Package 'ICESat2VegR'

December 2, 2025

Type Package

Title NASA's Ice, Cloud, and Elevation Satellite (ICESat-2) Data Analysis for Land and Vegetation Applications

Version 0.0.1

Description Set of tools for downloading, reading, visualizing, processing and exporting NASA's ICESat-2 ATL03 (Global Geolocated Photon Data) and ATL08 (Land and Vegetation Height) products for Land and Vegetation Applications.

License GPL (>= 3)

Imports curl, data.table, sf, fs, getPass, jsonlite, hdf5r, httr2, magrittr, methods, Rcpp, Rdpack, R6, randomForest, reticulate, terra, xml2

Encoding UTF-8

URL https://github.com/carlos-alberto-silva/ICESat2VegR

BugReports https://github.com/carlos-alberto-silva/ICESat2VegR/issues

LazyData true

NeedsCompilation yes

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

RdMacros Rdpack

Suggests chromote, devtools, ggplot2, grid, gridExtra, htmlwidgets, knitr, leaflet, leafsync, lidR, lwgeom, mapview, png, rmarkdown, rstudioapi, signal, stars, testthat (>= 3.0.0), webshot, viridis

LinkingTo Rcpp

Collate 'ANNIndex.R'
'lazy_applier.R'
'ATL03_ATL08_compute_seg_attributes_dt_segStat.R'
'utmTools.R'
'ATL03_ATL08_photons_attributes_dt_LAS.R'
'ATL03_ATL08_photons_attributes_dt_clipBox.R'
'ATL03_ATL08_photons_attributes_dt_clipGeometry.R'
'ATL03_ATL08_photons_attributes_dt_gridStat.R'
'ATL03_ATL08_photons_attributes_dt_join.R'
'ATL03_ATL08_photons_attributes_dt_polyStat.R'
'ATL03_ATL08_photons_seg_dt_fitground.R'
'ATL03_ATL08_photons_seg_dt_height_normalize.R'

'ATL03_ATL08_seg_attributes_dt_clip.R'
'ATL03_ATL08_seg_cover_dt_compute.R'
'ATL03_ATL08_segment_create.R'
'ATL03_h5_clip.R'
'clipTools.R'
'ATL03_h5_clipBox.R'
'ATL03_h5_clipGeometry.R'
'ATL03_photons_attributes_dt.R'
'lasTools.R'
'ATL03_photons_attributes_dt_LAS.R'
'ATL03_photons_attributes_dt_clipBox.R'
'ATL03_photons_attributes_dt_clipGeometry.R'
'class_tools.R'
'class.icesat2.R'
'zzz.R'
'ATL03_read.R'
'ATL03_seg_metadata_dt.R'
'ATL08_read.R'
'ATL08_h5_clip.R'
'ATL08_h5_clipBox.R'
'ATL08_photons_attributes_dt.R'
'ATL08_photons_attributes_dt_LAS.R'
'ATL08_seg_attributes_dt.R'
'ATL08_seg_attributes_dt_LAS.R'
'ATL08_seg_attributes_dt_clipBox.R'
'ATL08_seg_attributes_dt_clipGeometry.R'
'ATL08_seg_attributes_dt_gridStat.R'
'ATL08_seg_attributes_dt_polyStat.R'
'ATL08_seg_attributes_h5_gridStat.R'
'ATLAS_dataDownload.R'
'ATLAS_dataFinder.R'
'earthaccess.R'
'ATLAS_dataFinder_cloud.R'
'ATLAS_dataFinder_direct.R'
'ICESat2VegR-package.R'
'ICESat2VegR_configure.R'
'addEEImage.R'
'argParse.R'
'class.icesat2.h5ds_cloud.R'
'class.icesat2.h5_cloud.R'
'class.icesat2.h5ds_local.R'
'class.icesat2.h5_local.R'
'clip.R'
'ee_build_AlphaEarth_embedding_terrain_stack.R'
'ee_build_hls_s1c_terrain_stack.R'
'extract.R'
'fit_metrics.R'
'fit_model.R'
'gdalBindings.R'
'gee-base-feature.R'
'gee-base-list.R'
'gee-base-number.R'

'gee-base.R'
'gee-search.R'
'map_tools.R'
'model_tools.R'
'predict_h5.R'
'rasterize_h5.R'
'rgt_extract.R'
'sample.R'
'to_vect.R'
'varSel.R'
'vect_as_ee.R'

Contents

`ICESat2VegR-package` *ICESat2VegR: NASA ICESat-2 data for land & vegetation*

Description

Tools to download, read, process, model, and visualize ICESat-2 ATL03/ATL08 for land and vegetation applications.

Author(s)

Maintainer: Caio Hamamura <caiohamamura@gmail.com> [copyright holder]

Authors:

- Carlos Alberto Silva <c.silva@ufl.edu> [copyright holder]

See Also

Useful links:

- https://github.com/carlos-alberto-silva/ICESat2VegR
- Report bugs at https://github.com/carlos-alberto-silva/ICESat2VegR/issues

---

`.as_ee_geom` *Convert R geometry objects to an Earth Engine geometry*

Description

`.as_ee_geom()` converts a variety of R geometry representations (including `sf`, `sfc`, `terra` objects, bounding boxes, WKT/GeoJSON, and even existing Earth Engine python objects) into a Python `ee$Geometry` suitable for use in Earth Engine workflows.

This helper is designed to be flexible and permissive: it accepts most common spatial formats used in R and normalizes them into a standard Earth Engine geometry. When a buffer distance is supplied, the geometry is optionally buffered in meters on the Earth Engine side.

Usage

```
.as_ee_geom(geom, xcol = "lon", ycol = "lat", crs = 4326, buffer_m = NULL)
```

Arguments

`geom`	Geometry input. Supported types include: • An Earth Engine python object (e.g. `ee$Geometry`, `ee$Feature`, `ee$FeatureCollection`). Features/FeatureCollections are reduced to their geometry via `$geometry()`. • An `sf` or `sfc` object. • A `terra::SpatVector`. • A `terra::SpatExtent` (or numeric vector of length 4 specifying a bounding box as `c(xmin, ymin, xmax, ymax)`). • A `data.frame` with longitude/latitude columns (`xcol`, `ycol`). • A single-character WKT string. • A single-character GeoJSON string. • A parsed GeoJSON list with a non-`NULL` `type` element.
`xcol`	Character. Name of the longitude column when `geom` is a `data.frame`. Default is `"lon"`.
`ycol`	Character. Name of the latitude column when `geom` is a `data.frame`. Default is `"lat"`.
`crs`	Coordinate reference system of the input geometry when `geom` is an `sf`/`sfc` object or a `data.frame`. Can be any `sf`-compatible CRS specification (default is EPSG 4326).
`buffer_m`	Optional numeric. Buffer distance in meters applied to the resulting Earth Engine geometry. If `NULL` (default) no buffering is applied. For sf/data.frame inputs, buffering is done on the EE side using `ee$Geometry$buffer()`.

Details

For sf/sfc and data.frame inputs, geometries are first transformed to EPSG:4326 and written to a temporary GeoJSON file, which is then read and wrapped into an ee$FeatureCollection(...)$geometry(). For terra::SpatVector and terra::SpatExtent inputs, conversion proceeds via terra::writeVector() / terra::as.polygons() and an EE rectangle geometry, respectively.

Existing Earth Engine python objects are returned as-is, except that ee$Feature and ee$FeatureCollection objects are coerced to their underlying geometry via $geometry().

Value

A Python ee$Geometry object (or compatible geometry-like EE object) suitable for use in Earth Engine operations.

Examples

```
## Not run:
  ee <- reticulate::import("ee", delay_load = FALSE)

  # 1) From sf polygon
  library(sf)
  poly <- st_as_sfc(st_bbox(c(
    xmin = -82.4, xmax = -82.2,
    ymin =  29.6, ymax =  29.8
  ), crs = 4326))

  ee_geom1 <- .as_ee_geom(poly)


  # 2) From terra::SpatVector
  library(terra)
  v <- vect(system.file("extdata", "all_boundary.shp", package = "ICESat2VegR"))
  ee_geom2 <- .as_ee_geom(v, buffer_m = 30)


  # 3) From numeric extent (xmin, ymin, xmax, ymax)
  bbox_vec <- c(-82.4, 29.6, -82.2, 29.8)
  ee_geom3 <- .as_ee_geom(bbox_vec)


  # 4) From data.frame of points
  df <- data.frame(
    lon = c(-82.3, -82.25),
    lat = c(29.65, 29.7)
  )
  ee_geom4 <- .as_ee_geom(df, buffer_m = 1000)

## End(Not run)
```

.ee_ping *Check if Google Earth Engine is initialized*

Description

`.ee_ping()` verifies that the Earth Engine Python API is initialized and responsive. It attempts to evaluate a trivial Earth Engine expression (`ee$Image$constant(1)$getInfo()`) and returns invisibly if successful. If the call fails, an error is raised with a hint to authenticate and initialize Earth Engine.

Usage

```
.ee_ping(ee)
```

Arguments

`ee`	The Earth Engine Python module as returned by `reticulate::import("ee")`.

Value

Invisibly returns `TRUE` if Earth Engine is initialized and responsive. Otherwise, an error is thrown indicating that Earth Engine must be authenticated and initialized.

Examples

```
## Not run:
  library(reticulate)
  ee <- import("ee", delay_load = FALSE)

  # Will error if EE is not authenticated/initialized
  .ee_ping(ee)

## End(Not run)
```

---

addEEImage *Add an Earth Engine Image to a leaflet map*

Description

Add an Earth Engine Image to a leaflet map

Usage

```
addEEImage(
  map,
  img,
  bands = NULL,
  min_value = 0,
  max_value = 1,
  palette = c("#d55e00", "#cc79a7", "#f0e442", "#0072b2", "#009e73"),
  group = NULL,
  aoi = NULL,
  ...
)
```

Arguments

map	A leaflet::leaflet widget.
img	An Earth Engine image (reticulate python.builtin.object with $visualize, $select).
bands	Character or numeric vector (length 1 or 3). If NULL, uses all bands from the image.
min_value	max_value Numeric visualization range. Aliases: min, max.
palette	Character vector of colors for single-band rendering.
group	Optional overlay group name.
aoi	Optional EE geometry to clip before rendering.
...	Passed to leaflet::addTiles() (e.g., options = leaflet::tileOptions(opacity = 0.8)).

Value

The input leaflet map with the EE image layer added.

---

aspect *Compute terrain aspect (degrees) from a DEM image*

Description

Computes the terrain *aspect* in degrees for each pixel of an Earth Engine ee$Image representing a digital elevation model (DEM).

Aspect describes the downslope direction of the steepest gradient and is expressed in degrees clockwise from north. The computation is performed using Earth Engine's built-in ee$Terrain$aspect() function. As with slope, Earth Engine uses the 4-connected neighborhood, so missing values may occur near image edges.

Usage

```
aspect(x)
```

Arguments

x	An ee$Image representing a DEM from which aspect will be derived.

Value

An ee$Image with one band named "aspect" containing aspect values in degrees clockwise from north.

Examples

```
## Not run:
  ee <- reticulate::import("ee")
  dem <- ee$Image("NASA/NASADEM_HGT/001")
  asp <- aspect(dem)

## End(Not run)
```

`ATL03_ATL08_compute_seg_attributes_dt_segStat` *Statistics of ATL03 and ATL08 labeled photons at the segment level*

Description

Computes a series of statistics from ATL03 and ATL08 labeled photons within a given segment length.

Usage

```
ATL03_ATL08_compute_seg_attributes_dt_segStat(
  atl03_atl08_seg_dt,
  list_expr,
  ph_class = c(0, 1, 2, 3),
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  quality_ph = NULL,
  night_flag = NULL
)
```

Arguments

`atl03_atl08_seg_dt`	An S4 object of class `icesat2.atl03_atl08_seg_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).
`list_expr`	The function to be applied for computing the defined statistics
`ph_class`	Character vector indicating photons to process based on the classification (1=ground, 2=canopy, 3=top canopy), Default is c(2,3)
`beam`	Character vector indicating beams to process. Default is c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
`quality_ph`	Indicates the quality of the associated photon. 0 = nominal, 1 = possible_afterpulse, 2 = possible_impulse_response_ effect, 3=possible_tep. Default is 0
`night_flag`	Flag indicating the data were acquired in night conditions: 0=day, 1=night. Default is 1

Value

Returns an S4 object of class `icesat2.atl08_dt` Containing Statistics of ATL03 and ATL08 labeled photons

Examples

```
# Specifying ATL03 and ATL08 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl08_path <- system.file("extdata",
```

```
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 labeled photons
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

# Computing the max canopy height at 30 m segments
atl03_atl08_dt_seg <- ATL03_ATL08_segment_create(atl03_atl08_dt, segment_length = 30)

max_canopy <- ATL03_ATL08_compute_seg_attributes_dt_segStat(atl03_atl08_dt_seg,
  list_expr = max(ph_h),
  ph_class = c(2, 3),
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  quality_ph = 0,
  night_flag = 0
)

head(max_canopy)

# Computing a series of canopy height statistics from customized list expressions
canopy_metrics <- ATL03_ATL08_compute_seg_attributes_dt_segStat(atl03_atl08_dt_seg,
  list_expr = list(
    max_ph_elevation = max(h_ph),
    h_canopy = quantile(ph_h, 0.98),
    n_canopy = sum(classed_pc_flag == 2),
    n_top_canopy = sum(classed_pc_flag == 3)
  ),
  ph_class = c(2, 3),
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  quality_ph = 0,
  night_flag = 0 # there are no night photons in this dataset
)

head(canopy_metrics)

close(atl03_h5)
close(atl08_h5)
```

`ATL03_ATL08_photons_attributes_dt_clipBox` *Clip joined ATL03 and ATL08 photons by Bounding Box*

Description

This function clips joined ATL03 and ATL08 photon attributes within a given Bounding Box

Usage

```
ATL03_ATL08_photons_attributes_dt_clipBox(
  atl03_atl08_dt,
  lower_left_lon,
  upper_right_lon,
  upper_right_lat,
  lower_left_lat
)
```

Arguments

atl03_atl08_dt	An S4 object of class icesat2.atl03atl08_dt containing ATL03 and ATL08 data (output of ATL03_ATL08_photons_attributes_dt_join() function).
lower_left_lon	Numeric. West longitude (x) coordinate of bounding rectangle, in decimal degrees.
upper_right_lon	Numeric. East longitude (x) coordinate of bounding rectangle, in decimal degrees.
upper_right_lat	Numeric. North latitude (y) coordinate of bounding rectangle, in decimal degrees.
lower_left_lat	Numeric. South latitude (y) coordinate of bounding rectangle, in decimal degrees.

Value

Returns an S4 object of class icesat2.atl03atl08_dt containing a subset of the ATL03 and ATL08 photon attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
# Specifying the path to ATL03 and ATL08 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
```

```
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Joining ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

# Bounding rectangle coordinates
lower_left_lon <- -103.7604
lower_left_lat <- 59.4672
upper_right_lon <- -103.7600
upper_right_lat <- 59.4680

# Clipping ATL08-derived canopy metrics by boundary box extent
atl03_atl08_dt_clip <- ATL03_ATL08_photons_attributes_dt_clipBox(
  atl03_atl08_dt,
  lower_left_lon,
  upper_right_lon,
  upper_right_lat,
  lower_left_lat
)
head(atl03_atl08_dt_clip)

close(atl03_h5)
close(atl08_h5)
```

`ATL03_ATL08_photons_attributes_dt_clipGeometry`
Clip Joined ATL03 and ATL08 by Geometry

Description

This function clips joined ATL03 and ATL08 photon attributes within a given geometry.

Usage

```
ATL03_ATL08_photons_attributes_dt_clipGeometry(
  atl03_atl08_dt,
  clip_obj,
  split_by = NULL
)
```

Arguments

`atl03_atl08_dt`	An S4 object of class `icesat2.atl03atl08_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).
`clip_obj`	An object of class `terra::SpatVector`, which can be loaded as an ESRI shapefile using terra::vect function.
`split_by`	Optional. clip_obj ID. If defined, the data will be clipped by each clip_obj using the clip_obj ID from the attribute table.

Value

Returns an S4 object of class `icesat2.atl03atl08_dt` containing the clipped ATL08 attributes.

Examples

```
# Specifying the path to ATL03 and ATL08 files
atl03_path <- system.file("extdata", "atl03_clip.h5", package = "ICESat2VegR")
atl08_path <- system.file("extdata", "atl08_clip.h5", package = "ICESat2VegR")

# Reading ATL03 and ATL08 data (h5 files)
atl03_h5 <- ATL03_read(atl03_path)
atl08_h5 <- ATL08_read(atl08_path)

# Joining ATL03 and ATL08 photon attributes
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

# Specifying the path to the shapefile
clip_obj_filepath <- system.file("extdata", "clip_geom.shp", package = "ICESat2VegR")

# Reading shapefile as a SpatVector object
clip_obj <- terra::vect(clip_obj_filepath)

# Clipping ATL08 terrain attributes by geometry
atl03_atl08_dt_clip <- ATL03_ATL08_photons_attributes_dt_clipGeometry(
  atl03_atl08_dt,
  clip_obj,
  split_by = "id"
)
head(atl03_atl08_dt_clip)

close(atl03_h5)
close(atl08_h5)
```

ATL03_ATL08_photons_attributes_dt_gridStat

Statistics of ATL03 and ATL08 photon attributes

Description

This function computes a series of user defined descriptive statistics within each given grid cell for ATL03 and ATL08 photon attributes

Usage

```
ATL03_ATL08_photons_attributes_dt_gridStat(
  atl03_atl08_dt,
  func,
  res = 0.5,
  ph_class = c(2, 3),
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  quality_ph = 0,
  night_flag = 1
)
```

Arguments

atl03_atl08_dt	An S4 object of class icesat2.atl03atl08_dt containing ATL03 and ATL08 attributes (output of the ATL03_ATL08_photons_attributes_dt_join() function).
func	The function to be applied for computing the defined statistics
res	Spatial resolution in decimal degrees for the output SpatRast raster layer. Default is 0.5.
ph_class	Character vector indicating photons to process based on the classification (1=ground, 2=canopy, 3=top canopy), Default is c(2,3)
beam	Character vector indicating beams to process. Default is c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
quality_ph	Indicates the quality of the associated photon. 0=nominal, 1=possible_afterpulse, 2=possible_impulse_response_effect, 3=possible_tep. Default is 0
night_flag	Flag indicating the data were acquired in night conditions: 0=day, 1=night. Default is 1

Value

Return a SpatRast raster layer(s) of selected ATL03 and ATL08 photon attribute(s)

Examples

```
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)
# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# # Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

# Computing the mean of ph_h attribute at 0.0002 degree grid cell
mean_ph_h <- ATL03_ATL08_photons_attributes_dt_gridStat(atl03_atl08_dt,
  func = mean(ph_h),
  res = 0.0002
)

plot(mean_ph_h)

# Define your own function
mySetOfMetrics <- function(x) {
  metrics <- list(
```

```
    min = min(x), # Min of x
    max = max(x), # Max of x
    mean = mean(x), # Mean of x
    sd = sd(x) # Sd of x
  )
  return(metrics)
}

# Computing a series of ph_h stats at 0.0002 degree grid cell from customized function
ph_h_metrics <- ATL03_ATL08_photons_attributes_dt_gridStat(atl03_atl08_dt,
  func = mySetOfMetrics(ph_h), res = 0.0002
)

plot(ph_h_metrics)

close(atl03_h5)
close(atl08_h5)
```

`ATL03_ATL08_photons_attributes_dt_join`
Join ATL03 and ATL08 photons attributes

Description

This function joins ATL03 and ATL08 computed photons attributes

Usage

```
ATL03_ATL08_photons_attributes_dt_join(
  atl03_h5,
  atl08_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
)
```

Arguments

`atl03_h5`	A ICESat-2 ATL03 object (output of `ATL03_read()` function). An S4 object of class `icesat2.atl03_dt`.
`atl08_h5`	A ICESat-2 ATL08 object (output of `ATL08_read()` function). An S4 object of class `icesat2.atl08_dt`.
`beam`	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")

Details

These are the photons attributes extracted by default:

- `ph_segment_id`: Georeferenced segment id (20-m) associated with each photon.
- `lon_ph`: Longitude of each received photon. Computed from the ECEF Cartesian coordinates of the bounce point.
- `lat_ph`: Latitude of each received photon. Computed from the ECEF Cartesian coordinates of the bounce point.

- h_ph: Height of each received photon, relative to the WGS-84 ellipsoid including the geophysical corrections noted in section 6.0. Please note that neither the geoid, ocean tide nor the dynamic atmospheric corrections (DAC) are applied to the ellipsoidal heights.
- quality_ph: Indicates the quality of the associated photon. 0=nominal, 1=possible_afterpulse, 2=possible_impulse_response_effect, 3=possible_tep. Use this flag in conjunction with signal_conf_ph to identify those photons that are likely noise or likely signal.
- solar_elevation: Elevation of the sun above the horizon at the photon bounce point.
- dist_ph_along: Along-track distance of the photon from the beginning of the segment.
- dist_ph_across: Across-track distance of the photon from the center of the segment.
- night_flag: Flag indicating the data were acquired in night conditions: 0=day, 1=night. Night flag is set when solar elevation is below 0.0 degrees.
- classed_pc_indx: Indices of photons tracking back to ATL03 that surface finding software identified and used within the creation of the data products.
- classed_pc_flag: The L2B algorithm is run if this flag is set to 1 indicating data have sufficient waveform fidelity for L2B to run.
- ph_h: Height of photon above interpolated ground surface.
- d_flag: Flag indicating whether DRAGANN labeled the photon as noise or signal.
- delta_time: Mid-segment GPS time in seconds past an epoch. The epoch is provided in the metadata at the file level.
- orbit_number: Orbit number identifier to identify data from different orbits.
- beam: Beam identifier.
- strong_beam: Logical indicating if the beam is a strong beam.

Value

Returns an S4 object of class icesat2.atl03atl08_dt containing the ATL08 computed photons attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
# Specifying the path to ATL03 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
```

```
# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# # Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

close(atl03_h5)
close(atl08_h5)
```

`ATL03_ATL08_photons_attributes_dt_LAS` *Converts ATL03/ATL08 classified photon cloud to LAS*

Description

Converts ATL03/ATL08 classified photon cloud to LAS

Usage

```
ATL03_ATL08_photons_attributes_dt_LAS(
  atl03_atl08_dt,
  output,
  normalized = TRUE
)
```

Arguments

`atl03_atl08_dt`	An S4 object of class `icesat2.atl08_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).
`output`	character. The output path of for the LAS(Z) file(s). The function will create one LAS file per UTM Zone in WGS84 datum.
`normalized`	logical, default TRUE. Whether the output should be normalized LAS or raw altitude.

Value

Nothing, it just saves outputs as LAS file in disk

Examples

```
outdir <- tempdir()

# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
```

```
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

if (require("lidR")) {
  ATL03_ATL08_photons_attributes_dt_LAS(
    atl03_atl08_dt,
    output = file.path(outdir, "output.laz"),
    normalized = TRUE
  )
}

close(atl03_h5)
close(atl08_h5)
```

`ATL03_ATL08_photons_attributes_dt_polyStat` *Statistics of ATL03 and ATL08 joined photons attributes within a given area*

Description

Computes a series of statistics ATL03 and ATL08 joined photons attributes within area defined by a polygon

Usage

```
ATL03_ATL08_photons_attributes_dt_polyStat(
  atl03_atl08_dt,
  func,
  poly_id = NULL
)
```

Arguments

`atl03_atl08_dt`	An S4 object of class `icesat2.atl08_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).
`func`	The function to be applied for computing the defined statistics
`poly_id`	Polygon id. If defined, statistics will be computed for each polygon

Value

Returns an S4 object of class `icesat2.atl08_dt` Containing Statistics of ATL08 classified canopy photons

Examples

```
# Specifying the path to ATL03 and ATL08 files
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL03 and ATL08 photons and heights
atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)
head(atl03_atl08_dt)

# Specifying the path to shapefile
polygon_filepath <- system.file("extdata", "clip_geom.shp", package = "ICESat2VegR")

# Reading shapefile as sf object
polygon <- terra::vect(polygon_filepath)

# Clipping ATL08 terrain attributes by Geometry
atl03_atl08_dt_clip <- ATL03_ATL08_photons_attributes_dt_clipGeometry(atl03_atl08_dt,
  polygon, split_by = "id")

# Computing the maximum ph_h by polygon id
max_ph_h <- ATL03_ATL08_photons_attributes_dt_polyStat(atl03_atl08_dt_clip,
  func = max(ph_h), poly_id = "poly_id")
head(max_ph_h)

# Define your own function
mySetOfMetrics <- function(x) {
  metrics <- list(
    min = min(x), # Min of x
    max = max(x), # Max of x
    mean = mean(x), # Mean of x
    sd = sd(x) # Sd of x
  )
  return(metrics)
}

# Computing a series of ph_h statistics from customized function
ph_h_metrics <- ATL03_ATL08_photons_attributes_dt_polyStat(
  atl03_atl08_dt_clip,
  func = mySetOfMetrics(ph_h),
  poly_id = "poly_id"
)
```

```
head(ph_h_metrics)

close(atl03_h5)
close(atl08_h5)
```

`ATL03_ATL08_photons_dt_height_normalize` *Fit and estimate ground elevation for photons or arbitrary distances from the track beginning.*

Description

Function to estimate ground elevation using smoothing and interpolation functions

Usage

```
ATL03_ATL08_photons_dt_height_normalize(
  atl03_atl08_seg_dt,
  smoothing_window = NA,
  smoothing_func = median,
  interpolation_func = NA,
  xout_parameter_name = "xout",
  ...
)
```

Arguments

`atl03_atl08_seg_dt`	An S4 object of class `icesat2.atl03_atl08_seg_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).
`smoothing_window`	numeric. The smoothing window size in meters for smoothing the photon cloud. Default is NA, see details for more information.
`smoothing_func`	function. The smoothing function to be applied on the smoothing window.
`interpolation_func`	function. The interpolation function to estimate the ground elevation. Default `stats::approx()`.
`xout_parameter_name`	character. The parameter name used for the `interpolation_func` to which will be used to predict, default "xout".
`...`	parameters to be passed forward to `ATL03_ATL08_photons_seg_dt_fitground()`.

Details

The function for calculating the ground will first pass a smoothing window with `smoothing_window` size, applying the `smoothing_func` to aggregate the ground photons.

Then it will use an interpolation function between those aggregated photons to calculate a smooth surface.

The `smoothing_func` signature will depend on the function used. It is assumed that the first two arguments are vectors of `x` (independent variable) and `y` (the prediction to be interpolated). The remaining arguments are passed through `...`.

The interpolation functions need a third parameter which is the `x` vector to be interpolated. Functions from `stats` base package `stats::approx()` and `stats::spline()` name this argument as `xout`, so you can use:

```
ATL03_ATL08_photons_fitground_seg_dt(
  dt,
  interpolation_func = approx,
  xout = 1:30
)
```

For example, to interpolate the values for the 1:30 vector. However, other functions may name the parameter differently, such as `signal::pchip()`, which calls the parameter `xi` instead of `xout`. The `pchip` algorithm (as implemented in the **signal** package) is the one used by the ATL08 ATBD.

The `smoothing_window` can be left NA, which will use the ATBD algoritm for calculating the window size:

$Sspan = ceil[5 + 46 * (1 - e^{-a*length})]$, where *length* is the number of photons within segment.

$$a \approx 21x10^{-6}$$

$$window_size = \frac{2}{3}Sspan$$

This is not the same algorithm as used in ATL08 but is an adapted version that uses the ATL08 pre-classification

---

`ATL03_ATL08_photons_seg_dt_fitground`

Fit and estimate ground elevation for photons or arbitrary distances from the track beginning.

Description

Function to estimate ground elevation using smoothing and interpolation functions

Usage

```
ATL03_ATL08_photons_seg_dt_fitground(
  atl03_atl08_seg_dt,
  smoothing_window = NA,
  smoothing_func = median,
  interpolation_func = NA,
  xout_parameter_name = "xout",
  ...
)
```

Arguments

`atl03_atl08_seg_dt`
: An S4 object of class `icesat2.atl03_atl08_seg_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).

`smoothing_window`
: numeric. The smoothing window size in meters for smoothing the photon cloud. Default is NA, see details for more information.

`smoothing_func`
: function. The smoothing function to be applied on the smoothing window.

`interpolation_func`
: function. The interpolation function to estimate the ground elevation.

`xout_parameter_name`
: character. Optional, can be used to inform the parameter name that the interpolation_func uses for passing the prediction vector and already use the photons for prediction. Default NA will use the ...

`...`
: Optional parameters to pass to the interpolation_func, see details for more information.

Details

The function for calculating the ground will first pass a smoothing window with `smoothing_window` size, applying the `smoothing_func` to aggregate the ground photons.

Then it will use an interpolation function between those aggregated photons to calculate a smooth surface.

The `smoothing_func` signature will depend on the function used. It is assumed that the first two arguments are vectors of `x` (independent variable) and `y` (the prediction to be interpolated). The remaining arguments are passed through `...`.

The interpolation functions need a third parameter which is the `x` vector to be interpolated. Functions from `stats` base package `stats::approx()` and `stats::spline()` name this argument as `xout`, so you can use:

```
ATL03_ATL08_photons_fitground_seg_dt(
  dt,
  interpolation_func = approx,
  xout = 1:30
)
```

For example, to interpolate the values for the 1:30 vector. However, other functions may name the parameter differently, such as `signal::pchip()`, which calls the parameter `xi` instead of `xout`. The `pchip` algorithm (as implemented in the **signal** package) is the one used by the ATL08 ATBD.

The `smoothing_window` can be left NA, which will use the ATBD algoritm for calculating the window size:

$Sspan = ceil[5 + 46 * (1 - e^{-a*length})]$, where *length* is the number of photons within segment.

$$a \approx 21x10^{-6}$$

$$window_size = \frac{2}{3}Sspan$$

This is not the same algorithm as used in ATL08 but is an adapted version that uses the ATL08 pre-classification

`ATL03_ATL08_segment_create` *Compute segments id for a given segment length*

Description

This function reads the ICESat-2 Land and Vegetation Along-Track Products (ATL08) as h5 file.

Usage

```
ATL03_ATL08_segment_create(
  atl03_atl08_dt,
  segment_length,
  centroid = "mean",
  output = NA,
  overwrite = FALSE
)
```

Arguments

`atl03_atl08_dt`	`icesat2.atl03atl08_dt`. The output of the `ATL03_ATL08_photons_attributes_dt_join()`.
`segment_length`	`numeric`. The desired segment length to split the photons.
`centroid`	character. Method used to calculate the segment centroid, either "mean" or "mid-point", see details. Default 'mean'.
`output`	Character vector. The output vector file. The GDAL vector format will be inferred by the file extension using `terra::writeVector()`
`overwrite`	logical input to control if the output vector file should be overwritten. Default FALSE.

Details

The centroid will be computed using either the photons centroid or the approximate segment centroid.

- "mean": calculated using the average coordinates from all photons within the segment. This approach will better represent the mean statistics location.
- "mid-point": the minimum and maximum coordinates will be averaged to calculate a midpoint within the segment. This will give a better representation of the segment true mid-point

Value

Returns an S4 object of class `icesat2.atl03atl08_dt` containing ICESat-2 ATL08 data.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
# Specifying the path to ICESat-2 ATL03 and ATL08 data
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ICESat-2 ATL08 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

atl03_atl08_dt <- ATL03_ATL08_photons_attributes_dt_join(atl03_h5, atl08_h5)

atl03_atl08_dt_seg <- ATL03_ATL08_segment_create(atl03_atl08_dt,
  segment_length = 30,
  centroid = "mean",
  output = NA,
  overwrite = FALSE
)

head(atl03_atl08_dt_seg)

close(atl03_h5)
close(atl08_h5)
```

ATL03_ATL08_seg_cover_dt_compute

Compute segments id for a given segment length

Description

This function reads the ICESat-2 Land and Vegetation Along-Track Products (ATL08) as h5 file.

Usage

```
ATL03_ATL08_seg_cover_dt_compute(atl03_atl08_dt, reflectance_ratio = 1)
```

Arguments

`atl03_atl08_dt` `icesat2.atl03atl08_dt`. The output of the `ATL03_ATL08_photons_attributes_dt_join()`.

`reflectance_ratio`
: Numeric. The reflectance ratio to use to compute the coverage metric. ρ_v/ρ_g, where ρ_v is the vegetation reflectance and ρ_v is the ground reflectance. Default is 1 (same reflectance).

Details

Coverage is calculated with the formula

$$\frac{1}{1 + \frac{\rho_v N_g}{\rho_g N_v}}$$

Value

Returns an S4 object of class `data.table::data.table` containing ICESat-2 ATL08 data.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ICESat-2 ATL08 data (h5 file)
atl08 <- ATL08_read(atl08_path = atl08_path)

close(atl08)
```

ATL03_h5_clipBox *Clips ICESat-2 ATL03 H5 data*

Description

This function clips ATL03 HDF5 file within beam groups, but keeps metada and ancillary data the same.

Usage

```
ATL03_h5_clipBox(
  atl03,
  output,
  bbox,
  beam = c("gt1r", "gt2r", "gt3r", "gt1l", "gt2l", "gt3l"),
  additional_groups = c("orbit_info")
)
```

Arguments

atl03	`icesat2.atl03_h5` object, obtained through `ATL03_read()` for clipping
output	character. Path to the output h5 file.
bbox	`numeric` or `terra::SpatExtent` for clipping, the order of the bbox is the default from NASA's ICESat-2 CMS searching: (ul_lat, ul_lon, lr_lat, lr_lon).

beam character. The vector of beams to include, default all c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")

additional_groups character. Other addional groups that should be included, default c("orbit_info")

Value

Returns the clipped S4 object of class icesat2.atl03_h5

Examples

```
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Bounding rectangle coordinates
xmin <- -106.5723
xmax <- -106.5693
ymin <- 41.533
ymax <- 41.537

# Clipping ATL03 photons by boundary box extent
output <- tempfile(fileext = ".h5")
atl03_photons_dt_clip <- ATL03_h5_clipBox(atl03_h5, output, c(ymax, xmin, ymin, xmax))

close(atl03_h5)
```

ATL03_h5_clipGeometry *Clips ICESat-2 ATL03 H5 data*

Description

This function clips ATL03 HDF5 file within beam groups, but keeps metada and ancillary data the same.

Usage

```
ATL03_h5_clipGeometry(
  atl03,
  output,
  vect,
  clip_obj_id = NULL,
  beam = c("gt1r", "gt2r", "gt3r", "gt1l", "gt2l", "gt3l"),
  additional_groups = c("orbit_info")
)
```

Arguments

atl03	icesat2.atl03_h5 object, obtained through ATL03_read() for clipping
output	character. Path to the output h5 file, the attribute for clip_objs will be appended to the file name.
vect	terra::SpatVector for clipping
clip_obj_id	character. The attribute name used for identifying the different clip_objs. Default is "id"
beam	character. The vector of beams to include, default all c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")
additional_groups	character. Other addional groups that should be included, default c("orbit_info")

Value

Returns a list of clipped S4 object of class icesat2.atl03_h5

Examples

```
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

output <- tempfile(fileext = ".h5")

vect_path <- system.file("extdata",
  "clip_objs.shp",
  package = "ICESat2VegR"
)

vect <- terra::vect()

# Clipping ATL03 photons  by boundary box extent
atl03_photons_dt_clip <- ATL03_h5_clipGeometry(
  atl03_h5,
  output,
  vect,
  clip_obj_id = "id"
)

close(atl03_h5)
```

ATL03_photons_attributes_dt

ATL03 photon attributes

Description

Extract photon-level attributes from ICESat-2 ATL03 data.

Usage

```
ATL03_photons_attributes_dt(
  atl03_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
)
```

Arguments

`atl03_h5`	An ICESat-2 ATL03 object (output of `ATL03_read()`), i.e. an S4 object of class `icesat2.atl03_h5`.
`beam`	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r").

Details

Attributes extracted include:

- `lon_ph`: Longitude of each received photon, computed from ECEF Cartesian coordinates of the bounce point.
- `lat_ph`: Latitude of each received photon, computed from ECEF Cartesian coordinates of the bounce point.
- `h_ph`: Height of each received photon, relative to the WGS-84 ellipsoid, including the geophysical corrections noted in the ATL03 ATBD. (Geoid, ocean tide and DAC are *not* applied.)
- `quality_ph`: Photon quality flag (0 = nominal, 1 = possible_afterpulse, 2 = possible_impulse_response_effect, 3 = possible_tep). Use together with `signal_conf_ph` to identify likely noise vs likely signal.
- `solar_elevation`: Solar elevation interpolated from segment-level `geolocation/solar_elevation` to each photon.
- `dist_ph_along`: Along-track distance for each photon (segment cumulative length + `heights/dist_ph_along` when available).
- `beam`: Beam ID (e.g. "gt2l").
- `strong_beam`: Logical indicating whether the beam is classified as strong for that orbit.

Value

An S4 object of class `data.table::data.table` (with class `"icesat2.atl03_dt"` prepended) containing photon-level ATL03 attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
## Not run:
atl03_path <- system.file(
  "extdata", "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl03_h5 <- ATL03_read(atl03_path)

atl03_photons_dt <- ATL03_photons_attributes_dt(atl03_h5)
head(atl03_photons_dt)

close(atl03_h5)

## End(Not run)
```

`ATL03_photons_attributes_dt_clipBox`
Clip ATL03 photons by Coordinates

Description

This function clips ATL03 photons attributes within a given bounding coordinates

Usage

```
ATL03_photons_attributes_dt_clipBox(
  atl03_photons_dt,
  lower_left_lon,
  upper_right_lon,
  lower_left_lat,
  upper_right_lat
)
```

Arguments

`atl03_photons_dt`	A atl03_photons_dt object (output of `ATL03_photons_attributes_dt()` function). An S4 object of class `icesat2.atl03_dt`
`lower_left_lon`	Numeric. West longitude (x) coordinate of bounding rectangle, in decimal degrees.
`upper_right_lon`	Numeric. East longitude (x) coordinate of bounding rectangle, in decimal degrees.
`lower_left_lat`	Numeric. South latitude (y) coordinate of bounding rectangle, in decimal degrees.
`upper_right_lat`	Numeric. North latitude (y) coordinate of bounding rectangle, in decimal degrees.

Value

Returns an S4 object of class icesat2.atl03_dt containing the ATL03 photons attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
# Specifying the path to ATL03 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Extracting ATL03 photons attributes
atl03_photons_dt <- ATL03_photons_attributes_dt(atl03_h5 = atl03_h5)

# Bounding rectangle coordinates
lower_left_lon <- -106.57
lower_left_lat <- 41.53
upper_right_lon <- -106.5698
upper_right_lat <- 41.54

# Clipping ATL08-derived canopy metrics by boundary box extent
atl03_photons_dt_clip <- ATL03_photons_attributes_dt_clipBox(
  atl03_photons_dt,
  lower_left_lon,
  upper_right_lon,
  lower_left_lat,
  upper_right_lat
)

head(atl03_photons_dt_clip)

close(atl03_h5)
```

ATL03_photons_attributes_dt_clipGeometry

Clip ATL03 photons by Coordinates

Description

This function clips ATL03 photon attributes within given bounding coordinates.

Usage

```
ATL03_photons_attributes_dt_clipGeometry(
  atl03_photons_dt,
  clip_obj,
  split_by = "id"
)
```

Arguments

atl03_photons_dt	An ATL03 photon data table. An S4 object of class icesat2.atl03_dt.
clip_obj	Spatial clip_obj. An object of class terra::SpatVector, which can be loaded as an ESRI shapefile using the terra::vect function in the *sf* package.
split_by	clip_obj id. If defined, GEDI data will be clipped by each clip_obj using the clip_obj id from the attribute table defined by the user.

Value

Returns an S4 object of class icesat2.atl03_dt containing the ATL03 photon attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Extracting ATL03 photon attributes
atl03_photons_dt <- ATL03_photons_attributes_dt(atl03_h5 = atl03_h5)

# Specifying the path to shapefile
clip_obj_filepath <-
  system.file(
    "extdata",
    "clip_geom.shp",
    package = "ICESat2VegR"
  )

# Reading shapefile as sf object
clip_obj <- terra::vect(clip_obj_filepath)

# Clipping ATL03 photon attributes by Geometry
atl03_photons_dt_clip <-
  ATL03_photons_attributes_dt_clipGeometry(atl03_photons_dt, clip_obj, split_by = "id")

head(atl03_photons_dt_clip)
```

```
close(atl03_h5)
```

`ATL03_photons_attributes_dt_LAS` *Converts ATL03 photon cloud to LAS*

Description

Converts ATL03 photon cloud to LAS

Usage

```
ATL03_photons_attributes_dt_LAS(atl03_dt, output)
```

Arguments

`atl03_dt`	An S4 object of class `icesat2.atl03_dt` (output of `ATL03_photons_attributes_dt()` function).
`output`	character. The output path of for the LAS(Z) file(s) The function will create one LAS file per UTM Zone in WGS84 datum.

Details

As the las format expects a metric coordinate reference system (CRS) we use helper functions to define UTM zones to which the original ICESat-2 data will be converted.

The function credits go to Chuck Gantz- chuck.gantz@globalstar.com.

Value

Nothing, it just saves outputs as LAS file in disk

See Also

https://oceancolor.gsfc.nasa.gov/docs/ocssw/LatLong-UTMconversion_8cpp_source.html

Examples

```
# ATL03 file path
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Extracting ATL03 and ATL08 photons and heights
atl03_dt <- ATL03_photons_attributes_dt(atl03_h5, beam = "gt1r")

outdir <- tempdir()
ATL03_photons_attributes_dt_LAS(
  atl03_dt,
```

```
  file.path(outdir, "output.laz")
)

close(atl03_h5)
```

ATL03_read *Read ICESat-2 ATL03 data*

Description

This function reads the ICESat-2 Global Geolocated Photons (ATL03) Product (ATL03) as h5 file.

Usage

```
ATL03_read(atl03_path)
```

Arguments

atl03_path	Either file path pointing to ICESat-2 ATL03 h5 data or a granule resulting from ATLAS_dataFinder() with cloud_computing = TRUE.

Value

Returns an S4 object of class icesat2.atl03_dt containing ICESat-2 ATL03 data.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
# Specifying the path to ATL03 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ICESat-2 ATL03 data (h5 file)
ATL03 <- ATL03_read(atl03_path = atl03_path)
close(ATL03)
```

`ATL03_seg_metadata_dt` *ATL03 geolocation segment metadata*

Description

Extract geolocation segment–level metadata from ICESat-2 ATL03 data. Each row in the output corresponds to a single **20 m geolocation segment** along track, not to individual photons.

In addition to variables from the ATL03 `geolocation` group, this function derives a segment-level reference photon height (`h_ph`) using the reference photon index and the photon-height array in the `heights` group.

Usage

```
ATL03_seg_metadata_dt(
  atl03_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  attributes = c("h_ph", "altitude_sc", "bounce_time_offset", "delta_time",
   "full_sat_fract", "near_sat_fract", "neutat_delay_derivative", "neutat_delay_total",
    "neutat_ht", "ph_index_beg", "pitch", "podppd_flag", "range_bias_corr",
   "ref_azimuth", "ref_elev", "reference_photon_index", "roll", "segment_dist_x",
   "segment_id", "segment_length", "segment_ph_cnt", "sigma_across", "sigma_along",
   "sigma_h", "sigma_lat", "sigma_lon", "solar_azimuth", "solar_elevation", "surf_type",
    "tx_pulse_energy", "tx_pulse_skew_est",
     "tx_pulse_width_lower",
    "tx_pulse_width_upper", "yaw")
)
```

Arguments

`atl03_h5`	An ICESat-2 ATL03 object (output of `ATL03_read()`), i.e. an S4 object of class `icesat2.atl03_h5`.
`beam`	Character vector indicating beams to process (e.g. `"gt1l"`, `"gt1r"`, `"gt2l"`, `"gt2r"`, `"gt3l"`, `"gt3r"`). Only beams present in `atl03_h5$beams` are used.
`attributes`	Character vector naming the segment-level variables to extract. By default, a broad set of geolocation and quality variables is used, including a derived reference-photon height (`h_ph`).

Details

The following variables may be requested via `attributes`:

- `h_ph`: Height of the **reference photon** above the WGS84 ellipsoid for each geolocation segment. This is derived from `geolocation/ph_index_beg`, `geolocation/reference_photon_index`, and `heights/h_ph`.
- `altitude_sc`: Height of the spacecraft above the WGS84 ellipsoid.
- `bounce_time_offset`: Difference between the transmit time and the ground-bounce time of the reference photon.
- `delta_time`: Transmit time of the reference photon, measured in seconds from `atlas_sdp_gps_epoch`.
- `full_sat_fract`: Fraction of pulses within the segment that are fully saturated.

- near_sat_fract: Fraction of pulses within the segment that are nearly saturated.
- neutat_delay_derivative: Change in neutral atmospheric delay per unit height change.
- neutat_delay_total: Total neutral atmosphere delay correction (wet + dry).
- neutat_ht: Reference height of the neutral atmosphere range correction.
- ph_index_beg: 1-based index of the first photon in this segment within the photon-rate data.
- pitch: Spacecraft pitch (degrees), 3–2–1 Euler sequence.
- podppd_flag: Composite flag describing the quality of input geolocation products for the segment.
- range_bias_corr: Estimated range bias from geolocation analysis.
- ref_azimuth: Azimuth (radians) of the unit pointing vector for the reference photon in the local ENU frame.
- ref_elev: Elevation (radians) of the unit pointing vector for the reference photon in the local ENU frame.
- reference_photon_index: Index of the reference photon within the photon set for a segment.
- reference_photon_lat: Latitude of the reference photon.
- reference_photon_lon: Longitude of the reference photon.
- roll: Spacecraft roll (degrees), 3–2–1 Euler sequence.
- segment_dist_x: Along-track distance from the equator crossing to the start of the 20 m geolocation segment.
- segment_id: 7-digit along-track geolocation segment identifier.
- segment_length: Along-track length of the geolocation segment (typically 20 m).
- segment_ph_cnt: Number of photons in the segment.
- sigma_across: Estimated Cartesian across-track uncertainty (1-sigma) for the reference photon.
- sigma_along: Estimated Cartesian along-track uncertainty (1-sigma) for the reference photon.
- sigma_h: Estimated height uncertainty (1-sigma) for the reference photon bounce point.
- sigma_lat: Estimated geodetic latitude uncertainty (1-sigma) for the reference photon.
- sigma_lon: Estimated geodetic longitude uncertainty (1-sigma) for the reference photon.
- solar_azimuth: Azimuth (degrees east) of the sun position vector from the reference photon bounce point in the local ENU frame.
- solar_elevation: Elevation (degrees) of the sun position vector from the reference photon bounce point in the local ENU frame.
- surf_type: Flags describing which surface types the segment is associated with (land, ocean, sea ice, land ice, inland water).
- tx_pulse_energy: Average transmit pulse energy per beam.
- tx_pulse_skew_est: Difference between the averages of the lower and upper threshold crossing times, estimating transmit pulse skew.
- tx_pulse_width_lower: Average distance between lower threshold crossing times measured by the Start Pulse Detector.
- tx_pulse_width_upper: Average distance between upper threshold crossing times measured by the Start Pulse Detector.
- yaw: Spacecraft yaw (degrees), 3–2–1 Euler sequence.

Value

A data.table::data.table with one row per ATL03 geolocation segment, with class "icesat2.atl03_seg_dt" prepended. Columns include:

- beam – beam ID (e.g., "gt2l").
- strong_beam – logical flag indicating whether the beam is classified as strong for the orbit.
- selected geolocation fields (see below).
- a derived h_ph column: height of the reference photon for each segment (one value per segment).

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
## Not run:
atl03_path <- system.file(
  "extdata", "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl03_h5 <- ATL03_read(atl03_path)

# Extract ATL03 geolocation segment metadata
atl03_segment_dt <- ATL03_seg_metadata_dt(atl03_h5)

head(atl03_segment_dt)
close(atl03_h5)

## End(Not run)
```

ATL08_attributes_dt_LAS

Read ICESat-2 ATL08 data

Description

This function reads the ICESat-2 Land and Vegetation Along-Track Products (ATL08) as h5 file.

Usage

```
ATL08_attributes_dt_LAS(atl08_path)
```

Arguments

atl08_path	File path pointing to ICESat-2 ATL08 data. Data in HDF5 Hierarchical Data Format (.h5).

Value

Returns an S4 object of class icesat2.atl08_dt containing ICESat-2 ATL08 data.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ICESat-2 ATL08 data (h5 file)
atl08 <- ATL08_read(atl08_path = atl08_path)

close(atl08)
```

ATL08_h5_clipBox *Clips ICESat-2 ATL08 data*

Description

This function clips the Atl08 HDF5 file. This function will only clip the beam groups within hdf5, it won't change metadata or ancillary data.

Usage

```
ATL08_h5_clipBox(
  atl08,
  output,
  bbox,
  beam = c("gt1r", "gt2r", "gt3r", "gt1l", "gt2l", "gt3l"),
  additional_groups = c("orbit_info")
)
```

Arguments

atl08	icesat2.atl08_h5 object, obtained through ATL08_read() for clipping
output	character. Path to the output h5 file.
bbox	numeric or terra::SpatExtent for clipping, the order of the bbox is the default from NASA's ICESat-2 CMS searching: (ul_lat, ul_lon, lr_lat, lr_lon).
beam	character. The vector of beams to include, default all c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")
additional_groups	character. Other addional groups that should be included, default c("orbit_info")

Value

Returns the clipped S4 object of class icesat2.atl08_h5

Examples

```
# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Bounding rectangle coordinates
ul_lon <- -106.5723
lr_lon <- -106.5693
lr_lat <- 41.533
ul_lat <- 41.537

# Clipping ATL08 terrain and canopy attributes by boundary box
atl08_seg_att_dt_clip <- ATL08_h5_clipBox(
  atl08_h5,
  output = tempfile(fileext = ".h5"),
  c(ul_lat, ul_lon, lr_lat, lr_lon)
)

close(atl08_h5)
close(atl08_seg_att_dt_clip)
```

ATL08_h5_clipGeometry *Clips ICESat-2 ATL08 data*

Description

This function clips ATL08 HDF5 file within beam groups, but keeps metada and ancillary data the same.

Usage

```
ATL08_h5_clipGeometry(
  atl08,
  output,
  vect,
  clip_obj_id = "id",
  beam = c("gt1r", "gt2r", "gt3r", "gt1l", "gt2l", "gt3l"),
  additional_groups = c("orbit_info")
)
```

Arguments

atl08	icesat2.atl08_h5 object, obtained through ATL08_read() for clipping
output	character. Path to the output h5 file.
vect	terra::SpatVector for clipping
clip_obj_id	character. The attribute name used for identifying the different clip_objs. Default is "id"
beam	character. The vector of beams to include, default all c("gt1l", "gt2l", "gt3l", "gt1r", "gt2r", "gt3r")
additional_groups	character. Other addional groups that should be included, default c("orbit_info")

Value

Returns a list of clipped S4 object of class icesat2.atl08_h5

Examples

```
# ATL08 file path
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

output <- tempfile(fileext = ".h5")

vect_path <- system.file("extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)

vect <- terra::vect(vect_path)

# Clipping ATL08 photons by boundary box extent
atl08_photons_dt_clip <- ATL08_h5_clipGeometry(
  atl08_h5,
  output,
  vect,
  clip_obj_id = "id"
)

close(atl08_h5)
```

ATL08_photons_attributes_dt

ATL08 computed photons attributes

Description

This function extracts computed photons attributes from ICESat-2 ATL08 data

Usage

```
ATL08_photons_attributes_dt(
  atl08_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
 photon_attributes = c("ph_segment_id", "classed_pc_indx", "classed_pc_flag", "ph_h",
    "d_flag", "delta_time")
)
```

Arguments

atl08_h5	A ICESat-2 ATL08 object (output of ATL08_read() function). An S4 object of class icesat2.atl08_dt.
beam	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
photon_attributes	character vector indicating the attributes to extract from the ATL08 photons. Default all c("ph_segment_id", "classed_pc_indx", "classed_pc_flag", "ph_h", "d_flag", "delta_time").

Details

These are the photons attributes extracted by default:

- **ph_segment_id**: Georeferenced bin number (20-m) associated with each photon
- **classed_pc_indx**: Indices of photons tracking back to ATL03 that surface finding software identified and used within the creation of the data products.
- **classed_pc_flag**: The L2B algorithm is run if this flag is set to 1 indicating data have sufficient waveform fidelity for L2B to run
- **ph_h**: Height of photon above interpolated ground surface
- **d_flag**: Flag indicating whether DRAGANN labeled the photon as noise or signal
- **delta_time**: Mid-segment GPS time in seconds past an epoch. The epoch is provided in the metadata at the file level

Value

Returns an S4 object of class data.table::data.table containing the ATL08 computed photons attributes.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
```

```
atl08_h5 <- ATL08_read(atl08_path)

# Extracting ATL08 classified photons and heights
atl08_photons <- ATL08_photons_attributes_dt(atl08_h5 = atl08_h5)

head(atl08_photons)
close(atl08_h5)
```

`ATL08_read` *Read ICESat-2 ATL08 data*

Description

This function reads the ICESat-2 Land and Vegetation Along-Track Products (ATL08) as h5 file.

Usage

```
ATL08_read(atl08_path)
```

Arguments

`atl08_path`	File path pointing to ICESat-2 ATL08 data. Data in HDF5 Hierarchical Data Format (.h5).

Value

Returns an S4 object of class `icesat2.atl08_h5` containing ICESat-2 ATL08 data.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ICESat-2 ATL08 data (h5 file)
atl08 <- ATL08_read(atl08_path = atl08_path)
close(atl08)
```

`ATL08_seg_attributes_dt` *ATL08 Terrain and Canopy Attributes*

Description

This function extracts terrain and canopy attributes by segments from ICESat-2 ATL08 data

Usage

```
ATL08_seg_attributes_dt(
  atl08_h5,
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  attributes = c("delta_time", "h_canopy", "canopy_openness", "h_te_mean",
    "terrain_slope")
)
```

Arguments

`atl08_h5`	A ICESat-2 ATL08 object (output of `ATL08_read()` function). An S4 object of class `icesat2.atl08_dt`.
`beam`	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
`attributes`	A character vector containing the list of terrain and canopy attributes to be extracted. Default is attribute = c("h_canopy","canopy_h_metrics","canopy_openness","h_te_mean","h

Details

ATL08 canopy attributes:

- *"h_canopy"*
- *"canopy_rh_conf"*
- *"h_median_canopy_abs"*
- *"h_min_canopy"*
- *"h_mean_canopy_abs"*
- *"h_median_canopy"*
- *"h_canopy_abs"*
- *"toc_roughness"*
- *"h_min_canopy_abs"*
- *"h_dif_canopy"*
- *"h_canopy_quad"*
- *"h_canopy_20m"*
- *"n_ca_photons"*
- *"photon_rate_can"*
- *"centroid_height"*
- *"canopy_h_metrics_abs"*

- *"h_mean_canopy"*
- *"subset_can_flag"*
- *"canopy_h_metrics"*
- *"n_toc_photons"*
- *"h_max_canopy_abs"*
- *"h_canopy_uncertainty"*
- *"canopy_openness"*
- *"h_max_canopy"*
- *"segment_cover"*

ATL08 terrain attributes:

- *"h_te_best_fit"*
- *"h_te_best_fit_20m"*
- *"h_te_interp"*
- *"h_te_max"*
- *"h_te_mean"*
- *"h_te_median"*
- *"h_te_mode"*
- *"h_te_rh25"*
- *"h_te_skew"*
- *"h_te_std"*
- *"h_te_uncertainty"*
- *"n_te_photons"*
- *"photon_rate_te"*
- *"subset_te_flag"*
- *"terrain_slope"*

Value

Returns an S4 object of class `icesat2.atl08_dt` containing the ATL08-derived terrain and canopy attributes by segments.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL08_ATBD_r006.pdf

Examples

```
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)
```

```
# Extracting ATL08-derived terrain and canopy attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)
head(atl08_seg_att_dt)

close(atl08_h5)
```

`ATL08_seg_attributes_dt_clipBox` *Clip ATL08 Terrain and Canopy Attributes by Bounding Box*

Description

This function clips ATL08 Terrain and Canopy Attributes within a given bounding box coordinates

Usage

```
ATL08_seg_attributes_dt_clipBox(
  atl08_seg_att_dt,
  lower_left_lon,
  upper_right_lon,
  lower_left_lat,
  upper_right_lat
)
```

Arguments

`atl08_seg_att_dt`	A atl08_seg_att_dt object (output of `ATL08_seg_attributes_dt()` function). An S4 object of class `icesat2.atl08_dt`
`lower_left_lon`	Numeric. West longitude (x) coordinate of bounding rectangle, in decimal degrees.
`upper_right_lon`	Numeric. East longitude (x) coordinate of bounding rectangle, in decimal degrees.
`lower_left_lat`	Numeric. South latitude (y) coordinate of bounding rectangle, in decimal degrees.
`upper_right_lat`	Numeric. North latitude (y) coordinate of bounding rectangle, in decimal degrees.

Value

Returns an S4 object of class `icesat2.atl08_dt` containing the clipped ATL08 terrain and canopy attributes.

Examples

```
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path)

# Extracting ATL08-derived Canopy Metrics
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

# Bounding rectangle coordinates
lower_left_lon <- -103.7604
lower_left_lat <- 59.4672
upper_right_lon <- -103.7600
upper_right_lat <- 59.4680

# Clipping ATL08 terrain and canopy attributes by boundary box
atl08_seg_att_dt_clip <- ATL08_seg_attributes_dt_clipBox(
  atl08_seg_att_dt,
  lower_left_lon,
  upper_right_lon,
  lower_left_lat,
  upper_right_lat
)

close(atl08_h5)
```

ATL08_seg_attributes_dt_clipGeometry

Clip ATL08 Terrain and Canopy Attributes by Geometry

Description

This function clips ATL08 Terrain and Canopy Attributes within a given geometry

Usage

```
ATL08_seg_attributes_dt_clipGeometry(
  atl08_seg_att_dt,
  clip_obj,
  split_by = NULL
)
```

Arguments

atl08_seg_att_dt	A atl08_seg_att_dt object (output of ATL08_seg_attributes_dt() function). An S4 object of class icesat2.atl08_dt
clip_obj	clip_obj. An object of class terra::SpatVector, which can be loaded as an ESRI shapefile using terra::vect function in the *sf* package.

split_by clip_obj id. If defined, ATL08 data will be clipped by each clip_obj using the clip_obj id from table of attribute defined by the user

Value

Returns an S4 object of class icesat2.atl08_dt containing the clipped ATL08 Terrain and Canopy Attributes.

Examples

```
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08-derived terrain and canopy attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

clip_obj_path <- system.file("extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)

if (require(terra)) {
  clip_obj <- terra::vect(clip_obj_path)

  head(atl08_seg_att_dt)
  # Clipping ATL08 Terrain and Canopy Attributes by Geometry
  atl08_seg_att_dt_clip <- ATL08_seg_attributes_dt_clipGeometry(atl08_seg_att_dt,
      clip_obj, split_by = "id")

  hasLeaflet <- require(leaflet)

  if (hasLeaflet) {
    leaflet() %>%
      addCircleMarkers(atl08_seg_att_dt_clip$longitude,
        atl08_seg_att_dt_clip$latitude,
        radius = 1,
        opacity = 1,
        color = "red"
      ) %>%
      addScaleBar(options = list(imperial = FALSE)) %>%
      addclip_objs(
        data = clip_obj, weight = 1, col = "white",
        opacity = 1, fillOpacity = 0
      ) %>%
      addProviderTiles(providers$Esri.WorldImagery,
        options = providerTileOptions(minZoom = 3, maxZoom = 17)
      )
  }
}
close(atl08_h5)
```

`ATL08_seg_attributes_dt_gridStat` *Statistics of ATL08 Terrain and Canopy Attributes at grid level*

Description

This function computes a series of user defined descriptive statistics within each grid cell for ATL08 Terrain and Canopy Attributes

Usage

```
ATL08_seg_attributes_dt_gridStat(atl08_seg_att_dt, func, res = 0.5)
```

Arguments

`atl08_seg_att_dt`	An S4 object of class `icesat2.atl08_dt` containing ATL08 data (output of `ATL08_seg_attributes_dt()` functions).
`func`	The function to be applied for computing the defined statistics
`res`	Spatial resolution in decimal degrees for the output SpatRast raster layer. Default is 0.5.

Value

Return a SpatRast raster layer(s) of selected ATL08 terrain and canopy attribute(s)

Examples

```
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08-derived terrain and canopy attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

# Computing the top h_canopy at 0.05 degree grid cell
res <- 0.0001
mean_h_canopy <- ATL08_seg_attributes_dt_gridStat(
  atl08_seg_att_dt,
  func = mean(h_canopy),
  res = res
)

plot(mean_h_canopy)

# Define your own function
mySetOfMetrics <- function(x) {
  metrics <- list(
```

```
    min = min(x, na.rm = TRUE), # Min of x
    max = max(x, na.rm = TRUE), # Max of x
    mean = mean(x, na.rm = TRUE), # Mean of x
    sd = sd(x, na.rm = TRUE) # Sd of x
  )
  return(metrics)
}

res <- 0.05
# Computing h_canopy statistics at 0.05 degree grid cell from user-defined function
h_canopy_metrics <- ATL08_seg_attributes_dt_gridStat(
  atl08_seg_att_dt,
  func = mySetOfMetrics(h_canopy),
  res = res
)

plot(h_canopy_metrics)

close(atl08_h5)
```

`ATL08_seg_attributes_dt_LAS` *Converts ATL08 segments to LAS*

Description

Converts ATL08 segments to LAS

Usage

```
ATL08_seg_attributes_dt_LAS(atl08_dt, output)
```

Arguments

`atl08_dt`	An S4 object of class `icesat2.atl08_dt` containing ATL03 and ATL08 data (output of `ATL03_ATL08_photons_attributes_dt_join()` function).
`output`	character. The output path of for the LAS(Z) file(s) The function will create one LAS file per UTM Zone in WGS84 datum.

Details

As the las format expects a metric coordinate reference system (CRS) we use helper functions to define UTM zones to which the original ICESat-2 data will be converted.

The function credits go to Chuck Gantz- chuck.gantz@globalstar.com.

Value

Nothing, it just saves outputs as LAS file in disk

See Also

https://oceancolor.gsfc.nasa.gov/docs/ocssw/LatLong-UTMconversion_8cpp_source.html

Examples

```
# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# # Extracting ATL03 and ATL08 photons and heights
atl08_dt <- ATL08_seg_attributes_dt(atl08_h5)

outputLaz <- tempfile(fileext = ".laz")
ATL08_seg_attributes_dt_LAS(
  atl08_dt,
  outputLaz
)

close(atl08_h5)
```

ATL08_seg_attributes_dt_polyStat

Statistics of ATL08 Terrain and Canopy Attributes by Geometry

---

Description

Computes a series of statistics of ATL08 terrain and canopy attributes within area defined by a polygon

Usage

```
ATL08_seg_attributes_dt_polyStat(atl08_seg_att_dt, func, poly_id = NULL)
```

Arguments

atl08_seg_att_dt	An S4 object of class icesat2.atl08_dt containing ATL08 terrain and canopy attributes (output of ATL08_seg_attributes_dt() function).
func	The function to be applied for computing the defined statistics
poly_id	Polygon id. If defined, statistics will be computed for each polygon

Value

Returns an S4 object of class icesat2.atl08_dt Containing Statistics of ATL08 terrain and canopy attributes

Examples

```
# Specifying the path to ATL08 file
atl08_path <- system.file(
  "extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Extracting ATL08 terrain and canopy attributes
atl08_seg_att_dt <- ATL08_seg_attributes_dt(atl08_h5 = atl08_h5)

# Specifying the path to shapefile
polygon_filepath <- system.file(
  "extdata",
  "clip_geom.shp",
  package = "ICESat2VegR"
)

# Reading shapefile as sf object
polygon <- terra::vect(polygon_filepath)

# Clipping ATL08 terrain and canopy attributes by Geometry
atl08_seg_att_dt_clip <- ATL08_seg_attributes_dt_clipGeometry(
  atl08_seg_att_dt,
  polygon,
  split_by = "id"
)

# Computing the max h_canopy by polygon id
max_h_canopy <- ATL08_seg_attributes_dt_polyStat(
  atl08_seg_att_dt_clip,
  func = max(h_canopy),
  poly_id = "poly_id"
)
head(max_h_canopy)

# Define your own function
mySetOfMetrics <- function(x) {
  metrics <- list(
    min = min(x), # Min of x
    max = max(x), # Max of x
    mean = mean(x), # Mean of x
    sd = sd(x) # Sd of x
  )
  return(metrics)
}

# Computing a series of canopy statistics from customized function
h_canopy_metrics <- ATL08_seg_attributes_dt_polyStat(
  atl08_seg_att_dt_clip,
  func = mySetOfMetrics(h_canopy),
  poly_id = "poly_id"
)
```

```
head(h_canopy_metrics)

close(atl08_h5)
```

`ATL08_seg_attributes_h5_gridStat` *Rasterize ATL08 canopy attributes from h5 files at large scale*

Description

This function will read multiple ATL08 H5 files and create a stack of raster layers: count, and 1st, 2nd, 3rd and 4th moments (count, m1, m2, m3 and m4) for each metric selected, from which we can calculate statistics such as Mean, SD, Skewness and Kurtosis.

Usage

```
ATL08_seg_attributes_h5_gridStat(
  atl08_dir,
 metrics = c("h_canopy", "canopy_rh_conf", "h_median_canopy_abs", "h_min_canopy",
    "h_mean_canopy_abs", "h_median_canopy", "h_canopy_abs", "toc_roughness",
   "h_min_canopy_abs", "h_dif_canopy", "h_canopy_quad", "h_canopy_20m", "n_ca_photons",
   "photon_rate_can", "centroid_height", "canopy_h_metrics_abs", "h_mean_canopy",
    "subset_can_flag", "canopy_h_metrics", "n_toc_photons", "h_max_canopy_abs",
   "h_canopy_uncertainty", "canopy_openness", "h_max_canopy", "segment_cover"),
  beam = c("gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r"),
  out_root,
  ul_lat,
  ul_lon,
  lr_lat,
  lr_lon,
  res,
  creation_options = def_co,
  agg_function = default_agg_function,
  agg_join = default_agg_join,
  finalizer = default_finalizer
)
```

Arguments

`atl08_dir`	CharacterVector. The directory paths where the ATL08 H5 files are stored;
`metrics`	CharacterVector. A vector of canopy attributes available from ATL08 product (e.g. "h_canopy")
`beam`	Character vector indicating beams to process (e.g. "gt1l", "gt1r", "gt2l", "gt2r", "gt3l", "gt3r")
`out_root`	Character. The root name for the raster output files, the pattern is {out_root}*{metric}*{count/m1/m2}.t This should include the full path for the file.
`ul_lat`	Numeric. Upper left latitude for the bounding box
`ul_lon`	Numeric. Upper left longitude for the bounding box
`lr_lat`	Numeric. Lower right latitude for the bounding box

lr_lon	Numeric. Lower right longitude for the bounding box
res	NumericVector. Resolution lon lat for the output raster in coordinates decimal degrees
creation_options	CharacterVector. The GDAL creation options for the tif file. Default c("COMPRESS=PACKBITS", "BIGTIFF=IF_SAFER", "TILED=YES", "BLOCKXSIZE=512", "BLOCKYSIZE=512") will create BIGTIFF if needed, with DEFLATE compression and tiled by 512x512 pixels.
agg_function	Formula function-like. An aggregate function which should return a data.table with the aggregate statistics
agg_join	Function. A function to merge two different agg objects.
finalizer	List<name, formula>. A list with the final raster names and the formula which uses the base statistics.

Details

This function will create five different aggregate statistics (n, mean, variance, min, max). One can calculate mean and standard deviation with the following formulas according to Terriberry (2007) and Joanes and Gill (1998):

The `agg_function` is a formula which return a data.table with the aggregate function to perform over the data. The default is:

```
~data.table(
    n = length(x),
    mean = mean(x,na.rm = TRUE),
    var = var(x) * (length(x) - 1),
    min = min(x, na.rm=T),
    max = max(x, na.rm=T)
  )
```

The `agg_join` is a function to merge two data.table aggregates from the `agg_function`. Since the h5 files will be aggregated one by one, the statistics from the different h5 files should have a function to merge them. The default function is:

```
function(x1, x2) {
    combined = data.table()
    x1$n[is.na(x1$n)] = 0
    x1$mean[is.na(x1$mean)] = 0
    x1$variance[is.na(x1$variance)] = 0
    x1$max[is.na(x1$max)] = -Inf
    x1$min[is.na(x1$min)] = Inf

    combined$n = x1$n + x2$n

    delta = x2$mean - x1$mean
    delta2 = delta * delta

    combined$mean = (x1$n * x1$mean + x2$n * x2$mean) / combined$n
    combined$variance = x1$variance + x2$variance +
      delta2 * x1$n * x2$n / combined$n
```

```
    combined$min = pmin(x1$min, x2$min, na.rm=F)
    combined$max = pmax(x1$max, x2$max, na.rm=F)
    return(combined)
}
```

The `finalizer` is a list of formulas to generate the final rasters based on the intermediate statistics from the previous functions. The default `finalizer` will calculate the `sd`, `skewness` and `kurtosis` based on the `variance`, `M3`, `M4` and `n` values. It is defined as:

```
list(
  sd = ~sqrt(variance/(n - 1)),
)
```

Value

Nothing. It outputs multiple raster tif files to the out_root specified path.

References

Joanes DN, Gill CA (1998). "Comparing measures of sample skewness and kurtosis." *Journal of the Royal Statistical Society: Series D (The Statistician)*, **47**(1), 183–189. doi:10.1111/1467-9884.00122.

Terriberry, Timothy B. (2007), Computing Higher-Order Moments Online, archived from the original on 23 April 2014, retrieved 5 May 2008

Examples

```
# Specifying the path to GEDI leveatl08_canopy_dt data (zip file)
library(ICESat2VegR)
library(data.table)

# Specifying the path to ATL08 file
atl08_path <- system.file("extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL08 data (h5 file)
atl08_h5 <- ATL08_read(atl08_path = atl08_path)

# Bounding rectangle coordinates
ul_lat <- 41.5386848449707031
ul_lon <- -106.5708541870117188
lr_lat <- 41.5314979553222656
lr_lon <- -106.5699081420898438

res <- 100 # meters
lat_to_met_factor <- 1 / 110540
lon_to_met_factor <- 1 / 111320
xres <- lon_to_met_factor * res
yres <- lat_to_met_factor * res

agg_function <- ~ data.table(
  min = min(x),
  max = max(x),
```

```
  sum = sum(x),
  n = length(x)
)

agg_join <- function(agg1, agg2) {
  agg1[is.na(agg1)] <- 0
  data.table(
    min = pmin(agg1$min, agg2$min),
    max = pmax(agg1$max, agg2$max),
    sum = agg1$sum + agg2$sum,
    n = agg1$n + agg2$n
  )
}

finalizer <- list(
  mean = "sum/n",
  range = "max-min"
)

outdir <- tempdir()


gc()
file.remove(list.files(outdir, "*.tif"))
close(atl08_h5)
```

ATLAS_dataDownload *Download ICESat-2 ATL03/ATL08 data*

Description

Download ICESat-2 ATL03 and ATL08 data from the LP DAAC / NSIDC endpoints. Handles connection drops by resuming partial files and retrying with exponential backoff. Authentication is handled via a `.netrc` file: if it does not exist, the user is prompted for Earthdata Login credentials. If authentication fails (e.g., wrong username/password), the user is informed and can choose to re-enter credentials or cancel.

Usage

```
ATLAS_dataDownload(
  url,
  outdir = NULL,
  overwrite = FALSE,
  buffer_size = 512,
  timeout = 10,
  retries = 3,
  backoff = 2,
  quiet = FALSE,
  use_home_netrc = TRUE
)
```

Arguments

url	Character vector; URLs to ATL03/ATL08 files (e.g., returned by ATLAS_dataFinder()).
outdir	Character; output directory (default: tempdir()).
overwrite	Logical; overwrite existing files? Default FALSE.
buffer_size	Integer; chunk size in KB per readBin call (default 512).
timeout	Numeric; connection timeout (seconds) for establishing the connection (default 10).
retries	Integer; maximum attempts per file (default 3).
backoff	Numeric; exponential backoff base used as sleep = backoff^(attempt - 1) (default 2).
quiet	Logical; suppress per-file messages (default FALSE).
use_home_netrc	Logical; if TRUE (default), use a single '~/.netrc' file in the user's home directory for Earthdata credentials so they can be reused across projects. If FALSE, a '.netrc' file is created in outdir.

Value

(invisibly) a data.frame with columns:

- file: file name
- dest: full destination path
- status: one of "ok", "failed", "skipped"
- attempts: number of attempts used
- error: error message (if any)

References

Credits to Cole Krehbiel. Code adapted from: https://git.earthdata.nasa.gov/projects/LPDUR/repos/daac_data_download_r/browse/DAACDataDownload.R

Examples

```
## Not run:
urls <- c(
  "https://example.com/path/to/ATL03_001.h5",
  "https://example.com/path/to/ATL08_001.h5"
)
status <- ATLAS_dataDownload(urls,
                             outdir        = "data",
                             retries       = 5,
                             backoff       = 2,
                             use_home_netrc = TRUE)
subset(status, status == "failed")

## End(Not run)
```

ATLAS_dataFinder *ICESat-2 ATL03 and ATL08 data finder for either direct download or cloud computing*

Description

This function finds the exact granule(s) that contain ICESat-2 ATLAS data for a given region of interest and date range

Usage

```
ATLAS_dataFinder(
  short_name,
  lower_left_lon,
  lower_left_lat,
  upper_right_lon,
  upper_right_lat,
  version = "006",
  daterange = NULL,
  persist = TRUE,
  cloud_hosted = TRUE,
  cloud_computing = FALSE
)
```

Arguments

short_name	ICESat-2 ATLAS data level short_name; Options: "ATL03", "ATL08",
lower_left_lon	Numeric. Minimum longitude in (decimal degrees) for the bounding box of the area of interest.
lower_left_lat	Numeric. Minimum latitude in (decimal degrees) for the bounding box of the area of interest.
upper_right_lon	Numeric. Maximum longitude in lon (decimal degrees) for the bounding box of the area of interest.
upper_right_lat	Numeric. Maximum latitude in (decimal degrees) for the bounding box of the area of interest.
version	Character. The version of the ICESat-2 ATLAS product files to be returned (only V005 or V006). Default "006".
daterange	Vector. Date range. Specify your start and end dates using ISO 8601 [YYYY]-[MM]-[DD]T[hh]:[mm]:[ss]Z. Ex.: c("2019-07-01T00:00:00Z","2020-05-22T23:59:59Z"). If NULL (default), the date range filter will be not applied.
persist	Logical. If TRUE, it will create the .netrc
cloud_hosted	Logical. Flag to indicate use of cloud hosted collections.To be used when the cloud computing parameter is FALSE.
cloud_computing	Logical. If TRUE, it will return granules for cloud computing directly, otherwise it will return links to be passed in the function ATLAS_dataDownload for data download.

Value

Return either a vector object pointing out the path to ICESat-2 ATLAS data found within the boundary box coordinates provided for data download or a vector object containing the granules hosted on the cloud for cloud computing directly.

See Also

bbox: Defined by the upper left and lower right corner coordinates, in lat,lon ordering, for the bounding box of the area of interest (e.g. lower_left_lon,lower_left_lat,upper_right_lon,upper_right_lat)

This function relies on the existing CMR tool: https://cmr.earthdata.nasa.gov/search/site/docs/search/api.html

Examples

```
# ICESat-2 data finder is a web service provided by NASA
# usually the request takes more than 5 seconds

# Specifying bounding box coordinates
lower_left_lon <- -96.0
lower_left_lat <- 40.0
upper_right_lon <- -96.5
upper_right_lat <- 40.5

# Specifying the date range
daterange <- c("2022-05-01", "2022-05-02")

# Extracting the path to ICESat-2 ATLAS data for the specified boundary box coordinates
# for data download

# Shouldn't be tested because it relies on a web service which might be down

ATLAS02b_list <- ATLAS_dataFinder(
  short_name = "ATL08",
  lower_left_lon,
  lower_left_lat,
  upper_right_lon,
  upper_right_lat,
  version = "006",
  daterange = daterange
)
```

---

clip *Clip ICESat-2 data (h5, attributes, or ATL03–ATL08 join) by box or geometry*

Description

Unified clipping interface for ICESat-2 ATL03/ATL08 data. The generic `clip()` dispatches on the class of `x` and internally calls the appropriate helper:

- `ATL03_h5_clipBox`, `ATL03_h5_clipGeometry`
- `ATL03_photon_attributes_dt_clipBox`, `ATL03_photon_attributes_dt_clipGeometry`

- ATL08_h5_clipBox, ATL08_h5_clipGeometry
- ATL08_seg_attributes_dt_clipBox, ATL08_seg_attributes_dt_clipGeometry
- ATL03_ATL08_photons_attributes_dt_join_clipBox, ATL03_ATL08_photons_attributes_dt_join_clipGe

Usage

```
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl03_h5,numeric'
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl03_h5,ANY'
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl08_h5,numeric'
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl08_h5,ANY'
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl03_dt,numeric'
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl03_dt,ANY'
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl08_dt,numeric'
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl08_dt,ANY'
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl03atl08_dt,numeric'
clip(x, clip_obj, ...)

## S4 method for signature 'icesat2.atl03atl08_dt,ANY'
clip(x, clip_obj, ...)
```

Arguments

x	An ICESat-2 object to be clipped. Supported classes: • "icesat2.atl03_h5" — ATL03 HDF5 handle. • "icesat2.atl08_h5" — ATL08 HDF5 handle. • "icesat2.atl03_seg_dt" — ATL03 photon/segment attributes. • "icesat2.atl08_dt" — ATL08 segment attributes. • "icesat2.atl03_atl08_join_spec" — specification bundling ATL03 + ATL08 attributes for join+clip operations.
clip_obj	A clipping object: • Numeric bounding box (e.g., c(xmin, xmax, ymin, ymax)). • terra::SpatExtent, terra::SpatVector, **sf** geometry, or similar.

Bounding boxes are usually routed to `*_clipBox()` helpers, and geometries to `*_clipGeometry()` helpers.

... Additional arguments passed down to the underlying helper functions.

Methods (by class)

- `clip(x = icesat2.atl03_h5, clip_obj = numeric)`: Clip ATL03 HDF5 by bounding box (delegates to ATL03_h5_clipBox()).
- `clip(x = icesat2.atl03_h5, clip_obj = ANY)`: Clip ATL03 HDF5 by geometry (delegates to ATL03_h5_clipGeometry()).
- `clip(x = icesat2.atl08_h5, clip_obj = numeric)`: Clip ATL08 HDF5 by bounding box (delegates to ATL08_h5_clipBox()).
- `clip(x = icesat2.atl08_h5, clip_obj = ANY)`: Clip ATL08 HDF5 by geometry (delegates to ATL08_h5_clipGeometry()).
- `clip(x = icesat2.atl03_dt, clip_obj = numeric)`: Clip ATL03 photon attributes by bounding box (delegates to ATL03_photon_attributes_dt_clipBox()).
- `clip(x = icesat2.atl03_dt, clip_obj = ANY)`: Clip ATL03 photon attributes by geometry (delegates to ATL03_photon_attributes_dt_clipGeometry()).
- `clip(x = icesat2.atl08_dt, clip_obj = numeric)`: Clip ATL08 segment attributes by bounding box (delegates to ATL08_seg_attributes_dt_clipBox()).
- `clip(x = icesat2.atl08_dt, clip_obj = ANY)`: Clip ATL08 segment attributes by geometry (delegates to ATL08_seg_attributes_dt_clipGeometry()).
- `clip(x = icesat2.atl03atl08_dt, clip_obj = numeric)`: Join ATL03 photons and ATL08 segments, then clip by bounding box.
  This method delegates to `ATL03_ATL08_photons_attributes_dt_join_clipBox()`.
- `clip(x = icesat2.atl03atl08_dt, clip_obj = ANY)`: Join ATL03 photons and ATL08 segments, then clip by geometry.
  This method delegates to `ATL03_ATL08_photons_attributes_dt_join_clipGeometry()`.

Related clipping helpers

1. `ATL03_h5_clipBox`
2. `ATL03_h5_clipGeometry`
3. `ATL03_photon_attributes_dt_clipBox`
4. `ATL03_photon_attributes_dt_clipGeometry`
5. `ATL08_h5_clipBox`
6. `ATL08_h5_clipGeometry`
7. `ATL08_seg_attributes_dt_clipBox`
8. `ATL08_seg_attributes_dt_clipGeometry`
9. `ATL03_ATL08_photons_attributes_dt_join_clipBox`
10. `ATL03_ATL08_photons_attributes_dt_join_clipGeometry`

See Also

`ATL03_h5_clipBox`, `ATL03_h5_clipGeometry`, `ATL03_photon_attributes_dt_clipBox`, `ATL03_photon_attribute`
`ATL08_h5_clipBox`, `ATL08_h5_clipGeometry`, `ATL08_seg_attributes_dt_clipBox`, `ATL08_seg_attributes_dt_c`
`ATL03_ATL08_photons_attributes_dt_join_clipBox`, `ATL03_ATL08_photons_attributes_dt_join_clipGeomet`

`ee_build_AlphaEarth_embedding_terrain_stack` *Stack Alpha Earth embedding and terrain ancillary layers*

Description

Creates a Google Earth Engine image stack that combines:

- Annual satellite embeddings from `"GOOGLE/SATELLITE_EMBEDDING/V1/ANNUAL"` (AlphaEarth).
- Terrain derivatives (elevation, slope, aspect) from USGS 3DEP or NASADEM as fallbacks.
- Optional longitude/latitude bands from `ee$Image$pixelLonLat()`.

The embedding collection is filtered to the specified year range and AOI, reduced using a pixel-wise median, and clipped to the AOI. Terrain data are reprojected to a target scale (default `30` m).

Usage

```
ee_build_AlphaEarth_embedding_terrain_stack(
  geom,
  start_year,
  end_year,
  mask_outside = TRUE,
  terrain_scale = 30,
  multiply_slope_aspect_by10 = TRUE,
  add_lonlat = TRUE
)
```

Arguments

`geom` AOI geometry. Can be:

- an **sf** object (`sf` or `sfc`),
- a `terra::SpatVector`,
- or an Earth Engine geometry (Python object) already in geographic coordinates.

The geometry is internally converted to an Earth Engine geometry via an internal helper.

`start_year, end_year` Integer (or coercible to integer). Inclusive year range used to filter the AlphaEarth annual embedding collection.

`mask_outside` Logical. If `TRUE` (default), pixels outside the AOI are masked out using a binary mask image clipped to `geom`.

`terrain_scale` Numeric. Target scale (in meters) used to reproject the terrain data (default `30`).

`multiply_slope_aspect_by10` Logical. If `TRUE` (default), slope and aspect are multiplied by 10 to preserve conventions used elsewhere in the package and avoid small floating-point values.

`add_lonlat` Logical. If `TRUE` (default), add longitude and latitude bands named `"lon"` and `"lat"` derived from `ee$Image$pixelLonLat()`.

Details

The terrain source is chosen in the following order:

1. USGS 3DEP 1 m (`"USGS/3DEP/1m"`), if available for the AOI.
2. USGS 3DEP 10 m (`"USGS/3DEP/10m"`).
3. NASADEM (`"NASA/NASADEM_HGT/001"`) as a global fallback.

All terrain layers are clipped to the AOI and reprojected to `"EPSG:4326"` with the requested `terrain_scale`.

Value

A Python `ee$Image` object that contains:

- The median annual embedding bands.
- `"elevation"`, `"slope"`, and `"aspect"` bands.
- Optional `"lon"` and `"lat"` bands.

Examples

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

  # ------------------------------------------------------------
  # Example 1 - Using the function
  # ------------------------------------------------------------

  # Simple AOI polygon (WGS84)
  aoi <- st_as_sfc(st_bbox(c(
    xmin = -82.4, xmax = -82.2,
    ymin =  29.6, ymax =  29.8
  ), crs = 4326))

  img <- ee_build_AlphaEarth_embedding_terrain_stack(
    geom       = aoi,
    start_year = 2018,
    end_year   = 2020
  )


  # ------------------------------------------------------------
  # Example 2 - Full standalone script (no function)
  # ------------------------------------------------------------

  library(reticulate)
  ee <- import("ee")
  ICESat2VegR:::.ee_ping(ee)

  # AOI geometry (sf → EE geometry)
  library(sf)
  aoi <- st_as_sfc(
    st_bbox(c(
      xmin = -82.4,
      xmax = -82.2,
      ymin = 29.6,
      ymax = 29.8
```

```
  ), crs = 4326)
)

ee_geom <- ICESat2VegR:::.as_ee_geom(aoi)

# ------------------------------------------------------------
# AlphaEarth annual embedding
# ------------------------------------------------------------
start_year <- 2018
end_year   <- 2020

emb_ic <- ee$ImageCollection("GOOGLE/SATELLITE_EMBEDDING/V1/ANNUAL")$
  filterDate(
    sprintf("%04d-01-01", start_year),
    sprintf("%04d-12-31", end_year)
  )$
  filterBounds(ee_geom)

embedding <- emb_ic$median()$clip(ee_geom)


# ------------------------------------------------------------
# Terrain DEM via ICESat2VegR dataset search (NASA DEM)
# ------------------------------------------------------------
dem_search   <- search_datasets("nasa", "dem")
dem_id       <- get_catalog_id(dem_search)
elevation    <- ee$Image(dem_id)$clip(ee_geom)

# Compute slope and aspect using package helpers
terrain_scale <- 30

# Reproject elevation to target scale (if desired)
elev_proj <- elevation$reproject("EPSG:4326", scale = terrain_scale)

slp <- slope(elev_proj)    # returns ee.Image with band "slope"
asp <- aspect(elev_proj)   # returns ee.Image with band "aspect"

# Optional scaling by 10
slp <- slp$multiply(10)
asp <- asp$multiply(10)

slp <- slp$clip(ee_geom)
asp <- asp$clip(ee_geom)


# ------------------------------------------------------------
# Lon/lat bands
# ------------------------------------------------------------
lonlat <- ee$Image$pixelLonLat()$clip(ee_geom)
lon    <- lonlat$select("longitude")$rename("lon")
lat    <- lonlat$select("latitude")$rename("lat")


# ------------------------------------------------------------
# Final stack: embeddings + terrain + lon/lat
# ------------------------------------------------------------
stack <- embedding$
```

```
    addBands(elevation$rename("elevation"))$
    addBands(slp)$
    addBands(asp)$
    addBands(lon)$
    addBands(lat)

  # Optional mask outside AOI
  mask  <- ee$Image$constant(1)$clip(ee_geom)
  stack <- stack$updateMask(mask)

  print(stack)

## End(Not run)
```

`ee_build_hls_s1c_terrain_stack` *Build HLS, Sentinel-1C and terrain ancillary stack in Earth Engine*

Description

Constructs a multi-source ancillary stack in Earth Engine composed of: (1) optical features from the Harmonized Landsat and Sentinel-2 (HLS) product, including several vegetation indices and spectral unmixing fractions; (2) C-band SAR backscatter and radar-based indices from Sentinel-1C GRD; and (3) terrain variables (elevation, slope, aspect) from a DEM.

The user must provide a temporal window via `start_date` and `end_date`. The HLS image collection is filtered to this period and mosaicked using `median()`. Sentinel-1C is filtered to the same period and reduced using `ee$Reducer$firstNonNull()`, following the original workflow. The final stack includes neighborhood statistics and GLCM texture features.

Usage

```
ee_build_hls_s1c_terrain_stack(
  x,
  start_date,
  end_date,
  cloud_max = 10,
  buffer_m = 30
)
```

Arguments

`x`	Spatial input defining the area of interest. Can be: • a character path to a vector file readable by `terra::vect()` (e.g., SHP, GPKG), • a terra::SpatVector, • a terra::SpatRaster (its extent will be used), • an sf::sf or sf::sfc object, • a terra::SpatExtent object (e.g., from `terra::ext()`), • a numeric vector of length 4 giving an extent as `c(xmin, xmax, ymin, ymax)`.

`start_date`	Character. Start date of the temporal filter (`"YYYY-MM-DD"`). **Required.**
`end_date`	Character. End date of the temporal filter (`"YYYY-MM-DD"`). **Required.**
`cloud_max`	Numeric. Maximum allowed cloud coverage percentage for HLS scenes (used in the `"CLOUD_COVERAGE < cloud_max"` filter). Default is 10.
`buffer_m`	Numeric. Buffer distance in meters applied to the AOI bounding box before filtering and clipping. Default is 30.

Details

This function assumes that:

- Earth Engine access is provided via the **ICESat2VegR** internal `ee` handle.
- Dataset search and catalog ID retrieval are performed by `search_datasets()` and `get_catalog_id()` from **ICESat2VegR**.

The HLS component follows the original workflow: the collection is filtered by AOI and date, filtered by cloud coverage, cloud and water masks are applied, and a `median()` mosaic is computed. Reflectance bands are renamed (blue, green, red, nir, swir1, swir2) and a set of vegetation indices and linear spectral unmixing fractions are added.

Sentinel-1C (COPERNICUS/S1_GRD) is filtered by AOI, date, polarization (VV/VH), and instrument mode ("IW"), sorted by time, and reduced using `ee$Reducer$firstNonNull()`. VV and VH are scaled, and RVI and co-polarization indices are derived.

A DEM is retrieved from the NASA catalog via `search_datasets()` and `get_catalog_id()`, and slope and aspect are derived following the original code pattern. Neighborhood statistics (mean, min, max, stdDev) are computed for the HLS and DEM components using a fixed 3×3 kernel, and GLCM texture metrics are computed from scaled HLS reflectance bands.

#' @examples Not run: # ========== # Example 1 — Using the function # ==========
res <- ee_build_hls_s1c_terrain_stack(x = system.file("extdata", "all_boundary.shp", package = "ICESat2VegR"), start_date = "2019-04-01", end_date = "2019-05-31")

full_stack <- res$stack

========== # Example 2 — FULL SCRIPT (no function) # Users may copy/paste and modify as needed # ==========

library(ICESat2VegR) library(terra)

AOI geom <- terra::vect(system.file("extdata", "all_boundary.shp", package="ICESat2VegR")) bbox <- terra::ext(geom)

aoi <- ee$Geometry$BBox(west = bbox$xmin, south = bbox$ymin, east = bbox$xmax, north = bbox$ymax)$buffer(30)

———— # HLS # ———— search <- search_datasets("hls") id <- get_catalog_id(search)

cloudMask <- 2^1 + 2^2 + 2^3

hlsMask <- function(image) image$updateMask(!(image[["Fmask"]] & cloudMask)) waterMask <- function(image) image$updateMask(image[["B5"]] >= 0.2)

hls <- ee$ImageCollection(id)$ filterBounds(aoi)$ filterDate("2019-04-01", "2019-05-31")$ filter("CLOUD_COVERAG < 10")$ map(hlsMask)$ map(waterMask)$ median()$ clip(aoi)

hls <- hls[["B2","B3","B4","B5","B6","B7"]] names(hls) <- c("blue","green","red","nir","swir1","swir2")

Vegetation indices nir <- hls[["nir"]] red <- hls[["red"]] blue <- hls[["blue"]] green <- hls[["green"]] swir1 <- hls[["swir1"]] swir2 <- hls[["swir2"]] nir_red <- nir - red

```
hls[["ndvi"]] <- (nir - red) / (nir + red)

sigma <- 1 knr <- exp((nir_red)^2 / (2*sigma^2)) hls[["kndvi"]] <- (1 - knr) / (1 + knr)

hls[["evi"]] <- 2.5 * nir_red / (nir + 6*red - 7.5*blue + 1) hls[["savi"]] <- 1.5 * nir_red / (nir + red
+ 0.5)

p1 <- 2*nir + 1 hls[["msavi"]] <- p1 - sqrt(p1^2 - 8*nir_red)/2

# Spectral unmixing soil <- c(0.14, 0.16, 0.22, 0.39, 0.45, 0.27) veg <- c(0.086,0.062,0.043,0.247,0.109,0.039)
water <- c(0.07,0.039,0.023,0.031,0.011,0.007)

img <- hls[[c("blue","green","red","nir","swir1","swir2")]] unmixing <- img$unmix(list(soil,veg,water))
names(unmixing) <- c("f_soil","f_veg","f_water")

hls <- c(hls, unmixing)

# More indices hls[["sri"]] <- nir / red hls[["ndwi"]] <- (green - nir)/(green + nir) hls[["gci"]]
<- nir/green - 1 hls[["wdrvi"]] <- ((0.1*nir) - red)/((0.1*nir) + red) hls[["gvmi"]] <- ((nir+0.1)-
(swir1+0.02))/((nir+0.1)+(swir1+0.02)) hls[["cvi"]] <- nir * (red/(green^2)) hls[["cmr"]] <- swir1/swir2

# ———————————————————— # DEM (NASA) # ————————————————————— dem_search <- search_datasets("nasa","dem") dem_id <- get_catalog_id(dem_search)

elevation <- ee$Image(dem_id) the_slope <- as.integer(slope(as.integer(elevation)) * 1000) the_aspect
<- aspect(elevation)

stackDem <- c(elevation, the_slope, the_aspect)$clip(aoi)

# ———————————————————— # Sentinel-1C # ————————————————————— s1c <- ee$ImageCollection("COPERNICUS/S1_GRD")$ filterBounds(aoi)$
filterDate("2019-04-01","2019-05-31")$ filter(ee$Filter$listContains("transmitterReceiverPolarisation","VV"))$
filter(ee$Filter$listContains("transmitterReceiverPolarisation","VH"))$ filter(ee$Filter$eq("instrumentMode","IW"))

s1c <- s1c$sort("system:time_start", FALSE) s1c <- s1c$reduce(ee$Reducer$firstNonNull()) s1c
<- s1c[["VV_first","VH_first"]] names(s1c) <- c("vv","vh")

s1c <- as.integer(s1c * 100) vv <- s1c[["vv"]] vh <- s1c[["vh"]]

s1c[["rvi"]] <- as.integer(sqrt(vv/(vv+vh)) * (vv/vh) * 10000) s1c[["copol"]] <- as.integer((vv/vh) *
10000) s1c[["copol2"]] <- as.integer(((vv - vh)/(vv + vh)) * 10000) s1c[["copol3"]] <- as.integer((vh/vv)
* 10000)

# ———————————————————— # Final stack + neighborhood + texture # ———————————————————— fullStack <- c(hls, s1c, stackDem)

kernel <- ee$Kernel$fixed( 3, 3, list(c(1,1,1), c(1,1,1), c(1,1,1)), 3, 3, FALSE )

for (nm in c("mean","min","max","stdDev")) { reducer <- ee$Reducer[[nm]]()

fullStack <- c( fullStack, hls$reduceNeighborhood(reducer, kernel), stackDem$reduceNeighborhood(reducer,
kernel) ) }

img_tex <- as.integer( (hls[[c("blue","green","red","nir","swir1","swir2")]]) * 1e4 )

tex <- img_tex$glcmTexture(size=3)

fullStack <- c(fullStack, tex)

# fullStack is the ancillary layers image fullStack

End(Not run)
```

Value

A named list with the following Earth Engine objects:

aoi EE geometry representing the buffered AOI bounding box.

hls HLS image with reflectance bands, vegetation indices, and spectral unmixing fractions.

s1c Sentinel-1C image with VV/VH backscatter and radar indices.

dem Terrain stack including elevation, slope, and aspect.

stack Full ancillary stack combining HLS, Sentinel-1C, and terrain, including neighborhood statistics and texture metrics.

`ee_initialize` *Initializes the Google Earth Engine API Initialize Earth Engine for this R session*

Description

Initializes the Google Earth Engine API Initialize Earth Engine for this R session

Usage

```
ee_initialize(
  project = Sys.getenv("EE_PROJECT", unset = NA),
  service_account = NULL,
  keyfile = NULL,
  quiet = FALSE,
  force_auth = FALSE
)
```

Arguments

`project`	Character. **GCP Project ID** (e.g., "ice-map-2025") or a numeric project **number**. If NULL/NA, falls back to Sys.getenv("EE_PROJECT").
`service_account`	Optional service account email (use with `keyfile`).
`keyfile`	Path to service-account JSON key (required if `service_account` is set).
`quiet`	Logical. Suppress messages.
`force_auth`	Logical. If TRUE, perform OAuth before Initialize().

Value

TRUE on success; FALSE otherwise (invisibly).

ee_rect_to_sf *Convert an EE Rectangle to an sf polygon (EPSG:4326)*

Description

Convert an EE Rectangle to an sf polygon (EPSG:4326)

Usage

```
ee_rect_to_sf(aoi)
```

Arguments

aoi	An ee$Geometry$Rectangle.

Value

An sf polygon with CRS EPSG:4326.

ext_to_ee *Convert a terra extent or spatial object to an Earth Engine Rectangle*

Description

Converts a terra::ext, terra::SpatVector, or terra::SpatRaster into a Google Earth Engine geometry of type ee$Geometry$Rectangle. The resulting rectangle is always expressed in geographic coordinates (EPSG:4326), regardless of the input object's projection.

This function is mainly used internally to standardize spatial inputs before Earth Engine requests (e.g., filtering, clipping, or exporting data).

Usage

```
ext_to_ee(x)
```

Arguments

x	A spatial object of class terra::ext, terra::SpatVector, or terra::SpatRaster. Its extent is extracted and converted into an Earth Engine bounding box.

Value

A Python object representing ee$Geometry$Rectangle, suitable for use with Earth Engine Python API functions accessed via reticulate.

Examples

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

# Create an extent over Gainesville, FL
bb <- ext(c(-82.4, -82.2, 29.6, 29.8))

# Convert to EE geometry
aoi <- ext_to_ee(bb)

## End(Not run)
```

`fit_metrics` *Model fit metrics*

Description

Computes RMSE (absolute and relative), MAE (absolute and relative), bias (absolute and relative), Pearson correlation (r), and adjusted R^2 from a linear fit between predicted and observed values. Optionally draws a 1:1 plot with the regression line.

Usage

```
fit_metrics(
  observed,
  predicted,
  plotstat = FALSE,
  legend = "topleft",
  unit = "",
  ...
)
```

Arguments

`observed`	Numeric vector of observed values.
`predicted`	Numeric vector of predicted values (same length/order as `observed`).
`plotstat`	Logical; if TRUE, draws a 1:1 plot with the regression line. Default: FALSE.
`legend`	Character position for the plot legend (e.g., "topleft"). Default: "topleft".
`unit`	Character indicating the unit for RMSE/MAE/Bias (e.g., "Mg/ha"). Default: "".
`...`	Additional arguments passed to `graphics::plot()`.

Value

A data.frame with rows for `rmse`, `rmseR` (%), `mae`, `maeR` (%), `bias`, `biasR` (%), `r`, and `adj_r2`.

Examples

```
observed  <- c(178, 33, 60, 80, 104, 204, 146)
predicted <- c(184, 28.5, 55, 85, 105, 210, 155)
fit_metrics(observed, predicted,
            plotstat = TRUE, legend = "topleft", unit = "Mg/ha",
            xlab = "Observed AGBD (Mg/ha)", ylab = "Predicted AGBD (Mg/ha)", pch = 16)
```

`fit_model` *Fit a Random Forest with optional resampling, tuning, and progress bars*

Description

`fit_model()` trains a Random Forest regression model and optionally evaluates it via LOOCV, K-fold CV, a train/test split, or bootstrap OOB estimation. It can also tune core RF hyperparameters and shows progress bars for long-running loops.

Usage

```
fit_model(
  x,
  y,
  rf_args = list(ntree = 500, mtry = NULL, nodesize = 5, sampsize = NULL),
  test = list(method = "none", k = 5, test_size = 0.3, seed = NULL, folds = NULL,
    iterations = 200, correction = FALSE),
 tune = list(enable = FALSE, search = "grid", grid = NULL, n_random = 20, seed = NULL),
  verbose = TRUE,
  list_test_models = TRUE
)
```

Arguments

`x`	A data.frame of predictors (rows = samples, cols = features).
`y`	A numeric vector of responses, `length(y) == nrow(x)`.
`rf_args`	A named list of base Random Forest arguments used for all fits (full-data fit and each resample). Elements: • `ntree` (integer, default `500`): number of trees. • `mtry` (integer or `NULL`): number of variables sampled at each split. If `NULL`, a safe default `floor(sqrt(p))` is used. • `nodesize` (integer, default `5`): minimum terminal node size. • `sampsize` (integer, fraction in `(0,1]`, or `NULL`): sample size per tree. If a single fraction, it is multiplied by the current training size and rounded. If `NULL`, the package default is used.
`test`	A named list configuring evaluation: • `method` (character): one of `"none"`, `"loocv"`, `"k-fold"` (also accepts `"kfold"`/`"k fold"`), `"split"`, or `"bootstrap"`. • `k` (integer, default `5`): number of folds for K-fold CV. • `folds` (list or `NULL`): custom index list of test-fold indices. If supplied, overrides `k`.

- `test_size` (numeric in `(0,1)`, default `0.3`): test fraction for train/test split.
- `iterations` (integer, default `200`): number of bootstrap resamples.
- `correction` (logical, default `FALSE`): if `TRUE`, apply the .632 correction to the RMSE for the bootstrap estimate.
- `seed` (integer or `NULL`): RNG seed for reproducibility (folds, split, random tuning).

`tune` A named list configuring hyperparameter search on the *training* subset for each fit:

- `enable` (logical, default `FALSE`): turn tuning on/off.
- `search` (character, default `"grid"`): `"grid"` or `"random"`.
- `grid` (data.frame or `NULL`): explicit grid with columns `mtry`, `ntree`, `nodesize`, `sampsize`. If `NULL`, a sensible default grid is generated from `p = ncol(x)`.
- `n_random` (integer, default `20`): number of random draws from the grid when `search = "random"`.
- `seed` (integer or `NULL`): RNG seed for the tuning subset draw order.

`verbose` Logical (default `TRUE`): show progress bars/messages for tuning, LOOCV, K-fold, and bootstrap loops. Set `FALSE` to silence.

`list_test_models` Logical (default `TRUE`): if `TRUE`, save the *per-resample* fitted model objects as a list in `models_test`. This can be large, especially for `bootstrap` with many iterations.

Details

Tuning minimizes OOB MSE for each candidate configuration on the current training subset and then refits the model using the best settings. When `test$method = "bootstrap"` and `correction = TRUE`, the .632 corrected RMSE is reported while keeping other OOB statistics unchanged.

Value

A list with:

method `"rf"`

rf_args Final RF arguments used for the full-data fit (after tuning).

tune_table (data.frame or `NULL`) tuning results sorted by OOB MSE.

test Echo of `test` configuration (with resolved values).

model `randomForest` object fit on the full dataset (or train subset for `split`).

fitted_full Numeric vector of in-sample predictions from the full-data fit.

stats_train data.frame of training statistics (RMSE, Bias, %RMSE, %Bias, r, r2).

stats_test (data.frame or `NULL`) test-set or OOF statistics depending on method.

loocv_pred (numeric) LOOCV predictions (for `method = "loocv"`).

cv_pred (numeric) K-fold OOF predictions (for `method = "k-fold"`).

train_index,test_index (integer) indices for `method = "split"`.

pred_train,pred_test (numeric) predictions for train/test in `split`.

oob_pred (numeric) OOB mean prediction per observation in `bootstrap`.

models_test (list or `NULL`) models fitted per resample/fold/iteration when `list_test_models=TRUE`.

Workflow

1. Optionally tune RF hyperparameters on the *current training subset* (or on full data when `test$method = "none"`).
2. Always fit a model on the full dataset (`model` and `fitted_full`).
3. If a testing method is requested, refit on resampled training folds and compute out-of-fold predictions to summarize generalization performance.

Progress Bars

Uses `utils::txtProgressBar()`; disable with `verbose = FALSE`. The internal helper is lightweight and has no external dependencies.

See Also

`randomForest::randomForest()`, `stats::lm()`, `stats::cor()`

Examples

```
set.seed(1)
n <- 200
x <- data.frame(NDVI = runif(n, 0.2, 0.9),
                EVI  = runif(n, 0.1, 0.8),
                NBR  = runif(n, -0.5, 0.9),
                SLP  = runif(n, 0, 30))
y <- with(x, 5 + 20*NDVI + 10*EVI^1.5 - 0.05*SLP + rnorm(n, 0, 2))

# Full-data fit (no resampling)
fit_none <- fit_model(x, y, rf_args = list(ntree = 400, mtry = 2))
fit_none$stats_train

# LOOCV
fit_loocv <- fit_model(x, y, test = list(method = "loocv"))
fit_loocv$stats_test

# 5-fold CV
fit_k-fold <- fit_model(x, y, test = list(method = "k-fold", k = 5, seed = 42))
fit_k-fold$stats_test

# Train/Test split
fit_split <- fit_model(x, y, test = list(method = "split", test_size = 0.25, seed = 42))
fit_split$stats_test

# Bootstrap (with tuning and .632 correction)
fit_boot <- fit_model(
  x, y,
  rf_args = list(ntree = 400),
  test    = list(method = "bootstrap", iterations = 300, correction = TRUE),
  tune    = list(enable = TRUE, search = "random", n_random = 12)
)

# K-fold CV with saved models
fit_k <- fit_model(x, y, test = list(method = "k-fold", k = 5, seed = 42), list_test_models = TRUE)
length(fit_k$models_test)  # one model per fold
```

geomSampling *Get observations given a minimum radius distance between samples*

Description

Get observations given a minimum radius distance between samples

Usage

```
geomSampling(size, geom, split_id = NULL, chainSampling = NULL)
```

Arguments

size	the sample size. Either an integer of absolute number of samples or if it is between (0, 1) it will sample a percentage relative to the number of available observations within the group.
geom	a terra::SpatVector object opened with terra::vect()
split_id	character. The variable name of the geometry to use as split factor for sampling
chainSampling	chains different methods of sampling by providing the result of another samplingMethod randomSampling(), spacedSampling(), stratifiedSampling().

Value

A icesat2_sampling_method for defining the method used in sample()

get_catalog_id *Retrieve the Google Earth Engine image catalog id*

Description

Retrieve the Google Earth Engine image catalog id

Usage

```
get_catalog_id(id)
```

Arguments

id	character. The id retrieved from the data.table resulting from search_datasets().

Value

The catalog id to open within Google Earth Engine.

gridSampling *Get samples stratified by grid cells of specified size*

Description

Get samples stratified by grid cells of specified size

Usage

```
gridSampling(size, grid_size, chainSampling = NULL)
```

Arguments

size	the sample size. Either an integer of absolute number of samples or if it is between (0, 1) it will sample a percentage relative to the number of available observations within the group.
grid_size	generic params. The gridSampling() will take a grid_size parameter defining the grid size for the sampling
chainSampling	chains different methods of sampling by providing the result of another samplingMethod randomSampling(), spacedSampling(), stratifiedSampling().

Value

A icesat2_sampling_method for defining the method used in sample()

ICESat2VegR_configure *Configure Python environment for ICESat2VegR cloud features*

Description

This function sets up a robust, persistent Python environment for the cloud-related features of **ICESat2VegR**, including:

Usage

```
ICESat2VegR_configure(
  envname = "icesat2-env",
  py_ver = "3.10",
  prefer_conda = TRUE,
  force_conda_forge = TRUE,
  allow_session_installs = TRUE,
  manage_aiobotocore = TRUE,
  auto_restart = TRUE,
  verbose = TRUE,
  quick_probe = c("h5py", "earthaccess", "ee")
)
```

Arguments

envname	Character. Name of the conda/virtualenv environment to use or create. Default is "icesat2-env". If this is the default value and the environment variable CONDA_DEFAULT_ENV is set, that value is used instead (useful when running inside an existing conda environment).
py_ver	Character. Python version to request when creating a new env (default "3.10").
prefer_conda	Logical. If TRUE (default), use conda/Miniconda. If FALSE, a virtualenv is used instead.
force_conda_forge	Logical. If TRUE (default), configure conda to use conda-forge with strict channel priority (recommended).
allow_session_installs	Logical. If TRUE (default) and Python is already initialized in the current R session, attempt session-only installs first via reticulate::py_require() and, if needed, a pip call from the active interpreter. If that fails, a persistent env is prepared instead.
manage_aiobotocore	Logical. If TRUE (default), attempt to detect and resolve common aiobotocore/botocore/boto3 version conflicts by pinning compatible versions and optionally uninstalling aiobotocore (which is not required for ICESat2VegR).
auto_restart	Logical. If TRUE (default), attempts to restart the R session in RStudio via rstudioapi::restartSession() after preparing the conda environment, so that the new Python binding is cleanly established.
verbose	Logical. If TRUE (default), print progress messages.
quick_probe	Character vector of Python import names to test for a fast early exit (default: c("h5py", "earthaccess", "ee")).

Details

- Downloading ICESat-2 data via earthaccess
- Interfacing with Google Earth Engine via earthengine-api
- Handling S3 access via boto3/botocore

The function:

- Detects whether Python is already initialized in the R session.
- Performs a fast "quick probe" to see if required modules are present (h5py, earthaccess, ee by default).
- If everything is already available, it returns immediately.
- Otherwise, it:
  - Ensures Miniconda is installed (if prefer_conda = TRUE).
  - Creates or reuses a conda environment (default: "icesat2-env").
  - Installs required Python modules using conda-forge (and pip fallback).
  - Optionally tries in-session installs if Python is already initialized.
  - Optionally resolves common aiobotocore/botocore conflicts.
  - Optionally restarts the R session in RStudio to bind the new env.

Typical usage is simply:

```
ICESat2VegR_configure()
```

You usually only need to run this:

- Once per machine, or
- After major Python / Miniconda changes, or
- If the package reports missing Python modules.

Value

Logical `TRUE` on success, `FALSE` otherwise.

Checked/installed Python modules

The function checks for the following import names and installs the corresponding packages when missing:

Import name	Install name
numpy	numpy
h5py	h5py
packaging	packaging
ee	earthengine-api
earthaccess	earthaccess
googleapiclient	google-api-python-client
boto3	boto3
botocore	botocore
s3transfer	s3transfer

Examples

```
## Not run:
  # Basic configuration using defaults
  ICESat2VegR_configure()

  # Use existing conda env named "icesat2"
  ICESat2VegR_configure(envname = "icesat2")

## End(Not run)
```

---

map_create *Create a Prediction Map in Google Earth Engine Using a Fitted Random Forest Model*

Description

Applies a fitted Random Forest model (from R's `randomForest` package) to a Google Earth Engine (**EE**) image or image collection and returns an EE image containing predicted values.

The model is converted to an Earth Engine estimator using `build_ee_forest`, which internally serializes the R forest through the ICESat2VegR C++ module (`icesat2_module`) and constructs an EE `Classifier` via `ee$Classifier$decisionTreeEnsemble()`.

Usage

```
map_create(
  model,
  stack,
  aoi = NULL,
  reducer = c("mosaic", "median"),
  mode = c("auto", "classifier"),
  to_float = TRUE
)
```

Arguments

model	A fitted randomForest::randomForest model, or a wrapper object containing an element named model.
stack	An ee$Image or ee$ImageCollection providing the predictor variables (bands) matching those used to train the model.
aoi	Optional EE geometry. If provided, the output map is clipped to it.
reducer	Aggregation method when stack is an image collection. One of: "mosaic" (default) or "median".
mode	Controls how the estimator is applied. Currently supported: • "auto" — (default) applies the classifier with img$classify(). • "classifier" — explicitly applies img$classify(). The "regressor" option is not supported, as the current implementation only builds an Earth Engine classifier.
to_float	Logical; if TRUE (default) converts the output band to 32-bit float.

Details

This function works for both:

- EE images: predictors already merged into one raster.
- EE image collections: predictions computed for each image and reduced using mosaic or median.

The output band is always named "prediction_layer".

Properties from a zero-pixel placeholder image are copied to preserve band metadata.

Value

An Earth Engine ee$Image containing model predictions.

See Also

build_ee_forest randomForest::randomForest Earth Engine API reference: https://developers.google.com/earth-engine

Examples

```
## Not run:
  library(ICESat2VegR)
  library(randomForest)

  # Fit a model in R
  data(airquality)
  air <- na.omit(airquality)

  rf_model <- randomForest(
    x = air[, 2:6],
    y = air[, 1],
    ntree = 100,
    importance = TRUE
  )

  # Suppose 'stack_ee' is an Earth Engine image with matching bands
  pred <- map_create(
    model = rf_model,
    stack = stack_ee,
    aoi   = NULL,
    mode  = "auto"
  )

  # Now pred is: ee$Image with one band ("prediction_layer")

## End(Not run)
```

`map_download` *map_download: create task → start → (monitor) → download/return id*

Description

One function to export an EE image to Drive, Cloud Storage, or Asset; optionally monitor the task; and, for Drive/GCS, download the result locally. Returns a local file (or `SpatRaster` if `terra` is available) for Drive/GCS, or the `asset_id` (invisible) for Asset exports.

Usage

```
map_download(
  ee_image,
  method = c("drive", "gcs", "asset"),
  region,
  scale = NULL,
  file_name_prefix,
  dsn = NULL,
  drive_folder = NULL,
  gcs_bucket = NULL,
  asset_id = NULL,
  monitor = TRUE,
  task_time = 5,
```

```
  ...
)
```

Arguments

ee_image	An ee$Image to export.
method	One of "drive", "gcs", or "asset".
region	EE geometry/feature collection defining the export region.
scale	Numeric pixel size in meters.
file_name_prefix	Export file prefix (used to search/download).
dsn	Destination path on disk (Drive/GCS methods).
drive_folder	Drive folder name for Drive exports.
gcs_bucket	Cloud Storage bucket name for GCS exports.
asset_id	Destination asset id for Asset exports.
monitor	Logical; if TRUE, poll the task until completion.
task_time	Seconds between polls when monitoring.
...	Additional arguments forwarded to the corresponding export helper (ee_image_to_drive(), ee_image_to_gcs(), or ee_image_to_asset()), e.g., CRS, pyramiding policy, maxPixels, etc.

Value

For "drive"/"gcs", a local path or a SpatRaster if terra is installed. For "asset", returns invisible(asset_id).

Examples

```
## Not run:
out <- map_download(
  ee_image = img, method = "drive", region = aoi, scale = 10,
  file_name_prefix = "my_pred", dsn = "pred.tif",
  drive_folder = "EE_Exports", monitor = TRUE
)

## End(Not run)
```

---

map_view *Compose a leaflet map from multiple layers (EE rasters and/or vectors)*

Description

High-level map composer capable of adding Earth Engine raster tiles (optionally tiled by AOI grid) and vector overlays (sf/SpatVector), with layer control, legends, and per-group opacity sliders.

Usage

```
map_view(
  layers,
  base_tiles = c("OSM", "Carto.Light", "Carto.Dark"),
  add_layers_control = TRUE,
  add_opacity_controls = TRUE,
  fit_to = NULL
)
```

Arguments

layers	A list of layer specs. For rasters: `list(type="ee_image", x, bands, aoi=NULL, group=NULL, mi` For vectors: `list(type="vector", vect, group=NULL, border_color, border_weight, fill, fill_color, fill_opacity, color_field, palette, legend_title)`.
base_tiles	One of `"OSM"`, `"Carto.Light"`, `"Carto.Dark"`.
add_layers_control	Logical; add layers control panel.
add_opacity_controls	Logical; add per-group opacity sliders.
fit_to	Optional EE Rectangle to fit the initial view.

Value

A `leaflet` htmlwidget.

Examples

```
## Not run:
m <- map_view(list(
  list(type="ee_image", x=img, bands="prediction_layer", aoi=aoi,
       group="Prediction", min_value=0, max_value=30,
       legend=list(title="Height (m)", auto="quantile")),
  list(type="vector", vect=polys_sf, group="Sites", color_field="site")
))

## End(Not run)
```

---

plot.varSel *Plot variable importance for* `varSel` *objects*

Description

Plot scaled variable importance for a `varSel` object using a horizontal barplot. Variables are ordered so that the most important metrics appear at the **top** (largest bars).

When `which = "importance"` all variables are shown, and those selected by `varSel` are highlighted in a different color. An optional color palette (e.g., `viridis::inferno`) can be supplied to color the bars by importance.

Usage

```
## S3 method for class 'varSel'
plot(
  x,
  which = c("sel.importance", "importance"),
  main = "Random Forest variable importance",
  xlab = "Scaled importance",
  col.selected = "steelblue",
  col.other = "grey80",
  palette = NULL,
  legend.loc = "bottomright",
  ...
)
```

Arguments

x	An object of class "varSel".
which	Character; either "sel.importance" (default) to plot the importance of selected variables only, or "importance" to plot the full-model importance (all predictors).
main	Plot title.
xlab	Label for the x-axis (importance scale).
col.selected	Base color for selected variables when no palette is given.
col.other	Base color for non-selected variables when no palette is given.
palette	Optional color palette. Can be: • a function f(n) returning n colors (e.g., viridis::inferno), • or a character vector of colors used to build a gradient via colorRampPalette. The palette is applied to all bars and then the color of selected variables is overridden by col.selected when which = "importance".
legend.loc	Legend location, e.g. "bottomright"
...	Additional graphical arguments passed to barplot.

Examples

```
require(randomForest)

data(airquality)
airquality <- na.omit(airquality)

xdata <- airquality[, 2:6]
ydata <- airquality[, 1]

vs <- varSel(xdata, ydata, ntree = 200, min.imp = 0.2)

## Selected variables only
plot(vs, which = "sel.importance")

## All variables, highlighting selected ones, with inferno palette
if (requireNamespace("viridis", quietly = TRUE)) {
  plot(vs, which = "importance", palette = viridis::inferno)
}
```

predict_h5 *Model prediction over data.tables using HDF5 file as output*

Description

Model prediction over a data.table from ATL03 or ATL08 data containing geolocation data. It can both append results to an existing HDF5 file or create a new file, allowing to incrementally add predictions to the file to avoid memory issues.

Usage

```
predict_h5(model, dt, output)
```

Arguments

model	The trained model object
dt	The input data.table to run the model
output	The output HDF5 file path

Value

An icesat2.predict_h5, which is an h5 file with latitude, longitude and prediction datasets.

Examples

```
atl03_path <- system.file(
  "extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

atl03_h5 <- ATL03_read(atl03_path = atl03_path)
atl03_seg_dt <- ATL03_seg_metadata_dt(atl03_h5)

linear_model <- stats::lm(h_ph ~ segment_ph_cnt, data = atl03_seg_dt)
output_h5 <- tempfile(fileext = ".h5")
predicted_h5 <- predict_h5(linear_model, atl03_seg_dt, output_h5)

# List datasets
predicted_h5$ls()$name

# See predicted values
head(predicted_h5[["prediction"]][])

# Close the file
close(predicted_h5)
```

prepend_class *Prepend a class to an object's list of classes*

Description

Prepend a class to an object's list of classes

Usage

```
prepend_class(obj, className)
```

Arguments

obj	The object to which prepend the class.
className	character with the name of the class to prepend.

Value

Nothing, it replaces the class attribute in place

randomSampling *Pure random sampling method*

Description

Pure random sampling method

Usage

```
randomSampling(size)
```

Arguments

size	the sample size. Either an integer of absolute number of samples or if it is between (0, 1) it will sample a percentage relative to the number of available observations within the group.

Value

A icesat2_sampling_method for defining the method used in sample()

`rasterize_h5` *Rasterizes the model prediction saved in the HDF5 file*

Description

This is used after running the prediction using `predict_h5()` function to rasterize and aggregate the prediction within raster cells. By default it will calculate (n, mean, variance * (n - 1), min, max, sd) in a single raster file with those 6 bands in that order. You can modify this behavior by changing the `agg_function`, `agg_join` and `finalizer` parameters, see details section.

Usage

```
rasterize_h5(h5_input, output, bbox, res, ...)
```

Arguments

`h5_input`	The input HDF5 file path
`output`	The output raster file path
`bbox`	The bounding box of the raster
`res`	The resolution of the raster
`...`	Additional parameters (see details section)

Details

This function will create five different aggregate statistics (n, mean, variance, min, max).

Within `...` additional parameters we can use:

`agg_function`: is a formula which return a data.table with the aggregate function to perform over the data. The default is:

```
~data.table(
    n = length(x),
    mean = mean(x,na.rm = TRUE),
    variance = var(x) * (length(x) - 1),
    min = min(x, na.rm=T),
    max = max(x, na.rm=T)
  )
```

`agg_join`: is a function to merge two data.table aggregates from the `agg_function`. Since the h5 files will be aggregated in chunks to avoid memory overflow, the statistics from the different chunks should have a function to merge them.

The default function is:

```
function(x1, x2) {
    combined = data.table()
    x1$n[is.na(x1$n)] = 0
    x1$mean[is.na(x1$mean)] = 0
    x1$variance[is.na(x1$variance)] = 0
    x1$max[is.na(x1$max)] = -Inf
    x1$min[is.na(x1$min)] = Inf
```

```
    combined$n = x1$n + x2$n

    delta = x2$mean - x1$mean
    delta2 = delta * delta

    combined$mean = (x1$n * x1$mean + x2$n * x2$mean) / combined$n
    combined$variance = x1$variance + x2$variance +
      delta2 * x1$n * x2$n / combined$n

    combined$min = pmin(x1$min, x2$min, na.rm=F)
    combined$max = pmax(x1$max, x2$max, na.rm=F)
    return(combined)
}
```

`finalizer`: is a list of formulas to generate the final rasters based on the intermediate statistics from the previous functions. The default `finalizer` will calculate the sd, based on the `variance` and n values. It is defined as:

```
list(
  sd = ~sqrt(variance/(n - 1)),
)
```

Examples

```
atl08_path <- system.file(
  "extdata",
  "atl08_clip.h5",
  package = "ICESat2VegR"
)

atl08_h5 <- ATL08_read(atl08_path = atl08_path)
atl08_dt <- ATL08_seg_attributes_dt(atl08_h5)

xmin <- min(atl08_dt$longitude)
xmax <- max(atl08_dt$longitude)
ymin <- min(atl08_dt$latitude)
ymax <- max(atl08_dt$latitude)

linear_model <- stats::lm(h_canopy ~ canopy_openness, data = atl08_dt)
output_h5 <- tempfile(fileext = ".h5")
predicted_h5 <- predict_h5(linear_model, atl08_dt, output_h5)
output_raster <- tempfile(fileext = ".tif")

rasterize_h5(
  predicted_h5,
  output = output_raster,
  bbox = terra::ext(xmin, xmax, ymin, ymax),
  res = 0.003
)
```

rasterSampling *Get observations given a minimum radius distance between samples*

Description

Get observations given a minimum radius distance between samples

Usage

```
rasterSampling(size, raster, chainSampling = NULL)
```

Arguments

size	the sample size. Either an integer of absolute number of samples or if it is between (0, 1) it will sample a percentage relative to the number of available observations within the group.
raster	a terra::SpatRaster object opened with terra::rast()
chainSampling	chains different methods of sampling by providing the result of another samplingMethod randomSampling(), spacedSampling(), stratifiedSampling().

Value

A icesat2_sampling_method for defining the method used in sample()

rgt_extract *Extract reference ground track from ATL03 segments*

Description

This function extracts reference ground track from ICESat-2 ATL03 data and writes it as a GDAL vector format.

Usage

```
rgt_extract(h5)
```

Arguments

h5	A ICESat-2 ATL03 object (output of ATL03_read() or ATL08_read() function). An S4 object of class icesat2.atl03_dt or icesat2.atl08_dt.

Details

This function will use the reference photons from the segments as reference for deriving the ground tracks. The begining and end of the lines are interpolated from the information regarding the position of the reference photon within the segment and the segment length.

Value

Returns the ground track boundaries as terra::SpatVector extracted from "orbit_info", along with other orbit information.

See Also

https://icesat-2.gsfc.nasa.gov/sites/default/files/page_files/ICESat2_ATL03_ATBD_r006.pdf

Examples

```
# Specifying the path to ATL03 H5 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)

# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)

# Extracting ATL03 photons attributes
rgt <- rgt_extract(h5 = atl03_h5)
head(rgt)

terra::plet(rgt)

close(atl03_h5)
```

---

sample *Sample method for applying multiple sampling methods*

Description

Sample method for applying multiple sampling methods

Usage

```
sample(x, ..., method)
```

Arguments

x	the generic input data to be sampled
...	generic params to pass ahead to the specific sampling function
method	the sampling method to use.

Details

It is expected that the user pass a method parameter within ...

See Also

randomSampling(), spacedSampling(), gridSampling(), stratifiedSampling(), geomSampling(), rasterSampling()

`sample_ATL_granules_by_year` *Sample ICESat-2 ATL granule URLs by year*

Description

Given a vector (or first column of a matrix/data frame) of ICESat-2 ATL03/ATL08 granule URLs or file paths, this function:

- Attempts to parse the acquisition year from each URL/path.
- Groups granules by year.
- Samples up to `n_per_year` unique URLs per year.

This is useful for creating manageable subsets of ATL granules for testing, model fitting, or cross-validation.

Usage

```
sample_ATL_granules_by_year(urls, n_per_year = 5, seed = NULL)
```

Arguments

`urls`	Character vector, matrix, or data frame containing granule URLs or file paths. If a matrix or data frame is provided, the first column is used.
`n_per_year`	Integer. Maximum number of URLs to sample per year (default `5`).
`seed`	Optional integer seed passed to `set.seed()` to make the sampling reproducible. If `NULL` (default), the random state is left unchanged.

Details

The year is extracted using the following heuristics:

1. A path component of the form `/YYYY/` (e.g., `".../2020/..."`).
2. A filename pattern of the form `"ATL0[38]_YYYYMMDD..."`.

URLs for which no year can be detected are dropped with a warning.

Value

A data frame with columns:

- `year`: integer acquisition year.
- `url`: the sampled URL or file path.

Rows are ordered by `year` and then `url`.

Examples

```
## Not run:
  urls <- c(
    "https://example.org/ATL03_20200101000000_001.h5",
    "https://example.org/ATL03_20200102000000_002.h5",
    "https://example.org/ATL03_20210101000000_003.h5"
  )

  sample_df <- ee_sample_atl_granules_by_year(urls, n_per_year = 1, seed = 123)
  sample_df

## End(Not run)
```

`search_datasets` *Search for Google Earth Engine datasets*

Description

Search for Google Earth Engine datasets

Usage

```
search_datasets(
  ...,
  title = TRUE,
  description = TRUE,
  operator = "and",
  refresh_cache = FALSE
)
```

Arguments

`...`	character arguments to search for within title and/or description
`title`	logical. Whether should search within the title, default TRUE.
`description`	logical. Whether should search within the description of the dataset, default TRUE.
`operator`	character. Should be either "OR" or "AND" to tell if the search needs to include all the queries "AND" or any of the queries "OR". Default "AND".
`refresh_cache`	flag indicating if the results cache should be refreshed, default FALSE.

Value

A `data.table` containing the id, title and description of the datasets that matched the supplied query ordered by relevance.

seg_ancillary_extract *Given a stack image raster from GEE retrieve the point geometry with values for the images*

Description

Given a stack image raster from GEE retrieve the point geometry with values for the images

Usage

```
seg_ancillary_extract(stack, geom, scale = 10, chunk_size = 1000)
```

Arguments

stack	A single image or a vector/list of images from Earth Engine.
geom	A geometry from terra::SpatVector read with terra::vect.
scale	The scale in meters for the extraction (image resolution).
chunk_size	If the number of observations is greater than 1000, it is recommended to chunk the results for not running out of memory within GEE server, default is chunk by 1000.

Value

A data.table::data.table with the properties extracted from the ee images.

slope *Compute terrain slope (degrees) from a DEM image*

Description

Computes the terrain *slope* in degrees for each pixel of an Earth Engine ee$Image representing a digital elevation model (DEM).

This method is a thin wrapper around ee$Terrain$slope() and returns the same output structure. Slope is computed using Earth Engine's internal gradient operator, which relies on the 4-connected neighborhood around each pixel. As a result, edge pixels may contain missing values depending on the input DEM.

Usage

```
slope(x)
```

Arguments

x	An ee$Image representing a DEM from which slope will be derived.

Value

An ee$Image with one band named "slope" containing terrain slope values in degrees.

Examples

```
## Not run:
  ee <- reticulate::import("ee")
  dem <- ee$Image("NASA/NASADEM_HGT/001")
  slp <- slope(dem)

## End(Not run)
```

`spacedSampling` *Get observations given a minimum radius distance between samples*

Description

Get observations given a minimum radius distance between samples

Usage

```
spacedSampling(size, radius, spatialIndex = NULL, chainSampling = NULL)
```

Arguments

`size`	the sample size. Either an integer of absolute number of samples or if it is between (0, 1) it will sample a percentage relative to the number of available observations within the group.
`radius`	the minimum radius between samples.
`spatialIndex`	optional parameter. You can create a spatial index for accelerating the search space with `ANNIndex` and reuse that if needed elsewhere. It will create one automatically everytime if you don't specify one.
`chainSampling`	chains different methods of sampling by providing the result of another samplingMethod `randomSampling()`, `spacedSampling()`, `stratifiedSampling()`.

Value

A `icesat2_sampling_method` for defining the method used in `sample()`

`stratifiedSampling` *Get samples stratified by a variable binning histogram*

Description

Get samples stratified by a variable binning histogram

Usage

```
stratifiedSampling(size, variable, chainSampling = NULL, ...)
```

Arguments

size	the sample size. Either an integer of absolute number of samples or if it is between (0, 1) it will sample a percentage relative to the number of available observations within the group.
variable	Variable name used for the stratification
chainSampling	chains different methods of sampling by providing the result of another samplingMethod randomSampling(), spacedSampling(), stratifiedSampling().
...	forward to the graphics::hist(), where you can manually define the breaks.

Value

A icesat2_sampling_method for defining the method used in sample()

---

to_vect *Generic function to convert icesat2 classes and sf to terra::SpatVector*

Description

Generic function to convert icesat2 classes and sf to terra::SpatVector

Usage

```
to_vect(x, ...)
```

Arguments

x	The icesat2 or sf object to convert to terra::SpatVector
...	other potential parameters needed.

Value

The icesat2 or sf object as the appropriate terra::SpatVector

Examples

```
# Get path to ATL03 h5 file
atl03_path <- system.file("extdata",
  "atl03_clip.h5",
  package = "ICESat2VegR"
)
# Reading ATL03 data (h5 file)
atl03_h5 <- ATL03_read(atl03_path = atl03_path)
# Extracting ATL03 segment attributes
atl03_segment_dt <- ATL03_seg_metadata_dt(atl03_h5 = atl03_h5)

# Extract vector
atl03_segment_vect <- to_vect(atl03_segment_dt)

# Terra plot
terra::plot(atl03_segment_vect, col = atl03_segment_vect$segment_ph_cnt)
```

```
# Export as temp gpkg
temp_vect <- tempfile(fileext = ".gpkg")
terra::writeVector(atl03_segment_vect, temp_vect)

head(atl03_segment_dt)
close(atl03_h5)
```

`varSel` *Random Forest Variable Selection (Breiman-only)*

Description

Implements the Random Forest model selection approach of Murphy et al. (2010), using Breiman's original **randomForest** implementation. This is a simplified adaptation of `rfUtilities::rf.modelSel`, restricted to the Breiman implementation and modified for the ICESat2VegR package.

It returns the selected variables and importance metrics, but does not fit a final model.

Usage

```
varSel(
  xdata,
  ydata,
  imp.scale = c("mir", "se"),
  r = c(0.25, 0.5, 0.75),
  min.imp = NULL,
  seed = NULL,
  parsimony = NULL,
  kappa = FALSE,
  ...
)
```

Arguments

`xdata`	Matrix or data.frame of predictor variables (columns = predictors).
`ydata`	Response vector. For classification, `ydata` must be a factor; otherwise the model is fit in regression mode.
`imp.scale`	Character; type of scaling for importance values, either `"mir"` or `"se"`. Default is `"mir"`.
`r`	Numeric vector of importance percentiles to test, e.g., `c(0.25, 0.50, 0.75)`. These percentiles are used to define thresholds for building nested models during selection.
`min.imp`	Optional numeric in $[0, 1]$ used as a minimum scaled importance cutoff (in MIR or SE scale) to filter the final set of variables. Variables whose scaled importance is below this threshold are dropped from the selected set `selvars`. If `NULL` (default), no cutoff is applied after the Murphy-style model selection.
`seed`	Optional integer; sets the random seed in the global R environment. This is strongly recommended for reproducibility.

parsimony	Numeric in (0,1); threshold for selecting among competing models. If specified, models whose errors are within parsimony of the best model (OOB / class error for classification, variance explained / MSE for regression) are considered, and the one with the fewest parameters is chosen.
kappa	Logical; use the chance-corrected κ statistic as the primary optimization criterion for classification instead of percent correctly classified (PCC). This corrects PCC for random agreement.
...	Additional arguments passed to randomForest, e.g. ntree = 1000, replace = TRUE, proximity = TRUE.

Details

For classification, ensure that ydata is a factor; otherwise the model is fit in regression mode.

Selection strategy:

- For classification, candidate models are compared using OOB PCC (or κ, if kappa = TRUE) and maximum within-class error, with preference for more parsimonious models.
- For regression, candidate models are compared using percent variance explained and MSE, with preference for more parsimonious models.
- After the best model is chosen, the optional min.imp cutoff is applied to the scaled importance (MIR/SE) from the full model to remove weak predictors from the final set selvars.

Typical choices for min.imp:

- MIR: min.imp in $[0.1, 0.3]$ (e.g., keep variables with at least 20\
- SE: min.imp in $[0.01, 0.05]$, remembering that SE-scaled importance values sum to 1.

Value

An object of class "varSel" (a list) with components:

- selvars Character vector of selected variable names (after applying min.imp, if provided).
- test Data.frame of model-selection diagnostics, containing error metrics, threshold, number of parameters, and the variables used in each candidate model (for inspection only).
- importance Data.frame of scaled importance values for all variables in the full model (columns parameter, importance).
- sel.importance Data.frame of scaled importance values for the selected variables.
- parameters List of variables used in each candidate model.
- scaling Character; the importance scaling used ("mir" or "se").

See Also

randomForest and plot.varSel.

Examples

```
require(randomForest)

data(airquality)
airquality <- na.omit(airquality)

xdata <- airquality[, 2:6]
```

```
ydata <- airquality[, 1]

## Regression example with MIR scaling and importance cutoff
vs.regress <- varSel(
  xdata     = xdata,
  ydata     = ydata,
  imp.scale = "mir",
  ntree     = 500,
  min.imp   = 0.2
)

vs.regress$selvars

## Plot all variables, highlighting selected ones
plot(vs.regress, which = "importance")
```

`vect_as_ee` *Convert vector data to Google Earth Engine FeatureCollection (no rgee)*

Description

`vect_as_ee()` converts vector data to a Google Earth Engine `ee$FeatureCollection` using **reticulate** to call the official Python Earth Engine API-**without** relying on **rgee**. It accepts `sf`/`sfc`/`sfg` objects or `terra::SpatVector`, validates and reprojects geometries, and either returns an in-memory FeatureCollection or exports to an EE Asset.

Usage

```
vect_as_ee(
  x,
  via = c("getInfo", "getInfo_to_asset"),
  assetId = NULL,
  overwrite = TRUE,
  proj = "EPSG:4326",
  make_valid = TRUE,
  quiet = FALSE,
  monitoring = TRUE,
  poll_interval_sec = 5,
  poll_timeout_sec = 3600
)
```

Arguments

`x`	An input vector object: an `sf`, `sfc`, or `sfg` object, or a `terra::SpatVector`. For `sfg`/`sfc`, a simple point/line/polygon geometry is accepted; for `SpatVector`, it will be converted to `sf`.
`via`	Character; one of `c("getInfo","getInfo_to_asset")`.

- `"getInfo"` returns an in-memory `ee$FeatureCollection` built from the GeoJSON of `x`.

- "getInfo_to_asset" creates an EE export task to save the collection to an Earth Engine **Asset** (requires assetId).

assetId — Character; EE asset id (e.g., "users/you/my_fc"). **Required** when via = "getInfo_to_asset".

overwrite — Logical; if TRUE, attempt to delete an existing EE asset at assetId before export.

proj — Character CRS string (e.g., "EPSG:4326"). Input will be transformed to this CRS before conversion. Default is "EPSG:4326".

make_valid — Logical; if TRUE, attempt to repair invalid geometries (sf::st_make_valid() / terra::makeValid()).

quiet — Logical; if FALSE, print progress messages (e.g., task state).

monitoring — Logical; when exporting to asset, if TRUE poll the task until completion or timeout, otherwise return immediately after starting.

poll_interval_sec — Numeric; seconds to wait between task status checks when monitoring = TRUE. Default 5.

poll_timeout_sec — Numeric; maximum seconds to keep polling before timing out. Default 3600 (1 hour).

Details

- The function converts x to sf, enforces a defined CRS, optionally fixes invalid geometries, and transforms to proj (default WGS84).
- Properties are carried alongside geometry; however, Earth Engine does not accept POSIX* timestamp columns or property names containing '.'. The function stops with a clear error if such columns are found-convert them to character or rename before calling.
- GeoJSON is built in-memory (via **geojsonsf** when available) or via a temporary .geojson file as a fallback, then parsed to a list for ee$FeatureCollection.
- Requires a properly initialized EE Python environment (ee.Initialize() in the active Python session used by **reticulate**).

Value

If via = "getInfo", returns an in-memory ee$FeatureCollection object (reticulate Python object). If via = "getInfo_to_asset", returns an ee$FeatureCollection **referencing** assetId (after successful export), or returns immediately if monitoring = FALSE.

Errors & Constraints

- Undefined CRS: the function stops if x has no CRS set.
- Unsupported columns: the function stops if any property is POSIX* or if names contain dots (.).
- Export failures: if via = "getInfo_to_asset" and the EE task fails or is cancelled, an error is thrown when monitoring = TRUE.

See Also

- Earth Engine Python API docs: ee$FeatureCollection
- Geometry repair: sf::st_make_valid(), terra::makeValid()
- GeoJSON helpers: **geojsonsf**, sf::st_write()

Examples

```
## Not run:
# -- Prerequisites (Python side):
# import ee; ee.Initialize()
#
# Example with sf POINTS
library(sf)
pts <- st_as_sf(data.frame(
  id = 1:3,
  x  = c(-84.02171, -84.02025, -84.02026),
  y  = c(31.29891, 31.31945, 31.31938)
), coords = c("x","y"), crs = "EPSG:4326")

# In-memory FeatureCollection:
fc <- vect_as_ee(pts, via = "getInfo")

# Export to an EE asset (monitor until done):
# fc_asset <- vect_as_ee(
#   pts,
#   via = "getInfo_to_asset",
#   assetId = "users/you/demo_points",
#   overwrite = TRUE,
#   monitoring = TRUE
# )

## End(Not run)
```

`write_geojson` *Safely write a GeoJSON file from a lon/lat table*

Description

Converts a data frame or similar tabular object with longitude/latitude columns into a point layer and writes it to disk as a GeoJSON file. The function first attempts to use **terra** via `ICESat2VegR::to_vect()`, and falls back to **sf** if available.

Usage

```
write_geojson(
  dt,
  path,
  xcol = "lon",
  ycol = "lat",
  crs = "EPSG:4326",
  overwrite = TRUE
)
```

Arguments

`dt`	A data frame, `data.table`, or similar object containing at least two numeric columns for coordinates.
`path`	Character. Path to the output GeoJSON file.

xcol, ycol	Character. Names of the longitude and latitude columns in dt. Defaults are "lon" and "lat".
crs	Character. Coordinate reference system of the input coordinates. Default is "EPSG:4326" (longitude/latitude in WGS84).
overwrite	Logical. If TRUE (default), allow overwriting an existing file.

Value

Invisibly returns TRUE on success. An error is thrown if both the **terra**-based and **sf**-based write attempts fail.

Examples

```
## Not run:
  pts <- data.frame(
    id  = 1:3,
    lon = c(-82.35, -82.34, -82.33),
    lat = c( 29.65,  29.66,  29.67)
  )

  write_geojson_safe(pts, "points.geojson")

## End(Not run)
```

Index