Package 'pacu'

September 26, 2025

Type Package

Title Precision Agriculture Computational Utilities

Version 0.1.72

Description Support for a variety of commonly used precision agriculture operations. Includes functions to download and process raw satellite images from Sentinel-2 <https://documentation.dataspace.copernicus.eu/APIs/OData.html>. Includes functions that download vegetation index statistics for a given period of time, without the need to download the raw images <https://documentation.dataspace.copernicus.eu/APIs/SentinelHub/Statistical.html>. There are also functions to download and visualize weather data in a historical context. Lastly, the package also contains functions to process yield monitor data. These functions can build polygons around recorded data points, evaluate the overlap between polygons, clean yield data, and smooth yield maps.

Depends R (>= 4.0.0)

License GPL (>= 3)

Encoding UTF-8

VignetteBuilder knitr

BugReports https://github.com/cldossantos/pacu/issues

RoxygenNote 7.3.2

Imports apsimx, gstat, httr, jsonlite, sf, stars, units, XML, concaveman, tmap (>= 4.1)

Suggests knitr, ggplot2, patchwork, mgcv, nasapower, spData, rmarkdown, nlraa, car, minpack.lm, MASS

NeedsCompilation no

Author dos Santos Caio [aut, cre], Miguez Fernando [aut]

Maintainer dos Santos Caio <clsantos@iastate.edu>

Repository CRAN

Date/Publication 2025-09-26 06:40:03 UTC

Contents

`merge` *Merge trial objects*

Description

Generic merge functionalities for pacu objects

Usage

```
## S3 method for class 'trial'
merge(...)
```

Arguments

`...`	pa_trial objects

Value

object of class "trial"

pacu.options *Environment which stores PACU options*

Description

Environment which can store the path to the executable, warning settings and where examples are located. Creating an environment avoids the use of global variables or other similar practices which would have possible undesirable consequences.

Usage

```
pacu.options
```

Format

An object of class `environment` of length 6.

Details

Environment which stores PACU options

Value

This is an environment, not a function, so nothing is returned.

Examples

```
names(pacu.options)
## to suppress messages
pacu_options(suppress.messages = TRUE)
```

pacu_options *Setting some options for the package*

Description

Set settings regarding messages and default behaviors of the package

Usage

```
pacu_options(
  suppress.warnings = FALSE,
  suppress.messages = FALSE,
  apportion.size.multiplier = 1,
  minimum.coverage.fraction = 0.5
)
```

Arguments

suppress.warnings
: whether to suppress warning messages

suppress.messages
: whether to suppress messages

apportion.size.multiplier
: a multiplier used to determine the size of the apportioning polygons in the RITAS algorithm. A value of sqrt(2) will make polygons approximately the same size as the harvest polygons. Smaller values increase the resolution but also increase the computation time substantially.

minimum.coverage.fraction
: The minimum area of an apportioning polygon that needs to be covered to conduct the apportioning operation.

Details

Set pacu options

Value

as a side effect it modifies the 'pacu.options' environment.

Examples

```
## Not run:
names(pacu.options)
pacu_options(suppress.warnings = FALSE)
pacu.options$suppress.warnings

## End(Not run)
```

---

pa_2utm *Reproject a sf object to UTM coordinates*

Description

Reproject a sf object to UTM coordinates

Usage

```
pa_2utm(df, verbose = FALSE)
```

Arguments

df	sf object to be reprojected to UTM coordinates
verbose	whether to print operation details

Details

This function will attempt to automatically determine the adequate UTM zone and reproject a sf object,

Value

a sf object

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## for examples, see vignette pacu
```

`pa_adjust_obs_effective_area` *Adjust the effective area of each observation based on vehicular polygon overlap*

Description

Adjust the effective area of each observation based on vehicular polygon overlap

Usage

```
pa_adjust_obs_effective_area(
  polygons,
  obs.vector,
  var.label = "yield",
  overlap.threshold = 0,
  cores = 1L,
  verbose = FALSE
)
```

Arguments

`polygons`	sf object containing vehicular polygons
`obs.vector`	a vector containing the observations
`var.label`	a string used to label the columns (e.g., yield)
`overlap.threshold`	a fraction threshold to remove observations. A value of 0 does not remove any observations. A value of 1 removes all observations that overlap even minimally with neighboring observations.

cores	the number of cores used in the operation
verbose	whether to print operation details

Details

This function will make use of the vehicular polygons to evaluate the overlap between polygons and adjust the variable in obs.vector to the effective area in the polygon. This is primarely intended for yield.

Value

an sf object

`pa_apportion_mass` *Impose a regular grid over yield polygons*

Description

Impose a regular grid over yield polygons

Usage

```
pa_apportion_mass(
  polygons,
  mass.vector,
  cell.size = NULL,
  sum = FALSE,
  remove.empty.cells = TRUE,
  cores = 1L,
  verbose = FALSE
)
```

Arguments

polygons	sf object containing polygon geometries
mass.vector	a vector of mass observations
cell.size	optional numerical value (length 1) to be used as the width and height of the grid
sum	whether the apportioned values should be added together. This is useful in the case of overlaping polygons that have an additive effect. For example, polygons representing seeding rates.
remove.empty.cells	logical. Whether to remove empty cells, with NA values.
cores	the number of cores used in the operation
verbose	whether to print operation details

Details

This function will impose a regular grid over the yield polygons and compute the weighted average of the mass value represented by each polygon. The averages are weighted according to the polygon area.

Value

sf object

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## for examples, see vignette pacu
```

---

pa_browse_dataspace *Browse satellite products from the Copernicus Data Space Ecosystem*

Description

Browse satellite products from the Copernicus Data Space Ecosystem

Usage

```
pa_browse_dataspace(
  aoi,
  start.date,
  end.date,
  max.cloud.cover = 100,
  collection.name = c("SENTINEL-2"),
  product.name = c("MSIL2A"),
  max.results = 1000
)
```

Arguments

aoi	sf object used to filter satellite products
start.date	beginning of the time window to filter satellite products. The date format should be '%Y-%m-%d'.
end.date	end of the time window to filter satellite products. The date format should be '%Y-%m-%d'.

`max.cloud.cover`
: maximum cloud cover. Values should be between 0 and 100. Images with cloud cover assessment greater than this parameter will be removed from the list.

`collection.name`
: collection of products to filter. Currently, only SENTINEL-2 is supported.

`product.name`
: partial match of product name used to filter products. Currently, only supports MSIL2A. We plan to expand this in the future.

`max.results`
: maximum number of results to return

Details

'pa_browse_dataspace()' will use HTTP requests to communicate with the Data Space API and search for available satellite products matching the filters established by the function parameters.

Value

a list of entries available for download

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## Not run:
extd.dir <- system.file("extdata", package = "pacu")
area.of.interest <- sf::st_read(file.path(extd.dir, 'cobs_a_aoi.shp'), quiet = TRUE)
available.images <- pa_browse_dataspace(aoi = area.of.interest,
                                        max.cloud.cover = 10,
                                        start.date = '2023-01-01',
                                        end.date = '2023-12-31')

## End(Not run)
```

---

pa_cardinal_dates *Predict cardinal dates from satellite image data*

Description

Predict cardinal dates from satellite image data

Usage

```
pa_cardinal_dates(x, ...)

## S3 method for class 'numeric'
pa_cardinal_dates(
  x,
  y,
  baseline.months = c(1:3, 12),
  model = c("none", "card3", "scard3", "agauss", "harmonic"),
  prior.means,
  prior.vars,
  bias.correction,
  ...
)

## S3 method for class 'Date'
pa_cardinal_dates(
  x,
  y,
  baseline.months = c(1:3, 12),
  model = c("none", "card3", "scard3", "agauss", "harmonic"),
  prior.means,
  prior.vars,
  bias.correction,
  ...
)

## S3 method for class 'veg.index'
pa_cardinal_dates(
  x,
  y = NULL,
  baseline.months = c(1:3, 12),
  model = c("none", "card3", "scard3", "agauss", "harmonic"),
  prior.means,
  prior.vars,
  bias.correction,
  ...
)
```

Arguments

x	vector containing the date or day of the year of that the satellite data was collected
...	ignored
y	vector containing the satellite data value
baseline.months	vector containing the months used as a baseline reference for when there are no crops in the field. For example, c(1:3, 12) represent Jan, Feb, Mar, and Dec.

model	a string naming the model to be used to estimate cardinal dates
prior.means	a vector of length three containing the prior means for cardinal dates
prior.vars	a vector of length three containing the prior variances for cardinal dates
bias.correction	a vector of length three containing the bias correction factor for cardinal dates

Value

when x is a vector, returns a vector of length 3 with the predicted cardinal dates. When x is a veg.index object, returns a stars object with spatially distributed cardinal dates

Examples

```
## Not run:
x <- seq(1, 365, 6)
y <- nlraa::scard3(x, 120, 210, 300)
pa_cardinal_dates.vector(
  x = x,
  y = y,
  model = 'scard3',
  prior.means = c(130, 190, 297),
  prior.vars = c(11, 13, 18),
  bias.correction = c(10, 10, 10)
)

## End(Not run)
```

---

pa_check_yield *Check the yield data before processing with the pa_yield function*

Description

This function will check for red flags so the user can know of potential problems before using the pa_yield functions

Usage

```
pa_check_yield(input, algorithm = c("all", "simple", "ritas"))
```

Arguments

input	an sf object containing the input data from a yield monitor
algorithm	for which algorithm should the function check the data. Different algorithms require different information to be present in the input data set.

Details

This function will check the input yield data for any potential problems before the user runs the 'pa_yield()' function. Ideally, this function warn the user of potential problems.

Value

object of class check.yield

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
extd.dir <- system.file("extdata", package = "pacu")
raw.yield <- sf::read_sf(file.path(extd.dir, '2012-basswood.shp'),
                         quiet = TRUE)
chk <- pa_check_yield(raw.yield)
chk
```

---

pa_compute_vi *Compute vegetation indices from a zipped Sentinel 2 file*

Description

Compute vegetation indices from a zipped Sentinel 2 file.

Usage

```
pa_compute_vi(
  satellite.images,
  vi = c("ndvi", "ndre", "gcvi", "reci", "evi", "bsi", "other"),
  aoi = NULL,
  formula = NULL,
  check.clouds = FALSE,
  buffer.clouds = 100,
  downscale.to = NULL,
  pixel.res = c("default", "10m", "20m", "60m"),
  img.formats = c("jp2", "tif"),
  fun = function(x) mean(x, na.rm = TRUE),
  verbose = TRUE
)
```

Arguments

satellite.images	list of file paths to the Sentinel 2 zip files
vi	the vegetation index to be computed
aoi	NULL or an sf object used to crop the vegetation index raster to an area of interest
formula	an optional two-sided formula with the vegetation index name on the left side and the relationship between the bands on the right side. See example.
check.clouds	whether to check for clouds over the area of interest. If clouds are found, the function will skip cloudy images.
buffer.clouds	distance in meters around the area of interest within a cloud would be considered to interfere with the index calculation. This is useful to eliminate the effect of cloud shading from the analysis.
downscale.to	the resolution in meters to downscale the resolution of the vegetation index raster layer
pixel.res	pixel resolution used to compute the vegetation index. Can be one of 10m, 20m, 30m
img.formats	image formats to search for in the zipped file
fun	function to be applied to consolidate duplicated images
verbose	whether to display information on the progress of operations

Details

This is script that unzips the Sentinel 2 zipped file into a temporary folder, searches for the index-relevant bands, and then computes the index. If no 'aoi' is provided, the script will compute the vegetation index for the area covered by the image. The pre-specified vegetation indices are computed as follows:

$$BSI = \frac{(SWIR + RED) - (NIR + BLUE)}{(SWIR + RED) + (NIR + BLUE)}$$

$$EVI = \frac{2.5 \times (NIR - RED)}{(NIR + (6 \times RED) - (7.5 \times BLUE) - 1)}$$

$$GCVI = \frac{(NIR)}{(GREEN)} - 1$$

$$NDRE = \frac{(NIR - REDedge)}{(NIR + REDedge)}$$

$$NDVI = \frac{(NIR - RED)}{(NIR + RED)}$$

$$RECI = \frac{(NIR)}{(REDedge)} - 1$$

The user can also specify custom vegetation indices using the formula argument. The formula should be two-sided, with the left side naming the vegetation index and the right side defining the

mathematical operations used to calculate the vegetation index. The bands should be specified as B01, B02, ..., B12.

An important detail of this function is that, if there are duplicated dates, the function will consolidate the data into a single raster layer. The default behavior is to average the layers that belong to the same date. This can be changed with the 'fun' argument.

Value

an object of class veg.index and stars

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
extd.dir <- system.file("extdata", package = "pacu")
## List of zipped Sentinel files in a directory
s2a.files <- list.files(extd.dir, '\\.zip', full.names = TRUE)
area.of.interest <- sf::st_read(file.path(extd.dir, 'cobs_a_aoi.shp'))

## computing ndvi
ndvi <- pa_compute_vi(satellite.images = s2a.files,
                            vi = 'ndvi',
                            aoi = area.of.interest,
                            check.clouds = TRUE)

## computing ndre
ndre <- pa_compute_vi(satellite.images = s2a.files,
                            vi = 'ndre',
                            aoi = area.of.interest,
                            check.clouds = TRUE)

## specifying a differente vegetation index, in this case, the
## excess green index
egi <- pa_compute_vi(satellite.images = s2a.files,
                            vi = 'other',
                            formula = EGI ~ (2 * B03) - B02 - B04,
                            aoi = area.of.interest,
                            check.clouds = TRUE)
```

---

pa_download_dataspace *Download satellite products from the Copernicus Data Space Ecosystem*

Description

Download satellite products from the Copernicus Data Space Ecosystem to find satellite products

Usage

```
pa_download_dataspace(x, dir.path = NULL, aoi = NULL, verbose = TRUE)
```

Arguments

x	object of class 'dslist'
dir.path	directory path to which the files will be saved
aoi	NULL or an sf object. If an area of interest (aoi) is provided, the downloaded zip files will be cropped to the aoi. This was designed to save storage space
verbose	whether to display information on the progress of operations

Details

'pa_download_dataspace()' uses the object from 'pa_browse_dataspace()' to download the data from Copernicus Data Space. The aoi argument is optional but was designed to save storage space.

Value

a list of objects that could not be downloaded

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## Not run:
extd.dir <- system.file("extdata", package = "pacu")
area.of.interest <- sf::st_read(file.path(extd.dir, 'cobs_a_aoi.shp'), quiet = TRUE)
available.images <- pa_browse_dataspace(aoi = area.of.interest,
                                        max.cloud.cover = 10,
                                        start.date = '2023-01-01',
                                        end.date = '2023-12-31')
dwonloaded.images <- pa_download_dataspace(x = available.images)


## End(Not run)
```

---

pa_get_rgb *Retrieve an RGB image from a zipped Sentinel 2 file*

Description

Retrieve an RGB image from a zipped Sentinel 2 file

Usage

```
pa_get_rgb(
  satellite.images,
  aoi = NULL,
  pixel.res = "10m",
  check.clouds = FALSE,
  buffer.clouds = 100,
  img.formats = c("jp2", "tif"),
  rgb.bands = c("B04", "B02", "B03"),
  fun = function(x) mean(x, na.rm = TRUE),
  verbose = TRUE
)
```

Arguments

satellite.images	list of file paths to the Sentinel 2 zip files
aoi	NULL or an sf object used to crop the RGB raster to an area of interest
pixel.res	pixel resolution used to retrieve the RGB image. Can be one of 10m, 20m, 30m.
check.clouds	whether to check for clouds over the area of interest. If clouds are found, the function will skip cloudy images.
buffer.clouds	distance in meters around the area of interest within a cloud would be considered to interfere with the index calculation. This is useful to eliminate the effect of cloud shading from the analysis.
img.formats	image formats to search for in the zipped file
rgb.bands	a vector containing the order of the RGB bands
fun	function to be applied to consolidate duplicated images
verbose	whether to display information on the progress of operations

Details

This is script that unzips the Sentinel 2 zipped file into a temporary folder, searches for the RGB, and constructs a multi-band raster containing the RGB bands. If no 'aoi' is provided, the script will construct the RGB image for the area covered by the image. An important detail of this function is that, if there are duplicated dates, the function will consolidate the data into a single raster layer. The default behavior is to average the layers that belong to the same date. This can be changed with the 'fun' argument.

Value

an object of class rgb and stars

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
extd.dir <- system.file("extdata", package = "pacu")
## List of zipped Sentinel files in a directory
s2a.files <- list.files(extd.dir, '\\.zip', full.names = TRUE)
area.of.interest <- sf::st_read(file.path(extd.dir, 'cobs_a_aoi.shp'))
rgb.rast <- pa_get_rgb(satellite.images = s2a.files,
                           aoi = area.of.interest)
pa_plot(rgb.rast)
```

`pa_get_vi_stats` *Request vegetation index statistics from the Data Space Statistics API*

Description

Request vegetation index statistics from the Data Space Statistics API

Usage

```
pa_get_vi_stats(
  aoi,
  start.date,
  end.date,
  collection = c("sentinel-2-l2a"),
  vegetation.index = c("bsi", "evi", "gcvi", "ndre", "ndvi", "reci"),
  agg.time = c("P1D", "P5D", "P10D"),
  by.feature = FALSE
)
```

Arguments

`aoi`	sf object used to filter satellite products
`start.date`	beginning of the time window to filter satellite products. Date format should be '%Y-%m-%d'.
`end.date`	end of the time window to filter satellite products. Date format should be '%Y-%m-%d'.
`collection`	for now, it only supports 'sentinel2'.
`vegetation.index`	vegetation index to be requested from the Data Space
`agg.time`	aggregation time of the satellite products
`by.feature`	logical, indicating whether the statistics should be retrieved by each polygon when multiple polygons are supplied in 'aoi'

Details

'pa_get_vi_sentinel2()' will use HTTP requests to communicate with the Data Space Statistics API and request areal statistics for the specified vegetation index

Value

returns an object of class veg.index and stars

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## Not run:
extd.dir <- system.file("extdata", package = "pacu")
area.of.interest <- sf::st_read(file.path(extd.dir, 'cobs_a_aoi.shp'), quiet = TRUE)
ndvi <- pa_get_vi_stats(aoi = area.of.interest,
                        start.date = '2021-01-01',
                        end.date = '2021-12-31',
                        vegetation.index = 'ndvi')

## End(Not run)
```

---

pa_get_weather_sf *Downloads a met file using the apsimx package*

Description

This function retrieves weather data from NASA Power and the Iowa Environmental Mesonet using the apsimx package/

Usage

```
pa_get_weather_sf(
  aoi,
  source = c("none", "iem", "power"),
  start.date = "1990-01-01",
  end.date = "2021-12-31"
)
```

Arguments

aoi	a sf object
source	the weather source from which the data should be retrieved. 'iem' = Iowa Environmental Mesonet, 'power' = NASA POWER. Defaults to 'iem'.
start.date	first day to retrieve the weather data. Format should be %Y-%m-%d.
end.date	last day to retrieve the weather data. Format should be %Y-%m-%d.

Value

an object of class met

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## Not run:
extd.dir <- system.file("extdata", package = "pacu")
area.of.interest <- sf::st_read(file.path(extd.dir, 'cobs_a_aoi.shp'))
weather.met <- pa_get_weather_sf(aoi = area.of.interest,
                                 start.date = '1990-01-01',
                                 end.date = '2020-12-31',
                                 source = 'power')

## End(Not run)
```

---

pa_initialize_dataspace

Register the Data Space credentials to the R environment

Description

Register the Data Space credentials to the R environment

Usage

```
pa_initialize_dataspace(username, password, verbose = TRUE)
```

Arguments

username	username used to authenticate the HTTP request
password	password used to authenticate the HTTP request
verbose	whether to print information about this operation

Details

'pa_initialize_dataspace()' registers the username and password to the machine's R environment. All the other functions that rely on authentication will search for the username and password in the R environment. Do not share your R environment with others, as they will be able to read your username and password. You can register at https://dataspace.copernicus.eu/.

Value

No return value, called for side effects

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## Not run:
pa_initialize_dataspace('my-username', 'my-password')

## End(Not run)
```

---

pa_initialize_oauth *Register the Oauth2.0 credentials to the R environment*

Description

Register the Oauth2.0 credentials to the R environment

Usage

```
pa_initialize_oauth(client_id, client_secret)
```

Arguments

client_id	client id used to authenticate the HTTP request
client_secret	client secret used to authenticate the HTTP request

Details

initialize_oauth registers the client id and secret to the machine's R environment All the other functions that rely on authentication will search for the clients id ans secret in the R environment. Do not share your R environment with others, as they will be able to read your client id and secret. You can register at https://dataspace.copernicus.eu/news. Please see this section for how to create your Oauth2.0 client: https://documentation.dataspace.copernicus.eu/APIs/SentinelHub/Overview/Authentication.html.

Value

No return value, called for side effects

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## Not run:
pa_initialize_oauth('my-client-id', 'my-client-secret')

## End(Not run)
```

`pa_make_vehicle_polygons` *Make vehicular polygons for yield monitor observations*

Description

Make vehicular polygons for yield monitor observations

Usage

```
pa_make_vehicle_polygons(
  points,
  swath,
  distance,
  angle = NULL,
  cores = 1L,
  verbose = FALSE
)
```

Arguments

`points`	a vector of points
`swath`	a vector containing the swath of the vehicle in meters
`distance`	a vector containing the distance traveled by the vehicle in meters
`angle`	a vector containing the angle of the vehicle's trajectory. If not supplied, the function will attempt to estimate the trajectory angle using the geographical information contained in the georeferenced points/
`cores`	the number of cores used in the operation
`verbose`	whether to print operation details

Details

This function will create vehicular polygons based on the distance between points, angle of the vehicle's trajectory, and swath.

Value

an sf object

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## for examples, see vignette pacu
```

---

pa_plot *Create a plot from a pacu object*

Description

Create a plot from a pacu object

Usage

```
pa_plot(x, ...)

## S3 method for class 'yield'
pa_plot(
  x,
  ...,
  plot.type = c("yieldmap", "variogram", "steps"),
  palette = "Temps",
  main = "",
  plot.var = NULL,
  interactive = FALSE,
  border.col = "black",
  style = c("quantile", "pretty", "equal"),
  scale = 1,
  nbreaks = 5,
  breaks = NULL,
  frame = TRUE,
  extent = sf::st_bbox(x[["yield"]]),
  legend.outside = FALSE,
  ask = TRUE
)

## S3 method for class 'trial'
pa_plot(
  x,
  ...,
  plot.type = c("trial"),
  palette = "Temps",
  main = "",
```

```
  plot.var = NULL,
  interactive = FALSE,
  border.col = "black",
  style = c("quantile", "pretty", "equal"),
  scale = 1,
  nbreaks = 5,
  breaks = NULL,
  frame = TRUE,
  extent = sf::st_bbox(x[["trial"]]),
  legend.outside = FALSE
)

## S3 method for class 'veg.index'
pa_plot(
  x,
  ...,
  palette = ifelse(plot.type == "timeseries", "Dark 2", "Temps"),
  plot.type = c("spatial", "timeseries"),
  main = "",
  plot.var = NULL,
  by = "year",
  xlab = NULL,
  ylab = NULL,
  style = c("quantile", "pretty", "equal"),
  nbreaks = 5,
  border.col = "black",
  frame = TRUE,
  legend.outside = FALSE,
  legend.title = NULL,
  pch = 16
)

## S3 method for class 'rgb'
pa_plot(
  x,
  ...,
  main = "",
  interactive = FALSE,
  saturation = 1,
  alpha = 1,
  interpolate = FALSE
)

## S3 method for class 'met'
pa_plot(
  x,
  ...,
  plot.type = c("climate_normals", "monthly_distributions"),
```

```
  unit.system = c("international", "standard"),
  start = 1,
  end = 365,
  months = 1:12,
  vars = c("maxt", "mint", "crain", "cradn"),
  tgt.year = "last"
)
```

Arguments

x	object to be plotted
...	additional arguments. None used currently.
plot.type	type of plot to be produced Defaults to trial.
palette	a string representing a color palette from hcl.pals. Defaults to 'Temps'.
main	a main title for the plot
plot.var	the name of the column to be plotted. Defaults to 'yield'
interactive	logical. Whether to produce interactive plots.
border.col	color of the border for the polygons plotted in the yield map
style	style applied to the colors
scale	a numerical value indicating the magnification of the graph. A value of 1 produces a plot using the default magnification. Greater values will produce zoomed in plots.
nbreaks	numerical value indicating the number of breaks for the color scale.
breaks	a vector indicating numerical breaks for the color scale.
frame	logical. Whether to draw the frame around the plotting area.
extent	a bbox object indicating the geographical area to be plotted
legend.outside	logical. Whether to place the legend outside of the graph.
ask	whether to ask for user before starting a new page of output. If FALSE, plots are arranged using wrap_plots
by	a string or vector of strings used to group the data when plotting. Defaults to 'year'
xlab	a string used as label for x axis
ylab	a string used as label for y axis
legend.title	a string used as title for the legend
pch	an integer indicating which shape to use for points
saturation	numeric. Controls the image saturation. 0 maps to grayscale. 1 maps to the default value. See tm_rgb for details.
alpha	numeric between 0 and 1. See tm_rgb for details.
interpolate	logical. Whether the raster image should be interpolated. See tm_rgb for details.
unit.system	unit system to be used: international (metric) or stanrdard (imperial)
start	day of the year to start computing the climate normals. Defaults to 1.

end	day of the year to finish computing the climate normals. Defaults to 365.
months	a numerical vector indicating which months to produce a plot for in the case of monthly distribution plots. Defaults to 1:12.
vars	which variables to include in the summary plot
tgt.year	which year to focus and compare to the historical mean. Defaults to the last year in the data set.

Value

No return value, called for side effects

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## Not run:
## for examples, please see the pacu vignette

## End(Not run)
```

---

pa_trial *EXPERIMENTAL FUNCTION - Create an interpolated trial object from as-applied data*

Description

EXPERIMENTAL FUNCTION - Create an interpolated trial object from as-applied data

Usage

```
pa_trial(
  input,
  data.columns = NULL,
  data.units = NULL,
  grid = NULL,
  algorithm = c("none", "simple", "ritas"),
  var.label = "as.applied",
  boundary = NULL,
  smooth.method = c("none", "krige", "idw"),
  formula = NULL,
  out.units = NULL,
  conversion.factor = 1,
  na.to.zero = ifelse(algorithm == "ritas", TRUE, FALSE),
  cores = 1L,
```

```
  steps = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

input	an sf object containing the as applied trial
data.columns	When algorithm is 'simple', this argument should be a vector of length one indicating which column contains the 'trial or as-applied' data. When algorithm is 'ritas', an optional named vector with the column names for the variables 'trial, angle, swath, distance'. If a an unnamed vector is supplied, the vector is assumed to be in this order. The default is NULL, in which case the function attempts to guess the columns by using a dictionary of possible guesses. The column indicating the 'trial' information is not guessed, as there are too many possible options (seeds, fertilizer, soil amendments, etc).
data.units	When algorithm is 'simple', should be a vector of length one indicating the units of the trial column and the moisture column. Common values would be 'c('kg N/ha', 'seeds/acre')'. When algorithm is 'ritas', an optional named vector with strings representing units for the variables 'trial, angle, swath, distance'. If a an unnamed vector is supplied, the vector is assumed to be in this order. A typical value for this argument would be 'c(trial = 'kg N/ha', angle = 'degreeN', width = 'ft', distance = 'ft')'. Please see valid_udunits for help with specifying units. The default is NULL, in which case the function attempts to guess the units according to the values of the variable. The units of 'trial' are not guessed, as there are too many possible options (seeds, fertilizer, soil amendments, etc).
grid	an sf object containing the prediction grid. If the user is processing as-applied data coming from a research trial (i.e. follows a trial design), the user can pass the sf object containing the trial design information to this argument.
algorithm	algorithm used to generate the yield object.
var.label	optional string to name the final product. Defaults to 'as.applied'.
boundary	optional sf object representing the field's outer boundary. If it not supplied, the function attempts to generate a boundary from the observed points.
smooth.method	the smoothing method to be used. If 'none', no smoothing will be conducted. If 'idw', inverse distance weighted interpolation will be conducted. If 'krige', kriging will be conducted.
formula	formula defining the relationship between the dependent and independent variables. If the dependent variable is a linear function of the coordinates, the formula can be 'z ~ X + Y'. If the dependent variable is modeled only as a function of the mean spatial process, the formula can be 'z ~ 1'. If no formula is supplied, it defaults to 'z ~ 1'.
out.units	units of the output after being multiplied by the conversion factor. If conversion.factor is 1 and out.units is NULL, out.units will default to the units of the trial input.
conversion.factor	a conversion factor by which the input trial data will be multiplied. This is useful for cases in which the user wants the output in different units from the input. A

	trivial example is a fertilizer trial in which the fertilizer contained in the input is only 50 In this case, conversion.factor should be set to 0.5.
na.to.zero	whether areas in which the trial applicator has not covered should be assigned a value of zero. This is only effective when 'algorithm' is 'ritas'. Defaults to TRUE when 'algorithm' is 'ritas'.
cores	the number of cores used in the operation
steps	whether to return the intermediate steps of the trial processing algorithm
verbose	whether to print function progress. 'FALSE or 0' will suppress details. 'TRUE or 1' will print a progress bar. '>1' will print step by step messages.
...	additional arguments to be passed krige and idw

Details

This function will follow the steps in the selected algorithm to produce a map of as-applied trial from the raw data.

Value

an object of class trial

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## Not run:
## tbd

## End(Not run)
```

---

pa_yield *Create an interpolated yield object from raw data*

Description

Create an interpolated yield object from raw data

Usage

```
pa_yield(
  input,
  data.columns = NULL,
  data.units = NULL,
  grid = NULL,
  algorithm = c("none", "simple", "ritas"),
```

```
  formula = NULL,
  overlap.threshold = 0.5,
  var.label = "yield",
  boundary = NULL,
  clean = FALSE,
  clean.sd = 3,
  clean.edge.distance = 0,
  smooth.method = c("none", "krige", "idw"),
  fun = c("none", "log"),
  lbs.per.bushel = NULL,
  moisture.adj = NULL,
  lag.adj = 0,
  unit.system = c("none", "metric", "standard"),
  remove.crossed.polygons = FALSE,
  steps = FALSE,
  cores = 1L,
  verbose = TRUE,
  ...
)
```

Arguments

input	an sf object containing the raw yield monitor data
data.columns	When algorithm is 'simple', this argument should be a vector of length 2 or 3 (depends on whether the user wants to adjust for time lag) indicating which column contains the yield data , a column containing moisture information, and a column indicating the time between readings. When algorithm is 'ritas', an optional named vector with the column names for the variables 'mass, flow, moisture, interval, angle, swath, distance'. If a an unnamed vector is supplied, the vector is assumed to be in this order. The default is NULL, in which case the function attempts to guess the columns by using a dictionary of possible guesses.
data.units	When algorithm is 'simple', should be a vector of length two, indicating the units of the yield column and the moisture column. Common values would be 'c('bu/ac', '%')'. When algorithm is 'ritas', an optional named vector with strings representing units for the variables 'mass, flow, moisture, interval, angle, swath, distance'. If a an unnamed vector is supplied, the vector is assumed to be in this order. A typical value for this argument would be 'c(flow = 'lb/s', moisture = '%', interval = 's', angle = 'degreeN', width = 'ft', distance = 'ft')'. Please see valid_udunits for help with specifying units. The default is NULL, in which case the function attempts to guess the units according to the values of the variable.
grid	an sf or pa_trial object containing the prediction grid. If the user is processing yield data coming from a research trial (i.e. follows a trial design), the user can pass the sf object containing the trial design information to this argument. If the argument 'formula' contains any predictions, the predictor should be included in the sf object supplied to this argument. polygons for which the predictions generated.

`algorithm` algorithm used to generate the yield object.

`formula` formula defining the relationship between the dependent and independent variables. If the dependent variable is a linear function of the coordinates, the formula can be 'z ~ X + Y'. If the dependent variable is modeled only as a function of the mean spatial process, the formula can be 'z ~ 1'. If no formula is supplied, it defaults to 'z ~ 1'.

`overlap.threshold` a fraction threshold to remove observations when there is overlap between the vehicular polygons. A value of 0 does not remove any observations. A value of 1 removes all observations that overlap even minimally with neighboring observations.

`var.label` optional string to name the final product. Defaults to 'yield'.

`boundary` optional sf object representing the field's outer boundary. If it not supplied, the function attempts to generate a boundary from the observed points.

`clean` whether to clean the raw data based on distance from the field edge and global standard deviation.

`clean.sd` standard deviation above which the cleaning step will remove data. Defaults to 3.

`clean.edge.distance` distance (m) from the field edge above which the cleaning step will remove data. Defaults to 0.

`smooth.method` the smoothing method to be used. If 'none', no smoothing will be conducted. If 'idw', inverse distance weighted interpolation will be conducted. If 'krige', kriging will be conducted.

`fun` a function used to transform the data. Currently, the option are 'none' and 'log'. If none, data operations are carried out in the data scale. If log, the function will usekrigeTg to perform kriging in the log scale. For now, only relevant when 'method' is krige. the log scale and back transform predictions to the data scale. When TRUE, 'fomula' should be 'z ~ 1'.

`lbs.per.bushel` a numeric value representing the number of pounds in a bushel (e.g., 60 for soybean and 56 for corn). This argument can be ommitted when the input and output units are in the metric system. It is necessary otherwise.

`moisture.adj` an optional numeric value to set the moisture value to which the yield map predictions should be adjusted (e.g., 15.5 for corn, and 13.0 for soybean). If NULL, the function will adjust the moisture to the average moisture of the field.

`lag.adj` an optional numeric value used to account for the time lag between the crop being cut by the combine and the time at which the combine records a data point.

`unit.system` a string representing the unit system to be used in the function output. If 'standard', the function output will be in bushel/acre. Alternatively, if 'metric', outputs will be in metric tonnes/hectare.

`remove.crossed.polygons` logical, whether to remove vehicle polygons that crossed different experimental units of the grid. This is intented to prevent from diluting the treatment effects. When this argument is TRUE, the argument 'grid' must be supplied.

steps	whether to return the intermediate steps of the yield processing algorithm
cores	the number of cores used in the operation
verbose	whether to print function progress. 'FALSE or 0' will suppress details. 'TRUE or 1' will print a progress bar. '>1' will print step by step messages.
...	additional arguments to be passed krige and idw

Details

This function will follow the steps in the selected algorithm to produce a yield map from the raw data.

Value

an object of class yield

Author(s)

Caio dos Santos and Fernando Miguez

Examples

```
## Not run:
extd.dir <- system.file("extdata", package = "pacu")
raw.yield <- sf::read_sf(file.path(extd.dir, '2012-basswood.shp'),
                         quiet = TRUE)
## the simple algorithm
pa_yield(input = raw.yield,
algorithm = 'simple',
unit.system = 'metric',
lbs.per.bushel = 56) ## 56 lb/bushel of maize

## the ritas algorithm
pa_yield(input = raw.yield,
algorithm = 'ritas',
unit.system = 'metric',
lbs.per.bushel = 56)

## End(Not run)
```

---

print *Print a pacu object*

Description

These functions print meaningful information from pacu objects.

Usage

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

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

## S3 method for class 'check.yield'
print(x, ...)

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

Arguments

x	object to be printed
...	additional arguments. None used currently.

Value

No return value, called for side effects

---

summary *Produce result summaries of the various pacu objects*

Description

Produce summaries for the different pacu objects

Usage

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

## S3 method for class 'yield'
summary(object, ..., by = NULL)

## S3 method for class 'veg.index'
summary(object, ..., by, fun)
```

Arguments

object	object to be summarized
...	additional arguments. None used currently.
by	sf or stars object containing the geometries within which the vegetation index values should be summarized
fun	a function to be applied when summarizing the vegetation index data. For example, mean, median, max, min.

Value

when object is of class dslist, no return value. Called for side effects.

when object is of class yield, returns an object of class data.frame

when object is of class veg.index, returns an object of class stars

Index