Package 'oystermapR'

October 13, 2026

Title Predict and Map Oyster Growth Suitability from Environmental Data

Version 1.6.0

Description Predicts spatial suitability for oyster growth from environmental survey data using rank-reciprocal weighted scoring.
Users supply sensor data from Acoustic Doppler Current Profilers (ADCP), Conductivity-Temperature-Depth (CTD) sensors, bathymetric sonar, and sidescan sonar, specify a target species, and receive per-location suitability scores, a five-band 'GeoTIFF' heatmap for 'QGIS', contour lines, and a formatted PDF or HTML report. Supports seventeen species across global aquaculture regions, including Ostrea edulis, Magallana gigas, Crassostrea virginica, Crassostrea hongkongensis, and thirteen further species; see list_species(). Includes ocean acidification scoring via in-house aragonite saturation state (Omega_arag) calculation using Lueker et al. (2000) <doi:10.1016/S0304-4203(00)00022-0> and Mucci (1983) <doi:10.1357/002224083788520153> equilibrium constants (no external dependencies), variable impact diagnostics (variable_impact()), fine-scale habitat area analysis for restoration reporting in m2 with contiguous patch identification (area_summary()), tolerance curve visualisation (plot_tolerance()), season-aware scoring, tidal height correction, Bayesian tolerance parameter updating from field observations, spatial block cross-validation (Roberts et al., 2017, <doi:10.1111/ecog.02881>), permutation variable importance, wave exposure and sediment stability modules, Harmful Algal Bloom (HAB) risk and anthropogenic disturbance scoring with optional live International Council for the Exploration of the Sea (ICES) data integration, hybrid larval dispersal connectivity scoring (union-find Gaussian kernel plus optional 'OpenDrift' or Finite Volume Community Ocean Model ('FVCOM') connectivity matrix), batch multi-species comparison, and evidence-weighted scoring in which each variable's weight is its published rank multiplied by the measured reliability and realised spread of that layer, with the fraction of the species' evidence base the survey observed reported alongside the score rather than normalised away, and lineage-specific tolerance profiles for related populations, in which thinly sampled lineages borrow strength from relatives across a genetic distance matrix only where tolerance is shown to track relatedness, with inverse mapping of where each lineage is more likely

to occur and which source population suits a site.

License GPL (>= 3)

URL https://github.com/trissyboats/oystermapR

BugReports https://github.com/trissyboats/oystermapR/issues

Encoding UTF-8

Language en-US

VignetteBuilder knitr

RoxygenNote 7.3.3

Imports dplyr (>= 1.0.0), terra (>= 1.7.18), sf (>= 1.0.0), rlang (>= 1.0.0), cli (>= 3.0.0), stats, utils

Suggests ggplot2 (>= 3.4.0), testthat (>= 3.0.0), knitr, rmarkdown (>= 2.20), tinytex, httr (>= 1.4.0), jsonlite (>= 1.8.0), ncdf4 (>= 1.19), gstat (>= 2.1.0), sp (>= 1.6.0), lubridate (>= 1.9.0), suncalc (>= 0.5.0), oce (>= 1.7.0)

Depends R (>= 4.1.0)

NeedsCompilation no

Author T Tucker [aut, cre] (ORCID: <https://orcid.org/0009-0008-5190-6423>)

Maintainer T Tucker <tristantucker48@gmail.com>

Repository CRAN

Date/Publication 2026-10-13 10:40:02 UTC

Contents

add_intertidal_flag 4
add_season_column 5
add_shellfish_classification 6
add_suitability_ci 7
analyse_connectivity 8
area_summary 9
assess_gear_feasibility 11
as_lineage_distances 12
auto_tidal_correct 13
calculate_aragonite 14
check_exclusions 16
classify_substrate_from_backscatter 18
compare_species 19
compare_surveys 21
composite_seasonal 22
confidence_score 23
correct_to_chart_datum 24
detect_season 25
estimate_chlorophyll_from_backscatter 26

estimate_fetch . . . 28
evidence_coverage . . . 29
evidence_suitability . . . 30
evidence_weights . . . 31
export_contours . . . 32
export_geotiff . . . 33
export_qml_style . . . 35
fetch_live_environmental_data . . . 35
fit_lineages . . . 36
generate_report . . . 39
generate_summary_pdf . . . 40
get_species_tolerances . . . 41
get_tolerance_posteriors . . . 42
identify_resilient_sites . . . 43
interpolate_survey . . . 43
layer_reproducibility . . . 45
lineage_fits_as_list . . . 47
lineage_occurrence . . . 47
lineage_signal . . . 48
list_species . . . 49
load_tolerance_update . . . 50
merge_sensor_data . . . 51
oystermapR_help . . . 52
oystermapR_live_config . . . 53
parse_opendrift_connectivity . . . 54
permutation_importance . . . 55
plot_tolerance . . . 57
predict_oyster . . . 58
project_suitability . . . 62
qc_survey_data . . . 63
rank_source_populations . . . 65
read_aanderaa_csv . . . 66
read_generic_csv . . . 67
read_nortek_adcp . . . 69
read_nortek_aquadopp . . . 70
read_rdi_adcp . . . 72
read_sonar_tif . . . 73
read_soundings_xyz . . . 75
reliability_from_perturbation . . . 76
reliability_from_replicates . . . 77
reset_tolerance_update . . . 78
save_tolerance_update . . . 79
score_anthropogenic_disturbance . . . 80
score_disease_risk . . . 82
score_economic_viability . . . 83
score_hab_risk . . . 84
score_larval_connectivity . . . 86
score_locations . . . 89

score_predation_risk 89
score_sediment_stability 91
score_settlement 93
score_wave_exposure 94
sensitivity_analysis 95
smooth_suitability 97
spatial_block_cv 98
stack_surveys 100
tidal_port_info 101
update_species_tolerances 101
validate_against_records 104
variable_impact 106
weight_table_md 107

Index **108**

`add_intertidal_flag` *Add an intertidal zone flag to a survey dataframe*

Description

Classifies each survey location as intertidal, subtidal, or supratidal based on its depth below Chart Datum (CD) and the tidal range at the nearest harmonic reference port. Chart Datum is approximately Lowest Astronomical Tide (LAT), so:

- **Subtidal:** `depth > 0` (always submerged)
- **Intertidal:** `-MHWS_above_CD <= depth <= 0` (exposed at some states of the tide; depth is at or above CD)
- **Supratidal:** `depth < -MHWS_above_CD` (above the highest astronomical tide; excluded from scoring)

MHWS above CD is approximated from the harmonic constituents of the nearest port as: `Z0 + 1.1 * (M2_H + S2_H)` – the 10% factor accounts for minor constituents not included in the 5-constituent model.

This flag is used internally by `score_locations()` to give *Magallana gigas* (Pacific oyster) full depth scores for intertidal cells, reflecting its strong intertidal ecology in NW European waters.

Usage

```
add_intertidal_flag(
  df,
  max_port_dist_km = 75,
  depth_col = "depth",
  verbose = TRUE
)
```

Arguments

df	Dataframe with lat, lon, and depth (chart-datum corrected) columns. Typically the output of auto_tidal_correct() or correct_to_chart_datum().
max_port_dist_km	Numeric. Maximum distance to nearest harmonic port in km (default 75). If the survey centroid is further than this from all ports, the function falls back to a conservative intertidal window of 6 m above CD and issues a warning.
depth_col	Character. Name of the chart-datum depth column (default "depth").
verbose	Logical. Print the nearest port name and derived MHWS (default TRUE).

Value

The input dataframe with an added intertidal_zone column:

- "subtidal": depth > 0 m CD
- "intertidal": 0 m CD >= depth >= -MHWS_above_CD
- "supratidal": depth < -MHWS_above_CD

Examples

```
# Minimal survey with chart-datum corrected depths.
# Positive depth = below Chart Datum (always submerged).
# Depth near 0 = intertidal zone. Negative = above MHWS (supratidal).
df <- data.frame(
  lat   = c(51.50, 51.51, 51.52),
  lon   = c(-3.18, -3.18, -3.17),
  depth = c(8.2, 0.4, -0.5)
)
df <- add_intertidal_flag(df, verbose = FALSE)
table(df$intertidal_zone)
```

---

add_season_column *Detect season for each row in a dataframe*

Description

Vectorised wrapper around detect_season() for use on dataframes.

Usage

```
add_season_column(df, date_col = NULL, lat_col = "lat")
```

Arguments

df	A dataframe containing at minimum a date column and a latitude column.
date_col	Name of the date column (default "date").
lat_col	Name of the latitude column (default "lat").

Value

The input dataframe with an additional column `season`.

`add_shellfish_classification` *Add shellfish water quality classification to scored result*

Description

Attaches a regulatory shellfish harvesting classification (A/B/C/Prohibited) to each row of a scored result dataframe. This classification affects the economic viability score for aquaculture applications: Prohibited sites receive viability = 0; Class C sites receive a 0.60 multiplier; Class B a 0.80 multiplier; Class A is unpenalised.

Classification can be supplied three ways (in priority order):

1. `class_col` – name of an existing column in `result` already containing classification values ("A", "B", "C", "Prohibited", or NA).
2. `classified_areas` – a dataframe with `lat`, `lon`, and `shellfish_class` columns, spatially matched to result rows within `match_radius_deg` degrees.
3. `fetch_live = TRUE` – queries the FSA England/Wales open data API (requires internet). Requires `httr` package. Works only for sites within the FSA/DAERA coverage area (England, Wales, Northern Ireland).

Unclassified or unmatched sites receive `shellfish_class = "Unclassified"` and a penalty of 0.70 (precautionary principle).

Usage

```
add_shellfish_classification(
  result,
  class_col = NULL,
  classified_areas = NULL,
  fetch_live = FALSE,
  match_radius_deg = 0.05,
  verbose = TRUE
)
```

Arguments

`result`	Dataframe from `predict_oyster()` or `score_locations()`. Must contain `lat` and `lon`.
`class_col`	Character. Name of an existing column in `result` containing class values. If supplied, `classified_areas` and `fetch_live` are ignored.
`classified_areas`	Dataframe with `lat`, `lon`, `shellfish_class` columns. Used when `class_col` is NULL and `fetch_live = FALSE`.

`fetch_live` Logical. Query live FSA/DAERA API (default FALSE). Requires internet access and the `httr` package. If fetch fails, falls back to "Unclassified".

`match_radius_deg` Numeric. Spatial matching tolerance in decimal degrees when using `classified_areas` (default 0.05 = ~5 km).

`verbose` Logical. Print matching summary (default TRUE).

Value

`result` with two additional columns:

- `shellfish_class`: "A", "B", "C", "Prohibited", or "Unclassified"
- `shellfish_class_penalty`: multiplier applied to economic viability (1.0 for A, 0.80 for B, 0.60 for C, 0.0 for Prohibited, 0.70 for Unclassified)

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

# Option 1: manual class column already present in data
result$water_class <- sample(c("A", "B", "C"), nrow(result), replace = TRUE)
result <- add_shellfish_classification(result, class_col = "water_class")

# Option 2: separate classified areas dataframe (spatially matched)
areas <- data.frame(lat = c(50.3), lon = c(-4.2), shellfish_class = c("B"))
result2 <- add_shellfish_classification(result, classified_areas = areas)
```

---

add_suitability_ci *Add bootstrap confidence intervals to suitability scores*

Description

Re-scores each row of a survey dataframe `n_boot` times, each time adding Gaussian noise to every numeric variable proportional to its measurement uncertainty, and returns the 5th and 95th percentile suitability scores as `suit_ci_lower` and `suit_ci_upper` columns (90% interval).

Measurement uncertainty is specified via the `uncertainty` argument: a named list where each name matches a variable name in `tolerances$scored_factors` and the value is the 1-sigma absolute error (same units as the variable). Any variable not listed defaults to 2% of its observed value.

Usage

```
add_suitability_ci(
  result,
  tolerances,
  n_boot = 200L,
  uncertainty = NULL,
```

```
  seed = 42L,
  verbose = FALSE
)
```

Arguments

result	Dataframe returned by predict_oyster().
tolerances	Species tolerance list from get_species_tolerances().
n_boot	Integer. Number of bootstrap resamples (default 200).
uncertainty	Named numeric vector. 1-sigma measurement errors by variable name (e.g. c(temperature = 0.2, salinity = 0.5)).
seed	Integer or NULL. Random seed for reproducibility (default 42).
verbose	Logical. Print progress (default FALSE).

Value

The input dataframe with additional columns: suit_ci_lower, suit_ci_upper, suit_ci_width.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
tol    <- get_species_tolerances("ostrea_edulis")
result <- add_suitability_ci(result, tol,
  uncertainty = c(temperature = 0.3, salinity = 0.5))
# Columns suit_ci_lower, suit_ci_upper, suit_ci_width now present
head(result[, c("lat", "lon", "suitability", "suit_ci_lower", "suit_ci_upper")])
```

---

analyse_connectivity *Analyse spatial connectivity of suitable habitat cells*

Description

Builds a spatial graph of cells above a suitability threshold and identifies connected components – contiguous patches of suitable habitat. Isolated cells or small patches are flagged because they are at higher risk of:

- **Recruitment failure** – larvae from isolated populations may be flushed away before settling back in the patch.
- **Genetic bottlenecks** – small isolated populations have lower allelic diversity and reduced adaptive capacity.
- **Local extinction without recolonisation** – if a patch is lost, there are no connected source populations to recolonise.

Two cells are considered connected if they are within gap_m metres of each other (default 500 m – approximately the scale of larval dispersal during a tidal cycle at moderate current speeds). Connectivity is computed using a fast union-find (disjoint set) algorithm – no external graph packages required.

Usage

```
analyse_connectivity(
  result,
  min_suitability = 0.5,
  gap_m = 500,
  min_patch_cells = 3L,
  verbose = TRUE
)
```

Arguments

result	Dataframe from predict_oyster() with lat, lon, suitability, and suitability_class columns.
min_suitability	Numeric. Minimum suitability score for a cell to be included in the connectivity analysis (default 0.5, i.e. Moderate+).
gap_m	Numeric. Maximum distance in metres between two suitable cells for them to be considered connected (default 500 m).
min_patch_cells	Integer. Patches smaller than this are flagged as isolated (default 3 cells).
verbose	Logical. Print connectivity summary (default TRUE).

Value

Input dataframe with additional columns: patch_id (integer – unique patch identifier; NA for non-suitable cells), patch_size (number of cells in the patch), patch_area_km2 (approximate area based on cell density), connectivity_class ("isolated", "small", "moderate", "large"), is_hub (logical – TRUE for the highest-scoring cell in each patch, useful as candidate introduction points).

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
result <- analyse_connectivity(result, gap_m = 500)
table(result$connectivity_class)
```

---

area_summary *Fine-scale habitat area summary for restoration and science*

Description

Converts the point-based output of predict_oyster() into estimated areas per suitability class at sub-hectare resolution. Designed for restoration ecology and scientific reporting where the difference between 200 m2 and 800 m2 is operationally significant.

The function:

1. Estimates survey point spacing from the median nearest-neighbour distance.
2. Computes a cell area in m2 for each point (Voronoi approximation: each point is assumed to represent one cell of that area).
3. Aggregates by suitability class.
4. Optionally rasterises the result via terra::terra and runs connected-component labelling to identify and size contiguous habitat patches – critical for assessing whether areas meet minimum viable restoration unit thresholds.

Usage

```
area_summary(
  result,
  cell_size_m = NULL,
  viable_area_m2 = 100,
  patch_analysis = TRUE,
 classes = c("High", "Moderate", "Low", "Very Low", "Excluded", "Insufficient data"),
  verbose = TRUE
)
```

Arguments

result	A dataframe returned by predict_oyster(), containing at minimum columns lat, lon, suitability_class, suitability, and excluded.
cell_size_m	Numeric or NULL. Survey point spacing in metres. If NULL (default), estimated automatically from the median nearest- neighbour distance of the survey points (capped at 500 m). Provide an explicit value if you know your survey grid resolution (e.g. 5 for a 5 m AUV grid, 25 for a 25 m ADCP trackline spacing).
viable_area_m2	Numeric. Minimum viable area threshold in m2 for a contiguous patch to be flagged as restoration-relevant (default 100, = 0.01 ha, aligned with OSPAR oyster reef restoration guidelines). Patches below this threshold are reported but flagged viable = FALSE.
patch_analysis	Logical. If TRUE (default), runs contiguous patch analysis using terra::terra. Patches are identified within "High" and "Moderate" suitability zones. Set to FALSE to skip (faster for large datasets).
classes	Character vector. Which suitability classes to include in the main summary. Default: all six classes, including "Insufficient data" – locations that evidence weighting declined to classify. Those count toward the surveyed area but never toward the suitable area, and are excluded from patch analysis.
verbose	Logical. Print the summary table to the console (default TRUE).

Value

A named list with three elements:

class_summary Dataframe – one row per suitability class with columns: class, n_points, area_m2, area_ha, pct_total_area, pct_suitable_area (High + Moderate = 100%), mean_suitability, median_suitability.

`total` Named numeric vector: `surveyed_area_m2`, `surveyed_area_ha`, `suitable_area_m2`, `suitable_area_ha`, `pct_suitable`, `cell_size_m` (estimated or supplied), `n_points_total`.

`patches` Dataframe of contiguous habitat patches (High + Moderate) sorted by area descending. Columns: `patch_id`, `class_dominant`, `n_cells`, `area_m2`, `area_ha`, `mean_suitability`, `viable`. `NULL` if `patch_analysis = FALSE`.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

# Automatic cell size estimation
s <- area_summary(result)

# Access the patch table and class summary
s$patches
s$class_summary
s$total["suitable_area_m2"]


# Known grid spacing, 500 m2 minimum viable patch (slower patch analysis)
s2 <- area_summary(result, cell_size_m = 25, viable_area_m2 = 500)
```

---

assess_gear_feasibility

Assess gear deployment feasibility at survey locations

Description

Evaluates which aquaculture and restoration gear types are physically viable at each survey location based on depth, current velocity, substrate, and (optionally) wave height. Returns a feasibility score and binary flag for each gear type.

This function is part of the **planning module** aimed at commercial operators. For pure restoration work, the `restoration_reef` gear type is always included and is the most relevant output.

Usage

```
assess_gear_feasibility(
  result,
  gear_types = names(.gear_specs),
  depth_col = "depth",
  current_col = "current_velocity",
  substrate_col = "substrate_hardness_class",
  wave_col = NULL,
  verbose = TRUE
)
```

Arguments

result	Dataframe from predict_oyster() with environmental columns.
gear_types	Character vector of gear types to assess. Options: "longline", "bottom_culture", "intertidal_rack", "raft", "restoration_reef". Default: all five.
depth_col	Character. Column name for depth below CD in metres (default "depth"; positive = subtidal).
current_col	Character. Column name for current velocity in m/s (default "current_velocity").
substrate_col	Character. Column for substrate class (default "substrate_hardness_class"). Can be the output of classify_substrate_from_backscatter().
wave_col	Character or NULL. Column for significant wave height Hs in metres (default NULL – wave check skipped if absent).
verbose	Logical. Print feasibility summary (default TRUE).

Value

Input dataframe with added columns per gear type: gear_<type>_feasible (logical), gear_<type>_score [0-1], plus best_gear (most feasible option) and n_gear_options (count).

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
result <- assess_gear_feasibility(result)
# Restoration reef sites (no farming licence needed)
reef_sites <- subset(result, gear_restoration_reef_feasible)
nrow(reef_sites)
```

---

as_lineage_distances *Validate a genetic distance matrix between lineages*

Description

Accepts a matrix, a dist object, a data frame (first column lineage ids, or row names), or the path to a CSV in that layout, and returns a checked symmetric numeric matrix with lineage ids as dimnames. Fst, p-distance, patristic distance from a tree (ape::cophenetic()) – anything that is a non-negative pairwise dissimilarity works; the scale does not matter.

Usage

```
as_lineage_distances(x)
```

Arguments

x	Matrix, dist, data frame or CSV path.

Value

A numeric matrix with identical row and column names.

Examples

```
d <- matrix(c(0, .1, .4, .1, 0, .35, .4, .35, 0), 3,
            dimnames = list(c("A", "B", "C"), c("A", "B", "C")))
as_lineage_distances(d)
```

`auto_tidal_correct` *Automatically predict and apply tidal correction from harmonic constituents*

Description

Finds the nearest standard port to the survey area, checks whether it is within a configurable distance threshold, predicts tidal heights for every survey timestamp using embedded 5-constituent harmonic models (M2, S2, N2, K1, O1), and applies `correct_to_chart_datum()` automatically.

No external data or internet connection is required. Harmonic constituent data (amplitude and Greenwich phase lag) for 31 UK and European ports are embedded in the package.

Accuracy: 5-constituent predictions are typically accurate to +/-0.15–0.35 m at the standard port. Accuracy degrades with distance from the port and in areas with complex tidal dynamics (e.g. the Bristol Channel, inner fjords). For safety-critical work, use official UKHO EasyTide predictions and supply them manually via `correct_to_chart_datum()`.

Requirements: The dataframe must contain a datetime column. Values must be parseable by `as.POSIXct()` (e.g. `"2024-06-15 10:30:00"` or `"2024-06-15"`). Date-only strings default to noon UTC.

Usage

```
auto_tidal_correct(
  df,
  datetime_col = "date",
  depth_col = "depth",
  max_port_dist_km = 75,
  keep_raw = TRUE,
  verbose = TRUE
)
```

Arguments

`df`	A dataframe with at minimum a depth and a datetime column. Typically the output of `merge_sensor_data()`.
`datetime_col`	Character. Name of the datetime column (default `"date"`). Values should be UTC or treated as UTC.
`depth_col`	Character. Name of the depth column to correct (default `"depth"`).

`max_port_dist_km`
: Numeric. Maximum distance in km from the survey centroid to the nearest standard port. If the nearest port is further than this, no correction is applied and a warning is issued (default 75 km).

`keep_raw`
: Logical. Retain original depths in a `depth_raw_m` column (default `TRUE`).

`verbose`
: Logical. Print port selection, distance, and prediction summary (default `TRUE`).

Value

The input dataframe with depths corrected to Chart Datum, plus a `tidal_height_pred_m` column containing the predicted tidal height used for each row.

See Also

`correct_to_chart_datum()` for manual correction, `tidal_port_info()` for port datum reference data.

Examples

```
## Not run:
# Minimal survey dataframe near Oban, Scotland
df <- data.frame(
  lat   = c(56.41, 56.42, 56.43),
  lon   = c(-5.47, -5.47, -5.46),
  depth = c(5.5, 12.0, 8.3),
  date  = as.POSIXct("2024-06-15 10:30:00", tz = "UTC")
)
# Automatic correction: finds Oban as nearest port, predicts tidal heights
df_corrected <- auto_tidal_correct(df, datetime_col = "date")
df_corrected[, c("depth", "tidal_height_pred_m")]

# Tighten the distance threshold (only trust ports within 40 km)
df_corrected2 <- auto_tidal_correct(df, datetime_col = "date",
                                    max_port_dist_km = 40)

## End(Not run)
```

---

calculate_aragonite *Calculate aragonite saturation state from seawater carbonate chemistry*

Description

Computes Omega_aragonite (aragonite saturation state) from pH, total alkalinity, temperature, and salinity using in-house equilibrium constant expressions. No external carbonate chemistry packages are required.

Omega_aragonite is the ratio of the ion product of calcium and carbonate ions to the stoichiometric solubility product of aragonite:

$$\Omega_{arag} = \frac{[Ca^{2+}][CO_3^{2-}]}{K_{sp,arag}}$$

Values:

- **> 1**: supersaturated – shells grow/maintain normally.
- **~1**: marginal – dissolution risk, especially for larvae and juveniles.
- **< 1**: undersaturated – net shell dissolution occurs.

Usage

```
calculate_aragonite(temperature, salinity, pH, alkalinity)
```

Arguments

`temperature`	Numeric vector. Seawater temperature in degrees Celsius.
`salinity`	Numeric vector. Practical salinity (PSU).
`pH`	Numeric vector. Seawater pH on the **Total Scale** (recommended). If your sensor outputs NBS/free scale pH, convert before calling this function (typical NBS-to-Total correction is approximately -0.11 at 15 degC, 35 PSU; see Dickson et al. 2007 Guide to Best Practices).
`alkalinity`	Numeric vector. Total alkalinity in **micromol per kg seawater** (umol/kg-sw). Typical open-ocean value ~2300 umol/kg; typical shelf sea value 2100-2400 umol/kg. If your data are in umol/L, divide by seawater density (~1.025 kg/L) before passing in.

Value

Numeric vector of Omega_aragonite values (dimensionless). Returns `NA` where any input is `NA` or out of the valid temperature/ salinity range for the equilibrium constants (2-35 degC, 19-43 PSU). A warning is issued if values outside this range are detected.

Alkalinity assumptions

The calculation uses the carbonate alkalinity approximation:

$$TA_{carb} = TA - [B(OH)_4^-] - [OH^-] + [H^+]$$

Minor alkalinity contributions (phosphate, silicate, fluoride) are omitted. This is appropriate for typical coastal seawater. If your water has high nutrient loading (>5 umol/kg phosphate), the result may be slightly overestimated.

References

Lueker T.J., Dickson A.G. & Keeling C.D. (2000). Ocean pCO2 calculated from dissolved inorganic carbon, alkalinity, and equations for K1 and K2: validation based on laboratory measurements of CO2 in gas and seawater at equilibrium. Geochim Cosmochim Acta 64(4):575-590.

Dickson A.G. (1990). Thermodynamics of the dissociation of boric acid in synthetic seawater. Deep-Sea Res 37(5):755-766.

Millero F.J. (1995). Thermodynamics of the carbon dioxide system in the oceans. Geochim Cosmochim Acta 59(4):661-677.

Mucci A. (1983). The solubility of calcite and aragonite in seawater at various salinities, temperatures, and one atmosphere total pressure. Am J Sci 283(7):780-799.

See Also

`predict_oyster()` – pass omega_aragonite as a column in your survey data for automatic inclusion in suitability scoring.

Examples

```
# Typical shelf sea conditions -- should return Omega ~2.5-3.0
calculate_aragonite(
  temperature = 12,
  salinity    = 35,
  pH          = 8.1,
  alkalinity  = 2300
)

# Vectorised -- add omega_aragonite to a minimal survey dataframe
df <- data.frame(
  temperature = c(10, 12, 15),
  salinity    = c(34, 35, 33),
  ph          = c(8.05, 8.10, 8.15),
  alkalinity  = c(2280, 2300, 2310)
)
df$omega_aragonite <- calculate_aragonite(
  temperature = df$temperature,
  salinity    = df$salinity,
  pH          = df$ph,
  alkalinity  = df$alkalinity
)
df[, c("ph", "omega_aragonite")]
```

---

check_exclusions *Check exclusion criteria for all rows in a dataset*

Description

Applies hard-stop exclusion criteria for a given species tolerance profile. Any location that violates one or more exclusion rules receives a flag and is removed from the suitability scoring pipeline.

Exclusion factors checked (where data columns are present):

- General temperature range
- Season-specific temperature floors/ceilings (requires `season` column)
- Salinity (with temperature-dependent minimum)
- Dissolved oxygen (hypoxia threshold)

Usage

```
check_exclusions(df, tolerances)
```

Arguments

`df`	A dataframe. Must contain at minimum a subset of the standard column names (see **Column Naming** below). Any exclusion factor for which no column is present is silently skipped with a warning.
`tolerances`	A species tolerance list as returned by `get_species_tolerances()`.

Value

The input dataframe with two additional columns:

- `excluded`: logical – `TRUE` if the location fails any exclusion criterion.
- `exclusion_reason`: character – semicolon-separated list of failed criteria, or `NA` if not excluded.

Column Naming

The function recognises these column names (case-insensitive):

Variable	Expected column name(s)
Temperature	`temperature`, `temp`
Salinity	`salinity`, `sal`
Dissolved oxygen	`dissolved_oxygen`, `do`, `do_mgl`
Season	`season` (auto-added by `add_season_column()`)

Examples

```
tol <- get_species_tolerances("ostrea_edulis")
df <- data.frame(
  lat = 51.5, lon = -2.5,
  temperature = 8, salinity = 28, dissolved_oxygen = 7
)
check_exclusions(df, tol)
```

`classify_substrate_from_backscatter` *Classify seabed substrate hardness from near-seabed ADCP backscatter*

Description

Uses the intensity of acoustic backscatter from the near-seabed bin(s) to classify substrate type. Hard substrates (gravel, rock, shell hash) scatter more sound energy than soft substrates (mud, fine sand). The function converts raw backscatter (dB re 1 m^-1 or instrument counts) to a categorical substrate class and a continuous hardness index [0, 1].

Classification thresholds (default, adjustable):

Class	Hardness index	Typical substrate
Hard	0.8 – 1.0	Rock, cobble, shell hash
Mixed	0.5 – 0.8	Gravel/sand mix, maerl beds
Soft	0.2 – 0.5	Sand, sandy mud
Very Soft	0.0 – 0.2	Mud, fine silt

The hardness index is a min-max normalisation of the mean near-seabed backscatter across the bottom `n_bottom_bins` cells. If the column contains calibrated volume backscatter (Sv in dB), set `is_sv = TRUE` and a sign-flip is applied (less negative Sv = stronger return = harder).

Usage

```
classify_substrate_from_backscatter(
  df,
  backscatter_col,
  is_sv = FALSE,
  bs_min = NULL,
  bs_max = NULL,
  hard_thresh = 0.8,
  mixed_thresh = 0.5,
  soft_thresh = 0.2,
  output_col = "substrate_hardness",
  verbose = TRUE
)
```

Arguments

`df`	Dataframe with at least one column of seabed backscatter values.
`backscatter_col`	Character. Name of the backscatter column.
`is_sv`	Logical. If TRUE, treats values as Sv (dB, typically negative); hardness is derived from abs(Sv) after sign convention correction. Default FALSE (raw instrument counts, higher = harder).

bs_min	Numeric. Backscatter value mapped to hardness = 0 (softest). Defaults to 5th percentile of observed data.
bs_max	Numeric. Backscatter value mapped to hardness = 1 (hardest). Defaults to 95th percentile of observed data.
hard_thresh	Numeric. Hardness index above which substrate is "Hard" (default 0.8).
mixed_thresh	Numeric. Hardness index above which substrate is "Mixed" (default 0.5).
soft_thresh	Numeric. Hardness index above which substrate is "Soft" (default 0.2); below is "Very Soft".
output_col	Character. Base name for output columns. Two columns are added: <output_col>_index (numeric, 0-1) and <output_col>_class (character). Default "substrate_hardness".
verbose	Logical. Print classification summary (default TRUE).

Value

Input dataframe with substrate_hardness_index and substrate_hardness_class columns appended. These map directly to the substrate_hardness scored variable in the tolerance spec.

Examples

```
# Minimal dataframe with a mean-volume backscatter column (dB re 1 m-1)
df <- data.frame(
  lat           = c(51.5, 51.6, 51.7),
  lon           = c(-3.2, -3.2, -3.1),
  sv_backscatter = c(-55.0, -68.0, -75.0)
)
df <- classify_substrate_from_backscatter(df, backscatter_col = "sv_backscatter",
                                          is_sv = TRUE, verbose = FALSE)
df[, c("substrate_hardness_index", "substrate_hardness_class")]
```

---

compare_species *Compare suitability across multiple oyster species*

Description

Runs predict_oyster() for every specified species against the same survey dataset and returns a combined comparison table – one row per survey location, one suitability column per species. An optional summary prints which species is best-suited to each location.

This is the tool to use when you haven't yet decided which species to stock, or when you want to identify areas where multiple species overlap in suitability (robust site selection).

Usage

```
compare_species(
  data,
  species = NULL,
  output_dir = NULL,
  min_data_quality = "low",
  verbose = TRUE
)
```

Arguments

data	A dataframe or CSV path accepted by predict_oyster().
species	Character vector of species keys to compare. Defaults to all species currently in the oystermapR database (see list_species()). Passing NULL uses all available species.
output_dir	Character or NULL. If a directory path is supplied, individual GeoTIFF heatmaps are exported for each species into that folder. Default NULL (no rasters written).
min_data_quality	Character. Minimum data quality tier to include: "low", "medium", or "high" (default "low" – include all species). Set to "high" to restrict to well-characterised species only.
verbose	Logical. Print a comparison summary per species (default TRUE).

Value

A dataframe with all original survey columns plus:

- suit_<species_key>: suitability score [0, 1] per species
- class_<species_key>: suitability class per species
- best_species: key of the species with the highest suitability at each location (NA if all excluded)
- best_suitability: the highest suitability score across all species
- n_species_high: number of species scoring "High" at each location (useful for identifying robust multi-species sites)

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
comparison <- compare_species(
  sample_csv,
  species = c("ostrea_edulis", "magallana_gigas"),
  verbose = FALSE
)
# Locations suitable for both species
both_high <- subset(comparison, n_species_high >= 2)
nrow(both_high)
```

`compare_surveys` *Compare suitability scores across multiple surveys or monitoring years*

Description

Takes a named list of `predict_oyster()` result dataframes and produces a single comparative summary suitable for trend monitoring, multi-site selection, or regulatory reporting. Common spatial cells are matched across surveys and a change analysis is run between consecutive surveys if ordered chronologically.

The function returns a list with three elements:

- `summary` – one row per survey with mean suitability, class breakdown, and data completeness stats.
- `spatial` – all surveys joined on shared lat/lon cells, with suitability columns per survey and `trend_slope` (linear regression of suitability over survey index for each cell).
- `change` – pairwise change table (only if surveys are in order; each consecutive pair shows mean score difference and fraction of cells that improved, degraded, or were stable).

Usage

```
compare_surveys(
  surveys,
  cell_deg = 0.001,
  stable_threshold = 0.05,
  verbose = TRUE
)
```

Arguments

`surveys`	Named list of dataframes from `predict_oyster()`. Names should be meaningful labels (e.g. survey year or site name).
`cell_deg`	Numeric. Spatial rounding tolerance in decimal degrees for matching cells across surveys (default 0.001 approx. 111 m).
`stable_threshold`	Numeric. Absolute suitability change below which a cell is considered "stable" rather than improved/degraded (default 0.05).
`verbose`	Logical. Print comparison summary (default TRUE).

Value

Named list: `summary` (dataframe), `spatial` (dataframe), `change` (dataframe or NULL if only one survey).

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
r2022 <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
r2023 <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

comp <- compare_surveys(
  surveys = list("2022" = r2022, "2023" = r2023)
)

comp$summary   # mean scores by survey
comp$change    # change between surveys
```

`composite_seasonal` *Combine multi-season survey results into a composite suitability score*

Description

Oysters are permanent residents; a location is only truly suitable if conditions are adequate year-round. `composite_seasonal()` takes a named list of `predict_oyster()` result dataframes (one per season or survey visit) and produces a single composite dataframe with one row per spatial cell, summarising suitability across all input surveys.

Composite methods:

- `"min"` (default) – most conservative; a location must be suitable in *every* season to score well. Best for regulatory site selection.
- `"mean"` – average across seasons. Good for monitoring trend summaries.
- `"prob"` – fraction of seasons with suitability >= `prob_threshold`. Useful when surveys are irregularly spaced or some seasons are missing.

Spatial matching uses a grid cell tolerance of `cell_deg` decimal degrees (default 0.001 approx. 111 m). Cells not present in all seasons are flagged in `n_seasons_present`.

Usage

```
composite_seasonal(
  surveys,
  method = c("min", "mean", "prob"),
  prob_threshold = 0.5,
  cell_deg = 0.001,
  verbose = TRUE
)
```

Arguments

`surveys`	Named list of dataframes from `predict_oyster()`. Each must have `lat`, `lon`, and `suitability` columns. Names are used as season labels (e.g. `list(spring = r1, summer = r2, autumn = r3)`).

method	Character. One of "min", "mean", "prob" (default "min").
prob_threshold	Numeric [0,1]. Suitability threshold used when method = "prob" (default 0.5).
cell_deg	Numeric. Spatial tolerance in decimal degrees for matching cells across surveys (default 0.001).
verbose	Logical. Print summary (default TRUE).

Value

A dataframe with columns: lat, lon, suitability_composite, suitability_class, n_seasons_present, one suit_<season> column per input survey, and suit_range (max - min across seasons).

Examples

```
# Simulate three seasonal surveys using the bundled sample data
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
spring <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
summer <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
autumn <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

composite <- composite_seasonal(
  surveys = list(spring = spring, summer = summer, autumn = autumn),
  method  = "min"
)
head(composite[, c("lat", "lon", "suitability_composite", "suitability_class")])
```

---

confidence_score *Combine per-location confidence components*

Description

Per-location confidence **within the observed layers**, kept deliberately separate from coverage. Multiplying the two would drive every location below any usable class threshold on a low-coverage survey – technically honest, useless as a product. So coverage is reported once, loudly, at survey level, and this carries the location-to-location variation.

Usage

```
confidence_score(components, weights)
```

Arguments

components	A named list of numeric vectors on [0, 1], each a confidence component – for example how a class was derived (a fitted physical model, a cluster assignment, an ordinal bin) or whether a location had an independent second observation.
weights	Numeric vector of component weights, recycled by name if named. Normalised to sum to 1.

Value

Numeric vector of combined confidence.

Examples

```
route  <- c(1.0, 0.6, 0.3, 1.0)
passes <- pmin(pmax(c(3, 1, 1, 2) / 2, 0.4), 1)
confidence_score(list(route = route, passes = passes), c(0.65, 0.35))
```

---

correct_to_chart_datum

Correct survey depths to Chart Datum (LAT)

Description

Converts water depths recorded during a survey to depths below Chart Datum (approximately Lowest Astronomical Tide, LAT) by adding the tidal height above Chart Datum at the time of survey. This removes bias introduced by surveying at different states of the tide.

Formula: `depth_CD = depth_surveyed + tidal_height_m`

Finding your tidal height:

- UK: UKHO EasyTide (easytide.ukho.gov.uk) – free 7-day predictions for any standard port. Download the predicted heights for your survey period.
- Ireland: Marine Institute tide gauges (data.marine.ie)
- France/EU: SHOM (shom.fr) or Copernicus Marine (marine.copernicus.eu)
- Any port: apps such as PocketTides, Tides Near Me, or TidePod give height-above-CD predictions for named ports.

For a typical loch or sheltered bay survey, reading the tidal height from a tide table for the nearest standard port at the mid-point of the survey is sufficient. For multi-day surveys or those spanning a full tidal cycle, supply per-row heights matched to the survey timestamp.

Usage

```
correct_to_chart_datum(
  df,
  tidal_height_m = NULL,
  depth_col = "depth",
  datetime_col = NULL,
  keep_raw = TRUE,
  verbose = TRUE
)
```

Arguments

df	A dataframe containing at minimum a depth column. Typically the output of merge_sensor_data() before passing to predict_oyster().
tidal_height_m	Numeric scalar or vector. Tidal height above Chart Datum at the time of survey, in **metres**. If a scalar, the same offset is applied to all rows. If a vector, must be the same length as nrow(df). Use NULL to skip correction (a warning will be issued).
depth_col	Character. Name of the depth column to correct (default "depth").
datetime_col	Character or NULL. Name of a datetime column in df. If supplied, it is used only for informational messages – tidal heights must still be supplied by the user. Default NULL.
keep_raw	Logical. If TRUE, retains the original depth values in a column named depth_raw_m alongside the corrected depth column (default TRUE).
verbose	Logical. Print correction summary (default TRUE).

Value

The input dataframe with the depth column corrected to Chart Datum. If keep_raw = TRUE, the original values are preserved as depth_raw_m.

Examples

```
# Minimal survey dataframe with depths measured below the water surface
df <- data.frame(
  lat   = c(56.41, 56.42, 56.43),
  lon   = c(-5.47, -5.47, -5.46),
  depth = c(5.5, 12.0, 8.3)
)
# Survey conducted near high water at Oban; tidal height ~3.1 m above CD
df_corrected <- correct_to_chart_datum(df, tidal_height_m = 3.1)
df_corrected[, c("depth", "depth_raw_m")]

# Per-row heights (e.g. from a matched tide gauge series)
df_corrected2 <- correct_to_chart_datum(df, tidal_height_m = c(3.1, 3.0, 2.9))
```

---

detect_season *Detect meteorological season from date and latitude*

Description

Returns the meteorological season ("winter", "spring", "summer", "autumn") for a given date and latitude. Hemisphere is inferred from the sign of latitude (positive = Northern Hemisphere, negative = Southern Hemisphere).

Usage

```
detect_season(date, lat)
```

Arguments

date	A Date or POSIXct object, or a character string coercible to Date (e.g., "2024-03-15").
lat	Numeric. Latitude in decimal degrees. Positive = N hemisphere.

Value

A character string: one of "winter", "spring", "summer", "autumn".

Examples

```
detect_season("2024-01-15", lat = 51.5)   # "winter" (UK)
detect_season("2024-07-15", lat = 51.5)   # "summer"
detect_season("2024-01-15", lat = -33.9)  # "summer" (Sydney)
```

---

estimate_chlorophyll_from_backscatter

Estimate chlorophyll-a concentration from ADCP acoustic backscatter

---

Description

Uses the empirical log-linear relationship between volume backscatter strength (Sv) and chlorophyll-a concentration to add an estimated chlorophyll_a column to a survey dataframe. Intended for surveys where a fluorometer was not deployed and the ADCP backscatter intensity is available.

The conversion is: Chl_a (ugg/L) = 10 ^ (a * Sv + b)

Default a and b constants are derived from Deines (1999) for three common ADCP frequencies (300, 600, 1200 kHz). For best results, calibrate against concurrent water samples from your survey using the calibration argument.

Usage

```
estimate_chlorophyll_from_backscatter(
  df,
  backscatter_col,
  frequency_khz = 600,
  calibration = NULL,
  min_chl = 0.05,
  max_chl = 50,
  keep_raw = TRUE,
  verbose = TRUE
)
```

Arguments

df	Dataframe containing a backscatter column and optional temperature and depth columns.
backscatter_col	Character. Name of the column containing volume backscatter strength values (Sv in dB re 1 m^{-1}). For Nortek Signature outputs, this is typically the mean amplitude across clean bins, available as amp_mean_dB after read_nortek_adcp().
frequency_khz	Numeric. ADCP operating frequency in kHz. Used to select default calibration constants if calibration is not supplied. Common values: 300, 600, 1200. For other frequencies, supply calibration directly.
calibration	Numeric vector of length 2: c(a, b) for the log-linear model log10(Chl_a) = a * Sv + b. Supply to override the built-in frequency-based defaults. Derive from concurrent water samples: fit log10(observed_chl) ~ backscatter and extract intercept (b) and slope (a).
min_chl	Numeric. Floor for estimated chlorophyll-a in ugg/L (default 0.05). Prevents physically implausible negative estimates.
max_chl	Numeric. Ceiling for estimated chlorophyll-a in ugg/L (default 50). Values above this are likely turbidity artefacts (non-algal particles dominating backscatter).
keep_raw	Logical. If TRUE, preserves the input backscatter column (default TRUE).
verbose	Logical. Print conversion summary and warnings (default TRUE).

Value

The input dataframe with an additional chlorophyll_a column (ugg/L). If a chlorophyll_a column already exists, a warning is issued and the existing column is overwritten.

Improving accuracy with water samples

Collect 5–15 grab/Niskin water samples distributed across your survey area and depth range, then:

```
# Fit calibration from samples
fit  <- lm(log10(sample_chl) ~ backscatter, data = samples_df)
a    <- coef(fit)[["backscatter"]]
b    <- coef(fit)[["(Intercept)"]]
df   <- estimate_chlorophyll_from_backscatter(
          df, "amp_mean_dB", frequency_khz = 600,
          calibration = c(a, b))
```

Examples

```
adcp_f <- system.file("extdata", "example_bay_adcp.csv", package = "oystermapR")
adcp   <- read_nortek_adcp(adcp_f, verbose = FALSE)
# estimate_chlorophyll_from_backscatter requires an amp_mean_dB column
# produced by read_nortek_adcp when the ADCP data contains amplitude bins
if ("amp_mean_dB" %in% names(adcp)) {
  adcp <- estimate_chlorophyll_from_backscatter(adcp, "amp_mean_dB",
```

```
    frequency_khz = 300)
}
```

`estimate_fetch` *Estimate effective fetch from a set of lat/lon points and a bearing*

Description

Simple great-circle ray-tracing to estimate how far open water extends from a point in a given direction, up to `max_fetch_km`. This is a lightweight GIS-free approximation – for precise fetch, use QGIS or the `fetchR` R package.

Usage

```
estimate_fetch(
  lat,
  lon,
  bearing_deg,
  land_polygons = NULL,
  max_fetch_km = 200,
  step_km = 1
)
```

Arguments

`lat`	Numeric. Site latitude (decimal degrees).
`lon`	Numeric. Site longitude (decimal degrees).
`bearing_deg`	Numeric. Prevailing wind direction to check (0 = North).
`land_polygons`	SpatialPolygons or sf POLYGON object representing land. If NULL, returns max_fetch_km (fully open ocean assumed).
`max_fetch_km`	Numeric. Maximum fetch to search (default 200 km).
`step_km`	Numeric. Ray step size (default 1 km).

Value

Numeric. Estimated fetch in km.

`evidence_coverage` *Fraction of a species' evidence base observed by a survey*

Description

Coverage is the share of a species' **total prior weight** that a survey actually observed, where prior weight is `1 / rank` summed over every variable in the tolerance specification.

It is deliberately weight-based rather than count-based. The `data_completeness` column produced by `score_locations()` counts variables (`n_scored / n_defined`) and so treats a rank-1 variable and a rank-6 variable as equal losses. Coverage does not: missing the single most important variable costs six times what missing the least important one does.

Variables passed in `assigned` do **not** count toward coverage. Coverage measures how much of the *species'* evidence base was observed, and inventing a new layer cannot increase that. A tool that let you raise your own coverage by adding layers would be measuring its own enthusiasm.

Usage

```
evidence_coverage(tolerances, observed, attempted_failed = NULL)
```

Arguments

`tolerances`	A species tolerance list (see `get_species_tolerances()`).
`observed`	Character vector of tolerance-spec variable names the survey observed. Names not present in the spec are ignored with a warning.
`attempted_failed`	Optional character vector of variables that were instrumented but returned unusable data. Reported separately, and used to compute `coverage_if_recovered` – the coverage the survey would have had if that payload had worked. Quantifying the cost of an instrument failure in units of the species' evidence base is usually more informative than reporting the failure alone.

Value

An object of class `evidence_coverage`: a list with `coverage`, `prior_observed`, `prior_total`, `observed`, `missing`, `attempted_failed`, `coverage_if_recovered` and a `ledger` data frame giving rank, prior and status for every variable in the spec.

Examples

```
## Not run:
tol <- get_species_tolerances("ostrea_edulis")
evidence_coverage(
  tol,
  observed         = c("depth", "slope", "substrate_hardness", "roughness"),
  attempted_failed = c("current_velocity", "shear_stress")
)

## End(Not run)
```

`evidence_suitability` *Evidence-weighted suitability*

Description

Re-weights scores already produced by `score_locations()` using `prior * reliability * spread` instead of `1 / rank`, and attaches the coverage of the species' evidence base to the result.

The split of labour is deliberate: the existing scoring curves supply the per-variable scores, and this supplies only the weights. Curve shapes are where site-specific judgement legitimately lives and are unchanged.

Usage

```
evidence_suitability(
  df,
  tolerances,
  reliability,
  confidence = NULL,
  confidence_floor = 0.5,
  assigned = NULL,
  aliases = NULL,
  extra_scores = NULL,
  observed = NULL,
  attempted_failed = NULL
)
```

Arguments

`df`	A data frame already through `score_locations()`, carrying `score_<variable>` columns.
`tolerances`	A species tolerance list from `get_species_tolerances()`.
`reliability`	Named numeric vector of measured reliability per layer.
`confidence`	Optional numeric vector of per-location confidence, e.g. from `confidence_score()`. Locations below `confidence_floor` are classified `"Insufficient data"` rather than given a class – an uncertain location should not be handed a label it cannot support. The continuous score is still returned for every location.
`confidence_floor`	Numeric. Default 0.5.
`assigned, aliases`	Passed to `evidence_weights()`.
`extra_scores`	Optional data frame of additional score columns on [0, 1] not produced by the standard curves (layers declared in `assigned`, typically). Must have `nrow(df)` rows.

observed — Optional character vector of tolerance-spec variables the survey observed, for coverage. Defaults to the published layers found in df, which is right only if every observed variable was also scorable – pass it explicitly when a variable was measured but could not be scored.

attempted_failed — Passed to evidence_coverage().

Value

df with added columns suitability_evidence, suitability_class_evidence and, if confidence was supplied, confidence. The weight table and coverage are attached as attributes "evidence_weights" and "evidence_coverage" and are what belongs in a methods section.

---

evidence_weights *Evidence-weighted variable weights*

Description

Computes w = prior * reliability * spread, normalised to sum to 1, and returns the derivation as a table rather than only the weights. Every factor traces to either the published tolerance profile (the prior) or a measurement made on the survey itself (reliability, spread), so the weight table is auditable line by line.

A layer with no measured reliability receives **no weight**. That is the rule that keeps the method honest: nothing enters the score on the strength of an argument alone, however good the argument. It is also why every candidate layer can be offered at once – the ones that do not reproduce demote themselves, so inclusion costs nothing.

Usage

```
evidence_weights(
  scores,
  reliability,
  tolerances,
  assigned = NULL,
  aliases = NULL
)
```

Arguments

scores — A data frame of per-location scores on [0, 1], one column per layer, or a named list of numeric vectors. Column names are layer names. score_ prefixes as produced by score_locations() are stripped.

reliability — Named numeric vector of reliability per layer, from reliability_from_replicates() or reliability_from_perturbation(). Layers absent from this vector, or carrying NA, get zero weight.

tolerances — A species tolerance list from get_species_tolerances(), used for the published ranks.

`assigned` Optional named list for layers with **no counterpart** in the published profile, each element a list with `rank` and `rationale`. Assigning a rank is a judgement call, and the only honest way to include such a layer is to make that call explicit rather than smuggle it in under a neighbouring variable's prior. Assigned layers are flagged as such in the weight table and excluded from coverage.

`aliases` Optional named character vector mapping layer names in `scores` to tolerance-spec variable names, e.g. `c(hardness = "substrate_hardness")`.

Details

`spread` is the population standard deviation of the layer's **score**, not of its raw values. A variable whose every observation lands in the optimal band scores 1.0 everywhere: it has spread 0 and takes weight 0, however important the species finds it and however precisely it was measured. This is the intended behaviour – such a variable carries no information about *where within this survey* conditions differ – and it is why the method reports coverage separately. A zero weight means "did not discriminate here", never "does not matter".

Value

An object of class `evidence_weights`: a list with `weights` (named numeric) and `table` (data frame with layer, prior variable, provenance, rank, prior, reliability, spread, raw and normalised weight).

`export_contours` *Export optional contour lines as a GeoPackage for QGIS*

Description

Derives contour lines from the IDW suitability raster and saves them as a vector GeoPackage (`.gpkg`). Load alongside the heatmap in QGIS for the depth-contour look shown in BioBase outputs.

Usage

```
export_contours(tif_path, interval = 0.1, gpkg_path = NULL)
```

Arguments

`tif_path` Character. Path to the GeoTIFF produced by `export_geotiff()`.

`interval` Numeric. Contour interval in suitability units (default `0.1`, producing lines at 0.1, 0.2, ... 0.9).

`gpkg_path` Character. Output `.gpkg` path. Defaults to same directory as the `.tif` with `_contours.gpkg` suffix.

Value

The `.gpkg` file path (invisibly).

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
out_tif <- file.path(tempdir(), "oyster_heatmap.tif")
# A coarse grid keeps the example fast; use the default resolution for real work.
export_geotiff(result, out_tif, resolution = 0.001, contours = FALSE)
export_contours(out_tif)
```

`export_geotiff` *Export suitability scores as a smooth interpolated GeoTIFF heatmap for QGIS*

Description

Takes the scored point dataset from `predict_oyster()` and produces a **smooth, continuous raster heatmap** using Inverse Distance Weighting (IDW) interpolation – the same technique used by systems like Lowrance BioBase. The result looks like a fluid heat surface over your survey area rather than isolated point squares.

The output GeoTIFF contains seven bands:

1. **suitability** – IDW-interpolated score [0, 1]. Use this for the heatmap.
2. **excluded_mask** – 1 where a survey point was hard-excluded, 0 otherwise.
3. **n_observations** – how many survey points fell within each raster cell (data density diagnostic).
4. **dist_to_nearest_m** – distance in metres from each cell centre to the nearest survey point (use as a QGIS transparency mask to fade extrapolated areas).
5. **n_layers_scored** – number of environmental variables that contributed to the suitability score at each survey point (data completeness overlay).
6. **confidence** – per-location confidence [0, 1] under evidence weighting; `NA` throughout under rank weighting, which has no notion of it.
7. **insufficient_mask** – 1 where the model declined to classify the location (below `confidence_floor`), 0 otherwise. Those points are held out of band 1 rather than asserting a suitability into the surface, and `export_qml_style()` writes an overlay style for this band so they read as a class in QGIS instead of as a hole in the map.

Bands 3, 4, 6 and 7 are the ones that say how much to trust band 1. Band 1 will interpolate a confident-looking score across water nobody surveyed.

Usage

```
export_geotiff(
  df,
  path,
  resolution = 2e-04,
  idw_power = 2,
```

```
  idw_max_dist = NULL,
  buffer = 0.001,
  contours = TRUE,
  crs = "EPSG:4326",
  overwrite = TRUE
)
```

Arguments

df	A dataframe processed by predict_oyster(), containing columns lat, lon, suitability, excluded.
path	Character. Output .tif file path.
resolution	Numeric. Raster cell size in decimal degrees. Default 0.0002 approx. 22 m – matches the typical 4-decimal-place survey grid. For large estuary surveys increase to 0.001 (100 m); for very detailed harbour surveys decrease to 0.0001 (~11 m).
idw_power	Numeric. IDW distance decay exponent (default 2). Higher values make the surface honour nearby points more tightly and decay faster with distance – producing sharper gradients. Lower values (e.g. 1) give broader, smoother spread.
idw_max_dist	Numeric or NULL. Maximum search radius in degrees for IDW interpolation. If NULL (default), the radius is automatically set to 5x the median nearest-neighbour spacing between survey points – so the heatmap fills gaps between transect lines but does not bleed onto land or beyond the survey boundary. Override with an explicit value (e.g. 0.003 approx. 330 m) if needed.
buffer	Numeric. Padding in degrees added around the survey extent before building the raster grid (default 0.001 approx. 110 m). Keeps the heatmap from clipping at the outermost survey points without extending far beyond the survey boundary.
contours	Logical. If TRUE (default), automatically export contour lines as a GeoPackage alongside the GeoTIFF using export_contours().
crs	Character. CRS string (default "EPSG:4326" – WGS84). Use a projected CRS (e.g. "EPSG:32630") if your coordinates are in metres.
overwrite	Logical. Overwrite existing file (default TRUE).

Value

The output file path (invisibly). Also writes .qml QGIS style files alongside the .tif automatically.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
out_tif <- file.path(tempdir(), "oyster_heatmap.tif")
# A coarse grid keeps the example fast; use the default resolution for real work.
export_geotiff(result, out_tif, resolution = 0.001)
file.exists(out_tif)
```

export_qml_style *Export a QGIS colour style (.qml) matching the BioBase orange/red heatmap*

Description

Writes a QGIS Layer Style file (`.qml`) using a yellow -> orange -> red -> dark red colour ramp, closely matching the BioBase/Lowrance heatmap style. The file is automatically applied when its name matches the `.tif` – no manual styling needed in QGIS.

A companion `<name>_insufficient.qml` is written at the same time. It styles band 7 – the locations the model declined to classify under evidence weighting – as a flat grey overlay, transparent everywhere else. Load the raster a second time in QGIS and drop that style on it to show unclassified ground as its own class sitting over the heatmap, rather than leaving the viewer to read an absence as a low score.

Usage

```
export_qml_style(tif_path)
```

Arguments

`tif_path`	Character. Path to the `.tif` from `export_geotiff()`.

Value

The `.qml` file path (invisibly). The overlay style path is returned as the attribute `"overlay"`.

fetch_live_environmental_data *Fetch live environmental data for a survey extent*

Description

Convenience wrapper that fetches data from one or more live sources and returns a named list of dataframes that can be passed to the relevant scoring functions. Each source can be fetched independently or all at once.

This function requires internet access and registered API credentials where applicable (see `oystermapR_live_config()`). If a source fails, it is silently skipped and not included in the output - the function never aborts due to a single failed source.

Usage

```
fetch_live_environmental_data(
  survey,
  sources = "all",
  date_range = NULL,
  verbose = TRUE
)
```

Arguments

survey	Dataframe with lat and lon columns defining the survey extent. The bounding box of these coordinates is used as the query extent.
sources	Character vector. Which data sources to fetch. Options: "cmems", "ices_hab", "ices_vms", "emodnet_predators", "fsa_shellfish". Use "all" to attempt all sources.
date_range	Character vector of length 2: c("YYYY-MM-DD", "YYYY-MM-DD"). Date range for time-windowed queries (CMEMS, ICES HAB). Defaults to the last 5 years.
verbose	Logical. Print progress (default TRUE).

Value

Named list. Each successfully fetched source produces one element:

- $cmems: SST, salinity, chlorophyll grid
- $ices_hab: HAB event records within extent
- $ices_vms: Swept-area ratio per ICES c-square
- $emodnet_predators: Predator occurrence records
- $fsa_shellfish: Shellfish classification polygons / area descriptions

Examples

```
## Not run:
oystermapR_live_config(cmems_user = "u", cmems_password = "p")
live <- fetch_live_environmental_data(survey, sources = c("ices_hab", "ices_vms"))
result <- score_hab_risk(result, hab_data = live$ices_hab, species = "ostrea_edulis")
result <- score_anthropogenic_disturbance(result, trawling_data = live$ices_vms)

## End(Not run)
```

---

fit_lineages *Fit tolerance profiles for a set of related lineages*

Description

Fits one tolerance profile per lineage from occurrence records carrying a lineage identifier, using the same Bayesian update as update_species_tolerances(). Where the data show that tolerance tracks genetic relatedness, sparse lineages borrow a prior from their relatives, and lineages with no records at all get a profile predicted from genetic distance alone.

Usage

```
fit_lineages(
  records,
  distances,
  species,
  lineage_col = "lineage_id",
  presence_col = "presence",
  update_vars = NULL,
  pool = c("auto", "always", "never"),
  min_records = 20L,
  min_sparse = 8L,
  min_lineages = 6L,
  alpha = 0.05,
  n_perm = 999L,
  calibration = c("shared", "lineage"),
  prior_sd_fraction = 0.2,
  method = "map",
  seed = NULL,
  verbose = TRUE
)
```

Arguments

records	Data frame of occurrence records with lat, lon, a presence/absence column, the environmental columns to fit, and lineage_col.
distances	Pairwise genetic distance matrix between lineages (as_lineage_distances()). Lineages in the matrix with no records are allowed and receive borrowed profiles when pooling is on.
species	Species key, e.g. "ostrea_edulis".
lineage_col	Name of the lineage column.
presence_col	Name of the presence/absence column.
update_vars	Variables to fit; NULL fits every updatable variable present in records.
pool	"auto" (default), "always" (skip the signal test) or "never".
min_records	Records needed to fit a lineage alone.
min_sparse	Records needed to refit a lineage under a borrowed prior; below this it takes the prior itself.
min_lineages	Fitted lineages needed before any pooling.
alpha	Significance level for the signal test.
n_perm	Permutations for the signal test.
calibration	How suitability is turned into occurrence probability. "shared" (default) fits one logistic curve pooled over every lineage's records, each scored against its own profile, so lineages differ only in their tolerances. "lineage" fits one per lineage, which lets each lineage's sampling prevalence into the comparison: a lineage that happened to be sampled at good sites looks more widespread than one sampled at poor ones, and that is not a property of the populations.

prior_sd_fraction, method
: Passed to update_species_tolerances().

seed
: Optional integer for reproducible permutations.

verbose
: Print progress.

Value

An object of class lineage_fits: profiles (long table of every parameter with its mean, sd and source), tolerances (a scoring-ready tolerance object per lineage), calibration (per-lineage logistic intercept and gain), signal, hyper, pooled, pool_reason, distances, n_records. source is "own", "pooled", "borrowed" or "default".

How pooling is decided

1. Lineages with at least min_records records are fitted alone.
2. lineage_signal() tests whether their fitted parameters are more alike between close relatives. With pool = "auto" (default), pooling happens only if there are at least min_lineages fitted lineages **and** the signal is positive and significant. Otherwise every lineage is left on its own fit, and lineages too sparse to fit are returned with the species default, flagged as such.
3. When pooling, a Gaussian-process prior with covariance decaying with genetic distance is estimated from the fitted lineages. Each lineage's prior is what its relatives (never itself) predict. Lineages with at least min_sparse records are refitted under that prior; lineages with fewer take the prior as their profile (source = "borrowed"), with the wide uncertainty that implies.

Circularity

Lineage identifiers must come from genetics alone. If lineages are defined by where they were sampled, the inverse map recovers the sampling, not anything about the populations.

Limits

Borrowing works by interpolation. A lineage further out than any of its fitted relatives on the distance scale is predicted no better than the species default, because a distance-decay prior cannot extrapolate a trend; it is flagged borrowed and is a hypothesis. Below about six fitted lineages the signal test has little power and pooling is refused. The fitter itself is bounded: an optimum more than about three prior standard deviations from the species default cannot be reached.

See Also

lineage_occurrence(), rank_source_populations(), lineage_signal()

`generate_report` *Generate a formatted survey report from predict_oyster() results*

Description

Renders a structured assessment report from the output of `predict_oyster()`. The report includes an executive summary, suitability class breakdown, per-variable scoring table, most common limiting factors, top introduction sites, and a methodology section.

Output format is PDF by default (requires a LaTeX installation – see Note below). HTML is also supported and requires no additional system software.

Usage

```
generate_report(
  result,
  output = "oyster_report.html",
  title = "Oyster Suitability Assessment",
  author = "",
  species = "ostrea_edulis",
  top_n = 5L,
  validation = NULL,
  comparison = NULL,
  open = TRUE
)
```

Arguments

`result`	A dataframe returned by `predict_oyster()`.
`output`	Character. Output file path. Extension determines format: `.pdf` for PDF, `.html` for HTML. Default `"oyster_report.pdf"`.
`title`	Character. Report title. Default `"Oyster Suitability Assessment"`.
`author`	Character. Author name(s) to appear on the title page. Default `""` (omitted).
`species`	Character. Species key used in `predict_oyster()`. Default `"ostrea_edulis"`.
`top_n`	Integer. Number of top introduction sites to include in the report (default 5).
`validation`	Named list or `NULL`. Optional output from `validate_against_records()`. When supplied, adds a Validation section with ROC curve, AUC, TSS, and confusion-matrix metrics.
`comparison`	Dataframe or `NULL`. Optional output from `compare_species()`. When supplied, adds a Competition Adjustment section showing the *M. gigas* competitive pressure penalty on *O. edulis* scores.
`open`	Logical. Open the report in your default viewer after rendering (default `TRUE`).

Value

The output file path (invisibly).

Note

HTML reports (recommended) require `plotly` and `scales`: `install.packages(c("plotly","scales"))`. **PDF reports** additionally require a LaTeX installation; see the Note section below.

PDF output requires LaTeX. If you don't have LaTeX installed, either use `output = "report.html"` (no extra software needed), or install a minimal LaTeX distribution with:

```
install.packages("tinytex")
tinytex::install_tinytex()
```

Examples

```
## Not run:
result <- predict_oyster(survey, "ostrea_edulis",
                         output_geotiff = "kames_bay.tif")

# Standard HTML report (recommended -- Plotly charts, no LaTeX needed)
generate_report(result, "kames_bay_report.html",
                title  = "Kames Bay Oyster Habitat Assessment",
                author = "T. Tucker")

# With validation results embedded
val  <- validate_against_records(result, nbn_records)
generate_report(result, "kames_bay_validated.html",
                validation = val)

# With competition adjustment (when comparing species)
comp <- compare_species(survey,
          species = c("ostrea_edulis", "magallana_gigas"))
generate_report(result, "kames_bay_competition.html",
                comparison = comp)

# PDF report (requires LaTeX / tinytex)
generate_report(result, "kames_bay_report.pdf",
                title = "Kames Bay Oyster Habitat Assessment")

## End(Not run)
```

---

generate_summary_pdf *Generate a compact printable PDF summary of oystermapR results*

Description

Produces a self-contained multi-page A4 PDF directly from the output of `predict_oyster()` without requiring LaTeX, pandoc, or Rmd rendering. The PDF is designed for printing and contains four pages:

- **Page 1** – Title header, executive summary table, class breakdown bar
- **Page 2** – Suitability heatmap (survey points, continuous colour ramp)
- **Page 3** – Per-variable mean score chart (horizontal bar)
- **Page 4** – Score distribution histogram + class count bar chart

Usage

```
generate_summary_pdf(
  result,
  output = "oyster_summary.pdf",
  species = "ostrea_edulis",
  title = "Oyster Suitability Summary",
  author = "",
  open = TRUE,
  verbose = TRUE
)
```

Arguments

result	A dataframe returned by predict_oyster().
output	Character. Output .pdf file path. Default "oyster_summary.pdf".
species	Character. Species key used in predict_oyster(). Default "ostrea_edulis".
title	Character. Report title shown in the header. Default "Oyster Suitability Summary".
author	Character. Author name (optional, shown in header).
open	Logical. Open the PDF after writing (default TRUE).
verbose	Logical. Print progress messages (default TRUE).

Value

The output file path (invisibly).

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
out_pdf <- file.path(tempdir(), "kames_bay_summary.pdf")
generate_summary_pdf(result, out_pdf,
                     title  = "Kames Bay -- Spring Survey",
                     author = "T. Tucker",
                     open   = FALSE)
```

---

get_species_tolerances

Get species tolerance parameters

Description

Get species tolerance parameters

Usage

```
get_species_tolerances(species)
```

Arguments

species	Character. Species key (e.g. "ostrea_edulis"), latin name, or common name (partial, case-insensitive).

Value

Named list of tolerance parameters for the species.

Examples

```
tol <- get_species_tolerances("ostrea_edulis")
tol$exclusions$temperature
```

---

get_tolerance_posteriors

Retrieve current posterior tolerance parameters for a species

---

Description

Returns the tolerance parameter list currently held in the session cache, after any update_species_tolerances() calls. If no update has been run, returns the built-in prior parameters unchanged.

Usage

```
get_tolerance_posteriors(species, var = NULL)
```

Arguments

species	Character. Species key (e.g. "ostrea_edulis").
var	Character or NULL. If specified, returns parameters for that variable only; otherwise returns the full $scored list.

Value

Named list of tolerance parameters (same structure as get_species_tolerances()$scored).

Examples

```
# Returns NULL if no Bayesian update has been run this session
post <- get_tolerance_posteriors("ostrea_edulis")
post$temperature  # updated optimal_min, optimal_max with CIs
```

identify_resilient_sites *Identify climate-resilient locations*

Description

From a `project_suitability()` output, identifies locations that maintain at least `min_class` suitability across all projected scenarios. These are the most robust sites for long-term restoration or aquaculture investment.

Usage

```
identify_resilient_sites(projections, baseline, min_class = "Moderate")
```

Arguments

`projections`	Named list returned by `project_suitability()`.
`baseline`	Dataframe from `predict_oyster()` (the baseline result).
`min_class`	Character. Minimum suitability class required in all scenarios. One of `"High"`, `"Moderate"`, `"Low"` (default `"Moderate"`).

Value

Subset of `baseline` for locations that meet `min_class` in every scenario, with an added `n_scenarios_qualifying` column.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
tol    <- get_species_tolerances("ostrea_edulis")
proj   <- project_suitability(result, tol)
resilient <- identify_resilient_sites(proj, result, min_class = "Moderate")
nrow(resilient)
```

interpolate_survey *Interpolate survey measurements to a regular grid before scoring*

Description

Converts sparse point observations (CTD casts, ADCP profiles) to a regular grid by spatial interpolation. The interpolated grid is returned in the same dataframe format as the raw survey, and can be passed directly to `predict_oyster()`.

Method selection (applied per variable independently):

- **Ordinary kriging** is used when >= 50 non-NA observations are available for a variable, the `gstat` and `sp` packages are installed, and the variogram can be fitted without error. Kriging is the best linear unbiased estimator (BLUE) under the assumption of spatial stationarity – it minimises mean squared prediction error and returns a kriging variance for each cell.
- **IDW** (Inverse Distance Weighting, p = 2) is used as a fallback when kriging cannot be applied. It is fast, deterministic, and requires no model fitting, but produces no uncertainty estimate and can create bull's-eye artefacts around isolated observations.

Kriging variograms are fitted automatically using `gstat::fit.variogram()` with a Matern model (default) – the most flexible and statistically justified model for environmental data. If automatic fitting fails, a spherical model is tried; if that also fails, IDW is used.

Usage

```
interpolate_survey(
  survey,
  vars = NULL,
  resolution_m = 100,
  method = c("auto", "kriging", "idw"),
  kriging_min_n = 50L,
  idw_power = 2,
  idw_max_radius_m = NULL,
  verbose = TRUE
)
```

Arguments

`survey`	Dataframe with `lat`, `lon`, and measurement columns.
`vars`	Character vector of column names to interpolate. Default: all numeric columns except `lat` and `lon`.
`resolution_m`	Numeric. Grid cell size in metres (default 100 m). Coarsen for sparse surveys (> 500 m) or refine for dense ADCP grids (50 m).
`method`	Character. `"auto"` (default – kriging if possible, IDW fallback), `"kriging"` (force kriging; error if insufficient data), or `"idw"` (always use IDW).
`kriging_min_n`	Integer. Minimum observations required to attempt kriging (default 50).
`idw_power`	Numeric. IDW distance decay exponent (default 2).
`idw_max_radius_m`	Numeric. Maximum search radius for IDW neighbours in metres (default 5 * resolution_m).
`verbose`	Logical. Print per-variable method used and diagnostics (default TRUE).

Value

A dataframe on a regular grid with interpolated values, a `method` column per variable (`interp_method_<var>`), and kriging variance columns (`krige_var_<var>`) where kriging was used.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
survey <- read.csv(sample_csv)

# Force IDW (fast, no gstat needed)
grid <- interpolate_survey(survey, resolution_m = 200, method = "idw")
head(grid[, c("lat", "lon", "temperature")])

## Not run:
# Kriging (auto-selects where data are dense enough)
grid_k <- interpolate_survey(survey, resolution_m = 100)
unique(grid_k$interp_method_temperature)

## End(Not run)
```

---

layer_reproducibility *Reliability of each layer from overlapping survey lines*

Description

Measures, layer by layer, how well two independent passes over the same ground agree. This is the `reliability` that `predict_oyster(weighting = "evidence")` needs, taken from the survey rather than typed in.

The survey is binned to a grid. In every cell crossed by two or more lines each line's mean is taken, the lines are paired, and the pairs go to `reliability_from_replicates()`. Nothing is assumed about the instrument: a layer that does not reproduce between lines scores near zero, and a layer with too few overlapping cells to test returns `NA`, which gives it no weight in `evidence_weights()`.

Usage

```
layer_reproducibility(
  df,
  layers = NULL,
  line_col = NULL,
  cell_size_m = NULL,
  min_n = 30,
  method = c("pearson", "spearman"),
  conf_level = 0.95
)
```

Arguments

`df`	A survey data frame with coordinates (`lat`/`lon` or synonyms), a column identifying the survey line or pass, and the layers to test. Rows should be in acquisition order within each line: the automatic cell size is taken from the spacing between consecutive rows of a line.

layers	Character vector of survey column names to test. NULL (default) tests every numeric column recognised as a scored layer.
line_col	Name of the line/pass column. NULL looks for line_id, line, survey_line, pass, pass_id, transect or track.
cell_size_m	Grid cell size in metres. NULL uses three times the median spacing between consecutive points along a line, so that neighbouring lines land in a common cell.
min_n	Minimum overlapping pairs needed; passed to reliability_from_replicates().
method	Correlation method, "pearson" (default) or "spearman".
conf_level	Confidence level for the intervals.

Value

An object of class layer_reproducibility, a list with

- reliability: named numeric, one entry per tested layer, named as the scored layer (depth, not depth_m), NA where untestable. Pass this to predict_oyster(reliability =).
- table: data frame with layer, column, r, ci_low, ci_high, n_pairs, n_cells, established and note.
- cell_size_m, n_lines, line_col.

What it does not tell you

Agreement between lines is reproducibility, not accuracy. A bias shared by every line (a miscalibrated sensor) reproduces perfectly. A large-scale gradient across the site also raises the correlation, because cells differ from one another more than lines differ within a cell; for a layer where that matters, detrend first.

See Also

reliability_from_replicates(), evidence_weights()

Examples

```
set.seed(1)
g  <- expand.grid(lon = seq(0, 0.01, length.out = 40),
                  lat = seq(50, 50.01, length.out = 40))
truth <- sin(g$lon * 900) + cos(g$lat * 700)
pass <- function(id, sd) transform(g, line_id = id,
  depth_m = truth + rnorm(nrow(g), 0, sd),
  turbidity = rnorm(nrow(g)))
survey <- rbind(pass("A", 0.2), pass("B", 0.2))
layer_reproducibility(survey, cell_size_m = 5)
```

lineage_fits_as_list *Turn a fitted lineage model into plain data, and back*

Description

`lineage_fits_as_list()` reduces a `fit_lineages()` result to plain lists and numbers – the profile table, calibration, distance matrix and the signal test – that survive a round trip through JSON. `lineage_fits_from_list()` rebuilds a scoring-ready `lineage_fits` from that: the tolerance objects are regenerated from the species defaults and the stored profile means, so the numbers a user sees in the profile table are exactly the ones that score.

Storing the parameters rather than serialised R objects keeps a saved model inspectable and independent of the R version that fitted it.

Usage

```
lineage_fits_as_list(fits)

lineage_fits_from_list(model)
```

Arguments

`fits`	A `lineage_fits` object.
`model`	A list as returned by `lineage_fits_as_list()` (or read back from its JSON).

Value

A list, or a `lineage_fits` object.

lineage_occurrence *Where is each lineage more likely to occur than the others?*

Description

For every location, scores each lineage's tolerance profile, converts the score to an occurrence probability with that lineage's fitted calibration, and applies Bayes' rule across lineages:
`P(lineage k | present, x) = pi_k * p_k(x) / sum_j pi_j * p_j(x)`

where `pi_k` is a prior share (uniform by default). This is **environmental correspondence**, not evidence of local adaptation: telling adaptation from plasticity needs common-garden or transplant data.

Usage

```
lineage_occurrence(
  fits,
  env,
  prior_share = "uniform",
  ambiguity_threshold = 0.5
)
```

Arguments

fits	A lineage_fits object from fit_lineages().
env	Data frame of locations with lat, lon and the environmental columns the profiles use.
prior_share	"uniform" (default), "records" (proportional to records per lineage) or a named numeric vector.
ambiguity_threshold	The best lineage must hold at least this share of the posterior to be named; default 0.5 (more likely than all others together). With many lineages whose tolerances overlap that is a high bar – most locations will be "Ambiguous", which is an accurate statement that the environment does not tell them apart. lineage_best and lineage_prob are always returned, so a most-likely map is available regardless; lower the threshold, or group lineages, to name more cells.

Value

env (standardised) with, per lineage, suit_<id>, p_<id> and post_<id>, plus lineage_best, lineage_prob, lineage_ambiguity, lineage_class and lineage_source (how the best lineage's profile was obtained), and excluded. Locations that fail a hard exclusion are "Excluded".

Ambiguity

Where the environment does not discriminate between lineages, the result is "Ambiguous" rather than a name. lineage_ambiguity is the normalised entropy of the posterior (0 = one lineage certain, 1 = all equally likely).

See Also

fit_lineages(), rank_source_populations()

---

lineage_signal *Does tolerance track relatedness?*

Description

A Mantel test between the pairwise distance in fitted tolerance parameters and the pairwise genetic distance. Borrowing tolerance from relatives is only justified if relatives really are more alike; this is the measurement. A significant positive correlation supports pooling. No signal means the lineages are best fitted alone: genetic similarity is not tolerance similarity, and local adaptation can decouple them.

Usage

```
lineage_signal(
  estimates,
  distances,
  n_perm = 999L,
  min_lineages = 6L,
  alpha = 0.05,
  seed = NULL
)
```

Arguments

estimates	Numeric matrix, one row per lineage (named) and one column per fitted parameter (e.g. temperature.optimal_min).
distances	Genetic distance matrix (as_lineage_distances()).
n_perm	Number of permutations for the p-value.
min_lineages	Fewer fitted lineages than this and the test is not run: with so few, it has little power (five lineages give ten pairs) and the answer would mean nothing. Default 6.
alpha	Significance level used for usable.
seed	Optional integer for reproducible permutations.

Value

An object of class lineage_signal: r, p, n_lineages, n_perm, usable (enough lineages, r > 0 and p < alpha) and note.

---

list_species *List all supported species*

Description

List all supported species

Usage

```
list_species()
```

Value

Named character vector: names = latin names, values = species keys.

Examples

```
list_species()
```

`load_tolerance_update` *Load saved Bayesian tolerance updates into session cache*

Description

Reads a previously saved `.rds` file (from `save_tolerance_update()`) and restores it into the session cache. Subsequent calls to `predict_oyster()` and `score_locations()` will automatically use the loaded parameters.

Usage

```
load_tolerance_update(
  species = "all",
  path = tools::R_user_dir("oystermapR", which = "data"),
  verbose = TRUE
)
```

Arguments

`species`	Character. Species key or `"all"` to load all `.rds` files in `path`.
`path`	Character. Directory to load from (default: `tools::R_user_dir("oystermapR", "data")`).
`verbose`	Logical. Default TRUE.

Value

Invisibly: the loaded tolerance list.

Examples

```
# Save a Bayesian update to tempdir first, then load it back
records <- data.frame(
  lat = c(51.5, 51.6, 51.7, 51.8, 51.4, 51.55, 51.65, 51.72,
          51.83, 51.42, 51.50, 51.60, 51.70, 51.80, 51.40,
          51.53, 51.63, 51.73, 51.83, 51.43),
  lon = c(-4.1, -4.2, -4.0, -4.3, -4.15, -4.05, -4.25, -4.12,
          -4.08, -4.18, -4.11, -4.21, -4.01, -4.31, -4.16,
          -4.06, -4.26, -4.13, -4.09, -4.19),
  temperature = c(12, 13, 11, 14, 12.5, 13.5, 11.5, 12.8,
                  13.2, 11.8, 12.1, 13.1, 11.1, 14.1, 12.4,
                  13.4, 11.4, 12.7, 13.3, 11.7),
```

```
  salinity = c(34, 35, 33, 36, 34.5, 35.5, 33.5, 34.8,
               35.2, 33.8, 34.1, 35.1, 33.1, 36.1, 34.4,
               35.4, 33.4, 34.7, 35.3, 33.7),
  depth = c(5, 8, 12, 3, 7, 10, 6, 9, 4, 11, 5.5, 8.5,
            12.5, 3.5, 7.5, 10.5, 6.5, 9.5, 4.5, 11.5),
  presence = c(1L, 1L, 1L, 0L, 1L, 1L, 1L, 1L, 0L, 1L,
               1L, 1L, 0L, 0L, 1L, 1L, 1L, 1L, 0L, 1L)
)
update_species_tolerances(records, "ostrea_edulis",
  update_vars = c("temperature", "salinity", "depth"),
  min_records = 20L, verbose = FALSE)
tmp <- file.path(tempdir(), "bayes_updates")
save_tolerance_update("ostrea_edulis", path = tmp, verbose = FALSE)
reset_tolerance_update("ostrea_edulis")           # clear cache
load_tolerance_update("ostrea_edulis", path = tmp) # reload
```

`merge_sensor_data` *Merge multiple sensor datasets into a single survey table*

Description

Takes any number of sensor dataframes (from `read_nortek_adcp()`, `read_generic_csv()`, or custom sources) and merges them spatially into a single table ready for `predict_oyster()`.

Each dataset is first rounded to a common spatial resolution grid. Datasets are then joined using the grid cell key – cells that exist in multiple datasets are merged; cells that exist in only one dataset carry `NA` for variables from other sensors.

Usage

```
merge_sensor_data(..., spatial_res = 4L, date_source = NULL, verbose = TRUE)
```

Arguments

`...` Two or more dataframes to merge. Each must contain `lat` and `lon` columns. Pass them as named arguments for cleaner messages (e.g. `adcp = adcp_df, biobase = biobase_df`).

Multiple runs of the same sensor: If you have several surveys from the same sensor type, pass them as a list and they will be automatically stacked via `stack_surveys()` before merging:

```
merge_sensor_data(
  adcp    = list(adcp_run1, adcp_run2, adcp_run3),
  bathy   = bathy_df,
  biobase = biobase_df
)
```

`spatial_res` Integer. Decimal places for the common spatial grid (default 4, approx.11 m at 56degrees N). Must be the same or coarser than the resolution used when reading each individual sensor.

date_source	Character. Name of the dataset to take the date column from (default NULL – uses the first dataset that contains a date column).
verbose	Logical. Print a merge summary (default TRUE).

Value

A single dataframe with all oystermapR-compatible columns populated from their respective sensor sources. Columns that weren't available from any sensor will be absent from the result – predict_oyster() handles missing columns gracefully.

Examples

```
adcp_f <- system.file("extdata", "example_bay_adcp.csv", package = "oystermapR")
ctd_f  <- system.file("extdata", "example_bay_ctd.csv",  package = "oystermapR")
adcp   <- read_nortek_adcp(adcp_f, verbose = FALSE)
ctd    <- read_generic_csv(ctd_f,  verbose = FALSE)
survey <- merge_sensor_data(adcp = adcp, ctd = ctd)
head(survey[, c("lat", "lon", "current_velocity", "temperature")])
```

---

oystermapR_help *Print an interactive getting-started guide for oystermapR*

Description

Prints a step-by-step workflow guide to the console, covering sensor data ingestion, spatial merging, suitability prediction, and QGIS export. Run this any time you need a reminder of the available functions and typical usage patterns.

Usage

```
oystermapR_help(topic = NULL)
```

Arguments

topic	Character. Optionally focus on a specific topic: "sensors", "merge", "tides", "predict", "weighting", "qgis", "species". Default NULL prints the full guide.

Value

Called for its side-effect (console output). Returns invisible(NULL).

Examples

```
oystermapR_help()                # full guide
oystermapR_help("sensors")       # sensor ingestion only
oystermapR_help("weighting")     # rank vs evidence weighting
oystermapR_help("qgis")          # QGIS output only
```

`oystermapR_live_config` *Configure live data API credentials for oystermapR*

Description

Stores API credentials in R session options so live data fetching can authenticate with external services. Credentials are **not** written to disk and are lost when the R session ends.

All parameters are optional - only set credentials for the services you intend to use. Functions that require a missing credential will warn and fall back to manual data rather than aborting.

Usage

```
oystermapR_live_config(
  cmems_user = NULL,
  cmems_password = NULL,
  ices_key = NULL,
  show_config = FALSE
)
```

Arguments

`cmems_user`	Character. CMEMS username.
`cmems_password`	Character. CMEMS password.
`ices_key`	Character. ICES API key (currently optional; ICES APIs are largely open).
`show_config`	Logical. Print current configuration (credentials masked). Default FALSE.

Value

Invisibly returns a named list of the options set.

Free registrations

- **CMEMS**: Register at marine.copernicus.eu
- **ICES**: API access is open (no key required for HAB/VMS endpoints)
- **EMODnet Biology**: Open WFS, no key required
- **FSA/DAERA**: Open REST API, no key required

Examples

```
## Not run:
oystermapR_live_config(
  cmems_user     = "myusername",
  cmems_password = "mypassword"
)
# Then use fetch_live = TRUE in any supporting function
```

```
score_hab_risk(result, species = "ostrea_edulis", fetch_live = TRUE)

## End(Not run)
```

`parse_opendrift_connectivity` *Parse an OpenDrift particle tracking output into a connectivity matrix*

Description

Converts the output of an OpenDrift settlement simulation (a CSV file with particle origin and final position) into the connectivity matrix format expected by `score_larval_connectivity(connectivity_matrix = ...)`.

OpenDrift setup: Run OpenDrift (`OceanDrift` or `OpenOil` adapted for larvae) seeding particles at each source reef location. Export a CSV with at minimum:

- `origin_lat`, `origin_lon` – seed position
- `final_lat`, `final_lon` – settlement position (stranded or settled)
- `status` – filter to `"stranded"` or `"active"` as appropriate

The function bins particles into a grid and computes the fraction of particles from each source bin that arrive in each destination bin.

Usage

```
parse_opendrift_connectivity(
  opendrift_csv,
  bin_deg = 0.05,
  status_col = NULL,
  status_keep = c("stranded", "settled"),
  min_weight = 0.005,
  verbose = TRUE
)
```

Arguments

`opendrift_csv`	Character. Path to OpenDrift settlement CSV file. Must contain columns `origin_lat`, `origin_lon`, `final_lat`, `final_lon`.
`bin_deg`	Numeric. Grid cell size in decimal degrees for binning (default 0.05 approx. 5 km at mid-latitudes).
`status_col`	Character or NULL. Name of a status column to filter on. If NULL (default), all rows are used.
`status_keep`	Character. Value(s) of `status_col` to include (default `c("stranded","settled")`).
`min_weight`	Numeric. Minimum connectivity weight to include in output (default 0.005 = 0.5 percent). Filters very weak connections.
`verbose`	Logical. Default TRUE.

Value

A dataframe with columns source_lat, source_lon, dest_lat, dest_lon, weight – ready for passing to score_larval_connectivity().

Examples

```
## Not run:
# After running OpenDrift simulation and exporting CSV:
cm <- parse_opendrift_connectivity("opendrift_output.csv", bin_deg = 0.05)

result <- predict_oyster(survey, "ostrea_edulis")
result <- score_larval_connectivity(result,
            species             = "ostrea_edulis",
            connectivity_matrix = cm)

## End(Not run)
```

---

permutation_importance

Permutation variable importance for suitability model

Description

Estimates the importance of each scored environmental variable by randomly permuting its values and measuring the resulting drop in AUC against known presence/absence records. A large drop indicates the model strongly depends on that variable; a small drop indicates it could be removed with minimal impact.

This approach (Breiman 2001) is model-agnostic, requires no model refit, and is directly interpretable: "how much does AUC fall if I destroy the information in variable X?"

Usage

```
permutation_importance(
  predicted,
  records,
  presence_col = "presence",
  n_permutations = 50L,
  match_radius_deg = 0.002,
  seed = 42L,
  verbose = TRUE
)
```

Arguments

predicted	Dataframe from predict_oyster() with lat, lon, suitability, and per-variable score columns (e.g. score_temperature, score_salinity). The function uses the per-variable component scores already computed by predict_oyster().

records	Dataframe of presence/absence records (same format as validate_against_records()).
presence_col	Character. Name of presence/absence column (default "presence").
n_permutations	Integer. Number of permutation replicates per variable (default 50). More = more stable importance estimates; >= 100 recommended for publication.
match_radius_deg	Numeric. Matching radius (default 0.002degrees).
seed	Integer. Random seed (default 42).
verbose	Logical. Default TRUE.

Value

A dataframe (sorted by importance, descending) with columns:

- variable: scored variable name
- baseline_auc: AUC of unperturbed model
- mean_permuted_auc: mean AUC after permuting this variable
- importance: baseline_auc - mean_permuted_auc (drop in AUC)
- importance_sd: SD of AUC drop across permutation replicates
- importance_pct: importance as percentage of baseline AUC
- rank: rank by importance (1 = most important)

Interpretation

- importance > 0.10 (> 10% AUC drop): high importance – variable is load- bearing for model discrimination.
- importance 0.02–0.10: moderate importance.
- importance < 0.02: low importance – consider whether this variable adds value relative to its data collection cost.

References

Breiman (2001) Machine Learning 45:5-32. Zurell et al. (2020) Ecography 43:1261-1277.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
set.seed(42)
records <- data.frame(
  lat      = result$lat[1:60],
  lon      = result$lon[1:60],
  presence = as.integer(runif(60) > 0.4)
)
imp <- permutation_importance(result, records, n_permutations = 10)
print(imp)
```

plot_tolerance *Plot the tolerance scoring curve for a species and variable*

Description

Draws the suitability scoring function for any scored environmental variable directly from the species tolerance specification – no survey dataset is required. The curve shows exactly what score the model assigns to each value of the variable: 1 (optimal), declining gradients through acceptable and poor zones, and 0 at hard limits.

For a data-driven response curve derived from an actual survey (partial dependence), use `sensitivity_analysis()` instead.

Usage

```
plot_tolerance(
  species,
  variable,
  season = NULL,
  n_steps = 500L,
  show_exclusion = TRUE,
  title = NULL,
  colour = "#2166ac"
)
```

Arguments

species	Character string. Species key (e.g. `"ostrea_edulis"`). Accepts the same formats as `predict_oyster()`.
variable	Character string. Name of the scored variable to plot (e.g. `"temperature"`, `"salinity"`, `"ph"`, `"omega_aragonite"`). Use `get_species_tolerances(species)$scored` to see available variables.
season	Character or `NULL`. For seasonal variables (temperature), specifies which season's parameters to display. One of `"winter"`, `"spring"`, `"summer"`, `"autumn"`. If `NULL` (default), the base parameters are used; pass `"all"` to overlay all four seasons.
n_steps	Integer. Number of points on the x-axis (default `500`). More = smoother curve.
show_exclusion	Logical. If `TRUE` (default), overlays the hard exclusion threshold(s) from `tolerances$exclusions` as vertical red lines.
title	Character or `NULL`. Plot title. `NULL` = auto-generated.
colour	Character. Curve colour (default `"#2166ac"`, a marine blue).

Value

A `ggplot2` object (invisibly). Printed as a side effect. Can be further customised with standard `ggplot2` layers. Falls back to base R `plot()` if `ggplot2` is not installed.

Examples

```
# Temperature curve for O. edulis (base / year-round parameters)
plot_tolerance("ostrea_edulis", "temperature")

# Summer temperature only
plot_tolerance("ostrea_edulis", "temperature", season = "summer")

# All four seasons overlaid
plot_tolerance("ostrea_edulis", "temperature", season = "all")

# pH curve for the European flat oyster
plot_tolerance("ostrea_edulis", "ph")

# Aragonite saturation for Pacific oyster
plot_tolerance("magallana_gigas", "omega_aragonite")

# Save to a temporary file
p <- plot_tolerance("ostrea_edulis", "salinity")
ggplot2::ggsave(file.path(tempdir(), "salinity_tolerance.png"), p, width = 8, height = 5)
```

---

predict_oyster *Predict oyster growth suitability from environmental survey data*

Description

The primary user-facing function of oystermapR. Accepts a tabular dataset (CSV file path or dataframe) containing spatial coordinates and environmental measurements, applies species-specific exclusion and weighted scoring rules, and returns a scored dataframe alongside an optional GeoTIFF heatmap for QGIS visualisation.

Usage

```
predict_oyster(
  data,
  species,
  output_geotiff = FALSE,
  resolution = 2e-04,
  contours = TRUE,
  verbose = FALSE,
  weighting = c("rank", "evidence"),
  reliability = NULL,
  confidence = NULL,
  confidence_floor = 0.5,
  assigned = NULL,
  aliases = NULL,
  observed = NULL,
  attempted_failed = NULL
)
```

Arguments

data	A dataframe or a file path to a CSV file. Must contain at minimum: • lon / lng / longitude – longitude in decimal degrees • lat / latitude – latitude in decimal degrees • date – observation date (any format coercible by as.Date()) Additional environmental columns are matched automatically; see **Column Naming** below. **Aragonite auto-calculation:** if the data contains ph (or a recognised synonym) and alkalinity columns alongside temperature and salinity, predict_oyster() will automatically compute omega_aragonite using the in-house calculate_aragonite() function before scoring. You can also pre-compute it and supply omega_aragonite directly.
species	Character string identifying the target oyster species. Accepts the key ("ostrea_edulis"), latin name, or common name. Use list_species() to see all available options.
output_geotiff	Logical or character. If TRUE, exports a GeoTIFF to the current working directory as <species>_suitability.tif. If a character string, it is used as the file path. If FALSE (default), no raster file is written.
resolution	Numeric. Raster cell size in degrees for GeoTIFF output (default 0.001, approximately 100 m at mid-latitudes). Ignored when output_geotiff = FALSE.
contours	Logical. If TRUE (default), contour lines are computed and embedded in the GeoTIFF export. Ignored when output_geotiff = FALSE.
verbose	Logical. If TRUE, prints a per-variable scoring summary after processing (default FALSE).
weighting	Character. How the per-variable scores are combined. "rank" (default) is the 1.5.0 behaviour: weights are 1 / rank, normalised over the variables present in the data. "evidence" weights each variable by prior * reliability * spread (see evidence_weights()) and attaches the coverage of the species' evidence base to the result. The default is unchanged, so existing calls return exactly what they returned in 1.5.0. Note what renormalisation costs you under "rank": a survey measuring four of a species' seventeen scored variables still returns a confident number on [0, 1], because the weights are renormalised over whatever happens to be present. Absence becomes indistinguishable from irrelevance. "evidence" is the mode that says so.
reliability	Named numeric vector, one entry per scored layer, giving the correlation between two independent estimates of that layer – from reliability_from_replicates() where the survey has repeat coverage, or reliability_from_perturbation() where it does not. Required when weighting = "evidence". A layer with no measured reliability receives no weight rather than a default one.
confidence	Optional numeric vector of per-location confidence on [0, 1], one value per row, typically from confidence_score(). When NULL (the default) and weighting = "evidence", confidence is taken to be the share of the evidence weight actually observed at that location – which falls straight out of the weighting and needs no further assumption. Either way a confidence column is returned.

`confidence_floor`
: Numeric, default `0.5`. Locations below this are classified `"Insufficient data"` instead of being given a suitability class. The continuous score is still returned for them. Only used when `weighting = "evidence"`.

`assigned`
: Optional named list for layers with no counterpart in the published tolerance profile, each element a list with `rank` and `rationale`. Passed to `evidence_weights()`. Assigned layers are flagged in the weight table and do not count toward coverage.

`aliases`
: Optional named character vector mapping score column names to tolerance-spec variable names, e.g. `c(hardness = "substrate_hardness")`.

`observed`
: Optional character vector of tolerance-spec variables the survey observed, used for coverage. Defaults to the published layers found in the data – pass it explicitly when a variable was measured but could not be scored.

`attempted_failed`
: Optional character vector of variables that were instrumented but returned unusable data. Reported separately and used for `coverage_if_recovered`, which prices an instrument failure in units of the species' evidence base.

Value

A dataframe (invisibly) with all original columns plus:

- `season`: detected season at each location/date.
- `excluded`: logical, `TRUE` if the location fails a hard exclusion.
- `exclusion_reason`: character, reason(s) for exclusion or `NA`.
- `suitability`: numeric [0, 1], overall weighted suitability score.
- `suitability_class`: factor, one of "High", "Moderate", "Low", "Very Low", "Excluded".
- `score_<variable>`: per-variable component scores.

With `weighting = "evidence"` the result carries, in addition:

- `suitability_rank` / `suitability_class_rank`: what rank weighting would have returned, kept for comparison.
- `confidence`: per-location confidence [0, 1].
- `suitability_class` may take the further value `"Insufficient data"` for locations below `confidence_floor`.
- attributes `"evidence_weights"` and `"evidence_coverage"`, holding the weight derivation and the coverage ledger. These are what belongs in a methods section; render them with `weight_table_md()`.

If `output_geotiff` is set, a GeoTIFF is written to disk and its path is printed to the console.

Column Naming

Column names are matched case-insensitively. Recognised synonyms:

Variable	Recognised column names
Longitude	`lon`, `lng`, `longitude`, `x`
Latitude	`lat`, `latitude`, `y`
Date	`date`, `datetime`, `timestamp`
Temperature (degrees C)	`temperature`, `temp`, `temp_c`
Salinity (PSU)	`salinity`, `sal`, `salinity_psu`
Dissolved oxygen (mg/L)	`dissolved_oxygen`, `do`, `do_mgl`, `oxygen`
Depth (m)	`depth`, `depth_m`
Current velocity (m/s)	`current_velocity`, `velocity`, `current`, `u_mean`
Shear stress (N/m^2)	`shear_stress`, `tau`, `bed_shear`, `shear`
Chlorophyll-a (ugg/L)	`chlorophyll_a`, `chla`, `chl_a`, `chlorophyll`
Turbidity (NTU)	`turbidity`, `ntu`, `turb`
Slope (degrees)	`slope`, `slope_deg`
Roughness / rugosity	`roughness`, `rugosity`
Substrate hardness	`substrate_hardness`, `hardness`, `bottom_hardness`
Sediment type	`sediment_type`, `sediment`, `substrate_type`
Benthic communities	`benthic_communities`, `benthic`, `community`
Biotope	`biotope`, `biotopes`, `habitat`
Fishing intensity	`fishing_intensity`, `fishing`, `fishing_observed`
pH	`ph`, `pH`, `seawater_ph`, `sea_ph`, `water_ph`
Total alkalinity (umol/kg)	`alkalinity`, `total_alkalinity`, `ta`, `alk`, `alkalinity_umol_kg`
Aragonite saturation	`omega_aragonite`, `omega_arag`, `aragonite_saturation`

See Also

`evidence_weights()`, `evidence_coverage()`, `confidence_score()`, `weight_table_md()`

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
table(result$suitability_class)

## Not run:
# Evidence weighting: reliability measured per layer, not assumed.
rel <- c(depth = 0.930, slope = 0.712, substrate_hardness = 0.402)
ev  <- predict_oyster(sample_csv, "ostrea_edulis",
                      weighting   = "evidence",
                      reliability = rel)
attr(ev, "evidence_coverage")
cat(weight_table_md(ev), sep = "\n")

## End(Not run)

## Not run:
# Full pipeline from raw sensor files
adcp_f  <- system.file("extdata", "example_bay_adcp.csv",      package = "oystermapR")
bathy_f <- system.file("extdata", "example_bay_soundings.xyz", package = "oystermapR")
ctd_f   <- system.file("extdata", "example_bay_ctd.csv",       package = "oystermapR")
adcp   <- read_nortek_adcp(adcp_f,    verbose = FALSE)
```

```
bathy  <- read_soundings_xyz(bathy_f, verbose = FALSE)
ctd    <- read_generic_csv(ctd_f,     verbose = FALSE)
survey <- merge_sensor_data(adcp = adcp, bathy = bathy, ctd = ctd)
predict_oyster(survey, "ostrea_edulis", verbose = FALSE)

## End(Not run)
```

`project_suitability` *Project suitability under future climate scenarios*

Description

Re-scores survey locations under user-specified climate perturbations to assess how habitat suitability may change under warming and related oceanographic shifts. Perturbations are applied as additive deltas to temperature and salinity, and multiplicative factors to other variables.

Built-in scenarios align with UKCP18 marine projections for NW European shelf seas:

Scenario	SST delta	Salinity delta	Notes
RCP2.6/SSP1	+0.5degrees C	0.0 PSU	Low emissions, best case
RCP4.5/SSP2	+1.2degrees C	-0.2 PSU	Intermediate emissions
RCP8.5/SSP5	+2.5degrees C	-0.5 PSU	High emissions, worst case
Custom	user	user	Supply your own deltas

The function runs `score_locations()` for each scenario and returns a list with one result dataframe per scenario, plus a comparison summary and a `suitability_delta` column (projected minus baseline) for each.

Usage

```
project_suitability(
  result,
  tolerances,
  scenarios = c("rcp26", "rcp45", "rcp85"),
  custom_deltas = NULL,
  horizon = "2050s",
  verbose = TRUE
)
```

Arguments

`result`	Dataframe from `predict_oyster()` (the baseline).
`tolerances`	Species tolerance list from `get_species_tolerances()`.
`scenarios`	Character vector of built-in scenario names, or NULL to use `custom_deltas` only. Built-in: `"rcp26"`, `"rcp45"`, `"rcp85"`. Default: all three.

custom_deltas	Named list of custom scenarios. Each element is a named numeric vector of deltas keyed to column names, e.g. list(my_scenario = c(temperature = 1.5, salinity = -0.3)).
horizon	Character. Label for the time horizon, used in output names only (default "2050s").
verbose	Logical. Print scenario comparison table (default TRUE).

Value

Named list:

- One dataframe per scenario (named by scenario key) with suitability, suitability_class, and suitability_delta columns.
- "summary" – dataframe comparing mean suitability, class breakdown, and net change vs baseline across all scenarios.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
tol    <- get_species_tolerances("ostrea_edulis")
proj   <- project_suitability(result, tol)
proj$summary
# Cells currently High that become Low under RCP8.5
vulnerable <- subset(proj$rcp85,
  result$suitability_class == "High" & suitability_class == "Low")
nrow(vulnerable)
```

---

qc_survey_data *Run automated quality control on raw survey data*

Description

Detects and flags suspect sensor readings before they enter the suitability model. Three complementary checks are applied to each numeric column:

1. **Range check** – values outside the physically plausible range for that variable (e.g. temperature above 35degrees C in temperate coastal water, salinity above 42 PSU) are flagged as "range_fail".
2. **Statistical outlier** – values beyond iqr_k x IQR from the median are flagged as "outlier" (default k = 3, Tukey's outer fence).
3. **Temporal gradient** – when a datetime column is present and data is sorted chronologically, sequential differences exceeding the max_gradient threshold are flagged as "gradient_fail" (instrument spike or data entry error).

Additionally, **cross-variable sanity checks** flag physically implausible combinations:

- Dissolved oxygen > 20 mg/L with temperature < 5degrees C (likely sensor error)

- Salinity < 5 PSU with temperature > 25degrees C (likely freshwater intrusion instrument error, not a real coastal reading)
- Chlorophyll_a > 50 ugg/L (extreme bloom or sensor fouling)

Flagged values are **not removed** – they are marked in a `qc_flag_<col>` column so the user can decide whether to replace with NA, correct, or accept them. A summary `qc_n_flags` column counts total flags per row.

Usage

```
qc_survey_data(
  df,
  datetime_col = "datetime",
  iqr_k = 3,
  max_gradients = NULL,
  apply_flags = FALSE,
  verbose = TRUE
)
```

Arguments

`df`	Dataframe of raw survey measurements.
`datetime_col`	Character or NULL. Name of the datetime column for temporal gradient checks (default `"datetime"`). Set to NULL to skip.
`iqr_k`	Numeric. IQR multiplier for outlier detection (default 3.0; use 1.5 for stricter QC of high-precision CTD data).
`max_gradients`	Named numeric vector. Maximum allowed absolute change per unit time (seconds) per variable. Defaults are conservative for typical towed or lowered deployments.
`apply_flags`	Logical. If TRUE, replace flagged values with NA in the returned dataframe. If FALSE (default), flags are added but values preserved for manual review.
`verbose`	Logical. Print QC summary (default TRUE).

Value

Input dataframe with additional columns: `qc_flag_<varname>` for each checked variable (`"ok"`, `"range_fail"`, `"outlier"`, `"gradient_fail"`, `"cross_fail"`), `qc_n_flags` (integer count of flags per row), `qc_status` (`"pass"`, `"review"`, `"fail"` – fail = 3+ flags on one row).

Examples

```
# Minimal survey with one out-of-range temperature value
df <- data.frame(
  lat         = c(51.5, 51.6, 51.7),
  lon         = c(-3.2, -3.2, -3.1),
  temperature = c(12.1, 99.0, 11.8),   # 99 degC is an instrument spike
  salinity    = c(33.1, 33.4, 33.0),
  depth       = c(8.0, 9.0, 7.5)
```

```
)
qc <- qc_survey_data(df, apply_flags = FALSE, verbose = TRUE)
# Inspect any flags raised
flagged <- subset(qc, qc_status %in% c("review", "fail"))
flagged[, c("temperature", "qc_flag_temperature", "qc_status")]

# Apply flags: replace bad values with NA before scoring
df_clean <- qc_survey_data(df, apply_flags = TRUE, verbose = FALSE)
```

`rank_source_populations` *Which source population suits each site?*

Description

Ranks lineages as candidate sources for each restoration site. The score is the lineage's occurrence probability at the site, optionally discounted by genetic distance from the population that is locally native, to reflect the risk of introducing a distant lineage.

`score = p_k(x) * exp(-distance_penalty * d(k, local) / max(d))`

With `distance_penalty = 0` (default) genetic distance plays no part and the ranking is purely environmental. Whether and how strongly to penalise translocation distance is a management judgement, not something the data decide; the penalty is stated in the output.

Usage

```
rank_source_populations(
  fits,
  sites,
  local_lineage = NULL,
  distance_penalty = 0,
  n_draws = 0L,
  tie_tol = 0.02,
  seed = NULL
)
```

Arguments

`fits`	A `lineage_fits` object.
`sites`	Data frame, one row per site: `lat`, `lon`, the environmental columns, and optionally `site_id` and a local-lineage column.
`local_lineage`	`NULL` (no discount), a lineage id applied to every site, or the name of a column of `sites` giving each site's native lineage.
`distance_penalty`	Non-negative strength of the discount; the farthest pair of lineages is discounted by `exp(-distance_penalty)`.

n_draws	If positive, profile parameters are drawn this many times from their posteriors and the ranking repeated, adding p_top (share of draws in which the lineage scores within tie_tol of the best) and a 5-95% interval on the score. Calibration uncertainty is not propagated. Slow for many sites.
tie_tol	Scores within this of the best count as tied for first in p_top. Where several lineages all suit a site (scores near 1) the winner is decided by differences in the third decimal, and calling one of them "best" would overstate what the model can tell apart. p_top values then sum to more than 1 across lineages, which is the point.
seed	Optional integer for the draws.

Value

A data frame, one row per site x lineage, ordered by site then rank: site_id, rank, lineage, suitability, p_occurrence, posterior_share, distance_to_local, translocation_weight, score, source, evidence ("low" for borrowed/default profiles) and, with draws, p_top, score_lo, score_hi. Excluded sites return no rows and are listed in the excluded_sites attribute.

See Also

lineage_occurrence(), fit_lineages()

---

read_aanderaa_csv *Read an Aanderaa RCM current meter CSV export*

Description

Reads the CSV file exported by **Aanderaa Data Studio** from an Aanderaa Recording Current Meter (RCM Blue 5450, SeaGuard RCM, or compatible model). The export typically contains time, current speed, current direction, and optionally temperature, pressure/depth, salinity, and dissolved oxygen.

Because Aanderaa instruments are moored at a fixed location, latitude and longitude must be supplied by the user via lat and lon.

Current speed is returned as current_speed (m/s). If the export uses cm/s (common in older instrument firmware), set speed_unit = "cm/s" or leave as "auto" and the function will detect units from the column header or by range-checking the data.

Usage

```
read_aanderaa_csv(
  file,
  lat = NULL,
  lon = NULL,
  speed_unit = c("auto", "m/s", "cm/s"),
  depth_from_pressure = FALSE,
  verbose = TRUE
)
```

Arguments

file	Character. Path to the Aanderaa Data Studio CSV export.
lat	Numeric. Site latitude in decimal degrees (required).
lon	Numeric. Site longitude in decimal degrees (required).
speed_unit	Character. Units of the current speed column. "auto" (default) detects from the column header (looks for cm/s) or infers from data range; "m/s" and "cm/s" override auto-detection.
depth_from_pressure	Logical. If TRUE and a pressure column is present, approximate depth is derived as pressure_dbar / 1.025 (default FALSE; use only when a depth column is absent).
verbose	Logical. Print progress messages (default TRUE).

Value

A one-row dataframe (site-averaged) with columns lat, lon, current_speed, and any additional variables present in the file (temperature, salinity, dissolved_oxygen, pressure_dbar, depth) – ready for merge_sensor_data().

See Also

read_rdi_adcp(), read_nortek_aquadopp(), merge_sensor_data()

Examples

```
## Not run:
# Moored RCM Blue at Strangford Lough narrows
rcm <- read_aanderaa_csv(
  "strangford_rcm_2024.csv",
  lat = 54.398,
  lon = -5.558
)
survey <- merge_sensor_data(adcp = rcm, bathy = bathy)

## End(Not run)
```

---

read_generic_csv *Read a generic sensor CSV with flexible column mapping*

Description

Reads any sensor CSV (Lowrance BioBase, CTD/probe logger, bathymetric sonar, sidescan export, etc.) and maps its columns to the standard oystermapR variable names. Performs spatial averaging at a specified resolution.

Usage

```
read_generic_csv(
  file,
  col_map = NULL,
  spatial_res = 4L,
  min_obs = 3L,
  categorical_cols = c("sediment_type", "benthic_communities", "biotope",
    "fishing_intensity"),
  verbose = TRUE
)
```

Arguments

file	Character. Path to the CSV file.

col_map	Named character vector mapping oystermapR variable names to column names in the CSV. Only variables you want to extract need to be listed. Example:
	`col_map = c(`
	`  lat  = "Latitude",`
	`  lon  = "Longitude",`
	`  depth  = "Depth_m",`
	`  substrate_hardness  = "Hardness",`
	`  temperature  = "Temp_C"`
	`)`
	If `col_map` is `NULL` (default), the function attempts automatic column name matching using built-in synonym lists (case-insensitive).

spatial_res	Integer. Decimal places for spatial binning (default `4`).
min_obs	Integer. Minimum observations per cell (default `3`).
categorical_cols	Character vector. Column names (after mapping) that should be treated as categorical – mode (most common value) is returned instead of mean. Default: `c("sediment_type", "benthic_communities", "biotope", "fishing_intensity")`.
verbose	Logical. Print a summary after loading (default `TRUE`).

Value

A spatially averaged dataframe using oystermapR standard column names.

Examples

```
ctd_f <- system.file("extdata", "example_bay_ctd.csv", package = "oystermapR")
ctd <- read_generic_csv(ctd_f, verbose = FALSE)
head(ctd[, c("lat", "lon", "temperature", "salinity")])
```

`read_nortek_adcp` *Read and process a Nortek Signature 500 ADCP CSV file*

Description

Reads a raw merged Nortek Signature 500 ADCP CSV, automatically detects and excludes depth bins contaminated by acoustic sidelobe interference, computes current velocity and estimated bed shear stress from the deepest clean bin, and returns a spatially averaged dataframe ready to merge with other sensor data via `merge_sensor_data()`.

Sidelobe detection: Each bin is tested independently. A bin is flagged as contaminated if more than `sidelobe_threshold` fraction of its speed readings exceed `max_plausible_speed`. Once a bin is flagged, all deeper bins are also flagged (contamination propagates downward).

Shear stress accuracy: Bed shear stress (`tau = rho * Cd * U_bed^2`) is most accurate when computed from near-bed velocity. If near-bed bins are contaminated and only near-surface bins remain, a warning is issued and the `shear_stress_quality` column is set to `"low"`. If two or more clean bins exist, the deepest clean bin is used and quality is `"moderate"` or `"good"`.

Usage

```
read_nortek_adcp(
  file,
  spatial_res = 4L,
  min_obs = 5L,
  max_plausible_speed = 1.5,
  sidelobe_threshold = 0.1,
  rho = 1025,
  Cd = 0.002,
  verbose = TRUE
)
```

Arguments

`file`	Character. Path to the Nortek merged CSV file.
`spatial_res`	Integer. Decimal places for lat/lon binning (default `4`, approx.11 m cells at 56degrees N). Reduce to `3` (~111 m) for coarser surveys.
`min_obs`	Integer. Minimum number of raw ensembles per spatial cell to include in output (default `5`). Cells with fewer observations are dropped as unreliable.
`max_plausible_speed`	Numeric. Speed in m/s above which a reading is considered a sidelobe spike (default `1.5` m/s – appropriate for sheltered coastal/loch environments). Increase to `2.5` for open coast surveys.
`sidelobe_threshold`	Numeric. Fraction of readings in a bin that must exceed `max_plausible_speed` before the bin is flagged as contaminated (default `0.10`, i.e. 10%).
`rho`	Numeric. Seawater density in kg/m^3 (default `1025`).

Cd	Numeric. Drag coefficient for bed shear stress calculation (default 0.002, typical for mixed sandy/rocky coastal seabed).
verbose	Logical. Print processing summary (default TRUE).

Value

A dataframe with one row per spatial cell containing:

- lat, lon – cell centroid coordinates
- date – earliest observation date in cell (character, "YYYY-MM-DD")
- current_velocity – mean speed from deepest clean bin (m/s)
- current_velocity_sd – standard deviation within cell (m/s)
- current_velocity_p95 – 95th percentile speed (m/s)
- shear_stress – estimated bed shear stress (N/m^2)
- shear_stress_quality – "good", "moderate", or "low"
- n_ensembles – raw ensembles averaged into this cell
- bins_used – which velocity bins contributed (e.g. "bin1")
- bins_excluded – which bins were removed as contaminated

Examples

```
adcp_f <- system.file("extdata", "example_bay_adcp.csv", package = "oystermapR")
adcp <- read_nortek_adcp(adcp_f, verbose = FALSE)
head(adcp[, c("lat", "lon", "current_velocity")])
```

---

read_nortek_aquadopp *Read a Nortek Aquadopp Profiler ASCII export*

Description

Reads the ASCII velocity export produced by Nortek's **AquaPro** software from an Aquadopp Profiler deployment. The function accepts two common export layouts:

- **ENU CSV** – a single file with named columns for east, north, and up velocity per depth bin (e.g. East_1, North_1, East_2, ...) plus optional time, temperature, and pressure columns.
- **Separate beam files** – the companion .sen (sensor) file providing time, temperature, and pressure, and the .v1/.v2 files providing beam velocities. Supply the .sen path to file and set beam_files = c("survey.v1", "survey.v2", "survey.v3").

Because Aquadopp instruments are often moored at a single location rather than vessel-mounted, GPS coordinates may not be embedded in the export. In that case supply lat and lon directly; all output rows receive that fixed position.

Usage

```
read_nortek_aquadopp(
  file,
  beam_files = NULL,
  lat = NULL,
  lon = NULL,
  spatial_res = 4L,
  min_obs = 5L,
  max_plausible_speed = 2.5,
  rho = 1025,
  Cd = 0.002,
  verbose = TRUE
)
```

Arguments

file	Character. Path to the AquaPro ENU CSV export, or path to the .sen file when using separate beam files.
beam_files	Character vector. Paths to .v1, .v2, .v3 beam velocity files (optional). If NULL (default) the function reads velocity from named columns in file.
lat	Numeric. Fixed latitude for moored deployments (decimal degrees). Required when GPS columns are absent from the export.
lon	Numeric. Fixed longitude for moored deployments (decimal degrees). Required when GPS columns are absent from the export.
spatial_res	Integer. Decimal places for lat/lon binning (default 4). For moored deployments all ensembles share one position and this parameter has no practical effect.
min_obs	Integer. Minimum ensembles per grid cell (default 5).
max_plausible_speed	Numeric. Upper speed threshold for outlier removal in m/s (default 2.5; Aquadopp profilers can measure stronger tidal flows than vessel-mounted ADCPs).
rho	Numeric. Water density in kg/m^3 (default 1025).
Cd	Numeric. Drag coefficient for bed shear stress (default 0.002).
verbose	Logical. Print progress messages (default TRUE).

Value

A dataframe with columns lat, lon, current_speed, bed_shear_stress, shear_stress_quality, and optionally temperature, pressure_dbar, and date – ready for merge_sensor_data().

Examples

```
## Not run:
# Moored deployment (fixed position)
adcp <- read_nortek_aquadopp(
  "harbour_mouth_aqd.csv",
  lat = 56.512, lon = -5.403
```

```
)

# Vessel-mounted with GPS columns in the file
adcp <- read_nortek_aquadopp("transect_enu.csv")

survey <- merge_sensor_data(adcp = adcp, bathy = bathy)

## End(Not run)
```

`read_rdi_adcp` *Read a Teledyne RDI ADCP binary file (PD0 format)*

Description

Reads the binary **PD0** output file produced by Teledyne RDI instruments (WorkHorse II, Sentinel V, StreamPro, Ocean Surveyor, Long Ranger) via the the `oce` package's `read.adp.rdi()` parser. The function extracts east/north velocity profiles, applies sidelobe contamination detection, and derives current speed and bed shear stress – producing output in the same format as `read_nortek_adcp()`.

Optional dependency: reading RDI binary (PD0) files requires the `oce` package (`install.packages("oce")`). If `oce` is absent the function will stop with an informative message. ASCII WinRiver II exports (`.txt`/`.csv`) do not require `oce`.

GPS: vessel-mounted deployments embed GPS in the binary file and are handled automatically. For moored deployments supply `lat` and `lon`.

Usage

```
read_rdi_adcp(
  file,
  lat = NULL,
  lon = NULL,
  spatial_res = 4L,
  min_obs = 5L,
  max_plausible_speed = 2,
  sidelobe_threshold = 0.1,
  rho = 1025,
  Cd = 0.002,
  verbose = TRUE
)
```

Arguments

`file`	Character. Path to the RDI binary `.000` / `.PD0` / `.pd0` file, or to an ASCII WinRiver II export (`.txt` or `.csv`) with columns `VelEast_binN` and `VelNorth_binN`.
`lat`	Numeric. Fixed latitude for moored deployments (decimal degrees).
`lon`	Numeric. Fixed longitude for moored deployments (decimal degrees).
`spatial_res`	Integer. Decimal places for lat/lon binning (default 4).

min_obs	Integer. Minimum ensembles per grid cell (default 5).
max_plausible_speed	Numeric. Upper speed threshold in m/s for sidelobe / spike removal (default 2.0).
sidelobe_threshold	Numeric. Fraction of ensembles exceeding max_plausible_speed above which a bin is considered sidelobe-contaminated (default 0.10).
rho	Numeric. Water density in kg/m^3 (default 1025).
Cd	Numeric. Drag coefficient for bed shear stress (default 0.002).
verbose	Logical. Print progress messages (default TRUE).

Value

A dataframe with columns lat, lon, current_speed, bed_shear_stress, shear_stress_quality, and n_ensembles – ready for merge_sensor_data().

See Also

read_nortek_adcp(), read_nortek_aquadopp(), merge_sensor_data()

Examples

```
## Not run:
# Vessel-mounted WorkHorse II (GPS embedded in binary)
adcp <- read_rdi_adcp("survey_2024.000")

# Moored Sentinel V (fixed position)
adcp <- read_rdi_adcp("mooring_2024.000", lat = 52.41, lon = -9.20)

survey <- merge_sensor_data(adcp = adcp, bathy = bathy)

## End(Not run)
```

---

read_sonar_tif *Read a bathymetric or sidescan GeoTIFF and convert to oystermapR variables*

Description

Reads a single-band GeoTIFF raster using terra and returns a spatially averaged dataframe suitable for merge_sensor_data().

Two modes are supported, selected via the type argument:

"bathy" – Bathymetric DEM (e.g. from Ping 3DSS or post-processed soundings). Extracts:

- depth – raster cell value (metres)
- slope – computed using terra::terrain()

- roughness – Terrain Ruggedness Index via terra::terrain()

"sidescan" – Sidescan backscatter mosaic (normalised 0–1 or raw DN). Extracts:

- substrate_hardness – backscatter intensity normalised to [0, 1]. High backscatter -> hard/coarse substrate (rock, gravel, shell); low backscatter -> soft substrate (mud, fine sand).

Usage

```
read_sonar_tif(
  file,
  type = c("bathy", "sidescan"),
  spatial_res = 4L,
  band = 1L,
  nodata_val = NA,
  depth_positive = TRUE,
  verbose = TRUE
)
```

Arguments

file	Character. Path to the GeoTIFF file (.tif).
type	Character. Either "bathy" or "sidescan".
spatial_res	Integer. Decimal places for output lat/lon grid (default 4). The raster is resampled to this resolution before converting to a dataframe.
band	Integer. Raster band to read (default 1).
nodata_val	Numeric. Value to treat as no-data (default NA). Override if your raster uses a sentinel (e.g. -9999).
depth_positive	Logical. For type = "bathy": if TRUE (default), raster values are already positive-downward. Set FALSE if stored as negative elevation.
verbose	Logical (default TRUE).

Value

A dataframe with lat, lon, and the derived variable columns ready for merge_sensor_data().

Examples

```
## Not run:
bathy_df   <- read_sonar_tif("kames_bay_bathy.tif",    type = "bathy")
sidescan_df <- read_sonar_tif("kames_bay_sidescan.tif", type = "sidescan")
survey <- merge_sensor_data(bathy = bathy_df, sidescan = sidescan_df, adcp = adcp_df)

## End(Not run)
```

`read_soundings_xyz` *Read a bathymetric XYZ point cloud and derive depth, slope and roughness*

Description

Reads a plain-text XYZ soundings file (comma or space delimited, optional #-prefixed header), spatially averages onto a regular grid, and derives:

- `depth` – mean depth per cell (metres, positive downward)
- `roughness` – rugosity index per cell, derived from intra-cell depth variance using the surface-area approximation `rugosity = sqrt(1 + (depth_sd / cell_size_m)^2)`. Values near 1.0 are flat; higher values indicate more complex relief.
- `slope` – maximum downslope gradient in degrees, computed by finite differences between neighbouring grid cells (Horn 1981 method).

Usage

```
read_soundings_xyz(
  file,
  lon_col = NULL,
  lat_col = NULL,
  depth_col = NULL,
  spatial_res = 4L,
  min_soundings = 10L,
  depth_positive = TRUE,
  verbose = TRUE
)
```

Arguments

`file`	Character. Path to the `.xyz` (or `.csv`) soundings file. Expected columns: longitude, latitude, depth (in that order, or named). Lines beginning with # are treated as comments.
`lon_col, lat_col, depth_col`	Character or integer. Column names or positions for longitude, latitude, and depth. If `NULL` (default), the function tries common names automatically.
`spatial_res`	Integer. Decimal places for lat/lon binning (default `4`, approx.11 m at 56degrees N). The rugosity calculation uses the physical cell size implied by this resolution.
`min_soundings`	Integer. Minimum soundings per cell to include in output (default `10`). Cells with fewer soundings produce unreliable statistics.
`depth_positive`	Logical. If `TRUE` (default), depth values are already positive-downward. Set to `FALSE` if your sonar records depth as negative (elevation-style) – values will be negated.
`verbose`	Logical. Print processing summary (default `TRUE`).

Value

A dataframe with columns `lat`, `lon`, `depth`, `slope`, `roughness`, and `n_soundings` – ready for `merge_sensor_data()`.

Examples

```
bathy_f <- system.file("extdata", "example_bay_soundings.xyz", package = "oystermapR")
bathy <- read_soundings_xyz(bathy_f, verbose = FALSE)
head(bathy[, c("lat", "lon", "depth")])
```

---

reliability_from_perturbation

Reliability of a layer from a measured error model

Description

For a layer with no repeat coverage to correlate, the same statistic is obtained by injecting two independent realisations of the **measured** per-observation error and recomputing. This keeps every layer's reliability on one scale rather than mixing measured correlations with assumed ones.

The error SD must be the honest one. For depth from a multi-pass survey that is the between-pass disagreement over `sqrt(2)`, **not** the within-cell standard error: the standard error describes scatter within a cell and ignores errors that are systematic within a pass, so it understates the uncertainty that matters here.

`derive` handles quantities computed *from* the noisy layer rather than measured directly. Slope, curvature and bathymetric position index all inherit their error from the depth surface, and perturbing depth before deriving them propagates it correctly – whereas perturbing the derived layer directly would not.

Usage

```
reliability_from_perturbation(
  x,
  sigma,
  derive = NULL,
  n_rep = 1L,
  seed = NULL,
  ...
)
```

Arguments

`x`	Numeric vector (or matrix, for `derive` functions that need spatial structure) of the measured quantity.
`sigma`	Numeric. SD of the per-observation error, in the units of `x`.
`derive`	Optional function applied to each perturbed realisation before correlating, e.g. `function(z) terrain_slope(z, cell = 1)`. The default correlates `x` itself.

n_rep	Number of perturbation pairs to average over. The default of 1 matches a single measured comparison; raising it reduces Monte Carlo noise in the reliability estimate itself.
seed	Optional integer for reproducibility. Reliability enters the weights, so a run that cannot be reproduced produces a weight table that cannot be reproduced.
...	Passed to reliability_from_replicates().

Value

As reliability_from_replicates(), with attribute method set to "perturbation" and sigma recorded.

Examples

```
set.seed(42)
depth <- 8 + rnorm(1000, 0, 1.2)
reliability_from_perturbation(depth, sigma = 0.308 / sqrt(2), seed = 1)
```

---

reliability_from_replicates

Reliability of a layer from two independent estimates

Description

Reliability is put on one scale for every layer: the correlation between two independent estimates of the same quantity. Where a survey has repeat coverage, that is measured directly – line against line, pass against pass – and this function is how it is computed.

Usage

```
reliability_from_replicates(
  x1,
  x2,
  method = "pearson",
  min_n = 30,
  conf_level = 0.95
)
```

Arguments

x1, x2	Numeric vectors of equal length: two independent estimates of the same quantity at the same locations. Pairs where either is NA are dropped.
method	Correlation method passed to stats::cor().
min_n	Minimum number of complete pairs required. Below this the result is NA, which zeroes the layer's weight in evidence_weights() – a layer with too few overlapping observations to test has not earned any influence.
conf_level	Confidence level for the returned interval.

Value

A single numeric, with attributes `n`, `ci` and `established`. `established` is `FALSE` when the interval crosses zero. Note that an unestablished layer is **not** removed: leaving it in and letting the weighting demote it is preferable to excluding it by hand, which would be a second judgement call.

Examples

```
set.seed(1)
truth <- rnorm(500)
reliability_from_replicates(truth + rnorm(500, 0, 0.5),
                            truth + rnorm(500, 0, 0.5))
```

`reset_tolerance_update`

Reset Bayesian tolerance updates for a species

Description

Clears the session cache for a species, reverting to the built-in prior parameters. Does not affect saved files (use file deletion for those).

Usage

```
reset_tolerance_update(species = "all", verbose = TRUE)
```

Arguments

`species`	Character. Species key, or `"all"` to clear all cached updates.
`verbose`	Logical. Default TRUE.

Value

Invisibly NULL.

Examples

```
reset_tolerance_update("ostrea_edulis")
reset_tolerance_update("all")
```

`save_tolerance_update` *Save Bayesian tolerance updates to disk*

Description

Serialises the current session cache for a species to an `.rds` file so that updated parameters persist across R sessions. Load again with `load_tolerance_update()`.

Usage

```
save_tolerance_update(
  species = "all",
  path = tools::R_user_dir("oystermapR", which = "data"),
  verbose = TRUE
)
```

Arguments

`species`	Character. Species key or `"all"` to save all cached species.
`path`	Character. Directory to save into (default: `tools::R_user_dir("oystermapR", "data")`). Created if absent.
`verbose`	Logical. Default TRUE.

Value

Invisibly: path(s) to saved file(s).

Examples

```
# Build a Bayesian update first (populates the session cache)
records <- data.frame(
  lat = c(51.5, 51.6, 51.7, 51.8, 51.4, 51.55, 51.65, 51.72,
          51.83, 51.42, 51.50, 51.60, 51.70, 51.80, 51.40,
          51.53, 51.63, 51.73, 51.83, 51.43),
  lon = c(-4.1, -4.2, -4.0, -4.3, -4.15, -4.05, -4.25, -4.12,
          -4.08, -4.18, -4.11, -4.21, -4.01, -4.31, -4.16,
          -4.06, -4.26, -4.13, -4.09, -4.19),
  temperature = c(12, 13, 11, 14, 12.5, 13.5, 11.5, 12.8,
                  13.2, 11.8, 12.1, 13.1, 11.1, 14.1, 12.4,
                  13.4, 11.4, 12.7, 13.3, 11.7),
  salinity = c(34, 35, 33, 36, 34.5, 35.5, 33.5, 34.8,
               35.2, 33.8, 34.1, 35.1, 33.1, 36.1, 34.4,
               35.4, 33.4, 34.7, 35.3, 33.7),
  depth = c(5, 8, 12, 3, 7, 10, 6, 9, 4, 11, 5.5, 8.5,
            12.5, 3.5, 7.5, 10.5, 6.5, 9.5, 4.5, 11.5),
  presence = c(1L, 1L, 1L, 0L, 1L, 1L, 1L, 1L, 0L, 1L,
               1L, 1L, 0L, 0L, 1L, 1L, 1L, 1L, 0L, 1L)
)
```

```
update_species_tolerances(records, "ostrea_edulis",
  update_vars = c("temperature", "salinity", "depth"),
  min_records = 20L, verbose = FALSE)
tmp <- file.path(tempdir(), "bayes_updates")
save_tolerance_update("ostrea_edulis", path = tmp, verbose = FALSE)
```

`score_anthropogenic_disturbance`
Score anthropogenic disturbance at survey locations

Description

Calculates a disturbance index [0,1] from bottom trawling intensity, dredging activity, and anchor damage. The primary data input is the ICES swept-area ratio (SAR), which can be supplied manually or fetched live from the ICES VMS data portal.

This output is primarily relevant to the gear feasibility and economic viability modules. Very high trawling intensity (SAR > 0.5/yr) constitutes a hard gate for bottom culture and restoration reef deployment – physical gear deployment is not viable on actively trawled ground regardless of ecological suitability.

Input options:

1. `trawling_data` – ICES VMS SAR data, manually downloaded as CSV/dataframe.
2. `fetch_live = TRUE` – queries ICES VMS GeoServer endpoint.
3. Neither – returns zero disturbance with a warning.

Usage

```
score_anthropogenic_disturbance(
  result,
  trawling_data = NULL,
  anchor_data = NULL,
  dredge_areas = NULL,
  fetch_live = FALSE,
  match_radius_m = 5000,
  verbose = TRUE
)
```

Arguments

`result`	Dataframe from `predict_oyster()`. Must contain `lat`, `lon`.
`trawling_data`	Dataframe of ICES VMS SAR data (see Details), or NULL.
`anchor_data`	Dataframe of anchoring density data, or NULL.
`dredge_areas`	Dataframe of dredging/extraction area centroids, or NULL.
`fetch_live`	Logical. Query ICES VMS GeoServer (default FALSE).
`match_radius_m`	Numeric. Spatial matching radius in metres (default 5000 m, consistent with ICES c-square ~0.05degrees resolution).
`verbose`	Logical. Default TRUE.

Value

`result` with additional columns:

- `disturbance_sar`: swept-area ratio at or near each site (NA if no data)
- `disturbance_score` [0,1]: composite anthropogenic disturbance index
- `disturbance_class`: "Negligible" / "Low" / "Moderate" / "High" / "Active"
- `disturbance_gear_gate`: logical – TRUE if SAR exceeds gear deployment threshold (used by `assess_gear_feasibility()`)
- `disturbance_note`: management flag

trawling_data format

A dataframe with columns:

- `lat`, `lon` – centroid of c-square or fishing location
- `sar` – swept-area ratio (numeric; dimensionless ratio per year)
- `gear_type` (optional) – "otter", "beam", "dredge", "seine" etc. Dredge gear has higher impact per SAR unit.

Additional disturbance inputs

- `anchor_data`: dataframe with `lat`, `lon`, `density` (vessel anchoring density index) for recreational/commercial anchorage areas.
- `dredge_areas`: dataframe with `lat`, `lon` identifying licensed aggregate extraction areas (all sites within the polygon receive score 1.0).

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

# Score using depth-proxy only (no external data needed)
result <- score_anthropogenic_disturbance(result)
table(result$anthropogenic_class)

## Not run:
# With manual ICES VMS trawling intensity data
vms <- read.csv("ices_vms_sar_2022.csv")
result <- score_anthropogenic_disturbance(result, trawling_data = vms)

## End(Not run)
```

`score_disease_risk` *Score locations for disease and parasite risk*

Description

Assesses the environmental risk of two key bivalve pathogens at each survey location:

- **Bonamia ostreae** (*Ostrea edulis* only) – an intracellular haplosporidian parasite that is the primary constraint on flat oyster restoration in northern Europe. Risk is temperature-driven: transmission peaks 15-20degrees C and is negligible below 8degrees C. The function also accepts a `known_sites` dataframe of confirmed infected locations to apply a proximity multiplier.
- **OsHV-1 microvariant** (*Magallana gigas*) – a herpesvirus causing Pacific Oyster Mortality Syndrome (POMS). Mortality events occur when seawater exceeds 16degrees C for sustained periods. High turbidity amplifies risk by increasing larval stress.

Risk scores are returned on a 0-1 scale (0 = negligible, 1 = highest environmental risk) and classified as Low / Moderate / High / Critical. They are deliberately kept separate from the suitability score – a High suitability site with High disease risk is a meaningful regulatory red flag.

Usage

```
score_disease_risk(
  result,
  species = "ostrea_edulis",
  known_sites = NULL,
  temp_col = "temperature",
  salinity_col = "salinity",
  turbidity_col = "turbidity",
  verbose = TRUE
)
```

Arguments

`result`	Dataframe from `predict_oyster()` with `lat`, `lon`, and environmental variable columns.
`species`	Character. `"ostrea_edulis"` (Bonamia risk) or `"magallana_gigas"` (OsHV-1 risk).
`known_sites`	Dataframe or NULL. Known infected/positive sites with `lat` and `lon` columns. When supplied, a proximity multiplier is applied. For *O. edulis* these are Bonamia-positive sites; for *M. gigas* OsHV-1 outbreak sites.
`temp_col`	Character. Name of temperature column in `result` (default `"temperature"`).
`salinity_col`	Character. Name of salinity column (default `"salinity"`). Used for Bonamia only.
`turbidity_col`	Character. Name of turbidity column (default `"turbidity"`). Used for OsHV-1 amplification only.
`verbose`	Logical. Print risk summary (default TRUE).

Value

Input dataframe with additional columns: `disease_risk_score` [0-1], `disease_risk_class` (Low/Moderate/High/Critical) `disease_agent` (pathogen name), and `disease_risk_note` (plain-language interpretation).

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

# Basic risk scoring (temperature-driven only)
result <- score_disease_risk(result, "ostrea_edulis")

# With known Bonamia-positive site locations
infected <- data.frame(lat = c(50.3, 50.4), lon = c(-4.2, -4.1))
result <- score_disease_risk(result, "ostrea_edulis", known_sites = infected)
table(result$disease_risk_class)
```

---

score_economic_viability

Score economic viability for aquaculture or restoration

Description

Combines ecological suitability, gear feasibility, site accessibility, and patch size into a composite **economic viability index** [0-1]. This is a heuristic scoring – not a financial model – but it ranks sites consistently and can be used to shortlist candidates for detailed business case development.

The index weighs four components:

1. **Ecological suitability** (40%) – from `predict_oyster()`
2. **Gear feasibility** (25%) – best gear score from `assess_gear_feasibility()`
3. **Accessible patch area** (20%) – log-scaled area of connected suitable habitat (from `analyse_connectivity()` if available)
4. **Access score** (15%) – distance to nearest harbour/port (if `harbours` table supplied) or flat 0.5 if unknown

Usage

```
score_economic_viability(
  result,
  harbours = NULL,
  target = c("restoration", "aquaculture"),
  verbose = TRUE
)
```

Arguments

result	Dataframe from assess_gear_feasibility() (which itself requires predict_oyster() output). Optionally also run analyse_connectivity() first to enable area scoring.
harbours	Dataframe or NULL. Known harbour/landing locations with lat and lon columns. Used to score site accessibility.
target	Character. "restoration" (default) weights ecological suitability and connectivity higher. "aquaculture" weights gear feasibility and economic access higher.
verbose	Logical. Print viability summary (default TRUE).

Value

Input dataframe with viability_score [0-1], viability_class (Poor/Fair/Good/Excellent), and viability_notes.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
result <- assess_gear_feasibility(result)

harbours <- data.frame(
  name = c("Plymouth"),
  lat  = c(50.37),
  lon  = c(-4.14)
)
result <- score_economic_viability(result, harbours = harbours,
  target = "aquaculture")
table(result$viability_class)
```

---

score_hab_risk *Score harmful algal bloom risk at survey locations*

Description

Produces a HAB risk index (0 = negligible, 1 = severe) based on historical bloom event frequency and, where available, nutrient and stratification data.

Input options (in priority order):

1. hab_data – manually supplied dataframe of historical HAB events (see Details).
2. fetch_live = TRUE – queries the ICES HAB database for event records within the survey extent and a specified date range. Requires internet access and the httr package.
3. Neither – returns a fixed background-level risk (0.1) with a warning.

Usage

```
score_hab_risk(
  result,
  hab_data = NULL,
  fetch_live = FALSE,
  date_range = NULL,
  match_radius_m = 5000,
  species = "ostrea_edulis",
  verbose = TRUE
)
```

Arguments

result	Dataframe from predict_oyster(). Must contain lat, lon. Optional: din_ug_l (dissolved inorganic nitrogen ugg/L) and temp_surface_c / temp_bottom_c for stratification index.
hab_data	Dataframe of historical HAB events (see Details), or NULL.
fetch_live	Logical. Query ICES HAB database (default FALSE).
date_range	Character vector length 2 c("YYYY-MM-DD","YYYY-MM-DD"). Date range for live fetch or for filtering manually supplied hab_data. Default: last 10 years.
match_radius_m	Numeric. Radius in metres for event aggregation (default 5000 m = 5 km, consistent with ICES monitoring station spacing).
species	Character. Target species – affects toxin severity weights ("ostrea_edulis" or "magallana_gigas").
verbose	Logical. Default TRUE.

Value

result with additional columns:

- hab_risk [0,1]: composite risk index
- hab_risk_class: "Low" / "Moderate" / "High" / "Critical"
- hab_closure_days_per_year: estimated days/year with shellfish closures
- hab_dominant_toxin: most frequently recorded toxin class at the site
- hab_risk_note: management flag

hab_data format

A dataframe with columns:

- lat, lon – event coordinates (decimal degrees)
- date – event date (Date or character "YYYY-MM-DD")
- genus (optional) – bloom genus (e.g. "Alexandrium", "Pseudo-nitzschia")
- toxin (optional) – toxin class (PSP, ASP, DSP, AZP) – affects severity weight
- closure_days (optional) – number of days the area was closed; default 1 per event if absent

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

# Manual HAB data
hab <- data.frame(
  lat          = c(50.25),
  lon          = c(-4.12),
  date         = "2022-07-15",
  genus        = "Alexandrium",
  toxin        = "PSP",
  closure_days = 14
)
result <- score_hab_risk(result, hab_data = hab, species = "ostrea_edulis")
table(result$hab_risk_class)
```

---

score_larval_connectivity

Score larval dispersal connectivity at survey locations

Description

Estimates the larval connectivity of each survey location using one or both of two approaches:

Route 1 – Built-in gap-threshold union-find (no external data needed): Patches above `min_suitability` that lie within the species' effective dispersal kernel distance are grouped into dispersal clusters using union-find. Within each cluster, a connectivity score is derived from the number, quality, and distance-weighted contribution of potential larval sources (Gaussian decay kernel, sigma = dispersal_km / 2).

Route 2 – External connectivity matrix (particle tracking / literature): When `connectivity_matrix` is supplied (a dataframe of source -> destination pairs with weights), those weights replace the Gaussian kernel for matched pairs. Rows not covered by the matrix fall back to Route 1. The matrix can come from:

- **OpenDrift** particle tracking simulations (export as source/destination settlement density CSV)
- **FVCOM / ROMS** model connectivity matrices
- **Published empirical matrices** (e.g. Robins et al. 2017)

Usage

```
score_larval_connectivity(
  result,
  species = NULL,
  dispersal_km = NULL,
  pld_days = NULL,
  tidal_excursion_km = 5,
  min_suitability = 0.4,
  connectivity_matrix = NULL,
```

```
  matrix_match_radius_deg = 0.1,
  verbose = TRUE
)
```

Arguments

result	Dataframe from predict_oyster() with lat, lon, suitability columns.
species	Character. Species key (e.g. "ostrea_edulis"). Used to look up pelagic larval duration for dispersal kernel. Required if dispersal_km is not supplied.
dispersal_km	Numeric. Override for the effective dispersal kernel radius in km. When supplied, species PLD look-up is skipped.
pld_days	Numeric. Override for the mean pelagic larval duration in days. Takes precedence over species lookup; ignored if dispersal_km is supplied.
tidal_excursion_km	Numeric. Mean daily tidal dispersal distance in km (default 5 km/day approx. mean current 0.06 m/s x 24 h). Increase for highly tidal or estuarine sites (8–15 km/day). Ignored if dispersal_km is supplied.
min_suitability	Numeric. Minimum suitability score for a patch to qualify as a potential larval source or sink (default 0.40). Unsuitable patches still receive a connectivity score reflecting their proximity to sources.
connectivity_matrix	Dataframe or NULL. External connectivity weights (Route 2). Must contain columns: • source_lat, source_lon: coordinates of the larval source • dest_lat, dest_lon: coordinates of the destination • weight: connectivity weight [0, 1] (e.g. settlement probability or proportional larval flux). Higher = stronger connection. OpenDrift export tip: \donttest{write.csv(connectivity_matrix, "cm.csv")}.
matrix_match_radius_deg	Numeric. Spatial matching tolerance in decimal degrees for linking matrix entries to result rows (default 0.10 approx. 10 km).
verbose	Logical. Print dispersal parameters and connectivity summary (default TRUE).

Value

result with additional columns:

- larval_dispersal_km: effective kernel distance used
- larval_pld_days: PLD used (species default or override)
- larval_type: "lecithotrophic" or "planktotrophic"
- n_larval_sources: suitable patches within dispersal range
- source_quality_score [0,1]: Gaussian-weighted quality of nearby sources
- larval_cluster_id: dispersal network cluster ID (integer; NA if below min_suitability threshold and no sources nearby)

- `larval_cluster_size`: number of suitable patches in dispersal network
- `nearest_source_km`: km to nearest suitable source patch
- `larval_connectivity_score` [0,1]: composite dispersal connectivity score
- `larval_connectivity_class`: "Highly connected" / "Connected" / "Low connectivity" / "Isolated"
- `larval_connectivity_note`: plain-language ecological interpretation
- `larval_source`: "union_find", "matrix", or "matrix+union_find"

Dispersal distance

If `dispersal_km` is not supplied, the effective kernel distance is derived from the species' mean pelagic larval duration (PLD) multiplied by `tidal_excursion_km` (mean daily tidal dispersal distance):

```
dispersal_km = PLD_mean_days x tidal_excursion_km
```

PLD values are literature-derived:

Species	PLD	Larval type	Default dispersal
O. edulis	1–7 days	lecithotrophic	~20 km
M. gigas	14–28 days	planktotrophic	~105 km
C. angulata	14–21 days	planktotrophic	~85 km
O. stentina	2–7 days	lecithotrophic	~20 km
O. lurida	3–10 days	lecithotrophic	~30 km

References

Cowen R.K. & Sponaugle S. (2009) Annual Review of Marine Science 1:443–466. Robins P.E. et al. (2017) J Applied Ecology 54:1699–1710. doi:10.1111/13652664.12854 Siegel D.A. et al. (2008) PNAS 105:8974–8979. Woolmer A.P. et al. (2009) J Shellfish Research 28:107–116. Trimble A.C. et al. (2009) J Shellfish Research 28:43–53.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
result <- score_larval_connectivity(result, species = "ostrea_edulis")
table(result$larval_connectivity_class)
```

score_locations *Score all locations in a dataframe*

Description

Applies `.score_row()` across every non-excluded row. Excluded rows receive `suitability = 0`. Adds per-variable score columns and a `limiting_factors` column identifying the variables most constraining the score at each location.

Usage

```
score_locations(df, tolerances, verbose = FALSE)
```

Arguments

`df`	A dataframe processed by `check_exclusions()`.
`tolerances`	Species tolerance list from `get_species_tolerances()`.
`verbose`	Logical. Print per-variable scoring summary (default `FALSE`).

Value

The input dataframe with additional columns:

- `suitability`: weighted score [0, 1]
- `suitability_class`: "High" / "Moderate" / "Low" / "Very Low" / "Excluded"
- `limiting_factors`: comma-separated names of the variables most limiting the score (NA if all variables score >= 0.65)
- `score_<variable>`: one column per scored variable present in the data

score_predation_risk *Score predation and bioturbation pressure at survey locations*

Description

Produces a predation risk index (0 = negligible, 1 = severe) based on predator occurrence data, substrate rugosity, and depth. The output is intended as an **overlay** on ecological suitability – high predation risk does not reduce the habitat suitability score directly but is flagged in a separate column for management planning (e.g. cage exclusion, deep subtidal placement to avoid intertidal drills).

Input options (in priority order):

1. `predator_data` – manually supplied dataframe with predator records (see Details).
2. `fetch_live = TRUE` – queries EMODnet Biology for *Asterias rubens* and *Carcinus maenas* occurrence records within the survey extent. Requires internet access and the `httr` package.
3. Neither – returns a depth-proxy-only risk score with a warning.

Usage

```
score_predation_risk(
  result,
  predator_data = NULL,
  fetch_live = FALSE,
  match_radius_m = 1000,
  species = "ostrea_edulis",
  verbose = TRUE
)
```

Arguments

result	Dataframe from predict_oyster() with lat, lon, and optionally depth_m and substrate columns.
predator_data	Dataframe of predator records (see Details), or NULL.
fetch_live	Logical. Query EMODnet Biology API (default FALSE).
match_radius_m	Numeric. Radius in metres within which predator records are aggregated per survey cell (default 1000 m).
species	Character. Target oyster species – affects which predators are most relevant ("ostrea_edulis" or "magallana_gigas").
verbose	Logical. Default TRUE.

Value

result with additional columns:

- predation_risk [0,1]: composite risk index
- predation_risk_class: "Low" / "Moderate" / "High" / "Severe"
- predation_risk_note: text flag for management planning
- n_predator_records: number of predator records within match_radius_m

predator_data format

A dataframe with columns:

- lat, lon – coordinates
- species – species name or AphiaID (character)
- density (optional) – relative abundance / density index; if absent, each record is treated as presence-only (density = 1)
- date (optional) – survey date (used to filter to relevant season)

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

# Depth-proxy only (no external data needed)
result <- score_predation_risk(result)
table(result$predation_risk_class)

# With manual predator observations
pred <- data.frame(lat=50.25, lon=-4.12, species="Asterias rubens", density=3)
result <- score_predation_risk(result, predator_data=pred, species="ostrea_edulis")
```

`score_sediment_stability` *Score sediment stability and mobility at survey locations*

Description

Calculates the Shields parameter and a sediment stability score [0,1] for each survey location. High sediment mobility (theta » theta_cr) indicates that the seabed is frequently in motion, which is unfavourable for oyster settlement, juvenile survival, and restoration reef persistence.

Usage

```
score_sediment_stability(
  result,
  current_col = "current_ms",
  wave_hs_col = "wave_hs_m",
  depth_col = "depth_m",
  substrate_col = "substrate",
  d50_col = "d50_mm",
  wave_period_s = 8,
  drag_coef = 0.003,
  verbose = TRUE
)
```

Arguments

`result` Dataframe from `predict_oyster()`. Must contain `lat`, `lon`. Key optional columns (column names configurable via arguments):

- `current_ms`: depth-averaged current speed (m/s)
- `wave_hs_m`: significant wave height (m) – from `score_wave_exposure()` or direct measurement
- `depth_m`: water depth (m) – needed for wave orbital velocity
- `substrate`: substrate classification string – used to estimate d50
- `d50_mm`: median grain diameter (mm) – overrides substrate-derived d50

current_col	Character. Column name for depth-averaged current speed (m/s). Default "current_ms".
wave_hs_col	Character. Column name for significant wave height (m). Default "wave_hs_m".
depth_col	Character. Column name for depth (m). Default "depth_m".
substrate_col	Character. Column name for substrate type. Default "substrate".
d50_col	Character. Column name for measured median grain size (mm). Default "d50_mm". If present, overrides substrate-derived estimate.
wave_period_s	Numeric. Peak wave period in seconds for orbital velocity calculation. Default 8 s (typical NW European sea state).
drag_coef	Numeric. Quadratic bed drag coefficient C_f. Default 0.003 (smooth mixed seabed; increase to 0.005-0.010 for rough rock/reef).
verbose	Logical. Default TRUE.

Value

result with additional columns:

- shields_parameter: Shields number theta at each site
- shields_critical: critical Shields number theta_cr for the substrate d50
- mobility_ratio: theta / theta_cr (> 1.0 = mobile; < 1.0 = stable)
- d50_mm_estimated: grain size used (mm) – measured or substrate-derived
- sediment_stability_score [0,1]: 1 = fully stable, 0 = highly mobile
- sediment_mobility_class: "Stable" / "Marginally stable" / "Mobile" / "Highly mobile"
- sediment_stability_note: management flag

Inputs required

At minimum, the function needs current velocity to estimate bed shear stress. Grain size improves accuracy but can be estimated from the substrate type column if not directly measured.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

# Using current_velocity and sediment_type columns from the survey
result <- score_sediment_stability(result,
  current_col = "current_velocity", substrate_col = "sediment_type")
table(result$sediment_stability_class)
```

`score_settlement` *Score locations for spat settlement suitability*

Description

Assesses whether survey locations meet the conditions required for bivalve larval settlement and metamorphosis. Settlement scoring uses a separate, tighter tolerance specification than adult habitat scoring – larvae are more sensitive to turbidity, require minimum current flow for delivery, and need hard substrate for attachment.

The output can be compared directly against the adult suitability score from `predict_oyster()` to distinguish "good adult habitat" from "likely natural recruitment site."

Usage

```
score_settlement(survey, species = "ostrea_edulis", verbose = TRUE)
```

Arguments

`survey`	Dataframe of survey measurements. Same format as `predict_oyster()`.
`species`	Character. One of `"ostrea_edulis"` or `"magallana_gigas"`.
`verbose`	Logical. Print summary (default TRUE).

Value

Dataframe with `settlement_suitability`, `settlement_class`, and per-variable `settle_score_*` columns. Combine with adult result via `cbind()` or merge on `lat`/`lon`.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
survey_df  <- read.csv(sample_csv)
adult_result      <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
settlement_result <- score_settlement(survey_df, "ostrea_edulis")
# Cells suitable for both adult survival AND natural recruitment
combined <- merge(adult_result,
  settlement_result[, c("lat","lon","settlement_suitability","settlement_class")],
  by = c("lat","lon"))
combined$dual_suitable <- combined$suitability >= 0.6 &
                          combined$settlement_suitability >= 0.6
sum(combined$dual_suitable)
```

`score_wave_exposure` *Score wave exposure from fetch or measured wave height*

Description

Produces a wave exposure score [0,1] and significant wave height estimate for each survey location. High wave exposure reduces gear feasibility and can destabilise restoration reef structures and spat bags.

Input options:

- `fetch_km`: effective fetch in kilometres (distance to nearest land or obstruction in the prevailing wind direction). Most useful for estuarine, sea loch, or enclosed bay sites. Can be measured from GIS (e.g. in QGIS using the Fetch tool) or estimated from chart inspection.
- `wave_height_m`: directly measured or modelled significant wave height (Hs). If supplied, `fetch_km` is ignored for Hs computation.
- Both absent: wave exposure is estimated from depth alone (coarse proxy; a warning is issued).

Usage

```
score_wave_exposure(
  result,
  fetch_col = NULL,
  wave_height_col = NULL,
  fetch_km = NULL,
  wave_height_m = NULL,
  wind_speed_ms = 12,
  depth_limit = TRUE,
  verbose = TRUE
)
```

Arguments

`result`	Dataframe from `predict_oyster()`. Must contain `lat`, `lon`. Optional columns used if present: `depth_m`, `fetch_km`, `wave_height_m`.
`fetch_col`	Character. Name of a column in `result` containing fetch values in km. Alternative to supplying a scalar `fetch_km`.
`wave_height_col`	Character. Name of a column in `result` containing measured/modelled Hs in metres.
`fetch_km`	Numeric scalar. Uniform fetch in km applied to all rows (overridden by `fetch_col` if both supplied).
`wave_height_m`	Numeric scalar. Uniform Hs in metres applied to all rows (overridden by `wave_height_col` if both supplied).
`wind_speed_ms`	Numeric. Design wind speed in m/s used for JONSWAP calculation (default 12 m/s = moderate gale, Beaufort 6, typical design condition for coastal aquaculture siting).

depth_limit	Logical. Apply depth-limited wave breaking cap (Hs_max = 0.6 * depth_m). Default TRUE.
verbose	Logical. Default TRUE.

Value

result with additional columns:

- wave_hs_m: significant wave height (m) – computed or supplied
- wave_exposure_score [0,1]: 0 = fully sheltered, 1 = severely exposed
- wave_exposure_class: "Sheltered" / "Moderate" / "Exposed" / "Severe"
- wave_exposure_note: gear implication flag
- wave_source: method used ("measured", "jonswap_fetch", "depth_proxy")

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)

# Uniform fetch for a sheltered sea loch
result <- score_wave_exposure(result, fetch_km = 3.5)
table(result$wave_exposure_class)

# Depth proxy only (no fetch data)
result2 <- score_wave_exposure(result)
```

---

sensitivity_analysis *Partial dependence / sensitivity analysis for a scored variable*

Description

Computes the marginal response curve for a single environmental variable: suitability is predicted at a grid of values for the focal variable, while all other variables are held at their observed median (or a specified background value). This reveals the shape of the scoring function as actually applied to a real dataset.

Partial dependence is a standard diagnostic for SDM publication (Elith et al. 2008; Zurell et al. 2020 ODMAP protocol). Use it to verify that predicted responses match ecological expectations before submission.

Usage

```
sensitivity_analysis(
  predicted,
  species,
  variable,
  n_steps = 100L,
  background = NULL,
```

```
  season = NULL,
  verbose = TRUE
)
```

Arguments

predicted	Dataframe from predict_oyster() containing lat, lon, suitability, and per-variable score/weight columns.
species	Character. Species key (e.g. "ostrea_edulis"). Used to retrieve tolerance parameters for the scoring function.
variable	Character. Name of the focal variable to vary (e.g. "temperature", "salinity", "depth").
n_steps	Integer. Number of evenly spaced values across the variable's biological range (default 100).
background	Named list. Fixed values for all other variables. If NULL (default), uses column medians from predicted.
season	Character or NULL. Season to apply for seasonal variables (e.g. "summer"). NULL = no seasonal override.
verbose	Logical. Default TRUE.

Value

A dataframe with columns:

- x: focal variable value
- suitability: predicted suitability at that value
- variable: focal variable name
- species: species key

Suitable for plotting with ggplot2 or base R plot().

References

Elith et al. (2008) J Animal Ecology 77:802-813. Zurell et al. (2020) Ecography 43:1261-1277.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
temp_pd <- sensitivity_analysis(result, "ostrea_edulis", "temperature")
plot(temp_pd$x, temp_pd$suitability, type = "l",
     xlab = "Temperature (degrees C)", ylab = "Suitability [0-1]")
```

`smooth_suitability` *Apply Gaussian kernel smoothing to suitability scores*

Description

Individual CTD or ADCP casts are point measurements subject to instrument noise, micro-scale patchiness, and tidal state variability. `smooth_suitability()` applies a Gaussian distance-weighted kernel to the suitability scores, replacing each cell's score with a weighted mean of all cells within `bandwidth_m` metres. The result is a spatially coherent surface that better reflects broad habitat quality rather than single-cast noise.

The smoothed score is computed as:

$$\hat{s}_i = \frac{\sum_j w_{ij} \cdot s_j}{\sum_j w_{ij}}$$

where $w_{ij} = \exp(-d_{ij}^2/(2\sigma^2))$ and σ is the Gaussian bandwidth in metres. Excluded cells (suitability = 0, excluded flag) are down-weighted but included so that unsuitable barriers are preserved.

Usage

```
smooth_suitability(
  result,
  bandwidth_m = 500,
  max_radius_m = NULL,
  smooth_excluded = FALSE,
  verbose = TRUE
)
```

Arguments

`result`	Dataframe from `predict_oyster()` with `lat`, `lon`, `suitability`, and `excluded` columns.
`bandwidth_m`	Numeric. Gaussian kernel bandwidth (1 sigma) in metres. Default 500 m. Increase for sparser surveys; decrease to preserve fine detail.
`max_radius_m`	Numeric. Maximum search radius for neighbours in metres. Cells further than this are ignored (default `3 * bandwidth_m`).
`smooth_excluded`	Logical. If FALSE (default), excluded cells do not contribute to neighbours' smoothed scores (hard barriers are preserved).
`verbose`	Logical. Print summary (default TRUE).

Value

Input dataframe with `suitability_raw` (original) and `suitability` (smoothed, overwrites original) columns, plus a `suitability_class` column reclassified from the smoothed score.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
result_smooth <- smooth_suitability(result, bandwidth_m = 300)
head(result_smooth[, c("lat", "lon", "suitability")])
```

---

spatial_block_cv *Spatial block cross-validation for suitability model evaluation*

Description

Evaluates model performance using spatial block cross-validation (Roberts et al. 2017), which avoids inflated performance estimates that arise when nearby train and test records share similar environments.

Records are partitioned into `n_blocks` contiguous geographic blocks. In each fold, one block is held out as test data and the remaining blocks provide training data to re-score the suitability model. AUC, TSS, and Brier score are computed per fold and summarised across folds.

This function is purely evaluative – it does not refit a new model. Instead, it uses the spatial heterogeneity in the existing `predicted` suitability surface to estimate out-of-block performance.

Usage

```
spatial_block_cv(
  predicted,
  records,
  presence_col = "presence",
  n_blocks = 5L,
  match_radius_deg = 0.002,
  seed = 42L,
  plot = TRUE,
  verbose = TRUE
)
```

Arguments

`predicted`	Dataframe from `predict_oyster()` containing `lat`, `lon`, `suitability`.
`records`	Dataframe of known presence/absence records. Must contain `lat`, `lon`, and a presence/absence column.
`presence_col`	Character. Name of presence/absence column (default `"presence"`).
`n_blocks`	Integer. Number of spatial blocks (default 5). More blocks = finer spatial resolution but smaller test sets per fold.
`match_radius_deg`	Numeric. Matching radius in degrees (default 0.002 approx. 220 m). Passed to inner matching step.
`seed`	Integer. Random seed for reproducibility (default 42).

plot	Logical. Reserved for future visualisation output; currently unused (default TRUE).
verbose	Logical. Print fold-level diagnostics (default TRUE).

Value

A named list:

- mean_auc: mean AUC across blocks (primary reporting metric)
- auc_sd: standard deviation of per-block AUC
- mean_tss, tss_sd: mean / SD of True Skill Statistic
- brier_mean, brier_sd: mean / SD of Brier score
- n_blocks_actual: number of blocks with >= 5 records (used in CV)
- fold_results: dataframe with per-fold metrics
- spatial_autocorr_range_deg: estimated autocorrelation range (degrees)
- spatial_bias_warning: logical, TRUE if random CV would be misleading

Interpreting results

Compare mean_auc from spatial CV with standard AUC from validate_against_records(). A large drop (> 0.10) indicates strong spatial autocorrelation and means the model is less transferable than the standard validation suggested. This is common for coastal surveys with spatially clustered records.

References

Roberts et al. (2017) Ecography 40:913-929. doi:10.1111/ecog.02881

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
# Use all result rows with alternating presence/absence for a balanced dataset
set.seed(42)
idx <- seq_len(nrow(result))
records <- data.frame(
  lat      = result$lat[idx],
  lon      = result$lon[idx],
  presence = as.integer(runif(length(idx)) > 0.4)
)
sp_cv <- spatial_block_cv(result, records, n_blocks = 4)
sp_cv$mean_auc
```

`stack_surveys` *Stack multiple survey runs from the same sensor type*

Description

Combines two or more dataframes from the same sensor (e.g. an ADCP run on Monday and again on Thursday) by row-binding them and re-averaging overlapping cells onto a common spatial grid. Non-overlapping cells are kept as-is.

This is the right tool when you have repeated surveys of the same area – rather than picking one run, overlapping cells are averaged across all runs which improves noise reduction. Non-overlapping cells (different survey extents) are preserved.

After stacking, pass the result into `merge_sensor_data()` alongside your other sensor datasets as usual.

Usage

```
stack_surveys(..., spatial_res = 4L, verbose = TRUE)
```

Arguments

`...`	Two or more dataframes of the same sensor type. Each must contain `lat` and `lon` columns.
`spatial_res`	Integer. Decimal places for the common spatial grid (default 4, approx.11 m at 56degrees N).
`verbose`	Logical. Print a summary (default `TRUE`).

Value

A single spatially averaged dataframe combining all input surveys.

Examples

```
adcp_f <- system.file("extdata", "example_bay_adcp.csv", package = "oystermapR")
run1 <- read_nortek_adcp(adcp_f, verbose = FALSE)
run2 <- read_nortek_adcp(adcp_f, verbose = FALSE)
adcp_all <- stack_surveys(run1, run2)
nrow(adcp_all)
```

tidal_port_info *Look up approximate tidal range for major European survey ports*

Description

Returns the mean spring tidal range (MHWS - MLWS) and typical Chart Datum offset from Mean Sea Level for a named port. Useful for a quick sanity check on tidal height values before applying `correct_to_chart_datum()`.

This is a reference table only – always use official tide table predictions for actual corrections.

Usage

```
tidal_port_info(port)
```

Arguments

`port`	Character. Port name (partial, case-insensitive match).

Value

A one-row dataframe with columns `port`, `country`, `mhws_m`, `mlws_m`, `spring_range_m`, `cd_below_msl_m`.

Examples

```
tidal_port_info("oban")
tidal_port_info("brest")
```

update_species_tolerances *Update species tolerance parameters from field observations (Bayesian)*

Description

Fits a Bayesian logistic regression linking suitability scores to observed presence/absence, then updates the optimal-range tolerance parameters for one or more scored variables.

Default method (MAP + Laplace): finds the Maximum A Posteriori estimate via `optim()` (L-BFGS-B) and approximates the posterior as a multivariate normal using the numerical Hessian. Fast; suitable for routine use.

MCMC method: Random-Walk Metropolis-Hastings sampler for full posterior samples. Recommended when n < 50 or when credible intervals are needed for publication. May take several minutes for large parameter sets.

Posteriors are stored in a package-level cache so that repeated calls accumulate evidence (sequential Bayesian updating). Use `save_tolerance_update()` to persist between sessions.

Usage

```
update_species_tolerances(
  records,
  species,
  update_vars = NULL,
  presence_col = "presence",
  method = c("map", "mcmc"),
  n_iter = 5000L,
  n_warmup = 1000L,
  prior_sd_fraction = 0.2,
  min_records = 20L,
  verbose = TRUE,
  profile = NULL,
  prior = NULL
)
```

Arguments

records	Dataframe of field observations. Must contain columns `lat`, `lon`, a presence/absence column (1 = present, 0 = absent), and columns matching the environmental variables to update (e.g. `temperature`, `salinity`, `depth`).
species	Character. Species key (e.g. `"ostrea_edulis"`). See `list_species()`.
update_vars	Character vector. Variables to update parameters for. Defaults to all numeric scored variables present in `records`.
presence_col	Character. Name of presence/absence column (default `"presence"`).
method	Character. `"map"` (default, fast) or `"mcmc"` (full posterior).
n_iter	Integer. MCMC iterations (default 5000; ignored for MAP).
n_warmup	Integer. MCMC warm-up / burn-in (default 1000; ignored for MAP).
prior_sd_fraction	Numeric. Prior SD as fraction of parameter range (default 0.20 = 20 percent). Wider = weaker / more data-driven prior.
min_records	Integer. Minimum matched records required (default 20).
verbose	Logical. Print progress and diagnostics (default TRUE).
profile	Optional character label for one population or lineage. When given, the update is fitted from the species defaults and stored under `"<species>::<profile>"` rather than under the species, so several profiles of one species can coexist and the species-level posterior is left alone. Used by `fit_lineages()`. Default `NULL`: behaviour unchanged.
prior	Optional named list overriding the prior for particular variables, each element `list(optimal_min = c(mean, sd), optimal_max = c(mean, sd))`. Variables not listed keep the default prior (current bounds, SD = `prior_sd_fraction` of the range). Used to pass in a prior borrowed from related lineages. Default `NULL`: behaviour unchanged.

Value

Invisibly: a named list with elements:

- `species`: species key
- `method`: "map" or "mcmc"
- `n_records`: number of matched records used
- `loglik_null`: log-likelihood of intercept-only model
- `loglik_fit`: log-likelihood at MAP/posterior mean
- `mcfadden_r2`: 1 - loglik_fit / loglik_null (pseudo-R^2)
- `updated_params`: named list of updated parameter values per variable, with posterior SDs, 95% intervals and `at_bound_omin` / `at_bound_omax` flags (TRUE when the estimate sits on the search box, where the Laplace SD is meaningless and the prior SD is reported instead)
- `posterior_sd`: named list of posterior SD per parameter (from Laplace/MCMC)
- `prior_params`: the parameter values before updating (for comparison)
- `convergence`: optim() convergence code (0 = success; MAP only)
- `tolerances`: the full updated tolerance object, ready to score with
- `profile`: the `profile` label, or `NULL`

Side effect: updates the in-session cache accessed by `score_locations()`.

What gets updated

For each variable in `update_vars`, the function estimates shifts in `optimal_min` and `optimal_max` (the sweet-spot bounds), plus `acceptable_min` / `poor_min` and `acceptable_max` / `poor_max` (shoulder bounds, updated proportionally to maintain curve shape). Parameters are constrained to biologically plausible ranges via box constraints in the optimiser.

See Also

`get_tolerance_posteriors()`, `reset_tolerance_update()`, `save_tolerance_update()`, `load_tolerance_update()`

Examples

```
# Field records with presence/absence + environmental measurements
records <- data.frame(
  lat      = c(51.5, 51.6, 51.7, 51.8, 51.4, 51.55, 51.65, 51.72,
               51.83, 51.42, 51.50, 51.60, 51.70, 51.80, 51.40,
               51.53, 51.63, 51.73, 51.83, 51.43),
  lon      = c(-4.1, -4.2, -4.0, -4.3, -4.0, -4.15, -4.22, -4.01,
               -4.32, -4.03, -4.11, -4.21, -4.01, -4.31, -4.00,
               -4.13, -4.23, -4.02, -4.30, -4.01),
  presence = c(1,1,0,0,1, 1,1,0,0,1, 1,0,0,1,1, 0,1,1,0,1),
  temperature = c(16.2,17.1,12.3,11.0,15.8, 16.5,17.0,12.5,11.2,
                  15.5, 16.0,17.3,12.0,11.5,15.9, 16.8,17.2,12.1,
                  11.1,15.6),
  salinity = c(32,33,31,30,34, 32,33,31,30,34, 33,33,31,31,34,
               32,33,32,30,34),
```

```
  depth    = c(5,8,15,20,3, 6,9,14,18,4, 5,10,16,19,3,
               7,8,15,20,4)
)

fit <- update_species_tolerances(
  records     = records,
  species     = "ostrea_edulis",
  update_vars = c("temperature", "salinity", "depth"),
  min_records = 20L
)
get_tolerance_posteriors("ostrea_edulis")
```

`validate_against_records` *Validate suitability predictions against known presence/absence records*

Description

Compares the suitability scores from `predict_oyster()` against a presence/absence dataset to quantify how well the model discriminates between occupied and unoccupied habitat. Useful for assessing whether the rank-weighted scoring produces ecologically defensible outputs before using them for site selection.

Metrics computed:

- **AUC** (Area Under the ROC Curve): probability that a random presence location scores higher than a random absence location. 0.5 = random; 1.0 = perfect discrimination.
- **Optimal threshold:** Youden's J statistic (maximises sensitivity + specificity - 1). Used for the confusion-matrix-derived metrics below.
- **Sensitivity** (true positive rate): fraction of presences correctly predicted above threshold.
- **Specificity** (true negative rate): fraction of absences correctly predicted below threshold.
- **F1 score**: harmonic mean of precision and sensitivity.
- **Brier score**: mean squared error between predicted probability and observed presence/absence; 0 = perfect, 0.25 = uninformative.
- **TSS** (True Skill Statistic = sensitivity + specificity - 1): values above 0.4 generally indicate useful predictive skill.

Usage

```
validate_against_records(
  predicted,
  records,
  presence_col = "presence",
  match_radius_deg = 0.002,
  plot = TRUE,
  verbose = TRUE
)
```

Arguments

predicted	Dataframe from predict_oyster() containing lat, lon, and suitability columns.
records	Dataframe of known presence/absence records. Must contain: • Coordinate columns (lat/lon or equivalents) • A presence column: 1 = present, 0 = absent (or logical TRUE/FALSE).
presence_col	Character. Name of the presence/absence column in records (default "presence").
match_radius_deg	Numeric. Radius in decimal degrees within which a prediction cell is matched to a record (default 0.002 approx. 220 m). Records that fall outside all prediction cells at this radius are dropped with a warning.
plot	Logical. Print an ASCII ROC curve to the console (default TRUE).
verbose	Logical. Print full metric summary (default TRUE).

Value

A named list:

- auc: numeric [0,1]
- optimal_threshold: numeric [0,1] – Youden's J
- sensitivity: numeric [0,1] at optimal threshold
- specificity: numeric [0,1] at optimal threshold
- f1: numeric [0,1] at optimal threshold
- tss: numeric [-1,1] at optimal threshold
- brier_score: numeric [0,1]
- n_presences: integer
- n_absences: integer
- n_unmatched: integer – records dropped due to no nearby prediction
- roc_df: dataframe with columns threshold, sensitivity, specificity (for custom plotting)

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
# Synthetic balanced presence/absence records for demonstration
set.seed(42)
records <- data.frame(
  lat      = result$lat[1:60],
  lon      = result$lon[1:60],
  presence = as.integer(runif(60) > 0.4)
)
val <- validate_against_records(result, records, presence_col = "presence")
val$auc
```

`variable_impact` *Summarise variable-level impact on suitability scores*

Description

Returns a ranked summary table showing, for each environmental variable that contributed to the suitability score, how important it was (weight), how well it scored on average, and what its net contribution to the final score was. This is the primary diagnostic for understanding which data layers are driving or limiting the suitability output.

Columns returned:

`variable` Variable name (matches `score_<variable>` columns in the result).

`rank` Species-defined importance rank (1 = highest priority).

`norm_weight_pct` Normalised weight as a percentage of the total weight across all variables present in the data. Higher = more influence.

`mean_score` Mean per-variable score across all non-excluded points that had data for this variable. Range 0-1.

`mean_contribution` `mean_score * norm_weight_pct / 100`. The average weighted contribution to the final suitability value. Sum of `mean_contribution` across all variables = approximate mean suitability.

`n_points` Number of non-excluded points with data for this variable.

`pct_coverage` Percentage of non-excluded points with data for this variable.

Usage

```
variable_impact(
  result,
  species,
  sort_by = "mean_contribution",
  exclude_zero = TRUE
)
```

Arguments

`result`	A dataframe returned by `predict_oyster()`.
`species`	Character string identifying the target oyster species. Same format as the `species` argument to `predict_oyster()`.
`sort_by`	Character. Column to sort the output by. One of `"mean_contribution"` (default), `"rank"`, `"mean_score"`, `"norm_weight_pct"`, or `"n_points"`.
`exclude_zero`	Logical. If `TRUE` (default), exclude variables that had zero coverage (no points with data). Set to `FALSE` to see all variables defined in the species tolerance spec.

Value

A dataframe ordered by `sort_by` (descending). Print it directly or pass to `knitr::kable()` for a formatted table.

Examples

```
sample_csv <- system.file("extdata", "sample_survey.csv", package = "oystermapR")
result <- predict_oyster(sample_csv, "ostrea_edulis", verbose = FALSE)
variable_impact(result, "ostrea_edulis")

# Sort by lowest-scoring variables (potential site limiters)
variable_impact(result, "ostrea_edulis", sort_by = "mean_score")
```

---

weight_table_md *Weight derivation as a markdown table*

Description

Renders the weight derivation and the coverage ledger as markdown, for pasting into a methods section or a supplementary file. Every number in the output is read back from the objects rather than transcribed, so re-rendering after a change refreshes the whole table.

Usage

```
weight_table_md(weights, coverage = NULL, title = NULL, file = NULL)
```

Arguments

`weights`	An `evidence_weights` object, or a data frame returned by `evidence_suitability()` carrying it as an attribute.
`coverage`	An `evidence_coverage` object. Taken from the attribute if `weights` is a scored data frame.
`title`	Optional heading.
`file`	Optional path. Written there if given, and returned invisibly.

Value

A character vector of markdown lines.

Index

.score_row(), *89*

add_intertidal_flag, 4
add_season_column, 5
add_season_column(), *17*
add_shellfish_classification, 6
add_suitability_ci, 7
analyse_connectivity, 8
analyse_connectivity(), *83, 84*
area_summary, 9
as.POSIXct(), *13*
as_lineage_distances, 12
as_lineage_distances(), *37*, *49*
assess_gear_feasibility, 11
assess_gear_feasibility(), *83, 84*
auto_tidal_correct, 13
auto_tidal_correct(), *5*

calculate_aragonite, 14
calculate_aragonite(), *59*
check_exclusions, 16
check_exclusions(), *89*
classify_substrate_from_backscatter, 18
classify_substrate_from_backscatter(), *12*
compare_species, 19
compare_species(), *39*
compare_surveys, 21
composite_seasonal, 22
confidence_score, 23
confidence_score(), *30*, *59*, *61*
correct_to_chart_datum, 24
correct_to_chart_datum(), *5*, *13, 14*, *101*

detect_season, 25
detect_season(), *5*

estimate_chlorophyll_from_backscatter, 26
estimate_fetch, 28
evidence_coverage, 29
evidence_coverage(), *31*, *61*
evidence_suitability, 30
evidence_suitability(), *107*
evidence_weights, 31
evidence_weights(), *30*, *45, 46*, *59–61*, *77*
export_contours, 32
export_contours(), *34*
export_geotiff, 33
export_geotiff(), *32*, *35*
export_qml_style, 35
export_qml_style(), *33*

fetch_live_environmental_data, 35
fit_lineages, 36
fit_lineages(), *47, 48*, *66*, *102*

generate_report, 39
generate_summary_pdf, 40
get_species_tolerances, 41
get_species_tolerances(), *8*, *17*, *29–31*, *42*, *62*, *89*
get_tolerance_posteriors, 42
get_tolerance_posteriors(), *103*

identify_resilient_sites, 43
interpolate_survey, 43

layer_reproducibility, 45
lineage_fits_as_list, 47
lineage_fits_from_list *(*lineage_fits_as_list*)*, 47
lineage_occurrence, 47
lineage_occurrence(), *38*, *66*
lineage_signal, 48
lineage_signal(), *38*
list_species, 49
list_species(), *20*, *59*, *102*
load_tolerance_update, 50

load_tolerance_update(), *79*, *103*

merge_sensor_data, 51
merge_sensor_data(), *13*, *25*, *67*, *69*, *71*, *73*, *74*, *76*, *100*

oystermapR_help, 52
oystermapR_live_config, 53
oystermapR_live_config(), *35*

parse_opendrift_connectivity, 54
permutation_importance, 55
plot_tolerance, 57
predict_oyster, 58
predict_oyster(), *6*, *8–10*, *12*, *16*, *19–22*, *25*, *33*, *34*, *39–41*, *43*, *50–52*, *55*, *57*, *62*, *80*, *82–85*, *87*, *90*, *91*, *93*, *94*, *96–98*, *104–106*
project_suitability, 62
project_suitability(), *43*

qc_survey_data, 63

rank_source_populations, 65
rank_source_populations(), *38*, *48*
read_aanderaa_csv, 66
read_generic_csv, 67
read_generic_csv(), *51*
read_nortek_adcp, 69
read_nortek_adcp(), *27*, *51*, *72*, *73*
read_nortek_aquadopp, 70
read_nortek_aquadopp(), *67*, *73*
read_rdi_adcp, 72
read_rdi_adcp(), *67*
read_sonar_tif, 73
read_soundings_xyz, 75
reliability_from_perturbation, 76
reliability_from_perturbation(), *31*, *59*
reliability_from_replicates, 77
reliability_from_replicates(), *31*, *45*, *46*, *59*, *77*
reset_tolerance_update, 78
reset_tolerance_update(), *103*

save_tolerance_update, 79
save_tolerance_update(), *50*, *101*, *103*
score_anthropogenic_disturbance, 80
score_disease_risk, 82
score_economic_viability, 83
score_hab_risk, 84
score_larval_connectivity, 86
score_larval_connectivity(), *55*
score_locations, 89
score_locations(), *4*, *6*, *29–31*, *50*, *103*
score_predation_risk, 89
score_sediment_stability, 91
score_settlement, 93
score_wave_exposure, 94
sensitivity_analysis, 95
sensitivity_analysis(), *57*
smooth_suitability, 97
spatial_block_cv, 98
stack_surveys, 100
stack_surveys(), *51*
stats::cor(), *77*

terra::terra, *10*
terra::terrain(), *73*, *74*
tidal_port_info, 101
tidal_port_info(), *14*

update_species_tolerances, 101
update_species_tolerances(), *36*, *38*, *42*

validate_against_records, 104
validate_against_records(), *39*, *56*, *99*
variable_impact, 106

weight_table_md, 107
weight_table_md(), *60*, *61*