plot. Must be present in `data_frame$sites`.

site_color	Character. Colour of the entropy trajectory line. Default is "steelblue". Pass plot_entropy_trajectories()$Colors[as.character(site)] for cross-plot colour consistency.
xbreaks	Integer vector. Partition period indices for x-axis breaks. Typically plot_entropy_trajectories()$XBr
xlabels	Character vector. Partition label strings aligned with xbreaks. Typically plot_entropy_trajectories(
col_current	Character. Fill colour of the upper geom-label (current GMM class at each partition). Default is "springgreen2".
col_max_class	Character. Fill colour of the lower geom-label (highest-entropy class label at each partition). Default is "red2".
label_size	Numeric. Font size of the class integer text inside the geom-labels. Default is 3.
x_angle	Numeric. Rotation angle of x-axis tick labels in degrees. Default is 45.
line_size	Numeric. Width of the entropy trajectory line. Default is 1.5.
plot_title	Character or NULL. Plot title. If NULL (default), the title is auto-generated as "GMM Entropy Class Trajectory \u2014 Site <site>".
save	Logical. If TRUE, the plot is saved to disk via ggsave. Default is FALSE.
save_path	Character or NULL. Directory in which to save the file. Created recursively if it does not exist. Must be supplied when save = TRUE. Default is NULL.
save_extension	Character. File extension including the leading dot (e.g. ".jpeg", ".pdf", ".png"). Default is ".jpeg".
width	Numeric. Saved figure width in inches. Default is 20.
height	Numeric. Saved figure height in inches. Default is 15.
dpi	Numeric. Resolution of the saved raster output in dots per inch. Default is 600.

Details

The function operates on the $Data_Frame element returned by plot_entropy_trajectories, which contains columns sites, entropies, class, max_class, period, and coverage.

Class label interpretation. The green label (upper) records the GMM class assigned to the site in that partition; the red label (lower) records max_class — the label of the highest-entropy component for that partition. When the two labels are equal the site has entered the highest-entropy class. Without relabeling, max_class is the raw Mclust label carrying the highest mean entropy (i.e. max(classification) at clustering time). When the user has pre-relabeled the partitions via relabel_entropy_classes before calling plot_entropy_trajectories, max_class is 1L throughout (class 1 is the highest-entropy group by definition), and the sentinel 999L is preserved unchanged.

Axis padding. Both axes carry generous expansion margins so that geom-label boxes at extreme entropy values or at the first and last partitions do not clip the panel border. The returned ggplot object can be further adjusted by appending standard ggplot2 layers with + before printing or saving.

Value

A ggplot object (returned invisibly). Additional ggplot2 layers can be appended with + before printing or saving, for example:

```
p <- plot_site_class_trajectory(traj$Data_Frame, site = 681L, ...)
p + ggplot2::geom_vline(xintercept = 14, colour = "darkorange",
                        linetype = "dashed", linewidth = 1.5)
```

See Also

plot_entropy_trajectories, relabel_entropy_classes, partition_time_windows

Examples

```
# Three-period synthetic dataset (independent of plot_entropy_trajectories'
# example). Three sites with different entropy trajectories; site 1 features
# a monotone-up trajectory and progressively higher rank among sites,
# producing class changes across all three partitions.
#
# Site 1 follows a monotone-up trajectory (0.469 -> 0.881 -> 1.522),
# climbing from lowest rank in P1 to highest in P3 for a clear
# "emerging variant" class progression. Site 2 follows the
# opposite trajectory (1.522 -> 1.000 -> 0.881), starting as the dominant
# high-entropy site and declining as site 1 rises. Site 3 (featured)
# peaks at P2 (0.881 -> 1.522 -> 1.000).
#
# Per-partition Shannon entropy (bits):
#          P1     P2     P3
#   s1   0.469  0.881  1.522
#   s2   1.522  1.000  0.881
#   s3   0.881  1.522  1.000   <- featured site
df_cls <- data.frame(
  s1 = c(
    c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L),       # P1: 9:1
    c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L, 2L, 2L),       # P2: 7:3
    c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 3L, 3L)        # P3: 4:4:2
  ),
  s2 = c(
    c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 3L, 3L),       # P1: 4:4:2
    c(1L, 1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 2L),       # P2: 5:5
    c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L, 2L, 2L)        # P3: 7:3
  ),
  s3 = c(
    c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L, 2L, 2L),       # P1: 7:3
    c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 3L, 3L),       # P2: 4:4:2
    c(1L, 1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 2L)        # P3: 5:5
  ),
  Date = rep(
    seq(as.Date("2020-01-01"), by = "month", length.out = 3L),
    each = 10L
  )
)

part_data <- partition_time_windows(
  data          = df_cls,
```

```
  n_sites       = 3L,
  window_length = 1L,
  window_type   = 3L,
  start_date    = "2020-01-01",
  end_date      = "2020-04-01",
  removez       = FALSE,
  removesngl    = FALSE
)

# Apply class relabeling so class 1 = highest mean entropy.
# plot_entropy_trajectories does not relabel internally; the consumer does.
for (i in seq_along(part_data$Clusters)) {
  part_data$Clusters[[i]]$DataFrame <-
    relabel_entropy_classes(part_data$Clusters[[i]]$DataFrame)
}

traj <- plot_entropy_trajectories(part_data)

p <- plot_site_class_trajectory(
  data_frame = traj$Data_Frame,
  site       = 3L,
  site_color = traj$Colors["1"],
  xbreaks    = traj$XBreaks,
  xlabels    = traj$XLabels
)
print(p)
```

`relabel_entropy_classes` *Relabel Entropy Classes*

Description

Relabels GMM cluster labels so that class 1 is always the highest-entropy group, class 2 the second highest, and so on.

Usage

```
relabel_entropy_classes(df)
```

Arguments

`df`	A data frame containing clustering results. Must have columns `class` (integer cluster labels from `cluster_sites_by_entropy`) and `entropies` (numeric Shannon entropy values per row).

Details

For univariate input (the package's typical use case), `Mclust` orders cluster components by increasing mean. Applied to per-site Shannon entropy values, this places the highest-entropy group at the highest label number (label `G` for `G` fitted components). This function flips that convention so that label `1` always denotes the highest-entropy group, which is more natural for downstream filtering ("class 1 = top") and visual labeling. Cluster identities and means are unchanged; only the integer labels are remapped.

Sentinel preservation. The class label `999L` marks undifferentiated groups (all entropies equal in `cluster_sites_by_entropy`) and is never relabeled. If a data frame contains a mixture of real GMM labels and one or more `999` entries, only the real labels are ranked and overwritten; the `999` rows pass through unchanged.

No-op return paths. The input is returned unchanged (with at most a `num_classes` column added) in four situations:

- Missing required columns (`class` or `entropies`) — warns and returns input.
- Zero rows.
- All rows already carry the sentinel `999L`.
- Only one class label is present (relabeling is a no-op).

Value

The data frame with a relabeled `class` column (integer) and an added `num_classes` column reporting the count of distinct labels present after relabeling.

See Also

`cluster_sites_by_entropy` for the upstream clustering step; `plot_entropy_trajectories` and `plot_site_class_trajectory` for downstream consumers that rely on the relabeled class convention; `partition_time_windows` which calls `cluster_sites_by_entropy` per window.

Examples

```
# Standard case: three classes, ranked by mean entropy.
df <- data.frame(
  sites     = 1:6,
  entropies = c(0.1, 0.1, 0.5, 0.5, 1.2, 1.3),
  class     = c(1L, 1L, 2L, 2L, 3L, 3L)
)
relabel_entropy_classes(df)
# Class 3 (highest mean entropy 1.25) -> 1; class 2 (0.5) -> 2;
# class 1 (0.1) -> 3.

# Sentinel preservation: 999 rows pass through unchanged even when
# mixed with real classes.
df_mixed <- data.frame(
  sites     = 1:4,
  entropies = c(0.5, 1.2, 0.3, 0.4),
  class     = c(1L, 2L, 999L, 1L)
)
```

```
relabel_entropy_classes(df_mixed)
# Class 2 (highest) -> 1; class 1 -> 2; class 999 stays 999.
```

`sarscov2_sample` *SARS-CoV-2 Surface Glycoprotein Sequences – NCBI Demo Sample*

Description

A compressed FASTA file containing a random sample of size 100 of SARS-CoV-2 surface glycoprotein (Spike protein) amino acid sequences, drawn from a dataset of 137,132 complete sequences downloaded from the NCBI SARS-CoV-2 Data Hub on October 12, 2021. Intended to demonstrate the full `ViralEntropR` pipeline on real-world surveillance data.

Format

A gzip-compressed FASTA file (`.fasta.gz`) readable by `readAAStringSet`. Each record contains:

Header NCBI Virus format: accession, country, and collection date metadata separated by `|`.

Sequence Amino acid sequence using the standard IUPAC one-letter code. Ambiguous residues (B, X, Z) and gaps (-) may be present and can be removed with `filter_ambiguous_sequences`.

Details

The file is accessed at runtime via:

```
path  <- system.file("extdata", "sarscov2_sample.fasta.gz",
                     package = "ViralEntropR")
fasta <- Biostrings::readAAStringSet(path)
```

Sample file contents:

- **Sample size**: 100 random sequences
- **Protein**: Surface glycoprotein (Spike, S protein)
- **Organism**: *Severe acute respiratory syndrome coronavirus 2* (SARS-CoV-2), NCBI Taxonomy ID: 2697049
- **Completeness**: Complete sequences only
- **Format**: Gzip-compressed FASTA (`.fasta.gz`), readable directly by `readAAStringSet`

Full dataset: The complete 137,132-sequence dataset (~181.5 MB, uncompressed FASTA) is archived on Zenodo (DOI: doi:10.5281/zenodo.19040165); `readAAStringSet` reads it directly.

Header format: Sequence headers follow the NCBI Virus export format. Dates and countries can be extracted directly using the package helper functions:

```
dates     <- extract_fasta_dates(fasta, option = 1)
countries <- extract_fasta_countries(fasta, position = 2)
```

Subsampling: To keep the package within CRAN size limits, the bundled sample was generated by random sampling of 100 sequences. The full provenance and reproducible sampling script are in `data-raw/sarscov2_ncbi.R` in the package source.

Provenance

Downloaded from the NCBI SARS-CoV-2 Data Hub (https://www.ncbi.nlm.nih.gov/labs/virus/vssi/) on **October 12, 2021** with the following filters:

1. Virus: Severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2), taxid: 2697049
2. Nucleotide completeness: complete
3. Protein: surface glycoprotein

Result: **n = 137,132 sequences**, file size ~181.5 MB (uncompressed).

License

NCBI sequence data is a US Government work and is in the public domain within the United States. Data from international contributors is subject to the INSDC open-access policy (Karsch-Mizrachi et al., 2025; doi:10.1093/nar/gkae1058). The compiled dataset is released under CC0 1.0 Universal. Individual GenBank accession numbers in the FASTA headers provide full traceability to original submissions.

Source

https://www.ncbi.nlm.nih.gov/labs/virus/vssi/

References

Clark K, Karsch-Mizrachi I, Lipman DJ, Ostell J, Sayers EW (2016). GenBank. *Nucleic Acids Research*, 44(D1), D67–D72. doi:10.1093/nar/gkv1276

Sayers EW, Bolton EE, Brister JR, et al. (2022). Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 50(D1), D20–D26. doi:10.1093/nar/gkab1112

NCBI Virus [Internet]. Bethesda (MD): National Library of Medicine (US), National Center for Biotechnology Information; [2020] – [cited 2021 Oct 12]. Available from: https://www.ncbi.nlm.nih.gov/labs/virus/vssi/

Examples

```
path  <- system.file("extdata", "sarscov2_sample.fasta.gz",
                     package = "ViralEntropR")
fasta <- Biostrings::readAAStringSet(path)

# Number of sequences in the demo sample
length(fasta)

# Inspect first 3 headers
names(fasta)[1:3]

# Extract collection dates
dates <- extract_fasta_dates(fasta, option = 1)
head(dates$corrected_dates)
```

```
# Extract countries
countries <- extract_fasta_countries(fasta, position = 2)
table(countries$countries)

# Convert to character matrix for pipeline entry
char_mat <- fasta_to_char_matrix(fasta)
dim(char_mat)

# The full 137,132-sequence dataset (~181.5 MB) is available on Zenodo:
# https://zenodo.org/records/19040165
# Download sequences.fasta manually and load with:
# fasta_full <- Biostrings::readAAStringSet("path/to/sequences.fasta")
```

`sarscov2_variants` *SARS-CoV-2 VOC/VOI Curated Variant Metadata*

Description

A pre-built named list containing curated biological and epidemiological metadata for 12 SARS-CoV-2 Variants of Concern (VOC) and Variants of Interest (VOI): Alpha, Beta, Delta, Epsilon, Eta, Gamma, Iota, Kappa, Lambda, Omicron, Theta, Zeta. Includes mutation profiles, nomenclature, temporal detection records, defining SNPs, and a fully citable reference table with 21 verified references.

Usage

```
data(sarscov2_variants)
```

Format

A named list of 13 elements (12 metadata fields plus a structured References object). See Details for per-element descriptions.

Details

The object is available automatically after `library(ViralEntropR)` via lazy loading. It is produced by `get_variants` from the curated Excel workbook `SARS_CoV_2_VOC_VOI.xlsx` (not bundled with the package; see `data-raw/sarscov2_variants.R` for the reproducible build script).

List elements:

`WHO_Label` List of 12. WHO variant label strings (e.g. `"Alpha"`, `"Omicron"`).

`Pango_Lineage` List of 12. Pango lineage designations (e.g. `"B.1.1.7"`).

`GISAID_Clade` Character(12). GISAID clade strings (e.g. `"GRY"`, `"GR/484A"`).

`Nextstrain_Clade` Character(12). Nextstrain clade strings (e.g. `"20I/501Y.V1"`).

`Country_First_Detected` List of 12. Country of first documented detection per variant.

`Date_Earliest_Sample` Character(12). Month-Year of earliest documented sample (e.g. `"Sep-2020"`).

`Date_First_Detected` Character(12). Month-Year of world-level first detection.

`Date_First_Detected_US` Character(12). Month-Year of first US detection.

`Spike_Mutations` List of 12. Character vectors of Spike protein mutations per variant, read from the Excel workbook.

`Mutation_Sites` List of 12. Integer vectors of amino acid positions extracted from `Spike_Mutations`.

`Defining_SNPs` List of 12. Canonical defining SNP strings (`NA` for variants not characterised in the reference set).

`Defining_SNP_Sites` List of 12. Integer positions extracted from `Defining_SNPs`.

`References` Named list with three elements: `$data` – data frame of 21 verified references (columns: ID, Authors, Year, Title, Journal_Source, Volume_Issue_Pages, DOI, URL, Type, Variants_Covered, Data_Field, Citation); `$display(variant = NULL)` – renders an interactive `datatable`, optionally filtered by WHO label (requires the **DT** package); `$cite(variant)` – returns formatted citation strings for a given WHO label, suitable for manuscript use.

References

The 21 curated references cover peer-reviewed articles, CDC MMWR government reports, and outbreak.info surveillance database records. Full provenance is available via `sarscov2_variants$References$data` or the interactive display.

Source

Compiled from 21 peer-reviewed and surveillance sources; see `sarscov2_variants$References$data` for the full reference table with DOIs and URLs. Built from `SARS_CoV_2_VOC_VOI.xlsx` via `get_variants`.

Examples

```
# Object is available immediately after library(ViralEntropR)

# --- Basic access ------------------------------------------------------

# All 12 WHO labels
unlist(sarscov2_variants$WHO_Label)

# Pango lineage for Alpha
sarscov2_variants$Pango_Lineage[[
  which(unlist(sarscov2_variants$WHO_Label) == "Alpha")
]]

# World detection dates for all variants
data.frame(
  Variant  = unlist(sarscov2_variants$WHO_Label),
  Detected = sarscov2_variants$Date_First_Detected,
  Country  = unlist(sarscov2_variants$Country_First_Detected)
)

# --- Mutation sites ----------------------------------------------------
```

```
# Defining SNPs and sites for Delta
idx <- which(unlist(sarscov2_variants$WHO_Label) == "Delta")
sarscov2_variants$Defining_SNPs[[idx]]
sarscov2_variants$Defining_SNP_Sites[[idx]]

# All Spike mutation sites for Omicron
sarscov2_variants$Mutation_Sites[[
  which(unlist(sarscov2_variants$WHO_Label) == "Omicron")
]]

# --- Nomenclature ------------------------------------------------------

data.frame(
  WHO        = unlist(sarscov2_variants$WHO_Label),
  PANGO      = unlist(sarscov2_variants$Pango_Lineage),
  GISAID     = sarscov2_variants$GISAID_Clade,
  Nextstrain = sarscov2_variants$Nextstrain_Clade
)

# --- References --------------------------------------------------------

# Full reference data frame (21 rows)
sarscov2_variants$References$data[,
  c("ID", "Authors", "Year", "Journal_Source")
]

# Formatted citation strings for Gamma
sarscov2_variants$References$cite("Gamma")

if (interactive() && requireNamespace("DT", quietly = TRUE)) {
  # Interactive DT table of all references
  sarscov2_variants$References$display()

  # Filtered to Omicron references only
  sarscov2_variants$References$display(variant = "Omicron")
}
```

`simulate_variant_evolution`

Simulate Viral Variant Evolution

Description

Generates a synthetic, ground-truth-traceable time series of viral amino-acid sequences in which a reference strain is progressively challenged by emerging variants under stochastic growth and pairwise competition. Designed as a controllable signal generator for benchmarking the entropy, Hellinger, and change-point components of the **ViralEntropR** pipeline; the dynamics are deliberately simplified rather than a faithful biological model of evolutionary fitness.

Usage

```
simulate_variant_evolution(
  ref_sequences,
  n_ref_months = 3L,
  start_date,
  end_date,
  variants_config,
  variant_intervals,
  n_new_mutations = 1L,
  mutation_rate = 2,
  mutation_rate_variability = 0.25,
  deleterious_rate = 0,
  n_deleterious_limit = 1L,
  n_sequences_total = 140L,
  ref_variability = FALSE,
  n_ref_sequences = 100L,
  prob_deletion_event = 0,
  n_rows_to_delete = 0L,
  seed = NULL
)
```

Arguments

`ref_sequences`	Either a single character string giving the reference amino-acid sequence, or a data.frame with a `Date` column and site columns named with integer position labels (`"1"`, `"2"`, ..., `as.character(L)`; one column per site). The sequence length `L` must be at least 3, since variant mutations and random substitution events are sampled from sites > 2 (sites 1 and 2 are reserved for reference variability when `ref_variability = TRUE`).
`n_ref_months`	Integer. Duration of the initial reference phase (months). Default `3`.
`start_date`	Character or Date. Simulation start.
`end_date`	Character or Date. Simulation end (inclusive month).
`variants_config`	Integer vector. Number of mutations per variant, e.g., `c(2, 6)`.
`variant_intervals`	Integer vector. Months between consecutive variant emergences (length `length(variants_config) - 1`).
`n_new_mutations`	Integer. Number of de-novo sequence *copies* introduced at a variant's first appearance. Despite the name, this controls the count of seeded sequences, not the number of mutations carried by the variant (which is set per-variant by `variants_config`). Default `1`.
`mutation_rate`	Numeric scalar or vector. Monthly growth multiplier per variant. Default `2`.
`mutation_rate_variability`	Numeric in `[0, 1)`. Fractional spread around `mutation_rate`: growth is drawn from `Uniform(mult*(1-v), mult*(1+v))`. Set to `0` for deterministic growth. Default `0.25`.

deleterious_rate
: Numeric in [0, 1]. Per-mutation probability of being flagged as "deleterious". Default 0.

n_deleterious_limit
: Integer. Per-site cap on the number of variant rows allowed to retain a flagged ("deleterious") allele in a given month; overflow is reverted to the reference allele at the flagged site only. See **Capped "Deleterious" Sites** in Details. Default 1.

n_sequences_total
: Integer. Total sequences generated over the full mutation phase (spread evenly per month). Default 140.

ref_variability
: Logical. If TRUE, introduces low-level variability at sites 1 and 2 of the reference pool. Requires n_ref_sequences >= 6 * n_ref_months for the variability rows to be correctly captured by the multi-variant fill pool; smaller values trigger a warning. Default FALSE.

n_ref_sequences
: Integer. When ref_sequences is a single string, the number of reference rows to create. Default 100.

prob_deletion_event
: Numeric in [0, 1]. Per-month probability of triggering a substitution event affecting n_rows_to_delete rows. (Parameter name retained for backward compatibility; the operation is substitution, not literal deletion. See Details.) Default 0.

n_rows_to_delete
: Integer. Number of rows affected when a substitution event fires (see prob_deletion_event). Default 0.

seed
: Integer or NULL. Random seed for reproducibility. Default NULL.

Details

Simulates the trajectory of a viral population over discrete monthly time steps. All stochastic components are reproducible under a fixed seed and are designed to expose a known emergence pattern that downstream detection methods can be evaluated against.

Simulation Phases:

1. **Reference Phase:** A stable period in which only the reference strain exists. Reference months are resampled with replacement to assemble n_ref_months monthly batches.
2. **Variant Emergence:** New variants are introduced at the monthly intervals specified by variant_intervals.
3. **Pairwise Growth:** Each month the newest variant grows alongside its immediate predecessor. Both grow stochastically: each variant's next-month count is its previous-month count times Uniform(mult*(1-variability), mult*(1+variability)). The two are not coupled by a shared frequency constraint — only by the hard monthly quota ceiling(n_sequences_total / (n_periods - n_ref_months)) that trims any overshoot. Older, non-paired variants enter the fill pool with sampling weights 2^(i+1).
4. **Capped "Deleterious" Sites:** A Bernoulli-random subset of each variant's mutated positions is flagged with probability deleterious_rate. At each post-emergence month the count of

variant rows still carrying the flagged allele is capped at `n_deleterious_limit` by reverting overflow sequences to the reference allele *at the flagged site only*. This is a per-site cap, not a genotype-level fitness penalty.

5. **Random Substitution Events:** Each month independently fires a Bernoulli coin flip with probability `prob_deletion_event`. If it fires, the last `n_rows_to_delete` rows of that month's batch receive a single non-reference amino-acid substitution at a randomly chosen site (sites 1 and 2 excluded), and those rows are flagged with `Delet = "Yes"`. Parameter names retain "deletion" / "delete" for backward compatibility; the operation itself is a substitution.

Value

An object of class `"viralSim"`, a named list:

`Simulation_Output`
: Data frame of all sequences, one column per site (named `"1"`, ..., `as.character(L)`) plus `Variant` (per-row strain label), `Phase` (`"R"` or `"RV"` for reference rows depending on whether the row matches the wild-type sequence; `"M1"`, `"M2"`, ... in mutation-phase months — note that `"Mn"` is a *month label* and applies to every row in that month regardless of strain), `Date`, `Period` (1-indexed month), and `Delet` (`"Yes"` for rows affected by substitution events, `"No"` otherwise).

`Variant_Details`
: List of per-variant metadata. Each element has `em` (emergence-month index), `pos` (mutated site indices), `flags` (logical vector of "deleterious" flags per mutated site), `vseq` (full mutated reference sequence as character vector), `mult` (growth multiplier), `last` (count in the most recent month), `cum` (cumulative count), and `repl` (logical flag set when the variant has been displaced by its successor).

`Simulation_Dates`
: Date vector of all simulated months.

`Baseline_Ref_Sequence`
: Character. The wild-type reference string.

`Delet_Records`
: Named list of substitution events keyed by date (`"YYYY-MM-DD"`). Each entry holds `period`, `date`, `site`, `old_aa`, `new_aa`, and `rows` (absolute row indices in `Simulation_Output` that were modified).

`Pool`
: Fill-pool data frame from the final multi-variant competition period (includes `._weight` column); `NULL` if at most one variant was ever active.

See Also

`partition_time_windows`, `calculate_hellinger_matrix`, `detect_changepoints_ecp`, `detect_changepoints_hdcp`

Examples

```
ref_seq <- "MKTIIALSYIFCLVFADYKDDDDK"

sim <- simulate_variant_evolution(
  ref_sequences   = ref_seq,
  n_ref_months    = 3,
  start_date      = "2021-01-01",
```

```
  end_date        = "2021-12-01",
  variants_config = c(3, 5),
  variant_intervals = c(4),
  n_sequences_total = 50,
  mutation_rate   = 1.5,
  seed            = 123
)
head(sim$Simulation_Output)
```

`tabulate_site_evolution` *Tabulate Site Frequency Evolution*

Description

Generates a frequency or proportion table showing the amino acid distribution at a specific site across multiple time partitions, optionally styled with **kableExtra** and saved as standalone HTML.

Usage

```
tabulate_site_evolution(
  partitions,
  site_index,
  labels = NULL,
  alphabet_size = 25L,
  zeros = TRUE,
  use_letters = TRUE,
  relative = FALSE,
  digits = 2L,
  col_width = "100px",
  highlight_col = NULL,
  background = "#f0f8ff",
  wrap_length = 10L,
  save = FALSE,
  save_extension = ".html",
  save_path = NULL,
  return_table = TRUE
)
```

Arguments

`partitions`	A list of data frames, typically produced by `partition_time_windows`. Each data frame must contain integer-encoded amino acid sequences as columns (values 1 to `alphabet_size`).
`site_index`	Integer. The column index (site) to analyse.
`labels`	Character vector. Column labels for each partition. Defaults to `names(partitions)`, or `"P1"`, `"P2"`, ... if unnamed.

alphabet_size	Integer. Total number of possible amino acid codes. Must match the encoding used during integer encoding (encode_aa_sequence produces values 1-25 by default). Default is 25L.
zeros	Logical. If TRUE (default), fills missing counts with 0. If FALSE, replaces zeros with ""; see Details for the type-coercion implication.
use_letters	Logical. If TRUE (default), uses decoded single-character codes from the 25-symbol alphabet (A, R, N, D, C, ..., V, B, Z, X, *, -) as row names. If FALSE, uses numeric codes 1 to alphabet_size.
relative	Logical. If TRUE, converts counts to proportions (column-wise division by partition size). Default is FALSE.
digits	Integer. Decimal places for rounding when relative = TRUE. Default is 2L.
col_width	Character. CSS width string applied to all columns (e.g. "100px"). Default is "100px".
highlight_col	Integer or NULL. 1-based column index (relative to the data columns, not counting the row-name column) of a partition to highlight with background. Out-of-range values trigger a warning and no highlight is applied. NULL (default) means no highlight.
background	Character. CSS background colour for the highlighted column. Default is "#f0f8ff" (light blue).
wrap_length	Integer. Character width at which to wrap long column labels using HTML line breaks. Default is 10L.
save	Logical. If TRUE, saves the rendered HTML table to disk via save_kable. Default is FALSE.
save_extension	Character. File extension for the saved file (including leading dot). Default is ".html".
save_path	Character or NULL. Directory in which to save the file. ... Must be supplied when save = TRUE. Default is NULL.
return_table	Logical. If TRUE (default), returns a named list with both the raw data frame and the styled kable object. If FALSE, returns only the styled kable.

Details

Aggregates amino acid counts per partition using get_site_counts, optionally converts to relative frequencies, applies kableExtra styling (column width, column highlighting, striped rows), and optionally saves to disk. Row names are decoded amino acid codes via decode_aa_sequence when use_letters = TRUE.

Empty partitions. Partitions containing no observations contribute all-zero columns. When relative = TRUE, division by zero is avoided by treating empty-column sums as 1, leaving proportions at zero. Inspect partition sizes via sapply(partitions, nrow) before interpreting the table.

Note on zeros = FALSE. Setting zeros = FALSE replaces numeric zeros with empty strings ("") for visual clarity in the kable. This conversion forces the underlying data frame to character storage; numeric operations (sum, mean, etc.) will not work on the returned table element. Use zeros = TRUE (default) if downstream numerical use is intended.

Value

If `return_table = TRUE`, a named list:

`table`	The raw count (or proportion) data frame, with row names corresponding to amino acid codes and column names corresponding to partition labels.
`styled`	The **kableExtra** HTML kable object.

If `return_table = FALSE`, returns only the styled kable object.

See Also

`partition_time_windows` for producing the typical input list of partitions; `get_site_counts` for the count-tabulation primitive; `decode_aa_sequence` for the alphabet code mapping; `calculate_hellinger_matrix` for the related cross-partition distance calculation on the same data shape.

Examples

```
p1 = data.frame(s1 = c(1L, 1L, 1L, 1L, 2L))
p2 = data.frame(s1 = c(1L, 1L, 2L, 2L, 2L))
parts = list(T1 = p1, T2 = p2)

# Default: counts, letters, no save
tbl = tabulate_site_evolution(parts, site_index = 1)
tbl$table

# Relative frequencies, highlight second partition
tbl2 = tabulate_site_evolution(parts, site_index = 1,
                               relative = TRUE, highlight_col = 2)
tbl2$table

# Numeric codes (skip alphabet decoding)
tbl3 = tabulate_site_evolution(parts, site_index = 1, use_letters = FALSE)
rownames(tbl3$table)
```

Index