# Package 'h3sdm'

April 15, 2026

**Type** Package

**Title** Species Distribution Modeling with H3 Grids

**Version** 0.1.0

**Description** Provides tools for species distribution modeling using H3 hexagonal grids (Uber Technologies Inc., 2022, <https://h3geo.org>). Facilitates retrieval of species occurrence records, generation of H3 grids, computation of landscape metrics, and preparation of spatial data for modern species distribution models workflows. Designed for biodiversity and landscape ecology research.

**URL** https://github.com/ManuelSpinola/h3sdm

**BugReports** https://github.com/ManuelSpinola/h3sdm/issues

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Depends** R (>= 4.1)

**Config/Needs/website** tidyverse/tidytemplate

**Imports** sf, dplyr, purrr, tibble, rlang, terra, spatialsample, recipes, rsample, tune, workflows, yardstick, ecospat, DALEX, stacks

**Suggests** ggplot2, paisaje, knitr, rmarkdown, here, tidyr, themis, DALEXtra, ingredients, exactextractr, landscapemetrics, h3jsr, tidyterra, spocc, tidymodels, workflowsets, ranger, xgboost, ggbrick, parsnip, tidyverse

**VignetteBuilder** knitr

**LazyData** true

**Language** en-US

**NeedsCompilation** no

**Author** Manuel Spínola [aut, cre]

**Maintainer** Manuel Spínola <mspinola10@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-04-15 13:00:29 UTC

# Contents



---

`bioclim_current` *Current bioclimatic raster*

---

## Description

A GeoTIFF with current bioclimatic variables for Costa Rica.

## Format

GeoTIFF file, readable with `terra::rast()`.

## Details

This file is stored in `inst/extdata/` and can be accessed with: `terra::rast(system.file("extdata", "bioclim_current.tif", package = "h3sdm"))`

### Examples

```
library(terra)
bio <- terra::rast(system.file("extdata", "bioclim_current.tif", package = "h3sdm"))
```

---

## bioclim_future *Future bioclimatic raster*

---

### Description

A GeoTIFF with projected bioclimatic variables for Costa Rica.

### Format

GeoTIFF file, readable with `terra::rast()`.

### Details

This dataset corresponds to the climate projection:

- Model: INM-CM4-8
- Scenario: SSP1-2.6
- Period: 2021–2040

The file is stored in `inst/extdata/` and can be accessed with: `terra::rast(system.file("extdata", "bioclim_future.tif", package = "h3sdm"))`

### Examples

```
library(terra)
bio <- terra::rast(system.file("extdata", "bioclim_future.tif", package = "h3sdm"))
```

---

## cr_outline_c *Costa Rica Continental Outline*

---

### Description

A simplified outline of Costa Rica as an `sf` object.

### Usage

```
cr_outline_c
```

### Format

An `sf` object containing polygon geometry of Costa Rica.

## Source

Adapted from publicly available geographic data.

## Examples

```
library(sf)
plot(cr_outline_c)
```

---

`h3sdm_calculate_it_metrics` *Calculate Information Theory Landscape Metrics for Hexagonal Grid*

---

## Description

Calculates 5 Information Theory (IT)-based landscape metrics (`condent`, `ent`, `joinent`, `mutinf`, `relmutinf`) for each hexagon in a given H3 hexagonal grid.

## Usage

```
h3sdm_calculate_it_metrics(landscape_raster, sf_grid)
```

## Arguments

| | |
|---|---|
| `landscape_raster` | A categorical SpatRaster containing land-cover data. |
| `sf_grid` | An `sf` object containing the hexagonal grid with species or land-cover data. |

## Details

This function computes landscape metrics using the `landscapemetrics::sample_lsm()` workflow. The results are pivoted to a wide format for easy use.

## Value

An `sf` object containing the input hex grid with new columns for each calculated metric.

## References

Hesselbarth et al., 2019. landscapemetrics: an open-source R tool to calculate landscape metrics. Ecography 42: 1648–1657.

Nowosad & Stepinski, 2019. Information theory as a consistent framework for landscape patterns. doi:10.1007/s1098001900830x

## Examples

```
library(sf)
library(terra)

# Create a categorical SpatRaster (land-cover map)
landscape_raster <- terra::rast(
  nrows = 30, ncols = 30,
  xmin = -85.0, xmax = -83.0,
  ymin = 9.0,  ymax = 11.0,
  crs = "EPSG:4326"
)
terra::values(landscape_raster) <- sample(1:4, terra::ncell(landscape_raster),
                                          replace = TRUE)
names(landscape_raster) <- "landcover"

# Create a simple hexagon grid as sf polygons
hex_grid <- sf::st_make_grid(
  sf::st_as_sfc(sf::st_bbox(c(
    xmin = -84.5, xmax = -83.5,
    ymin = 9.5,  ymax = 10.5
  ), crs = sf::st_crs(4326))),
  n = c(3, 3),
  square = FALSE
)
sf_grid <- sf::st_sf(h3_address = paste0("hex_", seq_along(hex_grid)),
                     geometry = hex_grid)

# Calculate Information Theory (IT) landscape metrics per hexagon
result_sf <- h3sdm_calculate_it_metrics(landscape_raster, sf_grid)
head(result_sf)
```

---

# h3sdm_classify *Classify predictions based on an optimal threshold*

---

## Description

Converts continuous probability predictions into binary presence/absence based on a specified threshold.

## Usage

```
h3sdm_classify(predictions_sf, threshold)
```

## Arguments

| | |
|---|---|
| predictions_sf | An sf object containing a numeric column named prediction, typically produced by h3sdm_predict(). |

threshold A numeric value representing the probability threshold (e.g., 0.45) above which predictions are classified as presence (1).

## Details

This function is useful for converting continuous probability outputs into binary presence/absence data for mapping or model evaluation purposes.

## Value

An sf object with the same geometry and all original columns, plus a new integer column predicted_presence with values 0 (absence) or 1 (presence).

## Examples

```
## Not run:
library(sf)
library(dplyr)

# Crear un sf de ejemplo
df <- data.frame(
  id = 1:5,
  prediction = c(0.2, 0.6, 0.45, 0.8, 0.3),
  lon = c(-75, -74, -73, -72, -71),
  lat = c(10, 11, 12, 13, 14)
)

df_sf <- st_as_sf(df, coords = c("lon", "lat"), crs = 4326)

# Clasificar usando un umbral
classified_sf <- h3sdm_classify(df_sf, threshold = 0.5)

# Revisar resultados
print(classified_sf)

## End(Not run)
```

---

# h3sdm_compare_models *Compare multiple H3SDM species distribution models*

---

## Description

Computes and combines performance metrics for multiple species distribution models created with h3sdm_fit_models() or similar workflows. Metrics include standard yardstick metrics (ROC AUC, TSS, Boyce index, etc.). Returns a tibble summarizing model performance.

## Usage

```
h3sdm_compare_models(h3sdm_results)
```

### Arguments

| | |
|---|---|
| h3sdm_results | A list or workflow set containing fitted models with a metrics tibble. Typically, this object is the output of h3sdm_fit_models(). |

### Value

A tibble with one row per model per metric, containing:

**model** Model name

**.metric** Metric name (ROC AUC, TSS, Boyce, etc.)

**.estimator** Metric type (usually "binary")

**mean** Metric value

### Examples

```
# Minimal reproducible example
example_metrics <- tibble::tibble(
  model = c("model1", "model2"),
  .metric = c("roc_auc", "tss_max"),
  .estimator = c("binary", "binary"),
  mean = c(0.85, 0.7)
)
example_results <- list(metrics = example_metrics)
h3sdm_compare_models(example_results)
```

---

## h3sdm_data *Combine species and environmental data for SDMs using H3 grids*

---

### Description

Combines species presence–absence data with environmental predictors. It also calculates centroid coordinates (x and y) for each hexagon grid cell.

### Usage

```
h3sdm_data(pa_sf, predictors_sf)
```

### Arguments

| | |
|---|---|
| pa_sf | An sf object from h3sdm_pa() containing species presence–absence data. |
| predictors_sf | An sf object from h3sdm_predictors() containing environmental predictors. |

### Value

An sf object containing species presence–absence, environmental predictor variables, and centroid coordinates for each hexagon cell.

## Examples

```
## Not run:
my_species_pa <- h3sdm_pa("Panthera onca", res = 6)
my_predictors <- h3sdm_predictors(my_species_pa)
combined_data <- h3sdm_data(my_species_pa, my_predictors)

## End(Not run)
```

---

`h3sdm_eval_metrics` *Evaluate performance metrics for a fitted H3SDM model*

---

## Description

Computes a set of performance metrics for a single fitted species distribution model. Includes standard yardstick metrics such as ROC AUC, accuracy, sensitivity, specificity, F1-score, Kappa, as well as ecological metrics such as the True Skill Statistic (TSS) and Boyce index. This function is designed as a helper for evaluating models produced by `h3sdm_fit_model` or `h3sdm_fit_models`.

## Usage

```
h3sdm_eval_metrics(
  fitted_model,
  presence_data = NULL,
  truth_col = "presence",
  pred_col = ".pred_1"
)
```

## Arguments

| | |
|---|---|
| `fitted_model` | A fitted model object, typically the output of `h3sdm_fit_model()`. |
| `presence_data` | Optional. An `sf` object or tibble containing presence locations used to compute the Boyce index. If not provided, the Boyce index will not be calculated. |
| `truth_col` | Character. Name of the column containing the true presence/absence values (default `"presence"`). |
| `pred_col` | Character. Name of the column containing predicted probabilities (default `".pred_1"`). |

## Details

This function centralizes model evaluation for a single fitted H3SDM model, combining both general classification metrics and ecological indices. It is especially useful for systematically comparing model performance across species or modeling approaches.

## Value

A tibble with one row per metric, containing:

**.metric** Metric name (e.g., "roc_auc", "tss", "boyce").

**.estimator** Estimator type (usually "binary").

**mean** Metric value.

**std_err** Standard error (NA for TSS and Boyce).

**conf_low** Lower bound of the 95% confidence interval (NA for TSS and Boyce).

**conf_high** Upper bound of the 95% confidence interval (NA for TSS and Boyce).

## Examples

```
## Not run:
# Assuming 'fitted' is the result of h3sdm_fit_model()
metrics <- h3sdm_eval_metrics(
  fitted_model = fitted,
  presence_data = presence_sf,
  truth_col = "presence",
  pred_col = ".pred_1"
)
print(metrics)

## End(Not run)
```

---

`h3sdm_explain` *Create a DALEX explainer for h3sdm workflows*

---

## Description

Creates a DALEX explainer for a species distribution model fitted with `h3sdm_fit_model()`. Prepares response and predictor variables, ensuring that all columns used during model training (including `h3_address` and coordinates) are included. The explainer can be used for feature importance, model residuals, and other DALEX diagnostics.

## Usage

```
h3sdm_explain(model, data, response = "presence", label = "h3sdm workflow")
```

## Arguments

| | |
|---|---|
| `model` | A fitted workflow returned by `h3sdm_fit_model()`. |
| `data` | A `data.frame` or `sf` object containing the original predictors and response variable. If an `sf` object, geometry is dropped automatically. |
| `response` | Character string specifying the name of the response column. Must be a binary factor or numeric vector (0/1). Defaults to `"presence"`. |
| `label` | Character string specifying a label for the explainer. Defaults to `"h3sdm workflow"`. |

## Value

An object of class `explainer` from the **DALEX** package, ready to be used with `feature_importance()`, `model_performance()`, `predict_parts()`, and other DALEX functions.

## Examples

```
library(h3sdm)
library(DALEX)
library(parsnip)

dat <- data.frame(
  x1 = rnorm(20),
  x2 = rnorm(20),
  presence = factor(sample(0:1, 20, replace = TRUE))
)

model <- logistic_reg() |>
  fit(presence ~ x1 + x2, data = dat)

explainer <- h3sdm_explain(model, data = dat, response = "presence")
feature_importance(explainer)
```

---

# h3sdm_extract_cat *Calculate Area Proportions for Categorical Raster Classes*

---

## Description

Extracts and calculates the **area proportion** of each land-use/land-cover (LULC) category found within each input polygon of the `sf_hex_grid`. This function is tailored for categorical rasters and ensures accurate, sub-pixel weighted statistics.

## Usage

```
h3sdm_extract_cat(spat_raster_cat, sf_hex_grid, proportion = TRUE)
```

## Arguments

| | |
|---|---|
| `spat_raster_cat` | A single-layer `SpatRaster` object containing categorical values (e.g., LULC classes). |
| `sf_hex_grid` | An `sf` object containing polygonal geometries (e.g., H3 hexagons). Must contain a column named `h3_address` for joining and grouping. |
| `proportion` | Logical. If `TRUE` (default), the output values are the proportion of the polygon area covered by each category (summing to 1 for covered area). If `FALSE`, the output is the raw sum of the coverage fraction (area). |

## Details

The function uses a custom function with `exactextractr::exact_extract` to perform three critical steps:

1. **Filtering NA/NaN:** Raster cells with missing values (`NA`) are explicitly excluded from the calculation, preventing the creation of a `_prop_NaN` column.
2. **Area Consolidation:** It sums the coverage fractions for all fragments belonging to the same category within the same hexagon, which is essential when polygons have been clipped or fragmented.
3. **Numerical Ordering:** The final columns are explicitly sorted based on the numerical value of the category (e.g., `_prop_70` appears before `_prop_80`) to correct the default alphanumeric sorting behavior of `tidyr::pivot_wider`.

## Value

An `sf` object identical to `sf_hex_grid`, but with new columns appended for each categorical value found in the raster. Column names follow the pattern `<layer_name>_prop_<category_value>`. Columns are **numerically ordered** by the category value.

## Examples

```
library(sf)
library(terra)

# Create a simple categorical SpatRaster
lulc <- terra::rast(
  nrows = 20, ncols = 20,
  xmin = -85.0, xmax = -83.0,
  ymin = 9.0,  ymax = 11.0,
  crs = "EPSG:4326"
)
terra::values(lulc) <- sample(1:4, terra::ncell(lulc), replace = TRUE)
names(lulc) <- "landuse"

# Define categorical levels explicitly
levels(lulc) <- data.frame(
  value = 1:4,
  class = c("forest", "grassland", "urban", "water")
)

# Create a simple hexagon grid as sf polygons (smaller than raster extent)
hex_grid <- sf::st_make_grid(
  sf::st_as_sfc(sf::st_bbox(c(
    xmin = -84.5, xmax = -83.5,
    ymin = 9.5,  ymax = 10.5
  ), crs = sf::st_crs(4326))),
  n = c(3, 3),
  square = FALSE
)
h7 <- sf::st_sf(h3_address = paste0("hex_", seq_along(hex_grid)),
                geometry = hex_grid)
```

```
# Extract categorical raster values by hexagon
lulc_p <- h3sdm_extract_cat(lulc, h7, proportion = TRUE)
head(lulc_p)
```

---

h3sdm_extract_num *Extract Area-Weighted Mean from Numeric Raster Stack*

---

## Description

Calculates the **area-weighted mean** value for each layer in a numeric SpatRaster (or single layer) within each polygon feature of an sf object. This function is designed to efficiently summarize continuous environmental variables (such as bioclimatic data) for predefined spatial units (e.g., H3 hexagons). It utilizes exactextractr to ensure highly precise zonal statistics by accounting for sub-pixel coverage fractions.

## Usage

```
h3sdm_extract_num(spat_raster_multi, sf_hex_grid)
```

## Arguments

| | |
|---|---|
| spat_raster_multi | A SpatRaster object from the terra package. Must contain numeric layers (can be a single layer or a stack/brick). |
| sf_hex_grid | An sf object containing polygonal geometries (e.g., H3 hexagons). Must be a valid set of polygons for extraction. |

## Details

The function relies on exactextractr::exact_extract with fun = "weighted_mean" and weights = "area". This methodology is crucial for maintaining spatial accuracy when polygons are irregular or small relative to the raster resolution. A critical check (nrow match) is performed before binding columns to ensure data integrity and prevent misalignment errors.

## Value

An sf object identical to sf_hex_grid, but with new columns appended. The new column names match the original SpatRaster layer names. The values represent the area-weighted mean for that variable within each polygon.

## Examples

```
library(sf)
library(terra)

# Create a SpatRaster stack with two numeric layers (e.g., bioclimatic variables)
bio1 <- terra::rast(
  nrows = 10, ncols = 10,
  xmin = -84.5, xmax = -83.5,
  ymin = 9.5,  ymax = 10.5,
  crs = "EPSG:4326"
)
bio2 <- bio1
terra::values(bio1) <- runif(terra::ncell(bio1), 15, 30)
terra::values(bio2) <- runif(terra::ncell(bio2), 500, 3000)
names(bio1) <- "bio1_temp"
names(bio2) <- "bio12_precip"
bio <- c(bio1, bio2)

# Create a simple hexagon grid as sf polygons
hex_grid <- sf::st_make_grid(
  sf::st_as_sfc(sf::st_bbox(c(
    xmin = -84.5, xmax = -83.5,
    ymin = 9.5,  ymax = 10.5
  ), crs = sf::st_crs(4326))),
  n = c(3, 3),
  square = FALSE
)
h7 <- sf::st_sf(h3_address = paste0("hex_", seq_along(hex_grid)),
                geometry = hex_grid)

# Extract numeric raster values by hexagon (mean per cell)
bio_p <- h3sdm_extract_num(bio, h7)
head(bio_p)
```

---

`h3sdm_fit_model` *Fits an SDM workflow to data using resampling and prepares it for stacking.*

---

## Description

Fits a Species Distribution Model (SDM) workflow to resampling data (cross-validation). This function is the main training step and optionally configures the results to be used with the 'stacks' package.

## Usage

```
h3sdm_fit_model(
  workflow,
```

```
  data_split,
  presence_data = NULL,
  truth_col = "presence",
  pred_col = ".pred_1",
  for_stacking = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| workflow | A 'workflow' object from tidymodels (e.g., GAM or Random Forest). |
| data_split | An 'rsplit' or 'rset' object (e.g., result of vfold_cv or spatial_block_cv). |
| presence_data | (Optional) Original presence data (used for extended metrics). |
| truth_col | Column name of the response variable (defaults to "presence"). |
| pred_col | Column name for the prediction of the class of interest (defaults to ".pred_1"). |
| for_stacking | Logical. If TRUE, uses control_stack_resamples() to save all workflow information required for the 'stacks' package. If FALSE, uses the standard control with save_pred = TRUE. |
| ... | Arguments passed on to other functions (e.g., to tune::fit_resamples if needed). |

## Value

A list with three elements:

- cv_model: The result of fit_resamples().
- final_model: The model fitted to the entire training set (first split).
- metrics: Extended evaluation metrics (if presence_data is provided).

---

# h3sdm_fit_models *Fit and evaluate multiple H3SDM species distribution models*

---

## Description

Fits one or more species distribution models using tidymodels workflows and a specified resampling scheme, then computes standard metrics (ROC AUC, accuracy, sensitivity, specificity, F1-score, Kappa) along with TSS (True Skill Statistic) and the Boyce index for model evaluation. Returns both the fitted models and a comparative metrics table.

## Usage

```
h3sdm_fit_models(
  workflows,
  data_split,
  presence_data = NULL,
  truth_col = "presence",
  pred_col = ".pred_1"
)
```

## Arguments

| | |
|---|---|
| workflows | A named list of tidymodels workflows created with h3sdm_workflow() or manually. |
| data_split | A resampling object (e.g., from vfold_cv() or h3sdm_spatial_cv()) for cross-validation. |
| presence_data | An sf object or tibble with presence locations to compute the Boyce index (optional). |
| truth_col | Character. Name of the column containing true presence/absence values (default "presence"). |
| pred_col | Character. Name of the column containing predicted probabilities (default ".pred_1"). |

## Value

A list with two elements:

**models** A list of fitted models returned by h3sdm_fit_model().

**metrics** A tibble with one row per model per metric, including standard yardstick metrics, TSS, and Boyce index.

## Examples

```
## Not run:
# Example requires prepared recipes and resampling objects
mod_log <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

mod_rf <- rand_forest() %>%
  set_engine("ranger") %>%
  set_mode("classification")

workflows_list <- list(
  logistic = h3sdm_workflow(mod_log, my_recipe),
  rf       = h3sdm_workflow(mod_rf, my_recipe)
)

results <- h3sdm_fit_models(
  workflows     = workflows_list,
  data_split    = my_cv_folds,
  presence_data = presence_sf
)
metrics_table <- results$metrics

## End(Not run)
```

h3sdm_get_grid *Generar cuadrícula H3 para un área de interés*

## Description

Crea una cuadrícula de hexágonos H3 que cubre un área de interés (sf_object), asegurando que las celdas se ajusten a la extensión del área y se recorten opcionalmente al contorno del AOI.

Esta función es equivalente a la usada en los módulos de paisaje de h3sdm, pero con el nombre estandarizado para mantener consistencia en el paquete.

## Usage

```
h3sdm_get_grid(sf_object, res = 6, expand_factor = 0.1, clip_to_aoi = TRUE)
```

## Arguments

| | |
|---|---|
| sf_object | Objeto sf que define el área de interés (AOI). |
| res | Entero entre 1 y 16. Define la resolución del índice H3. Valores mayores producen hexágonos más pequeños. |
| expand_factor | Valor numérico que amplía ligeramente el bounding box del AOI antes de generar los hexágonos. Por defecto 0.1. |
| clip_to_aoi | Lógico (TRUE o FALSE), indica si los hexágonos deben recortarse exactamente al contorno del AOI. Por defecto TRUE. |

## Value

Un objeto sf con los hexágonos H3 correspondientes al área de interés, con geometrías válidas (MULTIPOLYGON).

## Examples

```
## Not run:
library(sf)

# Crear un polígono de ejemplo
cr <- st_as_sf(data.frame(
  lon = c(-85, -85, -83, -83, -85),
  lat = c(9, 11, 11, 9, 9)
), coords = c("lon", "lat"), crs = 4326) |>
  summarise(geometry = st_combine(geometry)) |>
  st_cast("POLYGON")

# Generar cuadrícula H3
h5 <- h3sdm_get_grid(cr, res = 5)
plot(st_geometry(h5))

## End(Not run)
```

`h3sdm_get_records` *Query Species Occurrence Records within an H3 Area of Interest (AOI)*

## Description

Downloads species occurrence records from providers (e.g., GBIF) using the spocc package, filtering the initial query by the exact polygonal boundary of the Area of Interest (AOI) for maximum efficiency and precision.

## Usage

```
h3sdm_get_records(
  species,
  aoi_sf,
  providers = NULL,
  limit = 500,
  remove_duplicates = FALSE,
  date = NULL
)
```

## Arguments

| | |
|---|---|
| species | Character string specifying the species name to query (e.g., "Puma concolor"). |
| aoi_sf | An sf object defining the Area of Interest (AOI). Its CRS will be transformed to WGS84 (EPSG:4326) before query. |
| providers | Character vector of data providers to query (e.g., "gbif", "bison"). If NULL (default), all available providers are used. |
| limit | Numeric. The maximum number of records to retrieve per provider. Default is 500. |
| remove_duplicates | Logical. If TRUE, records with identical longitude and latitude are removed using dplyr::distinct(). Default is FALSE. |
| date | Character vector specifying a date range (e.g., c('2000-01-01', '2020-12-31')). |

## Details

The function transforms the aoi_sf polygon into a WKT string, which is used in the spocc::occ geometry argument for **efficient WKT-based querying**. Final spatial filtering is performed using sf::st_intersection to ensure strict containment. A critical check is included to prevent errors when the API returns no data (addressing the 'column not found' error).

## Value

An sf object of points containing the filtered occurrence records, with geometry confirmed to fall strictly within the aoi_sf boundary. If no records are found or the download fails, an empty sf object with the expected structure is returned.

## Examples

```
library(sf)

# Create a simple AOI polygon in Costa Rica
aoi_sf <- sf::st_as_sf(
  data.frame(
    lon = c(-84.5, -83.5, -83.5, -84.5, -84.5),
    lat = c(9.5, 9.5, 10.5, 10.5, 9.5)
  ) |>
    {\(d) sf::st_sfc(sf::st_polygon(list(as.matrix(d))), crs = 4326)}(),
  data.frame(id = 1)
)

records <- h3sdm_get_records(
  species = "Puma concolor",
  aoi_sf = aoi_sf,
  providers = "gbif",
  limit = 100
)

print(records)
```

---

# h3sdm_get_records_by_hexagon

*Download Species Records and Count Occurrences per H3 Hexagon*

---

## Description

This function downloads occurrence records for one or more species and counts the number of records falling inside each H3 hexagon covering the specified Area of Interest (AOI).

## Usage

```
h3sdm_get_records_by_hexagon(
  species,
  aoi_sf,
  res = 6,
  providers = NULL,
  remove_duplicates = FALSE,
  date = NULL,
  expand_factor = 0.1,
  limit = 500
)
```

## Arguments

| | |
|---|---|
| species | Character vector of species names to query (e.g., `c("Puma concolor", "Panthera onca")`). |

| | |
|---|---|
| aoi_sf | An sf polygon defining the Area of Interest (AOI). |
| res | Numeric. H3 resolution level (default 6), determining hexagon size. |
| providers | Character vector of data providers (e.g., "gbif"). If NULL, all providers are used. |
| remove_duplicates | Logical. If TRUE, duplicate coordinates are removed before counting. Default is FALSE. |
| date | Character vector specifying a date range (e.g., c('2000-01-01','2020-12-31')). |
| expand_factor | Numeric. Factor to expand the AOI bounding box before generating the H3 grid. Default is 0.1. |
| limit | Numeric. Maximum number of records to retrieve per species per provider. Default is 500. |

## Details

Download Species Records and Count Occurrences per H3 Hexagon

For each species:

1. An H3 grid is generated across the AOI using h3sdm_get_grid().
2. Occurrence records are downloaded using h3sdm_get_records().
3. Points are joined to the hexagonal grid with sf::st_join().
4. Counts of points per hexagon are calculated.
5. Counts are merged into the main hex grid.

The function ensures column names derived from species names are safe in R by replacing spaces with underscores and handles API failures gracefully.

## Value

An sf object containing the H3 hexagonal grid (MULTIPOLYGON) with additional integer columns for each species (spaces replaced by underscores) showing the count of occurrence records in each hexagon. Hexagons with no records have 0.

## See Also

h3sdm_get_grid, h3sdm_get_records

## Examples

```
library(sf)

# Create a simple AOI polygon in Costa Rica
aoi_sf <- sf::st_as_sf(
  data.frame(id = 1),
  geometry = sf::st_sfc(
    sf::st_polygon(list(matrix(
      c(-84.5, 9.5,
        -83.5, 9.5,
        -83.5, 10.5,
```

```
        -84.5, 10.5,
        -84.5, 9.5),
      ncol = 2, byrow = TRUE
    ))),
    crs = 4326
  )
)

hex_counts <- h3sdm_get_records_by_hexagon(
  species = c("Agalychnis callidryas", "Smilisca baudinii"),
  aoi_sf  = aoi_sf,
  res     = 7,
  providers = "gbif",
  limit   = 100
)

print(hex_counts)
```

---

h3sdm_pa *Generate presence/pseudo-absence dataset for a species*

---

## Description

Generates a hexagonal grid over the AOI, assigns species presence records to hexagons, and samples pseudo-absences from hexagons with no records.

## Usage

```
h3sdm_pa(
  species,
  aoi_sf,
  res = 6,
  n_pseudoabs = 500,
  providers = NULL,
  remove_duplicates = FALSE,
  date = NULL,
  limit = 500,
  expand_factor = 0.1
)
```

## Arguments

| | |
|---|---|
| species | character Species name (single string) for which records are requested. |
| aoi_sf | sf AOI (area of interest) polygon. |
| res | integer H3 resolution for the hexagonal grid. |
| n_pseudoabs | integer Number of pseudo-absence hexagons to sample. |

| | |
|---|---|
| providers | character Optional vector of data providers (e.g., "gbif", "inat"). |
| remove_duplicates | logical Remove duplicate records at the same coordinates. |
| date | character Optional date filter for records. |
| limit | integer Maximum number of records to download. |
| expand_factor | numeric Factor to expand AOI before creating hex grid. |

## Value

sf object with columns:

- h3_address: H3 index of the hexagon.
- presence: factor with levels "0" (pseudo-absence) and "1" (presence).
- geometry: MULTIPOLYGON of each hexagon.

## Examples

```
## Not run:
data(cr_outline_c, package = "h3sdm")
dataset <- h3sdm_pa("Agalychnis callidryas", cr_outline_c, res = 7, n_pseudoabs = 100)

## End(Not run)
```

---

# h3sdm_predict *Predict species presence probability using H3 hexagons*

---

## Description

Uses a fitted tidymodels workflow (from h3sdm_fit_model or a standalone workflow) to predict species presence probabilities on a new spatial H3 grid. Automatically generates centroid coordinates (x and y) if missing. The new_data must contain the same predictor variables as used in model training.

## Usage

```
h3sdm_predict(fit_object, new_data)
```

## Arguments

| | |
|---|---|
| fit_object | A fitted tidymodels workflow or the output list from h3sdm_fit_model. |
| new_data | An sf object containing the spatial grid and the same predictor variables used for model training. |

## Value

An sf object with the original geometry and a new column prediction containing the predicted probability of presence for each hexagon.

## Examples

```
## Not run:
# Predict presence probabilities on a new hex grid
predictions_sf <- h3sdm_predict(
  fit_object = fitted_model,
  new_data   = grid_sf
)

## End(Not run)
```

---

h3sdm_predictors *Combine Predictor Data from Multiple sf Objects*

---

## Description

This function merges predictor variables from multiple `sf` objects into a single `sf` object. It preserves the geometry from the first input and joins columns from the other `sf` objects using a common key (`h3_address` or `ID`).

## Usage

```
h3sdm_predictors(...)
```

## Arguments

| | |
|---|---|
| `...` | Two or more `sf` objects containing predictor variables. The first object must contain the geometry to preserve. All objects must share a common key column (`h3_address` or `ID`). |

## Details

The function uses a left join based on the `h3_address` column if present, otherwise it falls back to `ID`. Geometries from the right-hand side `sf` objects are dropped to avoid conflicts, and the final geometry is cast to `MULTIPOLYGON`.

## Value

An `sf` object containing the geometry of the first input and all predictor columns from all provided `sf` objects.

## Examples

```
## Not run:
# Combine sf objects with different predictor types into one
combined <- h3sdm_predictors(num_sf, cat_sf, it_sf)
head(combined)

## End(Not run)
```

h3sdm_recipe *Create a tidymodels recipe for H3-based SDMs*

## Description

Prepares an `sf` object with H3 hexagonal data for modeling with the `tidymodels` ecosystem. Extracts centroid coordinates, assigns appropriate roles to the variables automatically, and returns a ready-to-use recipe for modeling species distributions.

## Usage

```
h3sdm_recipe(data)
```

## Arguments

| | |
|---|---|
| `data` | An `sf` object, typically the output of `h3sdm_data()`, including species presence-absence, H3 addresses, and environmental predictors. The geometry must be of type `MULTIPOLYGON`. |

## Details

This function prepares spatial H3 grid data for species distribution modeling:

- Extracts centroid coordinates (`x` and `y`) from MULTIPOLYGON geometries using sf functions.
- Removes the geometry column to create a purely tabular dataset for tidymodels.
- Assigns roles to columns:
  - `presence` → `"outcome"` (target variable)
  - `h3_address` → `"id"` (used for joining predictions later)
  - `x` and `y` → `"spatial_predictor"`
- All other columns are assigned `"predictor"` role.

## Value

A `tidymodels` recipe object (class `"h3sdm_recipe"`) ready for modeling.

## Examples

```
## Not run:
# Example: Prepare H3 hexagonal SDM data for modeling
# `combined_data` is typically the output of h3sdm_data()
sdm_recipe <- h3sdm_recipe(combined_data)
sdm_recipe  # inspect the recipe object

## End(Not run)
```

`h3sdm_recipe_gam` *Creates a 'recipe' object for Generalized Additive Models (GAM) in SDM.*

## Description

This function prepares an `sf` (Simple Features) object for use in a Species Distribution Model (SDM) workflow with the 'mgcv' GAM engine within the 'tidymodels' ecosystem.

The crucial step is extracting the coordinates (x, y) from the geometry and assigning them the **predictor** role so they can be used in the GAM's spatial smooth term (`s(x, y, bs = "tp")`). It also assigns special roles to the 'presence' and 'h3_address' variables.

## Usage

```
h3sdm_recipe_gam(data)
```

## Arguments

| | |
|---|---|
| `data` | An `sf` (Simple Features) object containing the species presence/absence/abundance data, environmental variables (e.g., bioclimatic), and the geometry (e.g., H3 centroids or points). |

## Details

**Assigned Roles:**

- `outcome`: "presence" (or the column containing the response variable).
- `id`: "h3_address" (cell identifier, not used for modeling).
- `predictor`: All other variables, including **x** and **y** for the GAM's smoothing function.

**Note on x and y:** The `x` and `y` coordinates are added to the recipe's internal data frame and are defined as **predictor** to meet the requirements of the `mgcv` engine.

## Value

A `recipe` object of class `h3sdm_recipe_gam`, ready to be chained with additional preprocessing steps (e.g., normalization).

## See Also

Other h3sdm_tools: `h3sdm_stack_fit()`, `h3sdm_workflow_gam()`

## Examples

```
library(sf)
library(recipes)

# Create a simple sf object with presence/absence data
# and simulated environmental variables
set.seed(42)
n <- 20

pts <- sf::st_as_sf(
  data.frame(
    h3_address = paste0("hex_", seq_len(n)),
    presence   = sample(0:1, n, replace = TRUE),
    bio1_temp  = runif(n, 15, 30),
    bio12_precip = runif(n, 500, 3000)
  ),
  geometry = sf::st_sfc(
    lapply(seq_len(n), function(i) {
      sf::st_point(c(runif(1, -84.5, -83.5), runif(1, 9.5, 10.5)))
    }),
    crs = 4326
  )
)

# Create a GAM recipe with spatial coordinates as predictors
gam_rec <- h3sdm_recipe_gam(pts)

# Optionally add normalization to bioclimatic variables
final_rec <- gam_rec |>
  recipes::step_normalize(recipes::starts_with("bio"))

print(final_rec)
```

---

# h3sdm_spatial_cv *Create a spatial-aware cross-validation split for H3 data*

---

## Description

Generates a spatially aware cross-validation split for species distribution modeling using H3 hexagonal grids. This helps avoid inflated model performance estimates caused by spatial autocorrelation, producing more robust model evaluation.

## Usage

```
h3sdm_spatial_cv(data, method = "block", v = 5, ...)
```

### Arguments

| | |
|---|---|
| data | An sf object, typically the output of h3sdm_data(). |
| method | Character. The spatial resampling method to use: |
| | **"block"** Use spatialsample::spatial_block_cv() for block-based spatial CV. |
| | **"cluster"** Use spatialsample::spatial_clustering_cv() for cluster-based spatial CV. |
| v | Integer. Number of folds (default = 5). |
| ... | Additional arguments passed to the underlying spatialsample function. |

### Details

Spatial cross-validation avoids overly optimistic performance estimates by ensuring that training and testing data are spatially separated.

- "block": Divides the spatial domain into contiguous blocks.
- "cluster": Groups locations into spatial clusters before splitting.

### Value

An rsplit object (from rsample) representing the spatial CV folds.

### Examples

```
## Not run:
# Example: Create spatial cross-validation splits for H3 data

# Block spatial CV with default folds
spatial_cv_block <- h3sdm_spatial_cv(combined_data, method = "block")

# Cluster spatial CV with 10 folds
spatial_cv_cluster <- h3sdm_spatial_cv(combined_data, method = "cluster", v = 10)

## End(Not run)
```

---

## h3sdm_stack_fit *Creates and fully fits an ensemble model (Stack).*

---

### Description

This function combines the process of creating the model stack, optimizing the weights (blend_predictions), and fitting the base models to the complete training set (fit_members()) into a single step.

**Warning:** It does not follow the canonical tidymodels flow but is convenient. It requires that the fitting results were generated using h3sdm_fit_model(..., for_stacking = TRUE).

## Usage

```
h3sdm_stack_fit(..., non_negative = TRUE, metric = NULL)
```

## Arguments

| | |
|---|---|
| ... | List objects that are the result of h3sdm_fit_model(). Each object must contain the cv_model element (result of fit_resamples). |
| non_negative | Logical. If TRUE (default), forces the candidate model weights to be non-negative. |
| metric | The metric used to optimize the combination of weights. |

## Value

A list containing two elements: blended_model (the stack after blending) and final_model (a fully fitted model_stack object). The final_model is ready for direct prediction with predict().

## See Also

Other h3sdm_tools: h3sdm_recipe_gam(), h3sdm_workflow_gam()

---

# h3sdm_workflow *Create a tidymodels workflow for H3-based SDMs*

---

## Description

Combines a model specification and a prepared recipe into a single tidymodels workflow. This workflow is suitable for species distribution modeling using H3 hexagonal grids and can be directly fitted or cross-validated.

## Usage

```
h3sdm_workflow(model_spec, recipe)
```

## Arguments

| | |
|---|---|
| model_spec | A tidymodels model specification (e.g., logistic_reg(), rand_forest(), or boost_tree()), describing the model type and engine to use for fitting. |
| recipe | A tidymodels recipe object, typically created with h3sdm_recipe(), which preprocesses the data and defines predictor/response roles. |

## Details

The function creates a workflow that combines preprocessing and modeling steps. This encapsulation allows consistent model training and evaluation with tidymodels functions like fit() or fit_resamples(), and is particularly useful when applying multiple models in parallel.

## Value

A `workflow` object ready to be used for model fitting with `fit()` or cross-validation.

## Examples

```
## Not run:
library(parsnip)
# Example: Create a tidymodels workflow for H3-based species distribution modeling

# Step 1: Define model specification
my_model_spec <- logistic_reg() %>%
  set_mode("classification") %>%
  set_engine("glm")

# Step 2: Create recipe
my_recipe <- h3sdm_recipe(combined_data)

# Step 3: Combine into workflow
sdm_wf <- h3sdm_workflow(model_spec = my_model_spec, sdm_recipe = my_recipe)

## End(Not run)
```

---

# h3sdm_workflows *Create multiple tidymodels workflows for H3-based SDMs*

---

## Description

Creates a list of tidymodels workflows from multiple model specifications and a prepared recipe. This is useful for comparing different modeling approaches in species distribution modeling using H3 hexagonal grids. The returned workflows can be used for model fitting and resampling.

## Usage

```
h3sdm_workflows(model_specs, recipe)
```

## Arguments

| | |
|---|---|
| `model_specs` | A named list of `tidymodels` model specifications (e.g., `logistic_reg()`, `rand_forest()`, `boost_tree()`), where each element specifies a different modeling approach to be included in the workflow set. |
| `recipe` | A `tidymodels` recipe object, typically created with `h3sdm_recipe()`, which prepares and preprocesses the data for modeling. |

## Details

This function automates the creation of workflows for multiple model specifications. Each workflow combines the same preprocessing steps (recipe) with a different modeling method. This facilitates systematic comparison of models and is especially useful in ensemble and stacking approaches.

## Value

A named list of workflow objects, one per model specification.

## Examples

```
## Not run:
# ... (examples are correct as is) ...

## End(Not run)
```

---

h3sdm_workflow_gam *Creates a tidymodels workflow for Generalized Additive Models (GAM).*

---

## Description

This function constructs a workflow object by combining a GAM model specification (gen_additive_mod with the mgcv engine) with either a recipe object or an explicit model formula.

It is optimized for Species Distribution Models (SDM) that use smooth splines, ensuring that the specialized GAM formula (containing s() terms) is correctly passed to the model, even when a recipe is provided for general data preprocessing.

## Usage

```
h3sdm_workflow_gam(gam_spec, recipe = NULL, formula = NULL)
```

## Arguments

| | |
|---|---|
| gam_spec | A parsnip model specification of type gen_additive_mod(), configured with set_engine("mgcv"). |
| recipe | (Optional) A recipes package recipe object (e.g., the output of h3sdm_recipe_gam). Used for general data preprocessing like normalization or dummy variable creation. |
| formula | (Optional) A formula object that defines the structure of the GAM, including smooth terms (e.g., y ~ s(x1) + s(x, y)). If provided alongside recipe, this formula overrides the recipe's implicit formula for the final model fit. |

## Details

**Formula Priority:**

- If **only** recipe is provided, the workflow uses the recipe's implicit formula (e.g., outcome ~ .).
- If recipe **and** formula are provided, the workflow uses the recipe for data preprocessing but explicitly passes the formula to the mgcv engine for fitting, enabling the use of specialized terms like s(x, y).

## Value

A `workflow` object, ready for fitting with `fit()` or resampling with `fit_resamples()` or `tune_grid()`.

## See Also

Other h3sdm_tools: `h3sdm_recipe_gam()`, `h3sdm_stack_fit()`

## Examples

```
## Not run:
library(parsnip)
# 1. Define the model specification
gam_spec <- gen_additive_mod() %>%
  set_engine("mgcv") %>%
  set_mode("classification")

# 2. Define a specialized GAM formula
gam_formula <- presence ~ s(bio1) + s(x, y, bs = "tp")

# 3. Define a base recipe (assuming 'data' exists)
# base_rec <- h3sdm_recipe_gam(data)

# 4. Create the combined workflow
# h3sdm_wf <- h3sdm_workflow_gam(
#   gam_spec = gam_spec,
#   recipe = base_rec,
#   formula = gam_formula
# )

## End(Not run)
```

---

# `records` *Presence/pseudo-absence records for Silverstoneia flotator*

---

## Description

A dataset containing presence and pseudo-absence records for the species *Silverstoneia flotator* in Costa Rica, generated using H3 hexagonal grids at resolution 7.

## Usage

```
records
```

## Format

An `sf` object with columns:

**h3_address** H3 index of the hexagon

**presence** factor with levels "0" (pseudo-absence) and "1" (presence)

**geometry** MULTIPOLYGON of each hexagon

## Source

Generated using `h3sdm_pa()` with occurrence data from GBIF (https://www.gbif.org).

## Examples

```
data(records)
head(records)
table(records$presence)
```

# Index