Package 'lineagefreq'

May 8, 2026

Title Lineage Frequency Dynamics from Genomic Surveillance Counts

Version 0.2.0

Description Models pathogen lineage frequency dynamics from genomic surveillance count data. Provides a unified interface for multinomial logistic regression, hierarchical partial-pooling models, and the Piantham approximation for relative reproduction number estimation. Features include rolling-origin backtesting, standardized forecast scoring, lineage collapsing, emergence detection, and sequencing power analysis. Designed for real-time public health surveillance of any variant-resolved pathogen. Methods described in Abousamra, Figgins, and Bedford (2024) <doi:10.1371/journal.pcbi.1012443>.

License MIT + file LICENSE

URL https://github.com/CuiweiG/lineagefreq

BugReports https://github.com/CuiweiG/lineagefreq/issues

Depends R (>= 4.1.0)

Imports cli (>= 3.4.0), dplyr (>= 1.1.0), grDevices, ggplot2 (>= 3.4.0), MASS, numDeriv, rlang (>= 1.1.0), stats, tibble (>= 3.1.0), tidyr (>= 1.3.0)

Suggests broom, cmdstanr, covr, knitr, posterior, rmarkdown, testthat (>= 3.0.0), withr

Additional_repositories https://mc-stan.org/r-packages/

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

NeedsCompilation no

Author Cuiwei Gao [aut, cre, cph]

Maintainer Cuiwei Gao <48gaocuiwei@gmail.com>

Repository CRAN

Date/Publication 2026-04-03 08:00:08 UTC

Contents

`as.data.frame.lfq_data` *Convert lfq_data to long-format tibble*

Description

Convert lfq_data to long-format tibble

Usage

```
## S3 method for class 'lfq_data'
as.data.frame(x, ...)
```

Arguments

`x`	An lfq_data object.
`...`	Ignored.

Value

A tibble with all columns.

Examples

```
data(sarscov2_us_2022)
x <- lfq_data(sarscov2_us_2022, lineage = variant,
              date = date, count = count, total = total)
as.data.frame(x)
```

`as_lfq_data` *Coerce to lfq_data*

Description

Generic function to convert various data formats into lfq_data objects. Methods can be defined for specific input classes to enable seamless interoperability with other genomic surveillance packages.

Usage

```
as_lfq_data(x, ...)

## S3 method for class 'lfq_data'
as_lfq_data(x, ...)

## S3 method for class 'data.frame'
as_lfq_data(x, ...)
```

Arguments

x	An object to coerce.
...	Additional arguments passed to methods. For the `data.frame` method, these are passed to `lfq_data()`.

Value

An lfq_data object.

An lfq_data object.

An lfq_data object.

See Also

`lfq_data()` for the primary constructor.

Examples

```
df <- data.frame(
  date    = rep(as.Date("2024-01-01") + c(0, 7), each = 2),
  lineage = rep(c("A", "B"), 2),
  count   = c(80, 20, 60, 40)
)
x <- as_lfq_data(df, lineage = lineage, date = date, count = count)
x
```

`augment.lfq_fit` *Augment data with fitted values from an lfq_fit object*

Description

Augment data with fitted values from an lfq_fit object

Usage

```
augment.lfq_fit(x, ...)
```

Arguments

x	An `lfq_fit` object.
...	Ignored.

Value

A tibble with columns: `.date`, `.lineage`, `.fitted_freq`, `.observed`, `.pearson_resid`.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
augment.lfq_fit(fit)
```

`autoplot.lfq_fit` *Plot lineage frequency model results*

Description

Plot lineage frequency model results

Usage

```
## S3 method for class 'lfq_fit'
autoplot(
  object,
  type = c("frequency", "advantage", "trajectory", "residuals"),
  generation_time = NULL,
  ...
)
```

Arguments

`object`	An `lfq_fit` object.
`type`	Plot type: `"frequency"` (default), `"advantage"`, `"trajectory"`, or `"residuals"`.
`generation_time`	Required when `type = "advantage"`.
`...`	Ignored.

Value

A ggplot object.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
autoplot(fit)
autoplot(fit, type = "advantage", generation_time = 5)
```

autoplot.lfq_forecast *Plot a lineage frequency forecast*

Description

Plot a lineage frequency forecast

Usage

```
## S3 method for class 'lfq_forecast'
autoplot(object, ...)
```

Arguments

object	An lfq_forecast object.
...	Ignored.

Value

A ggplot object.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
fc <- forecast(fit, horizon = 14)
autoplot(fc)
```

backtest *Rolling-origin backtesting of lineage frequency models*

Description

Evaluates forecast accuracy by repeatedly fitting models on historical data and comparing predictions to held-out observations. This implements the evaluation framework described in Abousamra et al. (2024).

Usage

```
backtest(
  data,
  engines = "mlr",
  origins = "weekly",
  horizons = c(7L, 14L, 21L, 28L),
  min_train = 42L,
  ...
)
```

Arguments

`data`	An lfq_data object.
`engines`	Character vector of engine names to compare. Default `"mlr"`.
`origins`	How to select forecast origins: • `"weekly"` (default): one origin per unique date, starting after `min_train` days. • An integer: use every Nth date as an origin. • A Date vector: use these specific dates as origins.
`horizons`	Integer vector of forecast horizons in days. Default `c(7, 14, 21, 28)`.
`min_train`	Minimum training window in days. Origins earlier than `min(date) + min_train` are skipped. Default 42 (6 weeks).
`...`	Additional arguments passed to `fit_model()` (e.g., `generation_time` for the Piantham engine).

Details

Implements the rolling-origin evaluation framework described in Abousamra et al. (2024), Section 2.4. At each origin date, the model is fit on data up to that date and forecasts are compared to held-out future observations. This avoids look-ahead bias and provides an honest assessment of real-time forecast accuracy.

Value

An `lfq_backtest` object (tibble subclass) with columns:

origin_date Date used as the training cutoff.

target_date Date being predicted.

horizon Forecast horizon in days.

engine Engine name.

lineage Lineage name.

predicted Predicted frequency (median).

lower Lower prediction bound.

upper Upper prediction bound.

observed Observed frequency at target_date.

References

Abousamra E, Figgins M, Bedford T (2024). Fitness models provide accurate short-term forecasts of SARS-CoV-2 variant frequency. *PLoS Computational Biology*, 20(9):e1012443. doi:10.1371/journal.pcbi.1012443

See Also

`score_forecasts()` to compute accuracy metrics, `compare_models()` to rank engines.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
bt <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
bt
```

---

cdc_ba2_transition *CDC SARS-CoV-2 variant proportions: BA.1 to BA.2 transition (US, 2022)*

Description

Real surveillance data covering the Omicron BA.1 to BA.2 variant replacement in the United States, December 2021 through June 2022. This is one of the best-documented variant replacement events and serves as an independent validation dataset.

Usage

```
cdc_ba2_transition
```

Format

A data frame with 150 rows and 4 columns:

date Biweek ending date (Date).

lineage Lineage name: BA.1, BA.2, BA.2.12.1, BA.4/5, Other.

count Approximate sequence count per biweek (integer).

proportion CDC weighted proportion estimate (numeric).

Source

CDC COVID Data Tracker (data.cdc.gov, public domain).

References

Lyngse FP, et al. (2022). Household transmission of SARS-CoV-2 Omicron variant of concern subvariants BA.1 and BA.2 in Denmark. *Nature Communications*, 13:5760. doi:10.1038/s41467-022334980

Examples

```
data(cdc_ba2_transition)
vd <- lfq_data(cdc_ba2_transition,
               date = date, lineage = lineage, count = count)
fit <- fit_model(vd, engine = "mlr", pivot = "BA.1")
# BA.2 Rt ~ 1.34 (consistent with published estimates)
growth_advantage(fit, type = "relative_Rt", generation_time = 3.2)
```

---

cdc_sarscov2_jn1 *CDC SARS-CoV-2 variant proportions: JN.1 emergence (US, 2023-2024)*

Description

Real surveillance data from the CDC's national genomic surveillance program covering the emergence and dominance of the SARS-CoV-2 JN.1 lineage in the United States, October 2023 through June 2024.

Usage

```
cdc_sarscov2_jn1
```

Format

A data frame with 171 rows and 4 columns:

date Biweek ending date (Date).

lineage Lineage name (character): JN.1, XBB.1.5, EG.5.1, HV.1, HK.3, BA.2.86, KP.2, KP.3, JN.1.11.1, Other.

count Approximate sequence count per biweek (integer).

proportion CDC weighted proportion estimate (numeric).

Details

Derived from CDC's published weighted variant proportion estimates. Approximate biweekly sequence counts were reconstructed from proportions using a reference total of 8,000 sequences per period. The original proportions are retained in the `proportion` column.

Source

CDC COVID Data Tracker, SARS-CoV-2 Variant Proportions. Dataset ID: jr58-6ysp. https://data.cdc.gov/Laboratory-Surveillance/SARS-CoV-2-Variant-Proportions/jr58-6ysp

Public domain (U.S. Government Work, 17 USC 105).

References

Ma KC, et al. (2024). Genomic Surveillance for SARS-CoV-2 Variants. *MMWR*, 73(42):941–948. doi:10.15585/mmwr.mm7342a1

Examples

```
data(cdc_sarscov2_jn1)
vd <- lfq_data(cdc_sarscov2_jn1,
               date = date, lineage = lineage, count = count)
fit <- fit_model(vd, engine = "mlr")
growth_advantage(fit, type = "relative_Rt", generation_time = 5)
```

---

coef.lfq_fit *Extract coefficients from a lineage frequency model*

Description

Extract coefficients from a lineage frequency model

Usage

```
## S3 method for class 'lfq_fit'
coef(object, type = c("growth_rate", "all"), ...)
```

Arguments

object	An lfq_fit object.
type	What to return: "growth_rate" (default) for growth rates only, or "all" for intercepts and growth rates.
...	Ignored.

Value

Named numeric vector.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
coef(fit)
```

`collapse_lineages` *Collapse rare lineages into an aggregate group*

Description

Merges lineages that never exceed a frequency or count threshold into a single group (default "Other"). Useful for reducing noise from dozens of low-frequency lineages.

Usage

```
collapse_lineages(
  x,
  min_freq = 0.01,
  min_count = 10L,
  other_label = "Other",
  mapping = NULL
)
```

Arguments

`x`	An lfq_data object.
`min_freq`	Minimum peak frequency a lineage must reach at any time point to be kept. Default 0.01 (1%).
`min_count`	Minimum total count across all time points to be kept. Default 10.
`other_label`	Label for the collapsed group. Default "Other".
`mapping`	Optional named character vector for custom grouping. Names = original lineage names, values = group names. If provided, `min_freq` and `min_count` are ignored.

Value

An lfq_data object with rare lineages merged.

Examples

```
sim <- simulate_dynamics(n_lineages = 6,
  advantages = c(A = 1.3, B = 1.1, C = 0.95, D = 0.5, E = 0.3),
  n_timepoints = 12, seed = 1)
collapsed <- collapse_lineages(sim, min_freq = 0.05)
attr(collapsed, "lineages")
```

compare_models *Compare model engines from backtest scores*

Description

Summarises and ranks engines across horizons based on forecast accuracy scores.

Usage

```
compare_models(scores, by = "engine")
```

Arguments

scores	Output of score_forecasts().
by	Grouping variable(s). Default "engine".

Value

A tibble with average scores per group, sorted by MAE.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
bt <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
sc <- score_forecasts(bt)
compare_models(sc)
```

filter_sparse *Filter sparse time points and lineages*

Description

Removes time points with very low total counts and lineages observed at too few time points.

Usage

```
filter_sparse(x, min_total = 10L, min_timepoints = 3L)
```

Arguments

x	An lfq_data object.
min_total	Minimum total sequences per time point. Default 10.
min_timepoints	Minimum number of time points a lineage must appear to be retained. Default 3.

Value

An lfq_data object with sparse entries removed.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
filtered <- filter_sparse(sim, min_total = 100)
```

---

fit_model *Fit a lineage frequency model*

Description

Unified interface for modeling lineage frequency dynamics. Supports multiple engines that share the same input/output contract.

Usage

```
fit_model(
  data,
  engine = c("mlr", "hier_mlr", "piantham", "fga", "garw"),
  pivot = NULL,
  horizon = 28L,
  ci_level = 0.95,
  ...
)
```

Arguments

data	An lfq_data object.
engine	Model engine to use: • "mlr" (default): Multinomial logistic regression. Fast, frequentist, no external dependencies. • "hier_mlr": Hierarchical MLR with partial pooling across locations. Requires .location column in data. • "piantham": Piantham approximation converting MLR growth rates to relative reproduction numbers. Requires generation_time argument.

- `"fga"`: Fixed growth advantage model (Bayesian via CmdStan). Requires 'CmdStan'; check with `lfq_stan_available()`.
- `"garw"`: Growth advantage random walk model (Bayesian via CmdStan). Allows fitness to change over time.

`pivot` Reference lineage name. Growth rates are reported relative to this lineage (fixed at 0). Default: the lineage with the highest count at the earliest time point.

`horizon` Forecast horizon in days (stored for later use by `forecast()`). Default 28.

`ci_level` Confidence level for intervals. Default 0.95.

`...` Engine-specific arguments passed to the internal engine function. For `engine = "mlr"`: `window`, `ci_method`, `laplace_smooth`. For `engine = "piantham"`: `generation_time` (required). For `engine = "hier_mlr"`: `shrinkage_method`.

Details

The MLR engine models the frequency of lineage v at time t as:

$$p_v(t) = \frac{\exp(\alpha_v + \delta_v t)}{\sum_k \exp(\alpha_k + \delta_k t)}$$

where α_v is the intercept, δ_v is the growth rate per `time_scale` days (default 7), and the pivot lineage has $\alpha = \delta = 0$. Parameters are estimated by maximum likelihood via `optim(method = "BFGS")` with the Hessian used for asymptotic Wald confidence intervals.

The constant growth rate assumption is appropriate for monotonic variant replacement periods (typically 2–4 months). For longer periods or non-monotonic dynamics, use the `window` argument to restrict the estimation window, or consider the `"garw"` engine which allows time-varying growth advantages.

Value

An `lfq_fit` object (S3 class), a list containing:

engine Engine name (character).

growth_rates Named numeric vector of growth rates per `time_scale` days (pivot = 0).

intercepts Named numeric vector of intercepts.

pivot Name of pivot lineage.

lineages Character vector of all lineage names.

fitted_values Tibble of fitted frequencies.

residuals Tibble with observed, fitted, Pearson residuals.

vcov_matrix Variance-covariance matrix.

loglik, aic, bic Model fit statistics.

nobs, n_timepoints, df Sample and model sizes.

ci_level, horizon As specified.

call The matched call.

References

Abousamra E, Figgins M, Bedford T (2024). Fitness models provide accurate short-term forecasts of SARS-CoV-2 variant frequency. *PLoS Computational Biology*, 20(9):e1012443. doi:10.1371/journal.pcbi.1012443

Piantham C, Linton NM, Nishiura H (2022). Predicting the trajectory of replacements of SARS-CoV-2 variants using relative reproduction numbers. *Viruses*, 14(11):2556. doi:10.3390/v14112556

See Also

`growth_advantage()` to extract fitness estimates, `forecast()` for frequency prediction, `backtest()` for rolling-origin evaluation.

Examples

```
sim <- simulate_dynamics(
  n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15, seed = 42
)
fit <- fit_model(sim, engine = "mlr")
fit
```

---

forecast *Forecast lineage frequencies (generic)*

Description

Forecast lineage frequencies (generic)

Usage

```
forecast(object, ...)
```

Arguments

`object`	A model object.
`...`	Additional arguments passed to methods.

Value

A forecast object.

`forecast.lfq_fit` *Forecast lineage frequencies*

Description

Projects lineage frequencies forward in time using the fitted model. Prediction uncertainty is quantified by parametric simulation from the estimated parameter distribution.

Usage

```
## S3 method for class 'lfq_fit'
forecast(
  object,
  horizon = 28L,
  ci_level = 0.95,
  n_sim = 1000L,
  sampling_noise = TRUE,
  effective_n = 100L,
  ...
)
```

Arguments

`object`	An `lfq_fit` object.
`horizon`	Number of days to forecast. Default 28 (4 weeks).
`ci_level`	Confidence level for prediction intervals. Default 0.95.
`n_sim`	Number of parameter draws for prediction intervals. Default 1000.
`sampling_noise`	Logical; add multinomial sampling noise to prediction intervals? Default `TRUE`. When `TRUE`, each draw includes both parameter uncertainty (from the MLE covariance) and observation-level multinomial sampling variability.
`effective_n`	Effective sample size for multinomial sampling noise. Default 100, corresponding to a typical weekly sequencing volume per reporting unit. Smaller values produce wider (more conservative) prediction intervals.
`...`	Unused.

Value

An `lfq_forecast` object (tibble subclass) with columns:

.date Date.

.lineage Lineage name.

.median Median predicted frequency.

.lower Lower prediction bound.

.upper Upper prediction bound.

.type "fitted" or "forecast".

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim, engine = "mlr")
fc <- forecast(fit, horizon = 21)
fc
```

---

glance.lfq_fit *Glance at an lfq_fit object*

Description

Returns a single-row tibble of model-level summary statistics.

Usage

```
glance.lfq_fit(x, ...)
```

Arguments

x	An lfq_fit object.
...	Ignored.

Value

A single-row tibble with columns: engine, n_lineages, n_timepoints, nobs, df, logLik, AIC, BIC, pivot, convergence.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
glance.lfq_fit(fit)
```

`growth_advantage` *Extract growth advantage estimates*

Description

Computes relative fitness of each lineage from a fitted model. Supports four output formats for different use cases.

Usage

```
growth_advantage(
  fit,
  type = c("growth_rate", "relative_Rt", "selection_coefficient", "doubling_time"),
  generation_time = NULL,
  ci_level = NULL
)
```

Arguments

`fit`	An `lfq_fit` object returned by `fit_model()`.
`type`	Output type: • `"growth_rate"` (default): raw growth rate delta per `time_scale` days (typically per week). • `"relative_Rt"`: relative effective reproduction number. Requires `generation_time`. • `"selection_coefficient"`: relative Rt minus 1. Requires `generation_time`. • `"doubling_time"`: days for frequency ratio vs pivot to double (positive = growing) or halve (negative = declining).
`generation_time`	Mean generation time in days. Required for `type = "relative_Rt"` and `"selection_coefficient"`. Common values: SARS-CoV-2 approximately 5 days, influenza approximately 3 days.
`ci_level`	Confidence level for intervals. Default uses the level from the fitted model.

Details

The `"relative_Rt"` and `"selection_coefficient"` types use the Piantham approximation (Piantham et al. 2022, doi:10.3390/v14112556), which assumes:

1. Variants are in their exponential growth phase (not saturating due to population-level immunity).
2. All variants share the same generation time distribution.
3. Growth advantage reflects transmissibility differences; immune escape is not separately identified.

Confidence intervals for `"relative_Rt"` are computed in log-space (Wald intervals on log-Rt), which is more accurate than the linear delta method for ratios.

Value

A tibble with columns:

lineage Lineage name.

estimate Point estimate.

lower Lower confidence bound.

upper Upper confidence bound.

type Type of estimate.

pivot Name of pivot (reference) lineage.

References

Piantham C, Linton NM, Nishiura H (2022). Predicting the trajectory of replacements of SARS-CoV-2 variants using relative reproduction numbers. *Viruses*, 14(11):2556. doi:10.3390/v14112556

Abousamra E, Figgins M, Bedford T (2024). Fitness models provide accurate short-term forecasts of SARS-CoV-2 variant frequency. *PLoS Computational Biology*, 20(9):e1012443. doi:10.1371/journal.pcbi.1012443

Examples

```
sim <- simulate_dynamics(
  n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15, seed = 42
)
fit <- fit_model(sim, engine = "mlr")

# Growth rates per week
growth_advantage(fit, type = "growth_rate")

# Relative Rt (needs generation time)
growth_advantage(fit, type = "relative_Rt", generation_time = 5)
```

---

influenza_h3n2 *Simulated influenza A/H3N2 clade frequency data*

Description

A simulated dataset of weekly influenza A/H3N2 clade sequence counts over a single Northern Hemisphere season (24 weeks).

Usage

```
influenza_h3n2
```

Format

A data frame with 96 rows and 4 columns:

date Collection date (Date, weekly).

clade Clade name (character).

count Number of sequences (integer).

total Total sequences for this week (integer).

Source

Simulated based on 'Nextstrain' influenza clade dynamics.

Examples

```
data(influenza_h3n2)
x <- lfq_data(influenza_h3n2, lineage = clade,
              date = date, count = count, total = total)
x
```

---

is_lfq_data *Test if an object is an lfq_data object*

Description

Test if an object is an lfq_data object

Usage

```
is_lfq_data(x)
```

Arguments

x	Object to test.

Value

Logical scalar.

Examples

```
d <- data.frame(date = Sys.Date(), lineage = "A", count = 10)
x <- lfq_data(d, lineage = lineage, date = date, count = count)
is_lfq_data(x)
is_lfq_data(d)
```

lfq_advantage *Pipe-friendly growth advantage extraction*

Description

Pipe-friendly growth advantage extraction

Usage

```
lfq_advantage(fit, type = "relative_Rt", generation_time = NULL, ...)
```

Arguments

fit	An lfq_fit object.
type	Output type. Default "relative_Rt".
generation_time	Mean generation time in days.
...	Passed to growth_advantage().

Value

A tibble of growth advantages.

lfq_data *Create a lineage frequency data object*

Description

Validates, structures, and annotates lineage count data for downstream modeling and analysis. This is the entry point for all lineagefreq workflows.

Usage

```
lfq_data(
  data,
  lineage,
  date,
  count,
  total = NULL,
  location = NULL,
  min_total = 10L
)
```

Arguments

data	A data frame containing at minimum columns for lineage identity, date, and count.
lineage	<tidy-select> Column containing lineage/variant identifiers (character or factor).
date	<tidy-select> Column containing collection dates (Date class or parseable character).
count	<tidy-select> Column containing sequence counts (non-negative integers).
total	<tidy-select> Optional column of total sequences per date-location. If NULL, computed as the sum of count per group.
location	<tidy-select> Optional column for geographic stratification.
min_total	Minimum total count per time point. Time points below this are flagged as unreliable. Default 10.

Details

Performs the following validation and processing:

1. Checks that all required columns exist and have correct types.
2. Coerces character dates to Date via ISO 8601 parsing.
3. Ensures counts are non-negative integers.
4. Replaces NA counts with 0 (with warning).
5. Aggregates duplicate lineage-date rows by summing (with warning).
6. Computes per-time-point totals and frequencies.
7. Flags time points below min_total as unreliable.
8. Sorts by date ascending, then lineage alphabetically.

Value

An lfq_data object (a tibble subclass) with standardized columns:

.lineage Lineage identifier (character).

.date Collection date (Date).

.count Sequence count (integer).

.total Total sequences at this time point (integer).

.freq Observed frequency (numeric).

.reliable Logical; TRUE if .total >= min_total.

.location Location, if provided (character).

All original columns are preserved.

Examples

```
d <- data.frame(
  date = rep(seq(as.Date("2024-01-01"), by = "week",
                 length.out = 8), each = 3),
  lineage = rep(c("JN.1", "KP.3", "Other"), 8),
  n = c(5, 2, 93, 12, 5, 83, 28, 11, 61, 50, 20, 30,
        68, 18, 14, 80, 12, 8, 88, 8, 4, 92, 5, 3)
)
x <- lfq_data(d, lineage = lineage, date = date, count = n)
x
```

---

lfq_engines *List available modeling engines*

Description

Returns information about all modeling engines available in lineagefreq. Core engines (mlr, hier_mlr, piantham) are always available. Bayesian engines (fga, garw) require 'CmdStan'.

Usage

```
lfq_engines()
```

Value

A tibble with columns: `engine`, `type`, `time_varying`, `available`, `description`.

Examples

```
lfq_engines()
```

---

lfq_fit *Pipe-friendly model fitting*

Description

Enables tidyverse-style chaining: `data |> lfq_fit("mlr") |> lfq_forecast(28) |> lfq_score()`

Usage

```
lfq_fit(data, engine = "mlr", ...)
```

Arguments

data	An lfq_data object.
engine	Engine name. Default "mlr".
...	Passed to fit_model().

Value

An lfq_fit object.

Examples

```
data(sarscov2_us_2022)
sarscov2_us_2022 |>
  lfq_data(lineage = variant, date = date, count = count, total = total) |>
  lfq_fit("mlr") |>
  lfq_advantage(generation_time = 5)
```

---

lfq_forecast *Pipe-friendly forecasting*

Description

Pipe-friendly forecasting

Usage

```
lfq_forecast(fit, horizon = 28L, ...)
```

Arguments

fit	An lfq_fit object.
horizon	Forecast horizon in days. Default 28.
...	Passed to forecast().

Value

An lfq_forecast object.

lfq_score *Pipe-friendly backtesting + scoring*

Description

Runs backtest and returns scores in one step.

Usage

```
lfq_score(
  data,
  engines = "mlr",
  horizons = c(14, 28),
  metrics = c("mae", "coverage"),
  ...
)
```

Arguments

data	An lfq_data object.
engines	Character vector of engine names.
horizons	Forecast horizons in days.
metrics	Score metrics.
...	Passed to backtest().

Value

A tibble of scores.

lfq_stan_available *Check if 'CmdStan' backend is available*

Description

Returns TRUE if 'CmdStanR' and 'CmdStan' are installed. Bayesian engines require this.

Usage

```
lfq_stan_available()
```

Value

Logical scalar.

Examples

```
lfq_stan_available()
```

`lfq_summary` *Convert lfq_fit results to a summary tibble*

Description

Returns a one-row-per-lineage summary with growth rates, fitted frequencies at first and last time points, and growth advantage in multiple scales.

Usage

```
lfq_summary(fit, generation_time = NULL)
```

Arguments

`fit`	An `lfq_fit` object.
`generation_time`	Generation time for Rt calculation.

Value

A tibble.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
lfq_summary(fit, generation_time = 5)
```

`lfq_version` *Package version and system information*

Description

Reports lineagefreq version and availability of optional backends. Useful for reproducibility and bug reports.

Usage

```
lfq_version()
```

Value

A list with components: `version`, `r_version`, `stan_available`, `engines`.

Examples

```
lfq_version()
```

`plot_backtest` *Plot backtest scores*

Description

Creates a panel plot of forecast accuracy by engine and horizon.

Usage

```
plot_backtest(scores)
```

Arguments

`scores`	Output of `score_forecasts()`.

Value

A ggplot object.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
bt <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
sc <- score_forecasts(bt)
plot_backtest(sc)
```

print.lfq_fit *Print a lineage frequency model*

Description

Print a lineage frequency model

Usage

```
## S3 method for class 'lfq_fit'
print(x, ...)
```

Arguments

x	An lfq_fit object.
...	Ignored.

Value

Invisibly returns x.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15, seed = 42)
fit <- fit_model(sim, engine = "mlr")
print(fit)
```

read_lineage_counts *Read lineage count data from a CSV file*

Description

A convenience wrapper for reading surveillance count data from CSV files into lfq_data format. Expects a file with at least columns for date, lineage, and count.

Usage

```
read_lineage_counts(
  file,
  date = "date",
  lineage = "lineage",
  count = "count",
  ...
)
```

Arguments

file	Path to CSV file.
date	Name of the date column. Default "date".
lineage	Name of the lineage column. Default "lineage".
count	Name of the count column. Default "count".
...	Additional arguments passed to lfq_data().

Value

An lfq_data object.

Examples

```
# Read the bundled example CSV
f <- system.file("extdata", "example_counts.csv",
                 package = "lineagefreq")
x <- read_lineage_counts(f)
x
```

---

register_engine *Register a custom modeling engine*

Description

Allows third-party packages to register custom engines with fit_model(). This enables an extensible plugin architecture similar to 'parsnip' engine registration.

Usage

```
register_engine(
  name,
  fit_fn,
  description = "",
  type = "frequentist",
  time_varying = FALSE
)
```

Arguments

name	Engine name (character scalar).
fit_fn	Function with signature function(data, pivot, ci_level, ...). Must return a list compatible with lfq_fit structure.
description	One-line description of the engine.
type	"frequentist" or "bayesian".
time_varying	Logical; does the engine support time-varying growth advantages?

Value

Invisibly returns the updated engine registry.

Examples

```
# Register a custom engine
my_engine <- function(data, pivot = NULL, ci_level = 0.95, ...) {
  # Custom implementation...
  .engine_mlr(data, pivot = pivot, ci_level = ci_level, ...)
}
register_engine("my_mlr", my_engine, "Custom MLR wrapper")
lfq_engines()
```

---

sarscov2_us_2022 *Simulated SARS-CoV-2 variant frequency data (US, 2022)*

Description

A simulated dataset of weekly SARS-CoV-2 variant sequence counts for the United States in 2022. Includes the BA.1 to BA.2 to BA.4/5 to BQ.1 transition dynamics. This is simulated data (not real GISAID data) to avoid license restrictions while preserving realistic statistical properties.

Usage

```
sarscov2_us_2022
```

Format

A data frame with 200 rows and 4 columns:

date Collection date (Date, weekly).

variant Variant name (character): BA.1, BA.2, BA.4/5, BQ.1, Other.

count Number of sequences assigned to this variant in this week (integer).

total Total sequences for this week (integer).

Source

Simulated based on parameters from published CDC MMWR genomic surveillance reports and 'Nextstrain' public data.

Examples

```
data(sarscov2_us_2022)
x <- lfq_data(sarscov2_us_2022, lineage = variant,
              date = date, count = count, total = total)
x
```

score_forecasts *Score backtest forecast accuracy*

Description

Computes standardized accuracy metrics from backtesting results.

Usage

```
score_forecasts(bt, metrics = c("mae", "rmse", "coverage", "wis"))
```

Arguments

bt	An lfq_backtest object from backtest().
metrics	Character vector of metrics to compute: • "mae": Mean absolute error of frequency. • "rmse": Root mean squared error. • "coverage": Proportion within prediction intervals. • "wis": Simplified weighted interval score for the single prediction interval stored in the backtest (typically 95%). For the full multi-quantile WIS per Bracher et al. (2021), use dedicated scoring packages such as 'scoringutils'.

Value

A tibble with columns: engine, horizon, metric, value.

References

Bracher J, Ray EL, Gneiting T, Reich NG (2021). Evaluating epidemic forecasts in an interval format. *PLoS Computational Biology*, 17(2):e1008618. doi:10.1371/journal.pcbi.1008618

See Also

compare_models() to rank engines based on scores.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
bt <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
score_forecasts(bt)
```

`sequencing_power` *Sequencing power analysis*

Description

Estimates the minimum number of sequences needed to detect a lineage at a given frequency with specified precision.

Usage

```
sequencing_power(target_precision = 0.05, current_freq = 0.02, ci_level = 0.95)
```

Arguments

`target_precision`	Desired half-width of the frequency confidence interval. Default 0.05 (plus/minus 5 percentage points).
`current_freq`	True or assumed frequency of the target lineage. Can be a vector for multiple scenarios. Default 0.02 (2%).
`ci_level`	Confidence level. Default 0.95.

Details

Uses the normal approximation to the binomial:

$$n = z^2 \cdot p(1-p)/E^2$$

where z is the critical value, p is frequency, E is precision.

Value

A tibble with columns: `current_freq`, `target_precision`, `required_n`, `ci_level`.

Examples

```
# How many sequences to estimate a 2% lineage within +/-5%?
sequencing_power()

# Multiple scenarios
sequencing_power(current_freq = c(0.01, 0.02, 0.05, 0.10))
```

`simulate_dynamics` *Simulate lineage frequency dynamics*

Description

Generates synthetic lineage frequency data under a multinomial sampling model with configurable growth advantages. Useful for model validation, power analysis, and teaching.

Usage

```
simulate_dynamics(
  n_lineages = 4L,
  n_timepoints = 20L,
  total_per_tp = 500L,
  advantages = NULL,
  reference_name = "ref",
  start_date = Sys.Date(),
  interval = 7L,
  overdispersion = NULL,
  seed = NULL
)
```

Arguments

`n_lineages`	Number of lineages including reference. Default 4.
`n_timepoints`	Number of time points. Default 20.
`total_per_tp`	Sequences per time point. A single integer (constant across time) or a vector of length `n_timepoints` (variable sampling effort). Default 500.
`advantages`	Named numeric vector of per-week multiplicative growth advantages for non-reference lineages. Length must equal `n_lineages - 1`. Values > 1 mean growing faster than reference, < 1 means declining. If `NULL`, random values in (0.8, 1.5).
`reference_name`	Name of the reference lineage. Default `"ref"`.
`start_date`	Start date for the time series. Default today.
`interval`	Days between consecutive time points. Default 7 (weekly data).
`overdispersion`	If `NULL` (default), standard multinomial sampling. If a positive number, Dirichlet-Multinomial sampling with concentration = 1 / overdispersion. Larger values = more overdispersion.
`seed`	Random seed for reproducibility.

Value

An lfq_data object with an additional `true_freq` column containing the true (pre-sampling) frequencies.

Examples

```
# JN.1 grows 1.3x per week, KP.3 declines at 0.9x
sim <- simulate_dynamics(
  n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15,
  seed = 42
)
sim
```

`summarize_emerging` *Summarize emerging lineages*

Description

Tests whether each lineage's frequency is significantly increasing over time using a binomial GLM. Useful for early warning of lineages that may warrant enhanced surveillance.

Usage

```
summarize_emerging(data, threshold = 0.01, min_obs = 3L, p_adjust = "holm")
```

Arguments

`data`	An lfq_data object.
`threshold`	Minimum current frequency to test. Default 0.01.
`min_obs`	Minimum time points observed. Default 3.
`p_adjust`	P-value adjustment method. Default `"holm"`.

Value

A tibble with columns: `lineage`, `first_seen`, `last_seen`, `n_timepoints`, `current_freq`, `growth_rate`, `p_value`, `p_adjusted`, `significant`, `direction`.

Examples

```
sim <- simulate_dynamics(
  n_lineages = 4,
  advantages = c(emerging = 1.5, stable = 1.0, declining = 0.7),
  n_timepoints = 12, seed = 42)
summarize_emerging(sim)
```

summary.lfq_fit *Summarise a lineage frequency model*

Description

Summarise a lineage frequency model

Usage

```
## S3 method for class 'lfq_fit'
summary(object, ...)
```

Arguments

object	An lfq_fit object.
...	Ignored.

Value

Invisibly returns object.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15, seed = 42)
fit <- fit_model(sim, engine = "mlr")
summary(fit)
```

tidy.lfq_fit *Tidy an lfq_fit object*

Description

Converts model results to a tidy tibble, compatible with the broom package ecosystem.

Usage

```
tidy.lfq_fit(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

x	An lfq_fit object.
conf.int	Include confidence intervals? Default TRUE.
conf.level	Confidence level. Default 0.95.
...	Ignored.

Value

A tibble with columns: `lineage`, `term`, `estimate`, `std.error`, `conf.low`, `conf.high`.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
tidy.lfq_fit(fit)
```

---

unregister_engine *Remove a registered engine*

Description

Remove a registered engine

Usage

```
unregister_engine(name)
```

Arguments

`name`	Engine name to remove.

Value

Invisibly returns the updated registry.

Index