Package ‘ebirdst’

May 9, 2023

Type Package

Title Access and Analyze eBird Status and Trends Data

Version 2.2021.3

Description Tools to download, load, plot, and analyze eBird Status and Trends Data Products (<https://science.ebird.org/en/status-and-trends>). eBird (<https://ebird.org/home>) is a global database of bird observations collected by member of the public. eBird Status and Trends uses these data to model global bird abundances, range boundaries, and habitat associations at a high spatial and temporal resolution.

License GPL-3

URL https://github.com/ebird/ebirdst

BugReports https://github.com/ebird/ebirdst/issues

Depends R (>= 4.0.0)

Imports DBI, dplyr (>= 1.0.0), ggplot2, grDevices, gridExtra, jsonlite, magrittr, rlang, RSQLite, sf (>= 1.0-0), stats, stringr, terra (>= 1.6-3), tidyr (>= 1.0.0), tools, utils, viridisLite

Suggests fields, gbm, geodata, knitr, mgcv, precrec, PresenceAbsence, rmarkdown, rnaturalearth, testthat, withr

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Matthew Strimas-Mackey [aut, cre] (<https://orcid.org/0000-0001-8929-7776>),
Shawn Ligocki [aut],
Tom Auer [aut] (<https://orcid.org/0000-0001-8619-7147>),
Daniel Fink [aut] (<https://orcid.org/0000-0002-8368-1248>),
Cornell Lab of Ornithology [cph]

Maintainer Matthew Strimas-Mackey <mes335@cornell.edu>

Repository CRAN

Date/Publication 2023-05-09 07:00:06 UTC

R topics documented:

abundance_palette *eBird Status and Trends color palettes for mapping*

Description

Generate the color palettes used for the eBird Status and Trends relative abundance maps.

Usage

```
abundance_palette(
  n,
  season = c("weekly", "breeding", "nonbreeding", "migration", "prebreeding_migration",
    "postbreeding_migration", "year_round")
)
```

Arguments

n	integer; the number of colors to be in the palette.
season	character; the season to generate colors for or "weekly" to get the color palette used in the weekly abundance animations.

Value

A character vector of hex color codes.

Examples

```
# breeding season color palette
abundance_palette(10, season = "breeding")
```

assign_to_grid *Assign points to a spacetime grid*

Description

Given a set of points in space and (optionally) time, define a regular grid with given dimensions, and return the grid cell index for each point.

Usage

```
assign_to_grid(
  points,
  coords = NULL,
  is_lonlat = FALSE,
  res,
  jitter_grid = TRUE,
  grid_definition = NULL
)
```

Arguments

- `points` — data frame; points with spatial coordinates `x` and `y`, and an optional time coordinate `t`.
- `coords` — character; names of the spatial and temporal coordinates in the input dataframe. Only provide these names if you want to overwrite the default coordinate names: `c("x", "y", "t")` or `c("longitude", "latitude", "t")` if `is_lonlat` = TRUE.
- `is_lonlat` — logical; if the points are in unprojected, lon-lat coordinates. In this case, the input data frame should have columns `"longitude"` and `"latitude"` and the points will be projected to an equal area sinusoidal CRS prior to grid assignment.
- `res` — numeric; resolution of the grid in the `x`, `y`, and `t` dimensions, respectively. If only 2 dimensions are provided, a space only grid will be generated. The units of `res` are the same as the coordinates in the input data unless `is_lonlat` is true in which case the `x` and `y` resolution should be provided in meters.
- `jitter_grid` — logical; whether to jitter the location of the origin of the grid to introduce some randomness.
- `grid_definition` — list; object defining the grid via the `origin` and `resolution` components. To assign multiple sets of points to exactly the same grid, `assign_to_grid()` returns a data frame with a `grid_definition` attribute that can be passed to subsequent calls to `assign_to_grid()`. `res` and `jitter` are ignored if `grid_definition` is provided.

Value

Data frame with the indices of the space-only and spacetime grid cells. This data frame will have a `grid_definition` attribute that can be used to reconstruct the grid.

Examples

```
set.seed(1)

# generate some example points
points_xyt <- data.frame(x = runif(100), y = runif(100), t = rnorm(100))
# assign to grid
cells <- assign_to_grid(points_xyt, res = c(0.1, 0.1, 0.5))

# assign a second set of points to the same grid
assign_to_grid(points_xyt, grid_definition = attr(cells, "grid_definition"))

# assign lon-lat points to a 10km space-only grid
points_ll <- data.frame(longitude = runif(100, min = -180, max = 180),
                        latitude = runif(100, min = -90, max = 90))
assign_to_grid(points_ll, res = c(10000, 10000), is_lonlat = TRUE)

# overwrite default coordinate names, 5km by 1 week grid
points_names <- data.frame(lon = runif(100, min = -180, max = 180),
                           lat = runif(100, min = -90, max = 90),
                           day = sample.int(365, size = 100))
assign_to_grid(points_names,
```

```
                  res = c(5000, 5000, 7),
                  coords = c("lon", "lat", "day"),
                  is_lonlat = TRUE)
```

`bernoulli_dev` *Bernoulli deviance*

Description

Bernoulli deviance

Usage

```
bernoulli_dev(obs, pred)
```

Arguments

`obs`	numeric; observed values.
`pred`	numeric; predicted values.

Value

A named numeric vector with three elements: model deviance, mean deviance, and deviance explained.

Examples

```
obs <- c(1, 1, 1, 0, 0, 0)
pred <- c(0.9, 0.8, 0.7, 0.3, 0.1, 0.2)
ebirdst:::bernoulli_dev(obs, pred)
```

`binom_test_p` *Binomial test for ensemble support*

Description

Binomial test for ensemble support

Usage

```
binom_test_p(x)
```

Arguments

`x`	numeric; named numeric vector with values for `pat`, `pi_es`, and `pat_cutoff`.

Value

A numeric p-value.

Examples

```
ebirdst:::binom_test_p(c(pat = 0.1, pi_es = 75))
```

`calculate_mcc_f1` *Calculate MCC and F1 score*

Description

Given binary observed and predicted response, estimate Matthews correlation coefficient (MCC) and the F1 score.

Usage

```
calculate_mcc_f1(observed, predicted)
```

Arguments

`observed`	logical or 0/1; the observed binary response.
`predicted`	logical or 0/1; the predicted binary response. This will typically need to be generated by applying a threshold to the continuous predicted response.

Value

A list with two elements: `mcc` and `f1`.

Examples

```
obs <- c(rep(1L, 1000L), rep(0L, 10000L))
pred <- c(rbeta(300L, 12, 2), rbeta(700L, 3, 4), rbeta(10000L, 2, 3))
calculate_mcc_f1(obs > 0, pred > 0.5)
```

date_to_st_week — *Get the Status and Trends week that a date falls into*

Description

Get the Status and Trends week that a date falls into

Usage

```
date_to_st_week(dates)
```

Arguments

`dates`	a vector of dates.

Value

An integer vector of weeks numbers from 1-52.

Examples

```
d <- as.Date(c("2016-04-08", "2018-12-31", "2014-01-01", "2018-09-04"))
date_to_st_week(d)
```

ebirdst — *`ebirdst`: Tools to Load, Map, Plot, and Analyze eBird Status and Trends Data Products*

Description

Tools to load, map, plot, and analyze eBird Status and Trends data products

ebirdst_data_dir — *Path to eBird Status and Trends data download directory*

Description

Identify and return the path to the default download directory for eBird Status and Trends data products. This directory can be defined by setting the environment variable `EBIRDST_DATA_DIR`, otherwise the directory returned by `tools::R_user_dir("ebirdst", which = "data")` will be used.

Usage

```
ebirdst_data_dir()
```

Value

The path to the data download directory.

Examples

```
ebirdst_data_dir()
```

---

ebirdst_download *Download eBird Status and Trends Data Products*

Description

Download an eBird Status and Trends data package for a single species, or for an example species. Accessing Status and Trends data requires an access key, consult `set_ebirdst_access_key()` for instructions on how to obtain and store this key. The example data consist of the results for Yellow-bellied Sapsucker subset to Michigan and are much smaller than the full dataset, making these data quicker to download and process. Only the low resolution data are available for the example data In addition, the example data are accessible without an access key.

Usage

```
ebirdst_download(
  species,
  path = ebirdst_data_dir(),
  tifs_only = TRUE,
  force = FALSE,
  show_progress = TRUE,
  pattern = NULL,
  dry_run = FALSE
)
```

Arguments

`species`	character; a single species given as a scientific name, common name or six-letter species code (e.g. woothr). The full list of valid species is can be viewed in the ebirdst_runs data frame included in this package. To download the example dataset, use "example_data".
`path`	character; directory to download the data to. All downloaded files will be placed in a sub-directory of this directory named for the data version year, e.g. "2020" for the 2020 Status Data Products. Each species' data package will then appear in a directory named with the eBird species code. Defaults to a persistent data directory, which can be found by calling `ebirdst_data_dir()`.
`tifs_only`	logical; whether to only download the GeoTIFFs for abundance and occurrence (the default), or download the entire data package, including data for predictor importance, partial dependence, and predictive performance metrics.

force	logical; if the data have already been downloaded, should a fresh copy be downloaded anyway.
show_progress	logical; whether to print download progress information.
pattern	character; regular expression pattern to supply to str_detect() to filter files to download. Note that some files are mandatory and will always be downloaded.
dry_run	logical; whether to do a dry run, just listing files that will be downloaded. This can be useful when testing the use of `pattern` to filter the files to download.

Value

Path to the folder containing the downloaded data package for the given species. If `dry_run = TRUE` a list of files to download will be returned.

Examples

```
## Not run:
# download the example data
ebirdst_download("example_data")

# download the data package for wood thrush, geotiffs only
ebirdst_download("woothr")
# download the data package for wood thrush, all data
ebirdst_download("woothr", tifs_only = FALSE)

# use pattern to only download low resolution data
# dry_run can be used to see what files will be downloaded
ebirdst_download("lobcur", pattern = "_lr_", dry_run = TRUE)
# use pattern to only download the high res weekly abundance data
ebirdst_download("lobcur", pattern = "abundance_median_hr", dry_run = TRUE)

## End(Not run)
```

---

ebirdst_extent — *Construct a spatiotemporal extent object to subset Status and Trends data*

Description

`ebirdst_extent` object are used to subset the eBird Status and Trends data spatially and/or temporally. This function constructs these objects.

Usage

```
ebirdst_extent(x, t, ...)

## S3 method for class 'bbox'
ebirdst_extent(x, t, ...)
```

```
## S3 method for class 'numeric'
ebirdst_extent(x, t, crs = 4326, ...)

## S3 method for class 'sfc'
ebirdst_extent(x, t, ...)

## S3 method for class 'sf'
ebirdst_extent(x, t, ...)
```

Arguments

x	the spatial extent; either a rectangular bounding box (defined as a vector of numbers representing the coordinates of the boundaries or an `st_bbox()` object) or a polygon (an sf object). See Details for further explanation of the format of x.
t	the temporal extent; a 2-element vector of the start and end dates of the temporal extent, provided either as dates (Date objects or strings in ISO format "YYYY-MM-DD") or numbers between 0 and 1 representing the fraction of the year. Note that dates can wrap around the year, e.g. ‘c("2021-12-01", "2021-01-31") is acceptable. See Details for further explanation of the format of t. **Leave the argument blank to include the full year of data.**
...	Additional arguments used by methods.
crs	coordinate reference system, provided as a `crs` object or argument to `st_crs()`. Defaults to unprojected, lat/long coordinates (crs = 4326). **Only required if x is given as a numeric vector defining the bounding box, ignored in all other cases.**

Details

The spatial extent, `x`, can be either a rectangular bounding box or a set of spatial polygons. The bounding box can be defined either as an `st_bbox()` object or by providing the coordinates of the rectangle edges directly as a named vector with elements xmin, xmax, ymin, and ymax (note that latitude and longitude correspond to y and x, respectively). In this latter case, a coordinate reference system must be provided explicitly via the `crs` argument (`crs = 4326` is the default and is a short form for unprojected lat/long coordinates). For a polygon spatial extent, `x` should be either an sf or sfc object (with feature type `POLYGON` or `MULTIPOLYGON`) from the `sf` package. To import data from a Shapefile or GeoPackage into this format, use `read_sf()`.

The temporal extent defines the start and end dates of the time period. These are most easily provided as Date objects or date strings in ISO format ("YYYY-MM-DD"). If dates are defined as strings, the year can be omitted (i.e. "MM-DD"). Alternatively, dates can be defined in terms of fractions of the year, e.g. `t = c(0.25, 0.5)` would subset to data within the second quarter of the year. In all cases, dates can wrap around the year, e.g. c("2021-12-01", "2021-01-31") would subset to data in December or January.

Value

An `ebirdst_extent` object consisting of a list with three elements: the spatial extent `extent`, the temporal extent `t`, and `type` (either "bbox" or "polygon").

Methods (by class)

- `ebirdst_extent(bbox)`: bounding box created with `st_bbox()`
- `ebirdst_extent(numeric)`: bounding box given as edges
- `ebirdst_extent(sfc)`: polygons as sfc spatial feature column
- `ebirdst_extent(sf)`: polygons as sf object

Examples

```
# bounding box of the north eastern united stats as a numeric vector
bb_vec <- c(xmin = -80, xmax = -70, ymin = 40, ymax = 47)
ebirdst_extent(bb_vec)

# bbox object
bb <- sf::st_bbox(bb_vec, crs = 4326)
ebirdst_extent(bb)

# polygon imported from a shapefile
poly <- sf::read_sf(system.file("shape/nc.shp", package="sf"))
ebirdst_extent(poly)

# subset to january
ebirdst_extent(bb, t = c("2021-01-01", "2021-01-31"))

# dates can wrap around, e.g. to use dec-jan
ebirdst_extent(bb, t = c("2021-12-01", "2021-01-31"))

# dates can also be given without an associated year
ebirdst_extent(bb, t = c("12-01", "01-31"))
```

`ebirdst_habitat` *eBird Status and Trends predictive habitat associations*

Description

Combine the predictor importance (PI) and partial dependence (PD) data to provide an estimate of the importance and directionality of the land cover classes (i.e. habitat) used as covariates in the occurrence probability model. **Note:** This is one of, if not the most, computationally expensive operations in the package.

Usage

```
ebirdst_habitat(path, ext, data = NULL, stationary_associations = FALSE)

## S3 method for class 'ebirdst_habitat'
plot(x, n_habitat_types = 15, ...)
```

Arguments

- `path` — character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.
- `ext` — ebirdst_extent object; the spatiotemporal extent over which to calculate the habitat associations. Note that **temporal component of `ext` is ignored is this function**, habitat associations are always calculated for the full year.
- `data` — as an alternative to providing the `path` argument specifying the location of the data package, the data required to calculate habitat associations can be provided explicitly as a named list of three data frames: `pis` containing PI data from `load_pis()`, `pds` containing PD data from `load_pds()`, and `stixels` containing stixel weights in a `weight` column. All data should be provided at the stixel level, identified by the `stixel_id` column, and only those stixels appearing in the `stixels` data frame will be used. Typically stixel weights are the proportion of the focal region that the given stixel overlaps. Ignored if `path` is provided. **In most cases, users will want to avoid using these arguments and simply provide `path` instead.**
- `stationary_associations` — logical; when the habitat association should be assumed to vary throughout the year and estimates should be made for each week of the year (the default) or habitat associations should be assumed constant throughout the year and a single set of estimates made for the full year. Annual estimates should only be made when you expect the associations to be constant throughout the year, e.g. for resident species.
- `x` — ebirdst_habitat object; habitat relationships as calculated by `ebirdst_habitat()`.
- `n_habitat_types` — number of habitat types to include in the cake plot. The most important set of predictors will be chosen based on the maximum weekly importance value across the whole year.
- `...` — ignored.

Details

The Status and Trends models use both effort (e.g. number of observers, length of checklist) and habitat (e.g. elevation, percent forest cover) covariates; for the full list consult ebirdst_predictors. This function calculates habitat associations only for the following covariates that most closely represent metrics of available habitat. In all cases these are calculated within a 1.5 km radius of each checklist:

- Land cover: percent of each landcover class
- Water cover: percent of each watercover class
- Intertidal: percent cover of intertidal mudflats
- Nighttime lights: total reflectance of nighttime lights
- Roads: road density. There are 5 covariates distinguishing between different road types; however, these are grouped together for the sake of the habitat associations.

The `plot()` method can be used to produce a cake plot, a stacked area chart showing habitat associations in which area indicates the importance of a given land cover class and the position above or below the x-axis indicates the direction of the relationship.

Value

An `ebirdst_habitat` object, consisting of a data frame giving the predictor importance and directionality for each predictor for each week of the year. The columns are:

- `predictor`: the name of the predictor
- `week`: the date of the center of the week, expressed as "MM-DD". This column will be missing if `stationary_associations = TRUE`.
- `importance`: the relative importance of the predictor, these values are scaled so they sum to 1 within each week.
- `prob_pos_slope`: the predicted probability that the slope of the PD
- `direction`: the direction of the relationship, either 1 for a positive relationship, -1 for a negative relationship, or NA when the direction of the relationship is not significant.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# define a spatial extent to calculate ppms over
e <- ebirdst_extent(c(xmin = -90, xmax = -82, ymin = 41, ymax = 48))

# compute habitat associations
habitat <- ebirdst_habitat(path = path, ext = e)
print(habitat)
# produce a cake plot
plot(habitat)

## End(Not run)
```

`ebirdst_ppms` — *eBird Status and Trends predictive performance metrics (PPMs)*

Description

Calculate a suite of predictive performance metrics (PPMs) for the eBird Status and Trends model of a given species within a spatiotemporal extent.

Usage

```
ebirdst_ppms(path, ext, es_cutoff, pat_cutoff)

## S3 method for class 'ebirdst_ppms'
plot(x, ...)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.
`ext`	ebirdst_extent object (optional); the spatiotemporal extent over which to calculate the PPMs.
`es_cutoff`	fraction between 0-1; the ensemble support cutoff to use in distinguishing zero and non-zero predictions. Optimal ensemble support cutoff values are calculated for each week during the modeling process and stored in the data package for each species. **In general, you should not specify a value for `es_cutoff` and instead allow the function to use the species-specific model-based values.**
`pat_cutoff`	numeric between 0-1; percent above threshold. Optimal PAT cutoff values are calculated for each week during the modeling process and stored in the data package for each species. **In general, you should not specify a value for `pat_cutoff` and instead allow the function to use the species-specific model-based values.**
`x`	ebirdst_ppms object; PPMs as calculated by `ebirdst_ppms()`.
`...`	ignored.

Details

During the eBird Status and Trends modeling process, a subset of observations (the "test data") are held out from model fitting to be used for evaluating model performance. Model predictions are made for each of these observations and this function calculates a suite of predictive performance metrics (PPMs) by comparing the predictions with the observed count on the eBird checklist.

Three types of PPMs are calculated: binary or range-based PPMs assess the ability of model to predict range boundaries, occurrence PPMs assess the occurrence probability predictions, and abundance PPMs assess the predicted abundance. Both the occurrence and count PPMs are within-range metrics, meaning the comparison between observations and predictions is only made within the range where the species occurs.

Prior to calculating PPMs, the test dataset is subsampled spatiotemporally using `ebirdst_subset()`. This process is performed for 25 monte carlo iterations resulting in 25 estimates of each PPM.

Value

An `ebirdst_pppms` object containing a list of three data frames: `binary_ppms`, `occ_ppms`, and `abd_ppms`. These data frames have 25 rows corresponding to 25 Monte Carlo iterations each estimating the PPMs using a spatiotemporal subsample of the test data. Columns correspond to the different PPMs. `binary_ppms` contains binary or range-based PPMs, `occ_ppms` contains within-range occurrence probability PPMs, and `abd_ppms` contains within-range abundance PPMs. In some cases, PPMs may be missing, either because there isn't a large enough test set within the spatiotemporal extent or because average occurrence or abundance is too low. In these cases, try increasing the size of the ebirdst_extent object.

`plot()` can be called on the returned `ebirdst_ppms` object to produce a boxplot of PPMs in all three categories: Binary Occurrence, Occurrence Probability, and Abundance.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# define a spatiotemporal extent to calculate ppms over
bb_vec <- c(xmin = -90, xmax = -82, ymin = 41, ymax = 48)
e <- ebirdst_extent(bb_vec, t = c("05-01", "07-31"))

# compute predictive performance metrics
ppms <- ebirdst_ppms(path = path, ext = e)
plot(ppms)

## End(Not run)
```

`ebirdst_ppms_ts` *Time series of eBird Status and Trends PPMs summarized temporally*

Description

Calculate a time series of predictive performance metrics (PPMs) for the eBird Status and Trends model. For each week or month of the year, PPMs will be summarized independently to produce a time series. For further details on eBird Status and Trends PPMs consult the help for ebirdst_ppms.

Usage

```
ebirdst_ppms_ts(path, ext, summarize_by = c("weeks", "months"), ...)

## S3 method for class 'ebirdst_ppms_ts'
plot(x, type = c("binary", "occurrence", "abundance"), metric = "kappa", ...)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.
`ext`	ebirdst_extent object (optional); the spatial extent over which to calculate the PPMs. Note that ebirdst_extent objects typically specify both a spatial and temporal extent, however, **within this function only the spatial component of the extent is used.**
`summarize_by`	character; periods over which to summarize PPMs. PPMs can either be calculated for eBird Status and Trends weeks (as defined in ebirdst_weeks) or for the months of the year.
`...`	ignored.
`x`	ebirdst_ppms_ts object; PPMs summarized by weeks or months as calculated by `ebirdst_ppms_ts()`.

`type` character; the PPM type to plot, either a binary, occurrence, or abundance PPM can be plotted.

`metric` character; the specific metric to plot, the list list of possible metrics varies by PPM type:

- Binary or occurrence: `auc`, `ppc`, `kappa`, `bernoulli_dev`, `sensitivity`, `specificity`
- Abundance: `poisson_dev_abd`, `poisson_dev_occ`, `spearman_abd`, `spearman_occ`

Value

An `ebirdst_pppms_ts` object containing a list of three data frames: `binary_ppms`, `occ_ppms`, and `abd_ppms`. Each row of these data frames corresponds to the PPMs from one Monte Carlo iteration for a given time period. Columns correspond to the different PPMs. `binary_ppms` contains binary or range-based PPMs, `occ_ppms` contains within-range occurrence probability PPMs, and `abd_ppms` contains within-range abundance PPMs. In some cases, PPMs may be missing, either because there isn't a large enough test set within the spatiotemporal extent or because average occurrence or abundance is too low. In these cases, try increasing the size of the ebirdst_extent object. `plot()` can be called on the returned `ebirdst_pppms_ts` object to plot a time series of a single PPM.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# define a spatial extent to calculate ppms over
e <- ebirdst_extent(c(xmin = -90, xmax = -82, ymin = 41, ymax = 48))

# compute predictive performance metrics, summarized by months
ppms <- ebirdst_ppms_ts(path = path, ext = e, summarize_by = "months")

# plot time series
# binary, kappa
plot(ppms, type = "binary", metric = "kappa")
# occurrence, sensitivity
plot(ppms, type = "occurrence", metric = "sensitivity")
#' # abundance, poisson deviance
plot(ppms, type = "abundance", metric = "poisson_dev_abd")

## End(Not run)
```

`ebirdst_predictors` *eBird Status and Trends predictors*

Description

A data frame of the predictors used in the eBird Status and Trends models. These include effort variables (e.g. distance traveled, number of observers, etc.) in addition to land and water cover variables. These landcover variables are derived from the MODIS MCD12Q1 500 m landcover product, and for each land cover class two FRAGSTATS metrics are calculated within a 1.5 km buffer around each checklist: % landcover (PLAND) and edge density (ED).

Usage

```
ebirdst_predictors
```

Format

A data frame with 74 rows and 5 columns:

predictor Predictor variable name.

predictor_label Descriptive labels for predictors for plotting and translating the cryptic variables names (e.g. `umd_fs_c1` is Evergreen Needleleaf Forest.

lc_class For the land and water cover FRAGSTATS variables, this gives the associated landcover class. It can be used for grouping and summarizing the four FRAGSTATS metrics to the level of the landcover class.

lc_class_label Similar to `predictor_label`; however, this variable gives the FRAGSTATS metrics a single name for the landcover class.

`ebirdst_runs` *Data frame of available eBird Status and Trends species*

Description

A dataset containing the species for which eBird Status and Trends data are available In addition, the dates defining the boundaries of the seasons are provided. These seasons are defined on a species-specific basis through expert review. For information on the details of defining seasons, please see the seasons section of the FAQ. Note that missing dates imply that a season failed expert review for that species within that season.

Usage

```
ebirdst_runs
```

Format

A data frame with 15 variables:

species_code Six letter eBird code in eBird Taxonomy v2018

scientific_name Scientific name from eBird Taxonomy v2018

common_name English common name from eBird Taxonomy v2018

resident Classifies this species a resident or a migrant

breeding_quality Breeding season quality

breeding_range_modeled Is the full range modeled?

breeding_start Breeding season start date

breeding_end Breeding season start date

nonbreeding_quality Non-breeding season quality

nonbreeding_range_modeled Is the full range modeled?

nonbreeding_start Non-breeding season start date

nonbreeding_end Non-breeding season start date

postbreeding_migration_quality Post-breeding season quality

postbreeding_migration_range_modeled Is the full range modeled?

postbreeding_migration_start Post-breeding season start date

postbreeding_migration_end Post-breeding season start date

prebreeding_migration_quality Pre-breeding season quality

prebreeding_migration_range_modeled Is the full range modeled?

prebreeding_migration_start Pre-breeding season start date

prebreeding_migration_end Pre-breeding season start date

resident_quality Resident quality

resident_start For resident species, the year-round start date

resident_end For resident species, the year-round end date

`ebirdst_subset` *Subset eBird Status and Trends data spatiotemporally*

Description

Spatiotemporally subset the raster or tabular eBird Status and Trends data. The spatiotemporal extent should be defined using `ebirdst_extent()`.

Usage

```
ebirdst_subset(x, ext)

## S3 method for class 'data.frame'
ebirdst_subset(x, ext)

## S3 method for class 'sf'
ebirdst_subset(x, ext)

## S3 method for class 'SpatRaster'
ebirdst_subset(x, ext)
```

Arguments

`x`	eBird Status and Trends data to subset; either a SpatRaster object with 52 layers (one for each week) or a data frame with PI or PD data.
`ext`	ebirdst_extent object; the spatiotemporal extent to filter the data to.

Value

eBird Status and Trends data in the same format as the input data, but subset in space and time.

Methods (by class)

- `ebirdst_subset(data.frame)`: PI or PD data
- `ebirdst_subset(sf)`: PI or PD data as an `sf` object
- `ebirdst_subset(SpatRaster)`: Status and Trends rasters

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data")
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# bbox for southern michigan in may
bb_vec <- c(xmin = -86, xmax = -83, ymin = 41.5, ymax = 43.5)
e <- ebirdst_extent(bb_vec, t = c("05-01", "05-31"))

# load and subset raster data
abd <- load_raster(path, product = "abundance", resolution = "lr")
abd_ss <- ebirdst_subset(abd, ext = e)

## End(Not run)
```

`ebirdst_version` *eBird Status and Trends Data Products version*

Description

Identify the version of the eBird Status and Trends Data Products that this version of the R package works with. Versions are defined by the year that all model estimates are made for. In addition, the release data and end date for access of this version of the data are provided. Note that after the given access end data you will no longer be able to download this version of the data and will be required to update the R package and transition to using a newer data version.

Usage

```
ebirdst_version()
```

Value

A list with three components: `version_year` is the year the model estimates are made for in this version of the data, `release_year` is the year this version of the data were released, and `access_end_date` is the last date that users will be able to download this version of the data.

Examples

```
ebirdst_version()
```

`ebirdst_weeks` *eBird Status and Trends weeks*

Description

eBird Status and Trends predictions are made for each of 52 weeks of the year. This data frame provides the boundaries of the weeks.

Usage

```
ebirdst_weeks
```

Format

A data frame with 52 rows and 5 columns:

week_number Integer week number from 1-52.

date Date of the midpoint of the week.

week_midpoint Date of the midpoint of the week expressed as a fraction of the year, i.e. a number from 0-1.

week_start Date of the start of the week expressed as a fraction of the year, i.e. a number from 0-1.

week_end Date of the end of the week expressed as a fraction of the year, i.e. a number from 0-1.

`get_species` *Get eBird species code for a set of species*

Description

Give a vector of species codes, common names, and/or scientific names, return a vector of 6-letter eBird species codes. This function will only look up codes for species for which eBird Status and Trends results exist.

Usage

```
get_species(x)
```

Arguments

x	character; vector of species codes, common names, and/or scientific names.

Value

A character vector of eBird species codes.

Examples

```
get_species(c("Black-capped Chickadee", "Poecile gambeli", "carchi"))
```

---

get_species_path *Get the path to the data package for a given species*

Description

This helper function can be used to get the path to a data package for a given species to be used by the various data loading functions.

Usage

```
get_species_path(species, path = ebirdst_data_dir())
```

Arguments

species	character; a single species given as a scientific name, common name or six-letter species code (e.g. woothr). The full list of valid species is can be viewed in the ebirdst_runs data frame included in this package. To download the example dataset, use "example_data".
path	character; directory to download the data to. All downloaded files will be placed in a sub-directory of this directory named for the data version year, e.g. "2020" for the 2020 Status Data Products. Each species' data package will then appear in a directory named with the eBird species code. Defaults to a persistent data directory, which can be found by calling `ebirdst_data_dir()`.

Value

The path to the data package directory.

Examples

```
## Not run:
# download the example data
ebirdst_download("example_data")

# get the path
path <- get_species_path("example_data")
```

```
# use it to load data
abd <- load_raster(path, "abundance")

# get the path to the full data package for yellow-bellied sapsucker
# common name, scientific name, or species code can be used
path <- get_species_path("Yellow-bellied Sapsucker")
path <- get_species_path("Sphyrapicus varius")
path <- get_species_path("yebsap")

## End(Not run)
```

`grid_sample` *Spatiotemporal grid sampling of observation data*

Description

Sample observation data on a spacetime grid to reduce spatiotemporal bias.

Usage

```
grid_sample(
  x,
  coords = c("longitude", "latitude", "day_of_year"),
  is_lonlat = TRUE,
  res = c(3000, 3000, 7),
  jitter_grid = TRUE,
  sample_size_per_cell = 1,
  cell_sample_prop = 0.75,
  keep_cell_id = FALSE,
  grid_definition = NULL
)

grid_sample_stratified(
  x,
  coords = c("longitude", "latitude", "day_of_year"),
  is_lonlat = TRUE,
  unified_grid = FALSE,
  keep_cell_id = FALSE,
  by_year = TRUE,
  case_control = TRUE,
  obs_column = "obs",
  sample_by = NULL,
  min_detection_probability = 0,
  maximum_ss = NULL,
  jitter_columns = NULL,
  jitter_sd = 0.1,
  ...
)
```

Arguments

x
: data frame; observations to sample, including at least the columns defining the location in space and time. Additional columns can be included such as features that will later be used in model training.

coords
: character; names of the spatial and temporal coordinates. By default the spatial spatial coordinates should be longitude and latitude, and temporal coordinate should be day_of_year.

is_lonlat
: logical; if the points are in unprojected, lon-lat coordinates. In this case, the points will be projected to an equal area sinusoidal CRS prior to grid assignment.

res
: numeric; resolution of the spatiotemporal grid in the x, y, and time dimensions. Unprojected locations are projected to an equal area coordinate system prior to sampling, and resolution should therefore be provided in units of meters. The temporal resolution should be in the native units of the time coordinate in the input data frame, typically it will be a number of days.

jitter_grid
: logical; whether to jitter the location of the origin of the grid to introduce some randomness.

sample_size_per_cell
: integer; number of observations to sample from each grid cell.

cell_sample_prop
: proportion (0-1]; if less than 1, only this proportion of cells will be randomly selected for sampling.

keep_cell_id
: logical; whether to retain a unique cell identifier, stored in column named .cell_id.

grid_definition
: list defining the spatiotemporal sampling grid as returned by assign_to_grid() in the form of an attribute of the returned data frame.

unified_grid
: logical; whether a single, unified spatiotemporal sampling grid should be defined and used for all observations in x or a different grid should be used for each stratum.

by_year
: logical; whether the sampling should be done by year, i.e. sampling N observations per grid cell per year, rather than across years, i.e. N observations per grid cell regardless of year. If using sampling by year, the input data frame x must have a year column.

case_control
: logical; whether to apply case control sampling whereby presence and absence are sampled independently.

obs_column
: character; if case_control = TRUE, this is the name of the column in x that defines detection (obs_column > 0) and non-detection (obs_column == 0).

sample_by
: character; additional columns in x to stratify sampling by. For example, if a landscape has many small islands (defined by an island variable) and we wish to sample from each independently, use sample_by = "island".

min_detection_probability
: proportion [0-1); the minimum detection probability in the final dataset. If case_control = TRUE, and the proportion of detections in the grid sampled dataset is below this level, then additional detections will be added via grid sampling the detections from the input dataset until at least this proportion of

detections appears in the final dataset. This will typically result in duplication of some observations in the final dataset. To turn this off this feature use min_detection_probability = 0.

maximum_ss
: integer; the maximum sample size in the final dataset. If the grid sampling yields more than this number of observations, maximum_ss observations will be selected randomly from the full set. Note that this subsampling will be performed in such a way that all levels of each strata will have at least one observation within the final dataset, and therefore it is not truly randomly sampling.

jitter_columns
: character; if detections are oversampled to achieve the minimum detection probability, some observations will be duplicated, and it can be desirable to slightly "jitter" the values of model training features for these duplicated observations. This argument defines the column names in x that will be jittered.

jitter_sd
: numeric; strength of the jittering in units of standard deviations, see jitter_columns.

...
: additional arguments defining the spatiotemporal grid; passed to grid_sample().

Details

grid_sample_stratified() performs stratified case control sampling, independently sampling from strata defined by, for example, year and detection/non-detection. Within each stratum, grid_sample() is used to sample the observations on a spatiotemporal grid. In addition, if case control sampling is turned on, the detections are oversampled to increase the frequecy of detections in the dataset.

The sampling grid is defined, and assignment of locations to cells occurs, in assign_to_grid(). Consult the help for that function for further details on how the grid is generated and locations are assigned. Note that by providing 2-element vectors to both coords and res the time component of the grid can be ignored and spatial-only subsampling is performed.

Value

A data frame of the spatiotemporally sampled data.

Examples

```
set.seed(1)

# generate some example observations
n_obs <- 10000
checklists <- data.frame(longitude = rnorm(n_obs, sd = 0.1),
                         latitude = rnorm(n_obs, sd = 0.1),
                         day_of_year = sample.int(28, n_obs, replace = TRUE),
                         year = NA_integer_,
                         obs = rpois(n_obs, lambda = 0.1),
                         forest_cover = runif(n_obs),
                         island = as.integer(runif(n_obs) > 0.95))
# add a year column, giving more data to recent years
checklists$year <- sample(seq(2016, 2020), size = n_obs, replace = TRUE,
                          prob = seq(0.3, 0.7, length.out = 5))
# create several rare islands
checklists$island[sample.int(nrow(checklists), 9)] <- 2:10
```

```
# basic spatiotemporal grid sampling
sampled <- grid_sample(checklists)

# plot original data and grid sampled data
par(mar = c(0, 0, 0, 0))
plot(checklists[, c("longitude", "latitude")],
     pch = 19, cex = 0.3, col = "#00000033",
     axes = FALSE)
points(sampled[, c("longitude", "latitude")],
       pch = 19, cex = 0.3, col = "red")

# case control sampling stratified by year and island
# return a maximum of 1000 checklists
sampled_cc <- grid_sample_stratified(checklists, sample_by = "island",
                                     maximum_ss = 1000)

# case control sampling increases the prevalence of detections
mean(checklists$obs > 0)
mean(sampled$obs > 0)
mean(sampled_cc$obs > 0)

# stratifying by island ensures all levels are retained, even rare ones
table(checklists$island)
# normal grid sampling loses rare island levels
table(sampled$island)
# stratified grid sampling retain at least one observation from each level
table(sampled_cc$island)
```

`load_config` *Load eBird Status and Trends configuration file*

Description

Load the configuration file for an eBird Status and Trends runs. This configuration file is mostly for internal use and contains a variety of parameters used in the modeling process.

Usage

```
load_config(path)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.

Value

A list with the run configuration parameters.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data")
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# load configuration file
cfg <- load_config(path)

## End(Not run)
```

---

load_fac_map_parameters

Load full annual cycle map parameters

Description

Get the map parameters used on the eBird Status and Trends website to optimally display the full annual cycle data. This includes bins for the abundance data, a projection, and an extent to map. The extent is the spatial extent of non-zero data across the full annual cycle and the projection is optimized for this extent.

Usage

```
load_fac_map_parameters(path)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.

Value

A list containing elements:

- `custom_projection`: a custom projection optimized for the given species' full annual cycle
- `fa_extent`: a SpatExtent object storing the spatial extent of non-zero data for the given species in the custom projection
- `res`: a numeric vector with 2 elements giving the target resolution of raster in the custom projection
- `fa_extent_sinu`: the extent in sinusoidal projection
- `weekly_bins`/`weekly_labels`: weekly abundance bins and labels for the full annual cycle
- `seasonal_bins`/‘seasonal_labels: seasonal abundance bins and labels for the full annual cycle

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# get map parameters
load_fac_map_parameters(path)

## End(Not run)
```

---

load_pds *Load eBird Status and Trends partial dependence data*

Description

Partial dependence (PD) plots depict the relationship between the modeled occurrence probability and each of the predictor variables used in the model. Status and Trends provides the data to generate these plots for every stixel.

Usage

```
load_pds(path, ext, model = c("occurrence", "count"), return_sf = FALSE)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.
`ext`	ebirdst_extent object; the spatiotemporal extent to filter the data to. The spatial component of the extent object must be provided in unprojected, latitude-longitude coordinates.
`model`	character; whether to make estimates for the occurrence or count model.
`return_sf`	logical; whether to return an sf object of spatial points rather then the default data frame.

Value

Data frame, or sf object if `return_sf = TRUE`, containing PD estimates for each stixel for either the occurrence or count model. The data frame will have the following columns:

- `stixel_id`: unique stixel identifier
- `latitude` and `longitude`: stixel centroid
- `day_of_year`: center day of year for stixel
- `predictor`: name of the predictor that the PD data correspond to, for a full list of predictors consult the ebirdst_predictors data frame

- `predictor_value`: value of the predictor variable at which PD is evaluated
- `response`: predicted response, occurrence or count, at the given value of the predictor averaged across all the values of the other predictors

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# load partial dependence data for occurrence model
pds <- load_pds(path)

# plot the top 15 predictor importances
# define a spatiotemporal extent to plot data from
bb_vec <- c(xmin = -86.6, xmax = -82.2, ymin = 41.5, ymax = 43.5)
e <- ebirdst_extent(bb_vec, t = c("05-01", "05-31"))
plot_pds(pds, "solar_noon_diff_mid", ext = e, n_bs = 5)

## End(Not run)
```

`load_pis` *Load eBird Status and Trends predictor importance data*

Description

Loads the predictor importance (PI) data from the stixel_summary.db sqlite database. PI estimates are provided for each stixel over which a model was run and are identified by a unique stixel ID in addition to the coordinates of the stixel centroid.

Usage

```
load_pis(path, ext, model = c("occurrence", "count"), return_sf = FALSE)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.
`ext`	ebirdst_extent object; the spatiotemporal extent to filter the data to. The spatial component of the extent object must be provided in unprojected, latitude-longitude coordinates.
`model`	character; whether to make estimates for the occurrence or count model.
`return_sf`	logical; whether to return an sf object of spatial points rather then the default data frame.

Value

Data frame, or sf object if `return_sf = TRUE`, containing PI estimates for each stixel for either the occurrence or count models. The data are provided in a 'wide' format, with each row corresponding to the PI estimates for a give stixel for the occurrence count model, and the relative importance of each predictor in columns. Stixels are identified by a unique `stixel_id`, and the centroid of the stixel in space and time is specified by the `latitude`, `longitude`, and `day_of_year` columns. The column `predictor` provides a code specifying the predictor variable. These codes can be looked up in `ebirdst_predictors` for a brief description.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# load predictor importance for the occurrence model
pis <- load_pis(path)

# plot the top 15 predictor importances
# define a spatiotemporal extent to plot data from
bb_vec <- c(xmin = -86.6, xmax = -82.2, ymin = 41.5, ymax = 43.5)
e <- ebirdst_extent(bb_vec, t = c("05-01", "05-31"))
plot_pis(pis, ext = e, n_top_pred = 15, by_cover_class = TRUE)

## End(Not run)
```

`load_predictions` *Load eBird Status and Trends test data predictions*

Description

During eBird Status and Trends modeling, predictions are made for checklists in a test dataset that is not included in the model fitting process. This function loads these predictions in addition to the actual observed count on the associated checklist. These data are used by `ebirdst_ppms()` for calculating predictive performance metrics.

Usage

```
load_predictions(path, return_sf = FALSE)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.
`return_sf`	logical; whether to return an sf object of spatial points rather then the default data frame.

Value

Data frame, or sf object if `return_sf = TRUE`, containing observed counts and model predictions for the test data.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# test data
test_predictions <- load_predictions(path)
dplyr::glimpse(test_predictions)

## End(Not run)
```

---

load_ranges *Load seasonal eBird Status and Trends range polygons*

Description

Range polygons are defined as the boundaries of non-zero seasonal relative abundance estimates, which are then (optionally) smoothed to produce more aesthetically pleasing polygons using the `smoothr` package.

Usage

```
load_ranges(path, resolution = c("mr", "lr"), smoothed = TRUE)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.
`resolution`	character; the raster resolution from which the range polygons were derived.
`smoothed`	logical; whether smoothed or unsmoothed ranges should be loaded.

Value

An `sf` update containing the seasonal range boundaries, with each season provided as a different feature.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data")
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# load smoothed ranges
# note that only low res data are provided for the example data
ranges <- load_range(path, resolution = "lr")

## End(Not run)
```

`load_raster` *Load eBird Status and Trends raster data cubes*

Description

Each of the eBird Status and Trends raster products is packaged as a GeoTIFF file representing predictions on a regular grid. The core products are occurrence, count, relative abundance, and percent of population. This function loads one of the available data products into R as a SpatRaster object.

Usage

```
load_raster(
  path,
  product = c("abundance", "count", "occurrence", "percent-population"),
  period = c("weekly", "seasonal", "full-year"),
  metric = NULL,
  resolution = c("hr", "mr", "lr")
)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.
`product`	character; Status and Trends product to load: occurrence, count, relative abundance, or percent of population. See Details for a detailed explanation of each of these products.
`period`	character; temporal period of the estimation. The Status and Trends models make predictions for each week of the year; however, as a convenience, data are also provided summarized at the seasonal or annual ("full-year") level.
`metric`	character; by default, the weekly products provide estimates of the median value (`metric = "median"`) and the summarized products are the cell-wise mean across the weeks within the season (`metric = "mean"`). However, additional

variants exist for some of the products. For the weekly relative abundance, confidence intervals are provided: specify `metric = "lower"` to get the 10th quantile or `metric = "upper"` to get the 90th quantile. For the seasonal and annual products, the cell-wise maximum values across weeks can be obtained with `metric = "max"`.

`resolution` character; the resolution of the raster data to load. The default is to load the native ~3 km resolution (`"hr"`); however, for some applications 9 km (`"mr"`) or 27 km (`"lr"`) data may be suitable.

Details

The core Status and Trends data products provide weekly estimates across a regular spatial grid. They are packaged as rasters with 52 layers, each corresponding to estimates for a week of the year, and we refer to them as "cubes" (e.g. the "relative abundance cube"). All estimates are the median expected value for a standard 1km, 1 hour eBird Traveling Count by an expert eBird observer at the optimal time of day and for optimal weather conditions to observe the given species. These products are:

- `occurrence`: the expected probability (0-1) of occurrence a species.
- `count`: the expected count of a species, conditional on its occurrence at the given location.
- `abundance`: the expected relative abundance of a species, computed as the product of the probability of occurrence and the count conditional on occurrence.
- `percent-population`: the percent of the total relative abundance within each cell. This is a derived product calculated by dividing each cell value in the relative abundance raster with the total abundance summed across all cells.

In addition to these weekly data cubes, this function provides access to data summarized over different periods. Seasonal cubes are produced by taking the cell-wise mean or max across the weeks within each season. The boundary dates for each season are species specific and are available in `ebirdst_runs`, and if a season failed review no associated layer will be included in the cube. In addition, full-year summaries provide the mean or max across all weeks of the year that fall within a season that passed review. Note that this is not necessarily all 52 weeks of the year. For example, if the estimates for the non-breeding season failed expert review for a given species, the full-year summary for that species will not include the weeks that would fall within the non-breeding season.

Value

For the weekly cubes, a SpatRaster with 52 layers for the given product, labeled by week. Seasonal cubes will have up to four layers labeled according to the seasons. The full-year products will have a single layer.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data")
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")
```

```
# weekly relative abundance
# note that only low resolution (lr) data are available for the example data
abd_weekly <- load_raster(path, "abundance", resolution = "lr")

# the weeks for each layer are stored in the layer names
names(abd_weekly)
# and can be converted to Date objects with
parse_raster_dates(abd_weekly)

# max seasonal abundance
abd_seasonal <- load_raster(path, "abundance",
                            period = "seasonal", metric = "max",
                            resolution = "lr")
# available seasons in stack
names(abd_seasonal)
# subset to just breeding season abundance
abd_seasonal[["breeding"]]

## End(Not run)
```

`load_stixels` *Load summary data for eBird Status and Trends stixels*

Description

eBird Status and Trends divides space and time into variably sized "stixels" within which individual base models are fit. The process of stixelization is performed many times and the prediction at any given point is the median of the predictions from all the stixels that that point falls in. This function loads summary statistics for each stixel, for example, the size of the stixels, the number of observations within each stixel, and a suite of predictive performance metrics (PPMs) for the model fit within that stixel.

Usage

```
load_stixels(path, ext, return_sf = FALSE)
```

Arguments

`path`	character; directory that the Status and Trends data for a given species was downloaded to. This path is returned by `ebirdst_download()` or `get_species_path()`.
`ext`	ebirdst_extent object; the spatiotemporal extent to filter the data to. The spatial component of the extent object must be provided in unprojected, latitude-longitude coordinates.
`return_sf`	logical; whether to return an sf object of spatial points rather then the default data frame.

Value

Data frame, or sf object if `return_sf = TRUE`, containing stixel summary data. Data are organized with one stixel per row and each stixel identified by a unique `stixel_id`, the centroid of each stixel in space and time is specified by `lat`, `lon`, and `date`.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# load stixel summary information
stixels <- load_stixels(path)
dplyr::glimpse(stixels)

## End(Not run)
```

---

parse_raster_dates *Parse weekly dates from raster layer names*

Description

The dates corresponding to each layer of a weekly data cube are stored as the layer names. This function converts these to Date objects.

Usage

```
parse_raster_dates(x)
```

Arguments

x	SpatRaster object; weekly Status and Trends data cube.

Value

Date vector.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data")
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# weekly relative abundance
abd_weekly <- load_raster(path, "abundance", resolution = "lr")
```

```
# dates corresponding to each week
parse_raster_dates(abd_weekly)

## End(Not run)
```

---

plot_pds *Plot partial dependency (PD) line plots*

Description

For a given eBird Status and Trends species, produce a line plot showing the partial dependence (PD) relationship for a given predictor. Two options for smoothing are provided.

Usage

```
plot_pds(
  pds,
  predictor,
  ext,
  bootstrap_smooth = TRUE,
  show_stixel_pds = FALSE,
  show_quantiles = FALSE,
  n_bs = 100,
  ss_equivalent = 10,
  k = 25,
  ci_alpha = 0.05,
  gbm_n_trees = 500,
  ylim = NULL,
  plot = TRUE
)
```

Arguments

`pds`	data frame; partial dependence data from `load_pds()`.
`predictor`	character; single predictor name to plot PD for. For a full list of predictors, and their associated definitions, see ebirdst_predictors.
`ext`	ebirdst_extent object; the spatiotemporal extent over which to calculate PDs. This is required, since results are less meaningful over large spatiotemporal extents.
`bootstrap_smooth`	logical; the ideal visualization of the PD data is a pointwise GAM smoothing of the individual stixel PD values. This argument specifies whether this should be done directly on the full PD dataset (`bootstrap_smooth = FALSE`) or by subsampling and bootstrapping. The latter approach deals with the randomness in the data and can be more efficient for large datasets.

show_stixel_pds
: logical; whether to plot the individual stixel PD values as semi-transparent lines.

show_quantiles
: logical; adds a band for the upper (90th) and lower (10th) quantiles of the individual stixel PD values. These are calculated using quantile regression.

n_bs
: int; number of GAM bootstrap iterations when estimating PD confidence intervals. Ignored if `bootstrap_smooth = FALSE`.

ss_equivalent
: int; when bootstrapping to estimate PD confidence intervals, this argument specifies the size of the subsample of the original data. In particular, `ss_equivalent` should be an integer representing the equivalent sampling size when averaging this number of PD estimates.

k
: integer; number of knots to use in the GAM when smooth the PD relationship.

ci_alpha
: numeric; alpha level of confidence intervals. Default is 0.05.

gbm_n_trees
: integer; number of trees to fit in the GBM when estimating quantiles. Ignored if `show_quantiles = FALSE`. Default is 500.

ylim
: numeric; 2-element vector to pre-define the y-limits of plotting. In the format `c(ymin, ymax)`.

plot
: logical; whether to plot the PD relationships or just return data.

Value

Plots the smoothed partial dependence relationship for the specified predictor and returns a data frame of the smoothed curve with confidence intervals.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# load predictor dependence data
pds <- load_pds(path)

# define a spatiotemporal extent to plot data from
bb_vec <- c(xmin = -90, xmax = -82, ymin = 41, ymax = 48)
e <- ebirdst_extent(bb_vec, t = c("05-01", "05-31"))

# for testing, run with 5 bootstrap iterations for speed
# in practice, best to run with the default number of iterations (100)
pd_smooth <- plot_pds(pds, "solar_noon_diff_mid", ext = e, n_bs = 5)
dplyr::glimpse(pd_smooth)

## End(Not run)
```

plot_pis *Plot predictor importance (PI) box plots*

Description

For a given eBird Status and Trends species, produce a box plot showing the predictor importance (PI) for each of the predictors used in the occurrence model. Predictors are plotted in order from highest to lowest importance. Many function parameters allow for customized plots.

Usage

```
plot_pis(
  pis,
  ext,
  by_cover_class = TRUE,
  n_top_pred = 15,
  pretty_names = TRUE,
  plot = TRUE
)
```

Arguments

pis	data frame; predictor importance data from load_pis().
ext	ebirdst_extent object; the spatiotemporal extent over which to calculate PIs. This is required, since results are less meaningful over large spatiotemporal extents.
by_cover_class	logical; whether to aggregate the FRAGSTATS metrics (PLAND and ED) for the land cover classes into single values for the land cover classes.
n_top_pred	integer; how many predictors to show.
pretty_names	logical; whether to convert cryptic land cover codes to readable land cover class names.
plot	logical; whether to plot predictor importance or just return top predictors.

Value

Plots a boxplot of predictor importance and invisibly returns the PI data subset to just the top predictors, grouped and renamed according to `by_cover_class` and `pretty_names`.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")
```

```
# load predictor importance
pis <- load_pis(path)

# define a spatiotemporal extent to plot data from
bb_vec <- c(xmin = -86, xmax = -83, ymin = 41.5, ymax = 43.5)
e <- ebirdst_extent(bb_vec, t = c("05-01", "05-31"))

top_pred <- plot_pis(pis, ext = e, n_top_pred = 10)
top_pred

## End(Not run)
```

---

poisson_dev *Poisson deviance*

Description

Poisson deviance

Usage

```
poisson_dev(obs, pred)
```

Arguments

obs	numeric; observed values.
pred	numeric; predicted values.

Value

A named numeric vector with three elements: model deviance, mean deviance, and deviance explained.

Examples

```
obs <- c(0, 0, 1, 3, 5, 2)
pred <- c(0.5, 0.1, 2.5, 3.3, 5.2, 2.5)
ebirdst:::poisson_dev(obs, pred)
```

project_extent — *Transform a spatiotemporal extent to a different CRS*

Description

Transform an eBird Status and Trends extent object to a different coordinate reference system. This is most commonly required to transform the extent to the sinusoidal CRS used by the eBird Status and Trends rasters.

Usage

```
project_extent(x, crs)
```

Arguments

x	ebirdst_extent object; a spatiotemporal extent.
crs	coordinate references system, given either as a proj4string, an integer EPSG code, or a `crs` object generated with `st_crs()`.

Value

An ebirdst_extent object in the new CRS.

Examples

```
# construct an ebirdst_extent object
bb_vec <- c(xmin = -80, xmax = -70, ymin = 40, ymax = 47)
bb <- sf::st_bbox(bb_vec, crs = 4326)
bb_ext <- ebirdst_extent(bb)

# transform to sinusoidal projection of rasters
sinu <- "+proj=sinu +lon_0=0 +x_0=0 +y_0=0 +a=6371007 +b=6371007 +units=m +no_defs"
project_extent(bb_ext, crs = sinu)

# also works on polygon extents
poly <- sf::read_sf(system.file("shape/nc.shp", package="sf"))
poly_ext <- ebirdst_extent(poly)
project_extent(poly_ext, crs = sinu)
```

set_ebirdst_access_key

Store the eBird Status and Trends access key

Description

Accessing eBird Status and Trends data requires an access key, which can be obtained by visiting https://ebird.org/st/request. This key must be stored as the environment variable `EBIRDST_KEY` in order for `ebirdst_download()` to use it. The easiest approach is to store the key in your `.Renviron` file so it can always be accessed in your R sessions. Use this function to set `EBIRDST_KEY` in your `.Renviron` file provided that it is located in the standard location in your home directory. It is also possible to manually edit the `.Renviron` file. **The access key is specific to you and should never be shared or made publicly accessible.**

Usage

```
set_ebirdst_access_key(key, overwrite = FALSE)
```

Arguments

`key`	character; API key obtained by filling out the form at https://ebird.org/st/request.
`overwrite`	logical; should the existing `EBIRDST_KEY` be overwritten if it has already been set in .Renviron.

Value

Edits .Renviron, then returns the path to this file invisibly.

Examples

```
## Not run:
# save the api key, replace XXXXXX with your actual key
set_ebirdst_access_key("XXXXXX")

## End(Not run)
```

stixelize

Generate polygons for eBird Status and Trends stixels

Description

eBird Status and Trends divides space and time into variably sized "stixels" within which individual base models are fit. The process of stixelization is performed many times and the prediction at any given point is the median of the predictions from all the stixels that that point falls in. `load_stixels()` loads information on all the stixels that compromise a species' Status and Trends model, with stixels identified by the location of their centroid. This function uses this information to define polygons for each stixel and attaches them to the original data in the form of an sf object.

Usage

```
stixelize(x)
```

Arguments

x	data frame; stixel summary data loaded with `load_stixels()`, or any other data frame with fields `lonitude_min`, `lontidue_max`, `latitude_min`, and `latitude_max`.

Value

sf object with geometry column storing polygons representing the stixels boundaries.

Examples

```
## Not run:
# download example data
path <- ebirdst_download("example_data", tifs_only = FALSE)
# or get the path if you already have the data downloaded
path <- get_species_path("example_data")

# load stixel summary information
stixels <- load_stixels(path)

# build stixel polygons
stixelize(stixels)

## End(Not run)
```

Index