

Package: gp3sequences (via r-universe)

October 9, 2026

Title Transparent Analysis of Ordered Categorical Sequences

Version 0.3.1

Description Provides transparent, reproducible, and auditable tools for validating, preparing, encoding, summarising, comparing, modelling, and diagnosing ordered categorical sequence data. Supports explicit preprocessing policies, contiguous motifs and bounded subsequences, consensus and group comparisons, edit and transition distances, clustering and stability diagnostics, transition networks, higher-order models, categorical, mixture, multichannel, and covariate hidden Markov models, longitudinal panel workflows, time-varying models, design-aware inference, analysis contracts and provenance audits, and guarded adapters to specialist sequence-analysis packages.

License MIT + file LICENSE

URL <https://stefanosbalaskas.github.io/gp3sequences/>,
<https://github.com/stefanosbalaskas/gp3sequences>

BugReports <https://github.com/stefanosbalaskas/gp3sequences/issues>

Suggests arules, arulesSequences, cluster, GrpString, igraph, knitr, mgcv, rmarkdown, seqHMM, testthat (>= 3.0.0), TraMineR

Config/testthat/edition 3

Config/Needs/website pkgdown

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Imports graphics, methods, stats, utils

VignetteBuilder knitr

Repository <https://stefanosbalaskas.r-universe.dev>

Date/Publication 2026-10-09 20:32:18 UTC

RemoteUrl <https://github.com/stefanosbalaskas/gp3sequences>

RemoteRef HEAD

RemoteSha a8494f9d09a95f701d59f356c69eb849767b1f5f

Contents

as_arules_sequences	3
as_grpstring_data	4
as_igraph_transition_network	5
as_seqhmm_sequences	6
as_traminer_sequences	7
audit_sequence_analysis	8
audit_sequence_data	9
bootstrap_sequence_clusters	10
bootstrap_sequence_group_difference	11
bootstrap_transition_network	12
cluster_sequences	13
compare_sequence_analysis_results	14
compare_sequence_groups	15
compare_sequence_hmms	17
compare_sequence_panel_changes	17
compare_sequence_subsequences	19
compute_sequence_distance	20
create_consensus_sequence	21
create_sequence_cluster_ensemble	22
create_transition_network	23
declare_sequence_comparison_design	24
decode_covariate_sequence_states	25
decode_multichannel_sequence_states	26
decode_sequence_states	27
detect_transition_communities	28
encode_sequence_data	29
extract_representative_sequences	30
extract_sequence_ngrams	31
extract_sequence_subsequences	33
filter_sequence_motifs	34
filter_sequence_subsequences	36
fit_covariate_sequence_hmm	37
fit_higher_order_transition_model	38
fit_multichannel_sequence_hmm	39
fit_sequence_hmm	41
fit_sequence_hmm_mixture	42
fit_time_varying_sequence_model	44
format_consensus_sequence	46
format_sequence_motif_positions	47
format_sequence_motifs	48
format_sequence_paths	50
plot_consensus_sequence	51
plot_multichannel_sequence_hmm	52
plot_sequence_cluster_silhouette	53
plot_sequence_distance_heatmap	54
plot_sequence_entropy	55

plot_sequence_group_comparison	56
plot_sequence_group_inference	57
plot_sequence_index	57
plot_sequence_motif_positions	58
plot_sequence_motifs	60
plot_sequence_panel_changes	61
plot_sequence_state_distribution	62
plot_sequence_subsequences	63
plot_time_varying_sequence_model	64
plot_transition_network	64
predict_covariate_transition_probabilities	65
predict_next_state	66
predict_time_varying_sequence_model	67
prepare_gp3tools_sequences	67
prepare_sequence_data	68
prepare_sequence_panel	70
sequence_capabilities	71
summarise_consensus_agreement	72
summarise_covariate_sequence_hmm	73
summarise_multichannel_sequence_hmm	74
summarise_sequence_cluster_stability	74
summarise_sequence_distance	75
summarise_sequence_group_inference	76
summarise_sequence_hmm	76
summarise_sequence_motif_positions	77
summarise_sequence_motifs	78
summarise_sequence_panel	80
summarise_sequence_states	80
summarise_sequence_subsequences	82
summarise_sequence_transitions	83
summarise_time_varying_sequence_model	84
summarise_transition_centrality	85
test_sequence_group_difference	86
validate_sequence_clusters	87
validate_sequence_data	88

Index	90
--------------	-----------

as_arules_sequences	<i>Convert sequence data to cSPADE transaction input</i>
---------------------	--

Description

Convert sequence data to cSPADE transaction input

Usage

```
as_arules_sequences(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state"
)
```

Arguments

`data` Long-format sequence data.
`sequence_id_col, order_col, state_col`
 Sequence columns.

Value

An arules transactions object whose transaction information contains positive integer sequenceID and eventID fields required by `arulesSequences::cspade()`.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
if (requireNamespace("arules", quietly = TRUE) &&
    requireNamespace("arulesSequences", quietly = TRUE)) {
  as_arules_sequences(sequences)
}
```

as_grpstring_data

Create GrpString-compatible event and string inputs

Description

Create GrpString-compatible event and string inputs

Usage

```
as_grpstring_data(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
```

```

    state_col = "state",
    alphabet = NULL
  )

```

Arguments

data Long-format sequence data.

sequence_id_col, order_col, state_col Sequence columns.

alphabet Optional single-character symbols. When omitted, printable ASCII characters are assigned deterministically.

Value

A list of class `gp3_grpstring_input` containing a wide event data frame, event-name vector, character vector, conversion key, and string vector.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
as_grpstring_data(sequences)

```

`as_igraph_transition_network`

Convert a transition network to an igraph object

Description

Convert a transition network to an igraph object

Usage

```
as_igraph_transition_network(network, directed = TRUE)
```

Arguments

network A first-order transition network.

directed Whether the resulting graph is directed.

Value

An igraph graph with edge attributes copied from the network.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
network <- create_transition_network(sequences)
if (requireNamespace("igraph", quietly = TRUE)) {
  as_igraph_transition_network(network)
}
```

as_seqhmm_sequences	<i>Convert sequence data to seqHMM observations</i>
---------------------	---

Description

Convert sequence data to seqHMM observations

Usage

```
as_seqhmm_sequences(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  ...
)
```

Arguments

data	Long-format sequence data.
sequence_id_col, order_col, state_col	Sequence columns.
...	Additional arguments passed to as_traminer_sequences() .

Value

A TraMineR stslist suitable for seqHMM::build_hmm().

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
if (requireNamespace("TraMineR", quietly = TRUE) &&
    requireNamespace("seqHMM", quietly = TRUE)) {
  as_seqhmm_sequences(sequences)
}

```

as_traminer_sequences *Convert sequence data to a TraMineR state-sequence object*

Description

Convert sequence data to a TraMineR state-sequence object

Usage

```

as_traminer_sequences(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  missing = NA,
  right = "DEL",
  ...
)

```

Arguments

data	Long-format sequence data.
sequence_id_col, order_col, state_col	Sequence columns.
missing	Missing-state code passed to <code>TraMineR::seqdef()</code> .
right	Right-missing policy passed to <code>TraMineR::seqdef()</code> .
...	Additional arguments passed to <code>TraMineR::seqdef()</code> .

Value

A TraMineR `stslst` object with original sequence identifiers as row names.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
if (requireNamespace("TraMineR", quietly = TRUE)) {
  as_traminer_sequences(sequences)
}

```

audit_sequence_analysis

Audit a gp3sequences analysis object

Description

Inspects supported gp3sequences objects for structural contract validity, recoverable provenance, identifiers, state levels, method settings, and randomness metadata. The audit does not reinterpret sequence structure as a psychological, diagnostic, cognitive, emotional, or causal construct.

Usage

```
audit_sequence_analysis(x, strict = FALSE, tolerance = 1e-08)
```

Arguments

<code>x</code>	A gp3sequences analysis result or supported structural object.
<code>strict</code>	Logical; if TRUE, fail when the audit status is "fail".
<code>tolerance</code>	Numerical tolerance used for matrix/probability checks.

Value

An object of class `gp3_sequence_analysis_audit` containing summary, issues, provenance, contract, and status.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3"), each = 3L),
  sequence_order = rep(1:3, times = 3L),
  state = c(
    "A", "B", "C",
    "A", "B", "B",
    "C", "B", "A"
  )
)

```

```

    )
  )
  distance <- compute_sequence_distance(
    sequences,
    method = "levenshtein"
  )
  audit_sequence_analysis(distance)

```

audit_sequence_data *Audit Long-Format Sequence Data*

Description

Examines a long-format data frame against the neutral gp3sequences sequence-data contract without modifying the input.

Usage

```

audit_sequence_data(
  data,
  sequence_id_col,
  order_col,
  state_col,
  duration_col = NULL,
  metadata_cols = NULL,
  expected_states = NULL
)

```

Arguments

data	A data frame containing ordered state observations.
sequence_id_col	Name of the sequence identifier column.
order_col	Name of the numeric sequence-order column.
state_col	Name of the categorical state column.
duration_col	Optional name of a numeric duration column.
metadata_cols	Optional character vector naming columns that should remain constant within each sequence.
expected_states	Optional vector of known or permitted state values.

Details

The audit checks column mappings, empty inputs, missing identifiers, missing or non-numeric order values, duplicated positions, integer order gaps, unordered rows, missing states, consecutive repeated states, single-row sequences, invalid durations, inconsistent metadata, unexpected states, and unused factor levels.

The function reports structural properties only. It does not infer psychological, cognitive, emotional, or diagnostic states.

Value

A data frame with one row per detected issue and the stable columns `sequence_id`, `row`, `column`, `issue_code`, `severity`, `value`, `message`, and `action`. Severity values are `error`, `review`, and `info`.

Examples

```
sequences <- data.frame(
  id = rep(c("s1", "s2"), each = 3L),
  position = rep(1:3, times = 2L),
  state = c("home", "search", "product", "home", "category", "product")
)

audit_sequence_data(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)
```

bootstrap_sequence_clusters

Bootstrap sequence-cluster stability

Description

Uses repeated subsampling without replacement and records pairwise co-clustering agreement relative to the full-data solution.

Usage

```
bootstrap_sequence_clusters(
  distance,
  k,
  method = c("hierarchical", "pam", "clara"),
  n_boot = 100L,
  sample_fraction = 0.8,
```

```

    seed = 1L,
    linkage = "average",
    ...
)

```

Arguments

distance	Sequence distance object.
k	Number of clusters.
method	Clustering method.
n_boot	Number of subsamples.
sample_fraction	Fraction of sequences sampled in each iteration.
seed	Reproducibility seed.
linkage	Hierarchical linkage.
...	Additional clustering arguments.

Value

An object of class `gp3_sequence_cluster_bootstrap`.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
distance <- compute_sequence_distance(sequences)
bootstrap_sequence_clusters(distance, k = 2L, n_boot = 5L, seed = 1L)

```

bootstrap_sequence_group_difference

Bootstrap a sequence group difference

Description

Resamples declared independent units within groups and returns a percentile interval for the mean difference. This interval does not by itself license causal interpretation.

Usage

```
bootstrap_sequence_group_difference(
  inference,
  n_boot = 999L,
  level = 0.95,
  seed = 1L
)
```

Arguments

<code>inference</code>	A result from <code>test_sequence_group_difference()</code> .
<code>n_boot</code>	Number of bootstrap samples.
<code>level</code>	Confidence level.
<code>seed</code>	Reproducibility seed.

Value

The inference object with a bootstrap component.

Examples

```
# See `test_sequence_group_difference()`.
```

<code>bootstrap_transition_network</code>	<i>Bootstrap transition-network edge weights</i>
---	--

Description

Bootstrap transition-network edge weights

Usage

```
bootstrap_transition_network(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  n_boot = 100L,
  level = 0.95,
  seed = 1L,
  include_self = TRUE
)
```

Arguments

data Long-format sequence data.
sequence_id_col, order_col, state_col Sequence columns.
n_boot Number of bootstrap samples of whole sequences.
level Confidence level for percentile intervals.
seed Reproducibility seed.
include_self Include self-transitions.

Value

A data frame containing observed edge weights, bootstrap means, standard deviations, and percentile intervals.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
bootstrap_transition_network(sequences, n_boot = 5L, seed = 1L)

```

cluster_sequences	<i>Cluster sequences from a distance object</i>
-------------------	---

Description

Cluster sequences from a distance object

Usage

```

cluster_sequences(
  distance,
  k,
  method = c("hierarchical", "pam", "clara"),
  linkage = "average",
  seed = 1L,
  ...
)

```

Arguments

distance	A dist object or square matrix.
k	Number of clusters.
method	"hierarchical", "pam", or "clara".
linkage	Hierarchical linkage method.
seed	Reproducibility seed for optional stochastic methods.
...	Additional arguments passed to the selected clustering function.

Value

A list of class `gp3_sequence_clustering` containing assignments, model object, medoid identifiers where available, and settings.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
distance <- compute_sequence_distance(sequences)
cluster_sequences(distance, k = 2L)
```

```
compare_sequence_analysis_results
```

Compare two gp3sequences analysis results

Description

Compares structural contracts and recoverable provenance from two analysis objects. This helper is intended for regression checks, sensitivity analyses, and reproducibility audits; it does not select a statistically or substantively "best" analysis.

Usage

```
compare_sequence_analysis_results(
  x,
  y,
  tolerance = 1e-08,
  compare_values = FALSE
)
```

Arguments

<code>x, y</code>	Two analysis objects.
<code>tolerance</code>	Numerical comparison tolerance.
<code>compare_values</code>	Logical; when TRUE, also run <code>all.equal()</code> on the complete objects after structural comparison.

Value

An object of class `gp3_sequence_analysis_comparison` containing per-field comparisons, both audits, optional whole-object comparison, and an overall equality flag.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3"), each = 3L),
  sequence_order = rep(1:3, times = 3L),
  state = c(
    "A", "B", "C",
    "A", "B", "B",
    "C", "B", "A"
  )
)
d1 <- compute_sequence_distance(
  sequences,
  method = "levenshtein"
)
d2 <- compute_sequence_distance(
  sequences,
  method = "levenshtein"
)
compare_sequence_analysis_results(d1, d2)
```

`compare_sequence_groups`

Compare sequence groups descriptively

Description

Produces descriptive state, transition and sequence-length comparisons. No hypothesis tests or causal interpretations are produced.

Usage

```
compare_sequence_groups(
  data,
  group_col,
  sequence_id_col = "sequence_id",
```

```

order_col = "sequence_order",
state_col = "state",
reference = NULL,
metrics = c("state", "transition", "length"),
include_self = TRUE,
transition_separator = " -> ",
zero_policy = c("missing", "infinite")
)

```

Arguments

<code>data</code>	Long-format sequence data.
<code>group_col</code>	Grouping column that is constant within sequence.
<code>sequence_id_col</code> , <code>order_col</code> , <code>state_col</code>	Sequence columns.
<code>reference</code>	Optional reference group. When supplied, each other group is reported as <code>group_1</code> and the reference as <code>group_2</code> ; differences and ratios are therefore other-minus-reference and other/reference. When omitted, all pairwise group contrasts are returned.
<code>metrics</code>	Any of "state", "transition", and "length".
<code>include_self</code>	Include self-transitions.
<code>transition_separator</code>	Separator used in reported transition labels. It must not occur inside an observed state label.
<code>zero_policy</code>	Ratio policy when the reference value is zero.

Value

A list of class `gp3_sequence_group_comparison` containing group summaries and pairwise descriptive contrasts.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
compare_sequence_groups(sequences, group_col = "group")

```

compare_sequence_hmms *Compare fitted sequence HMMs descriptively*

Description

Compare fitted sequence HMMs descriptively

Usage

```
compare_sequence_hmms(...)
```

Arguments

... Fitted HMM or mixture-HMM objects.

Value

A data frame containing log likelihood, AIC, BIC, parameter count, convergence status, and delta criteria. Criteria are descriptive and do not automatically select a substantive model.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
model_1 <- fit_sequence_hmm(sequences, 1L, max_iter = 5L)
model_2 <- fit_sequence_hmm(sequences, 2L, max_iter = 5L)
compare_sequence_hmms(one = model_1, two = model_2)
```

compare_sequence_panel_changes
Compare within-panel sequence changes

Description

Computes distance and simple structural changes between consecutive panel occasions. The output is descriptive and does not establish a causal or psychological change.

Usage

```
compare_sequence_panel_changes(
  panel,
  method = c("levenshtein", "lcs", "optimal_matching", "transition"),
  normalise = c("none", "max_length", "path_length"),
  indel_cost = 1,
  substitution_cost = 1,
  substitution_matrix = NULL,
  transition_smoothing = 0
)
```

Arguments

<code>panel</code>	A sequence panel.
<code>method</code>	Distance method supported by <code>compute_sequence_distance()</code> .
<code>normalise</code>	Distance normalisation.
<code>indel_cost</code> , <code>substitution_cost</code>	Costs for optimal matching.
<code>substitution_matrix</code>	Optional named substitution matrix.
<code>transition_smoothing</code>	Smoothing for transition-profile distance.

Value

A data frame of class `gp3_sequence_panel_changes`.

Examples

```
panel_data <- data.frame(
  participant_id = rep(c("p1", "p2"), each = 8L),
  occasion = rep(rep(c(1, 2), each = 4L), times = 2L),
  sequence_id = rep(c("a", "b", "c", "d"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "C", "C", "D",
            "D", "C", "B", "A", "D", "B", "B", "A")
)
compare_sequence_panel_changes(
  prepare_sequence_panel(panel_data, "participant_id", "occasion"),
  method = "lcs"
)
```

compare_sequence_subsequences

Compare subsequence prevalence between groups

Description

Performs transparent contingency-table tests on sequence-level presence. Multiple testing is adjusted explicitly. Results are associational unless a randomized design independently justifies causal interpretation.

Usage

```
compare_sequence_subsequences(
  occurrences,
  group_col,
  test = c("auto", "chisq", "fisher"),
  p_adjust = "BH",
  min_sequence_count = 1L
)
```

Arguments

occurrences	A non-contiguous subsequence occurrence table.
group_col	Sequence-constant group column retained through metadata_cols during extraction.
test	"auto", "chisq", or "fisher".
p_adjust	Multiple-testing adjustment method.
min_sequence_count	Minimum total sequence count per subsequence.

Value

A data frame of prevalence contrasts and adjusted p-values.

Examples

```
data <- data.frame(
  sequence_id = rep(paste0("s", 1:4), each = 4L),
  sequence_order = rep(1:4, 4L),
  state = c("A", "B", "C", "D", "A", "B", "D", "D",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L)
)
occurrences <- extract_sequence_subsequences(data, metadata_cols = "group")
compare_sequence_subsequences(occurrences, "group")
```

```
compute_sequence_distance
```

Compute pairwise sequence distances

Description

Provides transparent base-R implementations of Levenshtein edit distance, longest-common-subsequence distance, optimal matching, and a first-order transition-profile distance.

Usage

```
compute_sequence_distance(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  method = c("levenshtein", "lcs", "optimal_matching", "transition"),
  indel_cost = 1,
  substitution_cost = 1,
  substitution_matrix = NULL,
  transition_smoothing = 0,
  normalise = c("none", "max_length", "path_length")
)
```

Arguments

<code>data</code>	Long-format sequence data or a prepared <code>gp3sequences</code> result.
<code>sequence_id_col</code> , <code>order_col</code> , <code>state_col</code>	Sequence columns.
<code>method</code>	Distance method.
<code>indel_cost</code>	Non-negative insertion/deletion cost used by <code>method = "optimal_matching"</code> .
<code>substitution_cost</code>	Non-negative default substitution cost used by <code>method = "optimal_matching"</code> .
<code>substitution_matrix</code>	Optional named square substitution-cost matrix for <code>method = "optimal_matching"</code> .
<code>transition_smoothing</code>	Non-negative smoothing for transition profiles.
<code>normalise</code>	One of "none", "max_length", or "path_length".

Value

A `dist` object with method and preprocessing metadata attached.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
compute_sequence_distance(sequences, method = "lcs")

```

```
create_consensus_sequence
```

Create an aligned-position consensus sequence

Description

Computes a descriptive consensus state at each observed sequence position. The function does not treat the consensus as a behavioural norm and does not infer psychological or causal meaning.

Usage

```

create_consensus_sequence(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  group_cols = NULL,
  weight_col = NULL,
  missing_state_policy = c("exclude", "state", "error"),
  missing_state_label = "<MISSING>",
  tie_method = c("first", "last", "missing", "all"),
  state_levels = NULL,
  min_support = 1L
)

```

Arguments

<code>data</code>	A long-format data frame, a prepared gp3sequences result, or a data frame containing canonical columns.
<code>sequence_id_col</code> , <code>order_col</code> , <code>state_col</code>	Column names defining sequences.
<code>group_cols</code>	Optional columns defining independent consensus groups.
<code>weight_col</code>	Optional non-negative row-weight column.
<code>missing_state_policy</code>	One of "exclude", "state", or "error".

missing_state_label	Label used when missing states are retained.
tie_method	Deterministic tie policy: "first", "last", "missing", or "all".
state_levels	Optional preferred state ordering used for ties.
min_support	Minimum number of contributing sequences at a position.

Value

A data frame of class `gp3_consensus_sequence` containing group columns, sequence position, consensus state, support counts and weights, agreement proportion, tie count, tied states, and total group sequences.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
create_consensus_sequence(sequences, group_cols = "group")
```

`create_sequence_cluster_ensemble`

Create a sequence-cluster ensemble

Description

Combines multiple named cluster assignments through a co-association matrix and applies hierarchical clustering to 1 - co-association.

Usage

```
create_sequence_cluster_ensemble(..., k, linkage = "average")
```

Arguments

<code>...</code>	Two or more <code>gp3_sequence_clustering</code> objects or named assignment vectors.
<code>k</code>	Number of consensus clusters.
<code>linkage</code>	Hierarchical linkage applied to the co-association distance.

Value

An object of class `gp3_sequence_cluster_ensemble` containing the consensus assignments, co-association matrix, model, and source solutions.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
d1 <- compute_sequence_distance(sequences, method = "lcs")
d2 <- compute_sequence_distance(sequences, method = "transition")
create_sequence_cluster_ensemble(cluster_sequences(d1, 2L),
                                cluster_sequences(d2, 2L), k = 2L)

```

```
create_transition_network
```

Create a transition network from ordered sequences

Description

Create a transition network from ordered sequences

Usage

```

create_transition_network(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  group_cols = NULL,
  order = 1L,
  include_self = TRUE,
  normalise = c("count", "from", "global"),
  smoothing = 0,
  context_separator = " > "
)

```

Arguments

data	Long-format sequence data or a prepared result.
sequence_id_col, order_col, state_col	Sequence columns.
group_cols	Optional grouping columns constant within sequence.
order	Markov order. 1 creates ordinary state-to-state edges; larger values create context-to-next-state edges.
include_self	Include first-order self-transitions.

normalise	Edge weight scale: counts, conditional probabilities from each context, or global shares.
smoothing	Non-negative additive smoothing applied to observed edges.
context_separator	Separator used for higher-order contexts.

Value

A data frame of class `gp3_transition_network` containing context, next state, counts, weights, sequence prevalence, and group columns.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
create_transition_network(sequences, normalise = "from")
```

```
declare_sequence_comparison_design
```

Declare a sequence group-comparison design

Description

Records the unit of assignment and the design under which a sequence-group contrast will be evaluated. Causal interpretation is permitted only for an explicitly randomized design and still depends on the validity of the study implementation.

Usage

```
declare_sequence_comparison_design(
  group_col,
  unit_col,
  design = c("observational", "randomized", "paired_randomized"),
  pair_col = NULL,
  cluster_col = NULL
)
```

Arguments

group_col	Group or treatment column.
unit_col	Independent assignment or analysis-unit column.
design	"observational", "randomized", or "paired_randomized".
pair_col	Pair or block column required for paired randomization.
cluster_col	Optional higher-level assignment cluster.

Value

An object of class gp3_sequence_comparison_design.

Examples

```
declare_sequence_comparison_design("group", "participant_id",
                                   design = "randomized")
```

decode_covariate_sequence_states

Decode states from a covariate-dependent HMM

Description

Decode states from a covariate-dependent HMM

Usage

```
decode_covariate_sequence_states(
  model,
  data = NULL,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  method = c("viterbi", "posterior")
)
```

Arguments

model	A fitted covariate HMM.
data	Optional new data. Training data are used when omitted.
sequence_id_col, order_col, state_col	Core columns for new data.
method	"viterbi" or "posterior".

Value

A long decoded-state data frame.

Examples

```
# See `fit_covariate_sequence_hmm()`.
```

```
decode_multichannel_sequence_states
```

Decode latent states from a multichannel HMM

Description

Decode latent states from a multichannel HMM

Usage

```
decode_multichannel_sequence_states(
  model,
  data = NULL,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  channel_cols = NULL,
  method = c("viterbi", "posterior")
)
```

Arguments

model	A fitted multichannel HMM.
data	Optional new long-format multichannel data.
sequence_id_col, order_col	Core sequence columns for new data.
channel_cols	Channel columns for new data; defaults to training names.
method	"viterbi" or "posterior".

Value

A long data frame of decoded states and posterior probabilities.

Examples

```
# See `fit_multichannel_sequence_hmm()`.
```

 decode_sequence_states

Decode hidden states from a fitted HMM

Description

Decode hidden states from a fitted HMM

Usage

```
decode_sequence_states(
  model,
  data = NULL,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  method = c("viterbi", "posterior"),
  component = NULL
)
```

Arguments

model	A fitted single HMM or HMM mixture.
data	Optional new long-format data. Training sequences are used when omitted.
sequence_id_col, order_col, state_col	Sequence columns for new data.
method	"viterbi" or "posterior".
component	Mixture component to decode. When omitted for a mixture, each sequence uses its highest-responsibility component.

Value

A long data frame containing decoded latent states and posterior probabilities where available.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
model <- fit_sequence_hmm(sequences, 2L, max_iter = 5L)
decode_sequence_states(model)
```

detect_transition_communities

Detect descriptive transition communities

Description

Detect descriptive transition communities

Usage

```
detect_transition_communities(
  network,
  method = c("label_propagation", "components"),
  max_iter = 100L,
  seed = 1L
)
```

Arguments

network	A first-order transition network.
method	"label_propagation" or "components".
max_iter	Maximum label-propagation iterations.
seed	Reproducibility seed used only to rotate update order.

Value

A data frame with states and deterministic community labels.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
network <- create_transition_network(sequences)
detect_transition_communities(network)
```

encode_sequence_data	<i>Encode Ordered Sequence States</i>
----------------------	---------------------------------------

Description

Creates a deterministic dictionary and adds integer and labelled state codes to long-format sequence data.

Usage

```
encode_sequence_data(
  data,
  sequence_id_col,
  order_col,
  state_col,
  duration_col = NULL,
  metadata_cols = NULL,
  expected_states = NULL,
  state_levels = NULL,
  prefix = "S",
  width = NULL
)
```

Arguments

data	A data frame containing ordered state observations.
sequence_id_col	Name of the sequence identifier column.
order_col	Name of the numeric sequence-order column.
state_col	Name of the categorical state column.
duration_col	Optional name of a numeric duration column.
metadata_cols	Optional character vector naming columns that should remain constant within each sequence.
expected_states	Optional vector of known or permitted state values.
state_levels	Optional atomic vector defining the complete state ordering. When omitted, factor levels are respected; otherwise observed state labels are sorted alphabetically.
prefix	Character prefix used for labelled codes.
width	Optional positive integer width for the numeric part of each labelled code. The default is determined from the dictionary size.

Details

The function does not reinterpret states. State codes are transparent identifiers derived from an explicit or deterministic state ordering.

Value

A named list containing:

- data: deterministically sorted canonical data with state_index and state_code;
- dictionary: state labels, integer indices, labelled codes, and an observed-state indicator;
- audit, status, and mapping from input validation;
- settings: the resolved code prefix and width.

Examples

```
sequences <- data.frame(
  id = c("s1", "s1", "s2", "s2"),
  position = c(1, 2, 1, 2),
  state = c("home", "search", "home", "product")
)

encoded <- encode_sequence_data(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)

encoded$dictionary
encoded$data
```

```
extract_representative_sequences
```

Extract representative sequences from clusters

Description

Extract representative sequences from clusters

Usage

```
extract_representative_sequences(
  clustering,
  distance = NULL,
  n_per_cluster = 1L
)
```

Arguments

clustering	A clustering result.
distance	Optional distance when absent from clustering.
n_per_cluster	Number of representatives per cluster.

Value

A data frame with cluster, rank, representative ID, and mean within-cluster distance.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
distance <- compute_sequence_distance(sequences)
fit <- cluster_sequences(distance, k = 2L)
extract_representative_sequences(fit)
```

extract_sequence_ngrams

Extract Contiguous Sequence N-Grams

Description

Enumerates contiguous state motifs from validated long-format sequence data.

Usage

```
extract_sequence_ngrams(
  data,
  sequence_id_col,
  order_col,
  state_col,
  duration_col = NULL,
  metadata_cols = NULL,
  expected_states = NULL,
  min_length = 2L,
  max_length = 3L,
  overlap = c("allow", "disallow"),
  separator = " > ",
  state_levels = NULL
)
```

Arguments

data A data frame containing ordered state observations.

sequence_id_col Name of the sequence identifier column.

<code>order_col</code>	Name of the numeric sequence-order column.
<code>state_col</code>	Name of the categorical state column.
<code>duration_col</code>	Optional name of a numeric duration column.
<code>metadata_cols</code>	Optional character vector naming columns that should remain constant within each sequence.
<code>expected_states</code>	Optional vector of known or permitted state values.
<code>min_length</code>	Positive whole number giving the shortest motif length.
<code>max_length</code>	Positive whole number giving the longest motif length.
<code>overlap</code>	Character value specifying whether overlapping occurrences of the same motif within the same sequence are "allow"ed or "disallow"ed.
<code>separator</code>	Character value used only to display state labels in the human-readable motif column.
<code>state_levels</code>	Optional atomic vector defining the complete state ordering. When omitted, factor levels are respected; otherwise observed labels are sorted deterministically.

Details

Motifs are contiguous windows only. No subsequence gaps, edit distances, statistical tests, or substantive interpretations are introduced. Consecutive repeated states are used exactly as supplied; any repeat collapsing should be performed explicitly with `prepare_sequence_data()` before extraction.

With `overlap = "disallow"`, overlap is resolved independently within each sequence-motif pair by a deterministic left-to-right greedy rule. Different motifs and different motif lengths do not compete for positions.

The collision-resistant `motif_id` and `motif_key` columns are derived from deterministic state codes. The display separator may therefore also occur inside a state label without changing motif identity.

Value

A named list containing:

- `occurrences`: one row per retained contiguous motif occurrence;
- `motifs`: the distinct motif dictionary;
- `sequences`: sequence-level counts of candidate and retained occurrences;
- `state_dictionary`: the deterministic state dictionary;
- `audit`, `status`, and mapping from input validation;
- `settings`: the resolved motif extraction settings.

Examples

```

sequences <- data.frame(
  id = c(rep("s1", 5L), rep("s2", 4L)),
  position = c(1:5, 1:4),
  state = c("A", "B", "A", "B", "A", "A", "B", "A", "C")
)

ngrams <- extract_sequence_ngrams(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state",
  min_length = 2,
  max_length = 3,
  overlap = "allow"
)

ngrams$occurrences
ngrams$motifs

```

extract_sequence_subsequences

Extract bounded non-contiguous sequence subsequences

Description

Enumerates ordered non-contiguous subsequences under explicit length, gap, span, and safety limits. The implementation is transparent and intended for modest sequence collections; specialist mining packages remain preferable for very large search spaces.

Usage

```

extract_sequence_subsequences(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  metadata_cols = NULL,
  min_length = 2L,
  max_length = 5L,
  max_gap = Inf,
  max_span = Inf,
  repeated_state_policy = c("preserve", "collapse"),
  separator = ">",
  max_combinations_per_sequence = 100000L
)

```

Arguments

data Long-format sequence data or a prepared result.
sequence_id_col, order_col, state_col Core sequence columns.
metadata_cols Optional sequence-constant metadata retained as attributes.
min_length, max_length Minimum and maximum subsequence lengths.
max_gap Maximum number of skipped positions between adjacent selected states. Use Inf for no restriction.
max_span Maximum difference between the first and last selected sequence positions. Use Inf for no restriction.
repeated_state_policy Preserve or collapse consecutive repeated states before mining.
separator Separator used in stable motif labels.
max_combinations_per_sequence Safety limit for the number of index combinations considered for any one sequence.

Value

A data frame of class `gp3_sequence_subsequences`, with one row per qualifying occurrence.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2"), each = 5L),
  sequence_order = rep(1:5, times = 2L),
  state = c("A", "B", "C", "D", "E", "A", "C", "B", "D", "E")
)
extract_sequence_subsequences(sequences, min_length = 2L, max_length = 3L,
                              max_gap = 2L)

```

filter_sequence_motifs

Filter Sequence Motif Summaries

Description

Applies transparent count, prevalence, length, and top-ranking filters to sequence motif summaries.

Usage

```
filter_sequence_motifs(
  x,
  min_occurrences = 1L,
  min_sequences = 1L,
  min_prevalence = 0,
  motif_lengths = NULL,
  top_n = NULL,
  rank_by = c("sequence_prevalence", "n_occurrences", "n_sequences"),
  ties = c("include", "first")
)
```

Arguments

<code>x</code>	A motif extraction, motif summary, or filtered-motif object.
<code>min_occurrences</code>	Non-negative whole number giving the minimum total occurrence count.
<code>min_sequences</code>	Non-negative whole number giving the minimum number of sequences containing the motif.
<code>min_prevalence</code>	Minimum sequence prevalence between 0 and 1.
<code>motif_lengths</code>	Optional vector of positive whole-number motif lengths to retain.
<code>top_n</code>	Optional positive whole number giving the requested number of highest-ranked motifs.
<code>rank_by</code>	Metric used for deterministic top-ranking: one of "sequence_prevalence", "n_occurrences", or "n_sequences".
<code>ties</code>	Character value controlling the top-n boundary. "include" retains every motif tied on rank_by with the final selected motif; "first" retains exactly top_n motifs after deterministic secondary sorting.

Details

Filtering is descriptive and deterministic. When `ties = "first"`, ties are resolved by sequence prevalence, occurrence count, sequence count, shorter motif length, and finally the collision-resistant motif key.

Value

A named list containing the filtered `motifs` table, the matching sequence-level rows in `by_sequence`, sequence and state dictionaries, validation metadata, filter settings, and counts before and after filtering.

Examples

```
sequences <- data.frame(
  id = c(rep("s1", 5L), rep("s2", 4L)),
  position = c(1:5, 1:4),
  state = c("A", "B", "A", "B", "A", "A", "B", "A", "C")
)
```

```

)

extracted <- extract_sequence_ngrams(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)

filtered <- filter_sequence_motifs(
  extracted,
  min_sequences = 2,
  top_n = 5,
  ties = "include"
)

filtered$motifs

```

filter_sequence_subsequences

Filter non-contiguous subsequence summaries

Description

Filter non-contiguous subsequence summaries

Usage

```

filter_sequence_subsequences(
  summary,
  min_sequences = 1L,
  min_prevalence = 0,
  max_mean_gap = Inf,
  top_n = NULL,
  ties = c("include", "exclude")
)

```

Arguments

summary	A data frame from summarise_sequence_subsequences() .
min_sequences	Minimum sequence count.
min_prevalence	Minimum sequence prevalence.
max_mean_gap	Optional maximum mean gap.
top_n	Optional number of rows retained after deterministic sorting.
ties	Include or exclude ties at the top_n boundary.

Value

A filtered summary data frame.

Examples

```
data <- data.frame(sequence_id = rep(c("a", "b"), each = 4L),
                  sequence_order = rep(1:4, 2L),
                  state = c("A", "B", "C", "D", "A", "C", "B", "D"))
x <- summarise_sequence_subsequences(extract_sequence_subsequences(data))
filter_sequence_subsequences(x, min_sequences = 2L)
```

fit_covariate_sequence_hmm

Fit a covariate-dependent categorical hidden Markov model

Description

Fits a categorical HMM whose initial-state and transition probabilities may depend on explicitly declared numeric covariates. Multinomial-logit coefficients are estimated inside the EM algorithm with a small ridge penalty. Emission probabilities remain time-homogeneous.

Usage

```
fit_covariate_sequence_hmm(
  data,
  n_states,
  initial_covariate_cols = NULL,
  transition_covariate_cols = NULL,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  symbol_levels = NULL,
  state_names = NULL,
  emission_probs = NULL,
  max_iter = 100L,
  inner_maxit = 100L,
  tolerance = 1e-06,
  pseudocount = 1e-06,
  ridge = 1e-06,
  seed = 1L,
  keep_posteriors = FALSE
)
```

Arguments

data	Long-format sequence data.
n_states	Number of latent states.

`initial_covariate_cols` Numeric sequence-constant covariates for initial-state probabilities.
`transition_covariate_cols` Numeric row-level covariates for transition probabilities.
`sequence_id_col, order_col, state_col` Core sequence columns.
`symbol_levels` Optional observed-symbol order.
`state_names` Optional latent-state names.
`emission_probs` Optional starting emission matrix.
`max_iter` Maximum EM iterations.
`inner_maxit` Maximum BFGS iterations in each multinomial M-step.
`tolerance` Relative log-likelihood tolerance.
`pseudocount` Emission smoothing count.
`ridge` Non-negative coefficient penalty.
`seed` Reproducibility seed.
`keep_posteriors` Retain final posteriors.

Value

An object of class `gp3_covariate_sequence_hmm`.

Examples

```

sequences <- data.frame(
  sequence_id = rep(paste0("s", 1:8), each = 5L),
  sequence_order = rep(1:5, times = 8L),
  state = rep(c("A", "B", "C", "B", "A"), times = 8L),
  condition = rep(rep(c(0, 1), each = 4L), each = 5L),
  time_scaled = rep(seq(-1, 1, length.out = 5L), times = 8L)
)
fit_covariate_sequence_hmm(
  sequences, 2L,
  initial_covariate_cols = "condition",
  transition_covariate_cols = c("condition", "time_scaled"),
  max_iter = 3L, inner_maxit = 10L, seed = 1L
)

```

`fit_higher_order_transition_model`

Fit a higher-order transition model

Description

Fit a higher-order transition model

Usage

```
fit_higher_order_transition_model(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  order = 2L,
  smoothing = 0.5,
  backoff = TRUE,
  context_separator = " > "
)
```

Arguments

data	Long-format sequence data.
sequence_id_col, order_col, state_col	Sequence columns.
order	Context order.
smoothing	Additive smoothing over observed next states.
backoff	Retain lower-order context tables for prediction backoff.
context_separator	Context separator.

Value

An object of class `gp3_higher_order_transition_model`.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
fit_higher_order_transition_model(sequences, order = 2L)
```

Description

Fits a finite-state, time-homogeneous HMM to two or more categorical channels under conditional independence of channels given the latent state. Latent states are statistical model states only.

Usage

```
fit_multichannel_sequence_hmm(
  data,
  n_states,
  channel_cols,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  symbol_levels = NULL,
  state_names = NULL,
  initial_probs = NULL,
  transition_probs = NULL,
  emission_probs = NULL,
  max_iter = 200L,
  tolerance = 1e-06,
  pseudocount = 1e-06,
  seed = 1L,
  keep_posteriors = FALSE
)
```

Arguments

<code>data</code>	Long-format multichannel sequence data.
<code>n_states</code>	Number of latent states.
<code>channel_cols</code>	Names of categorical observation channels.
<code>sequence_id_col, order_col</code>	Core sequence columns.
<code>symbol_levels</code>	Optional named list of symbol orders by channel.
<code>state_names</code>	Optional latent-state names.
<code>initial_probs, transition_probs, emission_probs</code>	Optional starting values.
<code>max_iter</code>	Maximum EM iterations.
<code>tolerance</code>	Relative log-likelihood tolerance.
<code>pseudocount</code>	Non-negative smoothing count.
<code>seed</code>	Reproducibility seed.
<code>keep_posteriors</code>	Retain final forward-backward results.

Value

An object of class `gp3_multichannel_sequence_hmm`.

Examples

```

multichannel <- data.frame(
  sequence_id = rep(paste0("s", 1:4), each = 5L),
  sequence_order = rep(1:5, times = 4L),
  action = c("A", "B", "C", "C", "D", "A", "B", "B", "C", "D",
             "D", "C", "B", "A", "A", "D", "C", "C", "B", "A"),
  context = rep(c("x", "x", "y", "y", "z"), times = 4L)
)
fit_multichannel_sequence_hmm(multichannel, 2L,
                             channel_cols = c("action", "context"),
                             max_iter = 5L, seed = 1L)

```

fit_sequence_hmm	<i>Fit a categorical hidden Markov model</i>
------------------	--

Description

Fits a finite-state, time-homogeneous categorical HMM by Baum-Welch EM. Latent states are statistical model states only; they are not psychological, diagnostic, or causal constructs.

Usage

```

fit_sequence_hmm(
  data,
  n_states,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  symbol_levels = NULL,
  state_names = NULL,
  initial_probs = NULL,
  transition_probs = NULL,
  emission_probs = NULL,
  max_iter = 200L,
  tolerance = 1e-06,
  pseudocount = 1e-06,
  seed = 1L,
  keep_posteriors = FALSE
)

```

Arguments

data	Long-format sequence data.
n_states	Number of latent states.
sequence_id_col, order_col, state_col	Sequence columns.
symbol_levels	Optional observed-symbol ordering.

state_names	Optional latent-state names.
initial_probs, transition_probs, emission_probs	Optional starting values.
max_iter	Maximum EM iterations.
tolerance	Relative log-likelihood tolerance.
pseudocount	Non-negative smoothing count.
seed	Reproducibility seed.
keep_posteriors	Retain final forward-backward results.

Value

An object of class `gp3_sequence_hmm` containing fitted parameters, log likelihood, convergence diagnostics, symbol coding, and optional posteriors.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
fit_sequence_hmm(sequences, n_states = 2L, max_iter = 5L, seed = 1L)
```

```
fit_sequence_hmm_mixture
```

Fit a mixture of categorical hidden Markov models

Description

Fits sequence-level mixture components, each containing a categorical HMM. Component membership is a statistical clustering device and should not be interpreted as a substantive latent type without external validation.

Usage

```
fit_sequence_hmm_mixture(
  data,
  n_components,
  n_states,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
```

```

    state_col = "state",
    symbol_levels = NULL,
    max_iter = 200L,
    inner_initial_iter = 20L,
    tolerance = 1e-06,
    pseudocount = 1e-06,
    seed = 1L
  )

```

Arguments

<code>data</code>	Long-format sequence data.
<code>n_components</code>	Number of mixture components.
<code>n_states</code>	Number of hidden states per component; scalar or vector.
<code>sequence_id_col</code> , <code>order_col</code> , <code>state_col</code>	Sequence columns.
<code>symbol_levels</code>	Optional symbol ordering.
<code>max_iter</code>	Maximum mixture-EM iterations.
<code>inner_initial_iter</code>	Initial single-HMM iterations per component.
<code>tolerance</code>	Relative log-likelihood tolerance.
<code>pseudocount</code>	Smoothing count.
<code>seed</code>	Reproducibility seed.

Value

An object of class `gp3_sequence_hmm_mixture`.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
fit_sequence_hmm_mixture(sequences, n_components = 2L, n_states = 2L,
  max_iter = 5L, inner_initial_iter = 2L, seed = 1L)

```

fit_time_varying_sequence_model

Fit a time-varying sequence condition model

Description

Fits a binary generalized additive model for either occupancy of a target state or occurrence of a target transition over aligned sequence time. Group specific smooths model time-varying condition differences, and a participant random-effect smooth accounts for repeated observations. The optional mgcv package is required.

Usage

```
fit_time_varying_sequence_model(
  data,
  group_col,
  participant_id_col,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  time_col = NULL,
  outcome = c("state", "transition"),
  target_state = NULL,
  from_state = NULL,
  to_state = NULL,
  k = 5L,
  method = "REML",
  include_random_effect = TRUE
)
```

Arguments

data	Long-format sequence data or a prepared result.
group_col	Group or condition column.
participant_id_col	Participant or repeated-unit column.
sequence_id_col, order_col, state_col	Core sequence columns.
time_col	Optional numeric time column; defaults to order_col.
outcome	"state" or "transition".
target_state	Target state for occupancy models.
from_state, to_state	Target transition for transition models.
k	Basis dimension for time smooths.

method Smoothing-parameter estimation method passed to `mgcv::gam()`.
 include_random_effect Include a participant random-effect smooth.

Value

An object of class `gp3_sequence_time_model`.

Examples

```
set.seed(2026)

n_participants <- 24L
n_positions <- 12L

participant_index <- rep(
  seq_len(n_participants),
  each = n_positions
)

sequence_order <- rep(
  seq_len(n_positions),
  times = n_participants
)

group_by_participant <- rep(
  c("control", "treatment"),
  each = n_participants %/% 2L
)

group <- rep(
  group_by_participant,
  each = n_positions
)

probability <- stats::plogis(
  -0.8 +
    0.08 * sequence_order +
    0.4 * (group == "treatment")
)

observed_b <- stats::rbinom(
  length(probability),
  size = 1L,
  prob = probability
)

sequences <- data.frame(
  participant_id = paste0("p", participant_index),
  sequence_id = paste0("s", participant_index),
  sequence_order = sequence_order,
  state = ifelse(observed_b == 1L, "B", "A"),
```

```

    group = group
  )

  if (requireNamespace("mgcv", quietly = TRUE)) {
    fit <- fit_time_varying_sequence_model(
      sequences,
      group_col = "group",
      participant_id_col = "participant_id",
      target_state = "B",
      k = 4L
    )
  }
}

```

format_consensus_sequence

Format consensus sequences as paths

Description

Format consensus sequences as paths

Usage

```

format_consensus_sequence(
  consensus,
  separator = " -> ",
  include_order = FALSE,
  include_agreement = FALSE,
  digits = 3L
)

```

Arguments

consensus	A consensus result.
separator	State separator.
include_order	Include position labels.
include_agreement	Include rounded agreement values.
digits	Number of decimal places for agreement values.

Value

One row per consensus group with a formatted path and position count.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
consensus <- create_consensus_sequence(sequences)
format_consensus_sequence(consensus)

```

format_sequence_motif_positions

Format Sequence Motif Position Summaries

Description

Produces a deterministic report-ready table from positional motif summaries.

Usage

```

format_sequence_motif_positions(
  x,
  digits = 3L,
  position_units = c("proportion", "percent"),
  include_rank = TRUE
)

```

Arguments

<code>x</code>	An object returned by <code>summarise_sequence_motif_positions()</code> .
<code>digits</code>	Whole number from 0 to 15 controlling numeric rounding.
<code>position_units</code>	Display units for relative positions: "proportion" or "percent". Absolute positions remain one-based sequence indices.
<code>include_rank</code>	Logical value indicating whether to include an earlier-to-later rank based on mean position. Ranking is performed within each supplied by group.

Details

Formatting copies and transforms the summary table only. The input object is not modified. Rows are ordered deterministically by grouping values, mean and median position, occurrence and sequence counts, motif length, and motif key. Equal mean positions receive the same minimum rank.

Value

A named list containing the formatted table, validation metadata, and formatting settings.

Examples

```

sequences <- data.frame(
  id = c(rep("s1", 5L), rep("s2", 4L)),
  position = c(1:5, 1:4),
  state = c("A", "B", "A", "B", "C", "A", "B", "C", "B")
)

extracted <- extract_sequence_ngrams(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)

positions <- summarise_sequence_motif_positions(
  extracted,
  position = "centre",
  scale = "relative"
)

formatted <- format_sequence_motif_positions(
  positions,
  position_units = "percent",
  digits = 1
)

formatted$table

```

format_sequence_motifs

Format Sequence Motif Summaries

Description

Produces a stable, report-ready table from motif extraction, summary, or filtered-motif output.

Usage

```

format_sequence_motifs(
  x,
  digits = 3L,
  prevalence = c("proportion", "percent"),
  include_rank = TRUE,
  rank_by = c("sequence_prevalence", "n_occurrences", "n_sequences"),
  ties = c("min", "first"),
  include_ids = TRUE
)

```

Arguments

<code>x</code>	A motif extraction, motif summary, or filtered-motif object.
<code>digits</code>	Whole number from 0 to 15 controlling numeric rounding.
<code>prevalence</code>	Character value specifying whether prevalence and occurrence share are shown as "proportion"s or "percent"ages.
<code>include_rank</code>	Logical value indicating whether to add a rank column.
<code>rank_by</code>	Metric used to order and rank motifs: one of "sequence_prevalence", "n_occurrences", or "n_sequences".
<code>ties</code>	Character value specifying "min" shared ranks or deterministic "first" ranks.
<code>include_ids</code>	Logical value indicating whether motif_id and motif_key should be included in the formatted table.

Details

Formatting changes display precision and units only. It does not change the underlying motif counts or introduce substantive interpretation.

Value

A named list containing table, validation metadata, and formatting settings. The table contains only structural motif measurements.

Examples

```
sequences <- data.frame(
  id = c(rep("s1", 5L), rep("s2", 4L)),
  position = c(1:5, 1:4),
  state = c("A", "B", "A", "B", "A", "A", "B", "A", "C")
)

extracted <- extract_sequence_ngrams(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)

formatted <- format_sequence_motifs(
  extracted,
  prevalence = "percent",
  digits = 1
)

formatted$table
```

format_sequence_paths *Format Ordered Sequence Paths*

Description

Creates a compact one-row-per-sequence representation of ordered state paths.

Usage

```
format_sequence_paths(
  data,
  sequence_id_col,
  order_col,
  state_col,
  metadata_cols = NULL,
  expected_states = NULL,
  separator = " > ",
  collapse_repeats = FALSE
)
```

Arguments

data	A data frame containing ordered state observations.
sequence_id_col	Name of the sequence identifier column.
order_col	Name of the numeric sequence-order column.
state_col	Name of the categorical state column.
metadata_cols	Optional character vector naming columns that should remain constant within each sequence.
expected_states	Optional vector of known or permitted state values.
separator	Character value inserted between adjacent state labels.
collapse_repeats	Logical value indicating whether consecutive repeated states should be collapsed for path display.

Details

Repeat collapsing affects only the formatted representation. It does not modify the supplied data or alter non-consecutive repeated states. Input rows are ordered deterministically for formatting, but review-level diagnostics such as `unordered_rows` remain reflected in the returned status and audit.

Value

A named list containing:

- `paths`: one row per sequence with observation counts, formatted-state counts, unique-state counts, start and end states, and the path string;
- `audit`, `status`, and mapping from input validation;
- `settings`: the path separator and repeat-collapsing choice.

Examples

```
sequences <- data.frame(
  id = c("s1", "s1", "s1", "s2", "s2"),
  position = c(1, 2, 3, 1, 2),
  state = c("A", "B", "C", "A", "C")
)

paths <- format_sequence_paths(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)

paths$paths
```

plot_consensus_sequence

Plot a consensus sequence

Description

Plot a consensus sequence

Usage

```
plot_consensus_sequence(
  consensus,
  type = c("agreement", "states"),
  group = NULL,
  main = NULL,
  xlab = "Sequence position",
  ylab = NULL,
  ...
)
```

Arguments

consensus	A consensus result.
type	"agreement" or "states".
group	Optional group value, encoded group key, or named list of values when grouped consensus was created. It is required when more than one consensus group is present.
main, xlab, ylab	Plot labels.
...	Additional arguments passed to base graphics.

Value

The plotted data, invisibly.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
consensus <- create_consensus_sequence(sequences)
plot_consensus_sequence(consensus)
```

plot_multichannel_sequence_hmm

Plot multichannel HMM emission profiles

Description

Plot multichannel HMM emission profiles

Usage

```
plot_multichannel_sequence_hmm(model, channel = model$channel_names[1L], ...)
```

Arguments

model	A fitted multichannel HMM.
channel	Channel name to plot.
...	Additional arguments passed to <code>graphics::barplot()</code> .

Value

The selected emission matrix, invisibly.

Examples

```
# See `fit_multichannel_sequence_hmm()`.
```

```
plot_sequence_cluster_silhouette
```

Plot sequence-cluster silhouette values

Description

Plot sequence-cluster silhouette values

Usage

```
plot_sequence_cluster_silhouette(clustering, distance = NULL, ...)
```

Arguments

clustering	A clustering result or named assignment vector.
distance	Optional distance when clustering is an assignment vector.
...	Additional arguments passed to graphics::barplot() .

Value

The ordered per-sequence silhouette table, invisibly.

Examples

```
data <- data.frame(sequence_id = rep(paste0("s", 1:4), each = 4L),
  sequence_order = rep(1:4, 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
    "D", "C", "B", "A", "D", "C", "A", "A"))
distance <- compute_sequence_distance(data)
plot_sequence_cluster_silhouette(cluster_sequences(distance, 2L))
```

plot_sequence_distance_heatmap
Plot a sequence-distance heatmap

Description

Plot a sequence-distance heatmap

Usage

```
plot_sequence_distance_heatmap(  
  distance,  
  order_by = NULL,  
  palette = "Viridis",  
  show_labels = TRUE,  
  ...  
)
```

Arguments

distance	A sequence distance object or square matrix.
order_by	Optional clustering or named assignment vector used to order the matrix.
palette	Base HCL palette.
show_labels	Draw sequence identifiers.
...	Additional arguments passed to <code>graphics::image()</code> .

Value

The ordered distance matrix, invisibly.

Examples

```
data <- data.frame(sequence_id = rep(paste0("s", 1:4), each = 4L),  
                  sequence_order = rep(1:4, 4L),  
                  state = c("A", "B", "C", "D", "A", "B", "C", "C",  
                           "D", "C", "B", "A", "D", "C", "A", "A"))  
plot_sequence_distance_heatmap(compute_sequence_distance(data))
```

plot_sequence_entropy *Plot position-wise state entropy*

Description

Plot position-wise state entropy

Usage

```
plot_sequence_entropy(  
  data,  
  sequence_id_col = "sequence_id",  
  order_col = "sequence_order",  
  state_col = "state",  
  base = 2,  
  normalise = TRUE,  
  ...  
)
```

Arguments

data	Long-format sequence data or a prepared result.
sequence_id_col, order_col, state_col	Core sequence columns.
base	Logarithm base.
normalise	Divide entropy by the maximum entropy for the observed alphabet.
...	Additional arguments passed to <code>graphics::plot()</code> .

Value

A data frame of position-wise entropy values, invisibly.

Examples

```
data <- data.frame(sequence_id = rep(c("a", "b"), each = 4L),  
                  sequence_order = rep(1:4, 2L),  
                  state = c("A", "B", "C", "D", "A", "C", "C", "D"))  
plot_sequence_entropy(data)
```

plot_sequence_group_comparison

Plot a descriptive sequence-group comparison

Description

Plot a descriptive sequence-group comparison

Usage

```
plot_sequence_group_comparison(
  comparison,
  component = c("state", "transition", "length"),
  measure = NULL,
  top_n = 12L,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

comparison	A result from compare_sequence_groups() .
component	"state", "transition", or "length".
measure	Measure to display. Defaults depend on component.
top_n	Maximum states or transitions to display.
main, xlab, ylab	Plot labels.
...	Additional base-graphics arguments.

Value

The plotted summary data, invisibly.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
comparison <- compare_sequence_groups(sequences, group_col = "group")
plot_sequence_group_comparison(comparison, component = "state")
```

```
plot_sequence_group_inference
      Plot sequence group inference
```

Description

Plot sequence group inference

Usage

```
plot_sequence_group_inference(
  inference,
  type = c("permutation", "group_means"),
  ...
)
```

Arguments

<code>inference</code>	A sequence group-inference object.
<code>type</code>	"permutation" or "group_means".
<code>...</code>	Additional base-graphics arguments.

Value

The inference object, invisibly.

Examples

```
# See `test_sequence_group_difference()`.
```

```
plot_sequence_index      Plot a sequence index heatmap
```

Description

Plot a sequence index heatmap

Usage

```
plot_sequence_index(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  sort_by = c("input", "length", "path"),
  state_levels = NULL,
  palette = "Dark 3",
  show_sequence_labels = TRUE,
  ...
)
```

Arguments

data	Long-format sequence data or a prepared result.
sequence_id_col, order_col, state_col	Core sequence columns.
sort_by	"input", "length", or "path".
state_levels	Optional state ordering.
palette	Base HCL palette.
show_sequence_labels	Draw sequence identifiers.
...	Additional arguments passed to <code>graphics::image()</code> .

Value

The plotted state-code matrix, invisibly.

Examples

```
data <- data.frame(sequence_id = rep(c("a", "b"), each = 4L),
  sequence_order = rep(1:4, 2L),
  state = c("A", "B", "C", "D", "A", "C", "C", "D"))
plot_sequence_index(data)
```

plot_sequence_motif_positions

Plot Sequence Motif Positions

Description

Shows occurrence locations for selected contiguous motifs.

Usage

```
plot_sequence_motif_positions(
  x,
  motifs = NULL,
  position = c("start", "centre", "end"),
  scale = c("absolute", "relative"),
  top_n = 10L,
  display = c("strip", "distribution")
)
```

Arguments

<code>x</code>	An object returned by <code>extract_sequence_ngrams()</code> or <code>summarise_sequence_motif_positions()</code> .
<code>motifs</code>	Optional character vector of motif identifiers, motif keys, or displayed motif labels. When omitted, motifs are selected deterministically by occurrence count, sequence count, motif length, and motif key.
<code>position</code>	Occurrence position represented by motif "start", "centre", or "end".
<code>scale</code>	Position scale: one-based "absolute" sequence positions or "relative" positions from 0 to 1.
<code>top_n</code>	Positive whole number giving the number of motifs selected when motifs is NULL.
<code>display</code>	Base-R display type: occurrence "strip" or "distribution" boxplots.

Details

The strip display uses deterministic vertical stacking rather than random jitter. The distribution display uses one horizontal boxplot per motif. Empty inputs produce an informative blank plot and return an empty table. The function provides structural location summaries only.

Value

Invisibly returns the exact occurrence-level data used by the plot, including deterministic motif ranks and plotting coordinates where relevant.

Examples

```
sequences <- data.frame(
  id = c(rep("s1", 5L), rep("s2", 4L)),
  position = c(1:5, 1:4),
  state = c("A", "B", "A", "B", "C", "A", "B", "C", "B")
)

extracted <- extract_sequence_ngrams(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)
```

```

plot_sequence_motif_positions(
  extracted,
  position = "centre",
  scale = "relative",
  top_n = 5,
  display = "strip"
)

```

plot_sequence_motifs *Plot Sequence Motif Summaries*

Description

Draws a dependency-free base-R bar chart of structural motif measurements.

Usage

```

plot_sequence_motifs(
  x,
  metric = c("sequence_prevalence", "n_occurrences", "n_sequences", "occurrence_share"),
  top_n = 20L,
  motif_lengths = NULL,
  ties = c("include", "first"),
  horizontal = TRUE
)

```

Arguments

<code>x</code>	A motif extraction, summary, or filtered-motif object.
<code>metric</code>	Structural metric to plot: "sequence_prevalence", "n_occurrences", "n_sequences", or "occurrence_share".
<code>top_n</code>	Positive whole number giving the requested number of motifs.
<code>motif_lengths</code>	Optional vector of positive whole-number motif lengths to retain.
<code>ties</code>	Top-n boundary policy. "include" retains all motifs tied on the selected metric; "first" retains exactly top_n after deterministic secondary sorting.
<code>horizontal</code>	Logical value indicating whether bars should be horizontal.

Details

The plot is descriptive. It does not perform inferential testing or assign substantive meaning to motif frequency or prevalence. Empty inputs produce an informative blank plot and return an empty table.

Value

Invisibly returns the exact motif table used for plotting, including plot_rank, plot_value, plot_label, and bar_midpoint.

Examples

```

sequences <- data.frame(
  id = c(rep("s1", 5L), rep("s2", 4L)),
  position = c(1:5, 1:4),
  state = c("A", "B", "A", "B", "C", "A", "B", "C", "B")
)

extracted <- extract_sequence_ngrams(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)

plot_sequence_motifs(
  extracted,
  metric = "sequence_prevalence",
  top_n = 10
)

```

plot_sequence_panel_changes

Plot longitudinal sequence changes

Description

Plot longitudinal sequence changes

Usage

```

plot_sequence_panel_changes(
  changes,
  metric = c("distance", "length_change", "transition_change"),
  type = c("individual", "summary"),
  ...
)

```

Arguments

changes	A result from compare_sequence_panel_changes() .
metric	One of "distance", "length_change", or "transition_change".
type	Plot individual panel trajectories or occasion-transition means.
...	Additional graphical arguments passed to base plotting functions.

Value

The plotted data, invisibly.

Examples

```

panel_data <- data.frame(
  participant_id = rep(c("p1", "p2"), each = 6L),
  occasion = rep(rep(c(1, 2), each = 3L), times = 2L),
  sequence_id = rep(c("a", "b", "c", "d"), each = 3L),
  sequence_order = rep(1:3, times = 4L),
  state = c("A", "B", "C", "A", "C", "C", "C", "B", "A", "C", "B", "B")
)
changes <- compare_sequence_panel_changes(
  prepare_sequence_panel(panel_data, "participant_id", "occasion")
)
plot_sequence_panel_changes(changes)

```

plot_sequence_state_distribution

Plot state distributions over aligned positions

Description

Plot state distributions over aligned positions

Usage

```

plot_sequence_state_distribution(
  data,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  proportion = TRUE,
  state_levels = NULL,
  palette = "Dark 3",
  ...
)

```

Arguments

data	Long-format sequence data or a prepared result.
sequence_id_col, order_col, state_col	Core sequence columns.
proportion	Plot position-wise proportions rather than counts.
state_levels	Optional state ordering.
palette	Base HCL palette.
...	Additional arguments passed to <code>graphics::matplot()</code> .

Value

A position-by-state matrix, invisibly.

Examples

```
data <- data.frame(sequence_id = rep(c("a", "b"), each = 4L),
                  sequence_order = rep(1:4, 2L),
                  state = c("A", "B", "C", "D", "A", "C", "C", "D"))
plot_sequence_state_distribution(data)
```

plot_sequence_subsequences

Plot non-contiguous subsequence summaries

Description

Plot non-contiguous subsequence summaries

Usage

```
plot_sequence_subsequences(
  x,
  metric = "sequence_prevalence",
  top_n = 10L,
  decreasing = TRUE,
  ...
)
```

Arguments

x	A summary or group-comparison table.
metric	Numeric column to plot.
top_n	Maximum number of subsequences.
decreasing	Sort metric in decreasing order.
...	Additional arguments passed to <code>graphics::barplot()</code> .

Value

The plotted rows, invisibly.

Examples

```
data <- data.frame(sequence_id = rep(c("a", "b"), each = 4L),
                  sequence_order = rep(1:4, 2L),
                  state = c("A", "B", "C", "D", "A", "C", "B", "D"))
summary <- summarise_sequence_subsequences(extract_sequence_subsequences(data))
plot_sequence_subsequences(summary, top_n = 5L)
```

```
plot_time_varying_sequence_model
```

Plot predicted time-varying sequence probabilities

Description

Plot predicted time-varying sequence probabilities

Usage

```
plot_time_varying_sequence_model(
  model,
  time = NULL,
  level = 0.95,
  show_interval = TRUE,
  ...
)
```

Arguments

<code>model</code>	A fitted time-varying sequence model.
<code>time</code>	Optional prediction grid.
<code>level</code>	Pointwise confidence level.
<code>show_interval</code>	Draw pointwise confidence ribbons.
<code>...</code>	Additional arguments passed to <code>graphics::plot()</code> .

Value

Prediction data, invisibly.

Examples

```
# See `fit_time_varying_sequence_model()`.
```

```
plot_transition_network
```

Plot a first-order transition network

Description

Plot a first-order transition network

Usage

```
plot_transition_network(
  network,
  weight_col = "weight",
  minimum_weight = 0,
  vertex_cex = 1,
  edge_scale = 5,
  ...
)
```

Arguments

network	A first-order network from <code>create_transition_network()</code> .
weight_col	Edge-weight column.
minimum_weight	Minimum plotted edge weight.
vertex_cex	Vertex label size.
edge_scale	Edge-width multiplier.
...	Additional arguments passed to <code>graphics::plot.window()</code> .

Value

The plotted network rows, invisibly.

Examples

```
data <- data.frame(sequence_id = rep(c("a", "b"), each = 4L),
  sequence_order = rep(1:4, 2L),
  state = c("A", "B", "C", "D", "A", "C", "C", "D"))
plot_transition_network(create_transition_network(data, normalise = "from"))
```

predict_covariate_transition_probabilities

Predict covariate-dependent transition probabilities

Description

Predict covariate-dependent transition probabilities

Usage

```
predict_covariate_transition_probabilities(model, newdata)
```

Arguments

model	A fitted covariate HMM.
newdata	Data frame containing transition covariates.

Value

A long data frame with one row per input row, origin state, and destination state.

Examples

```
# See `fit_covariate_sequence_hmm()`.
```

predict_next_state	<i>Predict the next state from a transition model</i>
--------------------	---

Description

Predict the next state from a transition model

Usage

```
predict_next_state(model, history, top_n = NULL)
```

Arguments

- model A higher-order transition model.
- history Character vector of observed recent states.
- top_n Optional number of states to retain. Returned probabilities remain on the full-model scale and are not renormalised after truncation.

Value

A probability table ordered from highest to lowest probability, with the context order actually used.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
model <- fit_higher_order_transition_model(sequences, order = 2L)
predict_next_state(model, c("A", "B"))
```

`predict_time_varying_sequence_model`*Predict a time-varying sequence model*

Description

Predict a time-varying sequence model

Usage

```
predict_time_varying_sequence_model(  
  model,  
  time = NULL,  
  groups = NULL,  
  level = 0.95  
)
```

Arguments

<code>model</code>	A fitted time-varying sequence model.
<code>time</code>	Optional numeric prediction grid.
<code>groups</code>	Optional subset of fitted groups.
<code>level</code>	Confidence level for pointwise intervals.

Value

A data frame with fitted probabilities and pointwise intervals.

Examples

```
# See `fit_time_varying_sequence_model()`.
```

`prepare_gp3tools_sequences`*Prepare common gp3tools-style sequence outputs*

Description

This optional compatibility helper recognises ordinary data frames or lists containing a data-frame component and maps common sequence-column names to the neutral gp3sequences contract. It does not require a gp3tools class.

Usage

```
prepare_gp3tools_sequences(
  data,
  sequence_id_col = NULL,
  order_col = NULL,
  state_col = NULL,
  duration_col = NULL,
  metadata_cols = NULL,
  ...
)
```

Arguments

data A data frame or list with a data-frame data component.

sequence_id_col, order_col, state_col Optional explicit mappings.

duration_col Optional duration mapping.

metadata_cols Optional constant-within-sequence metadata columns.

... Additional arguments passed to [prepare_sequence_data\(\)](#).

Value

A standard gp3sequences preparation result.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
names(sequences)[names(sequences) == "sequence_order"] <- "position"
names(sequences)[names(sequences) == "state"] <- "aoi_label"
prepare_gp3tools_sequences(sequences)
```

prepare_sequence_data *Prepare Long-Format Sequence Data*

Description

Applies explicit preprocessing policies and returns a deterministic, canonical long-format representation.

Usage

```
prepare_sequence_data(
  data,
  sequence_id_col,
  order_col,
  state_col,
  duration_col = NULL,
  metadata_cols = NULL,
  expected_states = NULL,
  missing_state_policy = c("error", "drop"),
  duplicate_position_policy = c("error", "first", "last"),
  repeated_state_policy = c("preserve", "collapse"),
  zero_duration_policy = c("preserve", "drop", "error"),
  unknown_state_policy = c("preserve", "drop", "error"),
  unused_state_levels = c("preserve", "drop")
)
```

Arguments

<code>data</code>	A data frame containing ordered state observations.
<code>sequence_id_col</code>	Name of the sequence identifier column.
<code>order_col</code>	Name of the numeric sequence-order column.
<code>state_col</code>	Name of the categorical state column.
<code>duration_col</code>	Optional name of a numeric duration column.
<code>metadata_cols</code>	Optional character vector naming columns that should remain constant within each sequence.
<code>expected_states</code>	Optional vector of known or permitted state values.
<code>missing_state_policy</code>	Policy for missing states: "error" or "drop".
<code>duplicate_position_policy</code>	Policy for duplicated sequence positions: "error", "first", or "last".
<code>repeated_state_policy</code>	Policy for consecutive repeated states: "preserve" or "collapse".
<code>zero_duration_policy</code>	Policy for zero durations: "preserve", "drop", or "error".
<code>unknown_state_policy</code>	Policy for states absent from <code>expected_states</code> : "preserve", "drop", or "error".
<code>unused_state_levels</code>	Policy for unused factor levels: "preserve" or "drop".

Details

Rows are sorted deterministically by sequence identifier, sequence order, and original row number. When consecutive repeats are collapsed, the first row supplies non-duration values and available durations are summed.

Unresolved errors produce status = "fail" and data = NULL; diagnostics and decision records remain available.

Value

A named list containing:

- data: canonical prepared data, or NULL when unresolved errors remain;
- audit: input- and output-stage diagnostics;
- decisions: a machine-readable preprocessing decision log;
- mapping: source-to-contract column mappings;
- status: pass, review, or fail;
- row counts and final state levels.

The canonical columns are sequence_id, sequence_order, state, original_row, and optional duration. Unmapped columns are preserved.

Examples

```
sequences <- data.frame(
  id = c("s2", "s1", "s1", "s2"),
  position = c(2, 2, 1, 1),
  state = c("product", "search", "home", "home")
)

prepared <- prepare_sequence_data(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)

prepared$status
prepared$data
```

```
prepare_sequence_panel
```

Prepare longitudinal or panel sequence data

Description

Validates a collection of sequences observed repeatedly for the same panel units and creates an auditable panel index. Each sequence must map to exactly one panel unit and one occasion.

Usage

```
prepare_sequence_panel(
  data,
  panel_id_col,
  occasion_col,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  metadata_cols = NULL,
  require_unique_occasions = TRUE
)
```

Arguments

`data` Long-format sequence data or a prepared `gp3sequences` result.

`panel_id_col` Column identifying the repeatedly observed unit.

`occasion_col` Column identifying the wave, visit, or occasion.

`sequence_id_col, order_col, state_col` Core sequence columns.

`metadata_cols` Optional columns that must remain constant within each sequence.

`require_unique_occasions` Require at most one sequence per panel unit and occasion.

Value

An object of class `gp3_sequence_panel` containing canonical data, a panel index, state levels, and mapping information.

Examples

```
panel <- data.frame(
  participant_id = rep(c("p1", "p2"), each = 8L),
  occasion = rep(rep(c(1, 2), each = 4L), times = 2L),
  sequence_id = rep(c("p1_w1", "p1_w2", "p2_w1", "p2_w2"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "C", "C", "D",
            "D", "C", "B", "A", "D", "B", "B", "A")
)
prepare_sequence_panel(panel, "participant_id", "occasion")
```

sequence_capabilities *Report gp3sequences capabilities and optional integrations*

Description

Returns a deterministic, machine-readable inventory of native analytical capabilities, optional adapters, reference implementations, and specialist handoffs relevant to the current `gp3sequences` development series.

Usage

```
sequence_capabilities(include_optional = TRUE, check_versions = TRUE)
```

Arguments

`include_optional`

Logical; include optional/reference capabilities.

`check_versions` Logical; report installed optional-package versions.

Value

A data frame with capability family, capability, implementation role, backend, availability, and version information. The function never installs, attaches, or loads optional packages.

Examples

```
capabilities <- sequence_capabilities()
capabilities[c("family", "capability", "role", "available")]
```

```
summarise_consensus_agreement
```

Summarise consensus agreement

Description

Summarise consensus agreement

Usage

```
summarise_consensus_agreement(
  consensus,
  by = c("overall", "group", "position"),
  threshold = 0.5
)
```

Arguments

`consensus` A result from [create_consensus_sequence\(\)](#).

`by` Summary level: "overall", "group", or "position".

`threshold` Agreement threshold used to count low-agreement positions.

Value

A data frame with position counts, mean, median, minimum and maximum agreement, weighted agreement, tie counts, and low-agreement counts.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
consensus <- create_consensus_sequence(sequences)
summarise_consensus_agreement(consensus)

```

summarise_covariate_sequence_hmm

Summarise a covariate-dependent HMM

Description

Summarise a covariate-dependent HMM

Usage

```
summarise_covariate_sequence_hmm(model)
```

Arguments

model A fitted covariate HMM.

Value

A list of fit, coefficient, and emission summaries.

Examples

```
# See `fit_covariate_sequence_hmm()`.
```

`summarise_multichannel_sequence_hmm`*Summarise a multichannel sequence HMM*

Description

Summarise a multichannel sequence HMM

Usage

```
summarise_multichannel_sequence_hmm(model)
```

Arguments

`model` A fitted multichannel HMM.

Value

A list of convergence, fit, transition, initial, and channel-specific emission summaries.

Examples

```
# See `fit_multichannel_sequence_hmm()`.
```

`summarise_sequence_cluster_stability`*Summarise sequence-cluster stability*

Description

Summarise sequence-cluster stability

Usage

```
summarise_sequence_cluster_stability(bootstrap, threshold = 0.8)
```

Arguments

`bootstrap` A result from [bootstrap_sequence_clusters\(\)](#).
`threshold` Pairwise stability threshold.

Value

Overall, cluster-level, and low-stability pair summaries.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
distance <- compute_sequence_distance(sequences)
boot <- bootstrap_sequence_clusters(distance, k = 2L, n_boot = 5L)
summarise_sequence_cluster_stability(boot)

```

summarise_sequence_distance

Summarise a sequence-distance object

Description

Summarise a sequence-distance object

Usage

```
summarise_sequence_distance(distance)
```

Arguments

distance A distance object returned by `compute_sequence_distance()`.

Value

A list containing an overall summary and per-sequence mean, median, minimum, and maximum distances.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
distance <- compute_sequence_distance(sequences)
summarise_sequence_distance(distance)

```

```
summarise_sequence_group_inference
```

Summarise sequence group inference

Description

Summarise sequence group inference

Usage

```
summarise_sequence_group_inference(inference)
```

Arguments

`inference` A sequence group-inference object.

Value

A list containing the estimate, optional interval, design, and interpretation statement.

Examples

```
# See `test_sequence_group_difference()`.
```

```
summarise_sequence_hmm
```

Summarise a fitted sequence HMM

Description

Summarise a fitted sequence HMM

Usage

```
summarise_sequence_hmm(model)
```

Arguments

`model` A single or mixture HMM.

Value

A list of convergence, fit, initial, transition, emission, and mixture summaries.

Examples

```

sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
model <- fit_sequence_hmm(sequences, 2L, max_iter = 5L)
summarise_sequence_hmm(model)

```

summarise_sequence_motif_positions

Summarise Sequence Motif Positions

Description

Summarises where contiguous motif occurrences appear within sequences.

Usage

```

summarise_sequence_motif_positions(
  x,
  position = c("start", "centre", "end"),
  scale = c("absolute", "relative"),
  by = NULL
)

```

Arguments

<code>x</code>	An object returned by <code>extract_sequence_ngrams()</code> .
<code>position</code>	Position represented by each occurrence: motif "start", "centre", or "end".
<code>scale</code>	Position scale: one-based "absolute" sequence positions or "relative" positions from 0 to 1.
<code>by</code>	Optional character vector naming preserved metadata columns used to produce separate summaries, such as "group" or "condition".

Details

Absolute positions use the one-based state index within each validated sequence. Relative positions are calculated as $(\text{absolute_position} - 1) / (\text{n_states} - 1)$ and are constrained to the interval from 0 to 1. A sequence containing one state is assigned relative position 0.

Grouping is descriptive. The function does not test differences or attach behavioural, psychological, cognitive, or causal interpretations to motif location.

Value

A named list containing:

- **summary**: one row per motif and optional metadata group;
- **occurrences**: occurrence-level absolute, relative, and selected-scale positions;
- **sequences**, validation metadata, and extraction settings;
- **settings**: the resolved position basis, scale, and grouping columns.

The summary reports motif identifiers and labels, motif length, occurrence and sequence counts, and minimum, maximum, mean, and median positions.

Examples

```
sequences <- data.frame(
  id = c(rep("s1", 5L), rep("s2", 4L)),
  position = c(1:5, 1:4),
  state = c("A", "B", "A", "B", "C", "A", "B", "C", "B"),
  group = c(rep("g1", 5L), rep("g2", 4L))
)

extracted <- extract_sequence_ngrams(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state",
  metadata_cols = "group",
  min_length = 2,
  max_length = 3
)

positions <- summarise_sequence_motif_positions(
  extracted,
  position = "centre",
  scale = "relative",
  by = "group"
)

positions$summary
```

summarise_sequence_motifs

Summarise Contiguous Sequence Motifs

Description

Aggregates extracted contiguous motif occurrences by sequence and overall.

Usage

```
summarise_sequence_motifs(x)
```

Arguments

`x` An object returned by `extract_sequence_ngrams()`.

Details

Sequence prevalence uses every validated sequence in the extraction object as its denominator, including sequences too short to contain a requested motif. Results are sorted deterministically by sequence prevalence, occurrence count, motif length, and motif key.

The function reports structural recurrence only. It does not perform significance testing or infer psychological, cognitive, emotional, or diagnostic attributes.

Value

A named list containing:

- `by_sequence`: occurrence counts for each sequence-motif pair;
- `overall`: total occurrence counts, sequence counts, sequence prevalence, occurrence share, and mean occurrence rates;
- `sequences` and `state_dictionary` from extraction;
- `audit`, `status`, `mapping`, and extraction settings;
- scalar counts for sequences, occurrences, and distinct motifs.

Examples

```
sequences <- data.frame(
  id = c(rep("s1", 5L), rep("s2", 4L)),
  position = c(1:5, 1:4),
  state = c("A", "B", "A", "B", "A", "A", "B", "A", "C")
)

extracted <- extract_sequence_ngrams(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state",
  min_length = 2,
  max_length = 3
)

summaries <- summarise_sequence_motifs(extracted)
summaries$by_sequence
summaries$overall
```

`summarise_sequence_panel`*Summarise a sequence panel*

Description

Summarise a sequence panel

Usage

```
summarise_sequence_panel(panel)
```

Arguments

`panel` A result from `prepare_sequence_panel()`.

Value

A list with occasion-level sequence summaries and state prevalence.

Examples

```
panel_data <- data.frame(
  participant_id = rep(c("p1", "p2"), each = 6L),
  occasion = rep(rep(c(1, 2), each = 3L), times = 2L),
  sequence_id = rep(c("a", "b", "c", "d"), each = 3L),
  sequence_order = rep(1:3, times = 4L),
  state = c("A", "B", "C", "A", "C", "C", "C", "B", "A", "C", "B", "B")
)
summarise_sequence_panel(prepare_sequence_panel(panel_data, "participant_id", "occasion"))
```

`summarise_sequence_states`*Summarise Sequence States*

Description

Produces per-sequence and overall state-frequency summaries from validated ordered sequence data.

Usage

```
summarise_sequence_states(
  data,
  sequence_id_col,
  order_col,
  state_col,
  duration_col = NULL,
  metadata_cols = NULL,
  expected_states = NULL
)
```

Arguments

<code>data</code>	A data frame containing ordered state observations.
<code>sequence_id_col</code>	Name of the sequence identifier column.
<code>order_col</code>	Name of the numeric sequence-order column.
<code>state_col</code>	Name of the categorical state column.
<code>duration_col</code>	Optional name of a numeric duration column.
<code>metadata_cols</code>	Optional character vector naming columns that should remain constant within each sequence.
<code>expected_states</code>	Optional vector of known or permitted state values.

Details

Observation proportions use state rows as the denominator. Sequence proportions report the proportion of sequences in which each state occurs. Missing durations are excluded from duration calculations; an all-missing duration group returns NA.

Value

A named list containing:

- `by_sequence`: state counts and proportions within each sequence;
- `overall`: state counts and proportions across all sequences;
- `audit`, `status`, and mapping from input validation.

When `duration_col` is supplied, both tables also include duration sums, duration proportions, and mean durations.

Examples

```
sequences <- data.frame(
  id = c("s1", "s1", "s1", "s2", "s2"),
  position = c(1, 2, 3, 1, 2),
  state = c("A", "B", "A", "B", "C")
)
```

```

summaries <- summarise_sequence_states(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)

summaries$by_sequence
summaries$overall

```

summarise_sequence_subsequences

Summarise non-contiguous subsequences

Description

Summarise non-contiguous subsequences

Usage

```
summarise_sequence_subsequences(occurrences)
```

Arguments

`occurrences` A result from [extract_sequence_subsequences\(\)](#).

Value

A data frame with occurrence counts, sequence prevalence, and span diagnostics.

Examples

```

data <- data.frame(sequence_id = rep(c("a", "b"), each = 4L),
  sequence_order = rep(1:4, 2L),
  state = c("A", "B", "C", "D", "A", "C", "B", "D"))
summarise_sequence_subsequences(extract_sequence_subsequences(data))

```

summarise_sequence_transitions

Summarise Adjacent Sequence Transitions

Description

Counts transitions between adjacent ordered states for each sequence and across the complete data set.

Usage

```
summarise_sequence_transitions(
  data,
  sequence_id_col,
  order_col,
  state_col,
  metadata_cols = NULL,
  expected_states = NULL,
  include_self = TRUE
)
```

Arguments

<code>data</code>	A data frame containing ordered state observations.
<code>sequence_id_col</code>	Name of the sequence identifier column.
<code>order_col</code>	Name of the numeric sequence-order column.
<code>state_col</code>	Name of the categorical state column.
<code>metadata_cols</code>	Optional character vector naming columns that should remain constant within each sequence.
<code>expected_states</code>	Optional vector of known or permitted state values.
<code>include_self</code>	Logical value indicating whether transitions from a state to the same state should be included.

Details

A transition is defined only between adjacent rows after deterministic ordering by sequence identifier, sequence order, and original row. Sequences with one state contribute no transitions.

Value

A named list containing:

- `by_sequence`: transition counts, proportions within each sequence, and conditional proportions within each origin state;

- overall: transition counts, sequence coverage, global proportions, and conditional origin-state proportions;
- audit, status, mapping, and the resolved include_self setting.

Examples

```
sequences <- data.frame(
  id = c("s1", "s1", "s1", "s2", "s2"),
  position = c(1, 2, 3, 1, 2),
  state = c("A", "B", "C", "A", "C")
)

transitions <- summarise_sequence_transitions(
  sequences,
  sequence_id_col = "id",
  order_col = "position",
  state_col = "state"
)

transitions$by_sequence
transitions$overall
```

```
summarise_time_varying_sequence_model
```

Summarise a time-varying sequence model

Description

Summarise a time-varying sequence model

Usage

```
summarise_time_varying_sequence_model(model)
```

Arguments

model A fitted time-varying sequence model.

Value

A list containing model metadata, parametric terms, smooth terms, deviance explained, and convergence information.

Examples

```
# See `fit_time_varying_sequence_model()`.
```

```
summarise_transition_centrality
  Summarise transition-network centrality
```

Description

Summarise transition-network centrality

Usage

```
summarise_transition_centrality(
  network,
  directed = TRUE,
  pagerank_damping = 0.85,
  pagerank_tolerance = 1e-10,
  pagerank_max_iter = 1000L
)
```

Arguments

<code>network</code>	A first-order transition network.
<code>directed</code>	Treat the network as directed.
<code>pagerank_damping</code>	Damping factor for PageRank.
<code>pagerank_tolerance</code>	Convergence tolerance.
<code>pagerank_max_iter</code>	Maximum PageRank iterations.

Value

A data frame containing degree, strength, weighted closeness, unweighted betweenness, and PageRank centrality.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
network <- create_transition_network(sequences)
summarise_transition_centrality(network)
```

test_sequence_group_difference

Test a sequence group difference

Description

Aggregates sequence metrics to the declared independent unit and performs a permutation or randomization test. For observational data the p-value tests exchangeability-based association only; it is not a causal estimate.

Usage

```
test_sequence_group_difference(
  data,
  design,
  metric = c("sequence_length", "transition_count", "state_prevalence",
    "subsequence_presence"),
  target_state = NULL,
  target_subsequence = NULL,
  sequence_id_col = "sequence_id",
  order_col = "sequence_order",
  state_col = "state",
  separator = " > ",
  n_permutations = 999L,
  alternative = c("two.sided", "greater", "less"),
  seed = 1L
)
```

Arguments

data	Long-format sequence data.
design	A comparison design.
metric	"sequence_length", "transition_count", "state_prevalence", or "subsequence_presence".
target_state	Required for state prevalence.
target_subsequence	Required for subsequence presence, expressed using separator.
sequence_id_col, order_col, state_col	Core sequence columns.
separator	Subsequence label separator.
n_permutations	Number of permutations.
alternative	Alternative hypothesis.
seed	Reproducibility seed.

Value

An object of class `gp3_sequence_group_inference`.

Examples

```
data <- data.frame(
  participant_id = rep(paste0("p", 1:8), each = 4L),
  sequence_id = rep(paste0("s", 1:8), each = 4L),
  sequence_order = rep(1:4, times = 8L),
  state = c(rep(c("A", "B", "C", "D"), 4L),
            rep(c("A", "A", "C", "D"), 4L)),
  group = rep(rep(c("control", "treatment"), each = 4L), each = 4L)
)
design <- declare_sequence_comparison_design("group", "participant_id",
                                           design = "randomized")
test_sequence_group_difference(data, design, metric = "state_prevalence",
                              target_state = "A", n_permutations = 99L)
```

`validate_sequence_clusters`

Validate sequence clusters descriptively

Description

Validate sequence clusters descriptively

Usage

```
validate_sequence_clusters(clustering, distance = NULL)
```

Arguments

`clustering` A result from `cluster_sequences()` or a named assignment vector.
`distance` Optional distance object when clustering is an assignment vector.

Value

A list containing overall validation metrics, cluster sizes, and per-sequence silhouette values.

Examples

```
sequences <- data.frame(
  sequence_id = rep(c("s1", "s2", "s3", "s4"), each = 4L),
  sequence_order = rep(1:4, times = 4L),
  state = c("A", "B", "C", "D", "A", "B", "C", "C",
            "D", "C", "B", "A", "D", "C", "A", "A"),
  group = rep(c("g1", "g2"), each = 8L),
  stringsAsFactors = FALSE
)
```

```
distance <- compute_sequence_distance(sequences)
fit <- cluster_sequences(distance, k = 2L)
validate_sequence_clusters(fit)
```

validate_sequence_data

Validate Long-Format Sequence Data

Description

Produces a compact validation result based on `audit_sequence_data()` without modifying the input.

Usage

```
validate_sequence_data(
  data,
  sequence_id_col,
  order_col,
  state_col,
  duration_col = NULL,
  metadata_cols = NULL,
  expected_states = NULL
)
```

Arguments

<code>data</code>	A data frame containing ordered state observations.
<code>sequence_id_col</code>	Name of the sequence identifier column.
<code>order_col</code>	Name of the numeric sequence-order column.
<code>state_col</code>	Name of the categorical state column.
<code>duration_col</code>	Optional name of a numeric duration column.
<code>metadata_cols</code>	Optional character vector naming columns that should remain constant within each sequence.
<code>expected_states</code>	Optional vector of known or permitted state values.

Value

A named list containing `valid`, `status`, issue counts, the complete `audit` table, the column mapping, row and sequence counts, and observed `state_levels`. A result is valid when no error-severity issue is present. Review-severity issues do not automatically invalidate the input.

Examples

```
sequences <- data.frame(  
  id = rep(c("s1", "s2"), each = 2L),  
  position = rep(1:2, times = 2L),  
  state = c("home", "search", "home", "product")  
)  
  
validation <- validate_sequence_data(  
  sequences,  
  sequence_id_col = "id",  
  order_col = "position",  
  state_col = "state"  
)  
  
validation$status  
validation$valid
```

Index

as_arules_sequences, 3
as_grpstring_data, 4
as_igraph_transition_network, 5
as_seqhmm_sequences, 6
as_traminer_sequences, 7
as_traminer_sequences(), 6
audit_sequence_analysis, 8
audit_sequence_data, 9

bootstrap_sequence_clusters, 10
bootstrap_sequence_clusters(), 74
bootstrap_sequence_group_difference, 11
bootstrap_transition_network, 12

cluster_sequences, 13
cluster_sequences(), 87
compare_sequence_analysis_results, 14
compare_sequence_groups, 15
compare_sequence_groups(), 56
compare_sequence_hmms, 17
compare_sequence_panel_changes, 17
compare_sequence_panel_changes(), 61
compare_sequence_subsequences, 19
compute_sequence_distance, 20
compute_sequence_distance(), 18, 75
create_consensus_sequence, 21
create_consensus_sequence(), 72
create_sequence_cluster_ensemble, 22
create_transition_network, 23
create_transition_network(), 65

declare_sequence_comparison_design, 24
decode_covariate_sequence_states, 25
decode_multichannel_sequence_states, 26
decode_sequence_states, 27
detect_transition_communities, 28
encode_sequence_data, 29

extract_representative_sequences, 30
extract_sequence_ngrams, 31
extract_sequence_subsequences, 33
extract_sequence_subsequences(), 82

filter_sequence_motifs, 34
filter_sequence_subsequences, 36
fit_covariate_sequence_hmm, 37
fit_higher_order_transition_model, 38
fit_multichannel_sequence_hmm, 39
fit_sequence_hmm, 41
fit_sequence_hmm_mixture, 42
fit_time_varying_sequence_model, 44
format_consensus_sequence, 46
format_sequence_motif_positions, 47
format_sequence_motifs, 48
format_sequence_paths, 50

graphics::barplot(), 52, 53, 63
graphics::image(), 54, 58
graphics::matplot(), 62
graphics::plot(), 55, 64
graphics::plot.window(), 65

mgcv::gam(), 45

plot_consensus_sequence, 51
plot_multichannel_sequence_hmm, 52
plot_sequence_cluster_silhouette, 53
plot_sequence_distance_heatmap, 54
plot_sequence_entropy, 55
plot_sequence_group_comparison, 56
plot_sequence_group_inference, 57
plot_sequence_index, 57
plot_sequence_motif_positions, 58
plot_sequence_motifs, 60
plot_sequence_panel_changes, 61
plot_sequence_state_distribution, 62
plot_sequence_subsequences, 63
plot_time_varying_sequence_model, 64

plot_transition_network, [64](#)
predict_covariate_transition_probabilities,
 [65](#)
predict_next_state, [66](#)
predict_time_varying_sequence_model,
 [67](#)
prepare_gp3tools_sequences, [67](#)
prepare_sequence_data, [68](#)
prepare_sequence_data(), [68](#)
prepare_sequence_panel, [70](#)
prepare_sequence_panel(), [80](#)

sequence_capabilities, [71](#)
summarise_consensus_agreement, [72](#)
summarise_covariate_sequence_hmm, [73](#)
summarise_multichannel_sequence_hmm,
 [74](#)
summarise_sequence_cluster_stability,
 [74](#)
summarise_sequence_distance, [75](#)
summarise_sequence_group_inference, [76](#)
summarise_sequence_hmm, [76](#)
summarise_sequence_motif_positions, [77](#)
summarise_sequence_motifs, [78](#)
summarise_sequence_panel, [80](#)
summarise_sequence_states, [80](#)
summarise_sequence_subsequences, [82](#)
summarise_sequence_subsequences(), [36](#)
summarise_sequence_transitions, [83](#)
summarise_time_varying_sequence_model,
 [84](#)
summarise_transition_centrality, [85](#)

test_sequence_group_difference, [86](#)
test_sequence_group_difference(), [12](#)

validate_sequence_clusters, [87](#)
validate_sequence_data, [88](#)