Package 'scrubr'

April 7, 2020

Type Package

Title Clean Biological Occurrence Records

Description Clean biological occurrence records. Includes functionality for cleaning based on various aspects of spatial coordinates, unlikely values due to political 'centroids', coordinates based on where collections of specimens are held, and more.

Version 0.3.2

License MIT + file LICENSE

URL https://github.com/ropensci/scrubr (devel)
https://docs.ropensci.org/scrubr (docs)

BugReports https://github.com/ropensci/scrubr/issues

LazyData TRUE

VignetteBuilder knitr

Encoding UTF-8

Language en-US

Imports methods, stats, utils, Matrix, magrittr, qlcMatrix, data.table, fastmatch, lazyeval, crul, jsonlite, tibble, hoardr, curl

Suggests testthat, knitr, rmarkdown, rgbif, sf, mapview, rworldmap, maps

RoxygenNote 7.1.0

X-schema.org-applicationCategory Biodiversity

X-schema.org-keywords specimens, occurrences, data, data-cleaning

X-schema.org-isPartOf https://ropensci.org

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (<https://orcid.org/0000-0003-1444-9135>)

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2020-04-07 17:10:02 UTC

R topics documented:

`scrubr-package` *scrubr*

Description

Clean biological occurrence data

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

`coords` *Coordinate based cleaning*

Description

Coordinate based cleaning

Usage

```
coord_incomplete(x, lat = NULL, lon = NULL, drop = TRUE)

coord_imprecise(x, which = "both", lat = NULL, lon = NULL, drop = TRUE)

coord_impossible(x, lat = NULL, lon = NULL, drop = TRUE)

coord_unlikely(x, lat = NULL, lon = NULL, drop = TRUE)

coord_within(
  x,
  field = NULL,
  country = NULL,
```

```
  lat = NULL,
  lon = NULL,
  drop = TRUE
)

coord_pol_centroids(x, lat = NULL, lon = NULL, drop = TRUE)

coord_uncertain(
  x,
  coorduncertainityLimit = 30000,
  drop = TRUE,
  ignore.na = FALSE
)
```

Arguments

x	(data.frame) A data.frame
lat, lon	(character) Latitude and longitude column to use. See Details.
drop	(logical) Drop bad data points or not. Either way, we parse out bad data points as an attribute you can access. Default: TRUE
which	(character) one of "has_dec", "no_zeros", or "both" (default)
field	(character) Name of field in input data.frame x with country names
country	(character) A single country name
coorduncertainityLimit	(numeric) numeric threshold for the coordinateUncertainityInMeters variable. Default: 30000
ignore.na	(logical) To consider NA values as a bad point or not. Default: FALSE

Details

Explanation of the functions:

- coord_impossible - Impossible coordinates
- coord_incomplete - Incomplete coordinates
- coord_imprecise - Imprecise coordinates
- coord_pol_centroids - Points at political centroids
- coord_unlikely - Unlikely coordinates
- coord_within - Filter points within user input political boundaries
- coord_uncertain - Uncertain occurrances of measured through coordinateUncertaintyInMeters default limit= 30000

If either lat or lon (or both) given, we assign the given column name to be standardized names of "latitude", and "longitude". If not given, we attempt to guess what the lat and lon column names are and assign the same standardized names. Assigning the same standardized names makes downstream processing easier so that we're dealing with consistent column names. On returning the data, we return the original names.

For coord_within, we use countriesLow dataset from the **rworldmap** package to get country borders.

Value

Returns a data.frame, with attributes

coord_pol_centroids

Right now, this function only deals with city centroids, using the maps::world.cities dataset of more than 40,000 cities. We'll work on adding country centroids, and perhaps others (e.g., counties, states, provinces, parks, etc.).

Examples

```
df <- sample_data_1

# Remove impossible coordinates
NROW(df)
df[1, "latitude"] <- 170
df <- dframe(df) %>% coord_impossible()
NROW(df)
attr(df, "coord_impossible")

# Remove incomplete cases
NROW(df)
df_inc <- dframe(df) %>% coord_incomplete()
NROW(df_inc)
attr(df_inc, "coord_incomplete")


# Remove imprecise cases
df <- sample_data_5
NROW(df)
## remove records that don't have decimals at all
df_imp <- dframe(df) %>% coord_imprecise(which = "has_dec")
NROW(df_imp)
attr(df_imp, "coord_imprecise")
## remove records that have all zeros
df_imp <- dframe(df) %>% coord_imprecise(which = "no_zeros")
NROW(df_imp)
attr(df_imp, "coord_imprecise")
## remove both records that don't have decimals at all and those that
## have all zeros
df_imp <- dframe(df) %>% coord_imprecise(which = "both")
NROW(df_imp)
attr(df_imp, "coord_imprecise")


# Remove unlikely points
NROW(df)
df_unlikely <- dframe(df) %>% coord_unlikely()
NROW(df_unlikely)
attr(df_unlikely, "coord_unlikely")

# Remove points not within correct political borders
```

```
if (requireNamespace("rgbif", quietly = TRUE) && interactive()) {
   library("rgbif")
   wkt <- 'POLYGON((30.1 10.1,40 40,20 40,10 20,30.1 10.1))'
   res <- rgbif::occ_data(geometry = wkt, limit=300)$data
} else {
   res <- sample_data_4
}

## By specific country name
NROW(res)
df_within <- dframe(res) %>% coord_within(country = "Israel")
NROW(df_within)
attr(df_within, "coord_within")

## By a field in your data - makes sure your points occur in one
## of those countries
NROW(res)
df_within <- dframe(res) %>% coord_within(field = "country")
NROW(df_within)
head(df_within)
attr(df_within, "coord_within")

# Remove those very near political centroids
## not ready yet
# NROW(df)
# df_polcent <- dframe(df) %>% coord_pol_centroids()
# NROW(df_polcent)
# attr(df_polcent, "coord_polcent")

## lat/long column names can vary
df <- sample_data_1
head(df)
names(df)[2:3] <- c('mylon', 'mylat')
head(df)
df[1, "mylat"] <- 170
dframe(df) %>% coord_impossible(lat = "mylat", lon = "mylon")

df <- sample_data_6

# Remove uncertain occurances

NROW(df)
df1<-df %>% coord_uncertain()
NROW(df1)
attr(df, "coord_uncertain")

NROW(df)
df2<-df %>% coord_uncertain(coorduncertainityLimit = 20000)
NROW(df2)

NROW(df)
df3<-df %>% coord_uncertain(coorduncertainityLimit = 20000,ignore.na=TRUE)
NROW(df3)
```

`date` *Date based cleaning*

Description

Date based cleaning

Usage

```
date_standardize(x, format = "%Y-%m-%d", date_column = "date", ...)

date_missing(x, date_column = "date", drop = TRUE, ...)

date_create(x, ...)

date_create_(x, ..., .dots, format = "%Y-%m-%d", date_column = "date")
```

Arguments

`x`	(data.frame) A data.frame
`format`	(character) Date format. See `as.Date()`
`date_column`	(character) Name of the date column
`...`	Comma separated list of unquoted variable names
`drop`	(logical) Drop bad data points or not. Either way, we parse out bade data points as an attribute you can access. Default: `TRUE`
`.dots`	Used to work around non-standard evaluation

Details

- date_standardize - Converts dates to a specific format
- date_missing - Drops records that do not have dates, either via being NA or being a zero length character string
- date_create - Create a date field from

Value

Returns a data.frame, with attributes

Examples

```
df <- sample_data_1
# Standardize dates
dframe(df) %>% date_standardize()
dframe(df) %>% date_standardize("%Y/%m/%d")
dframe(df) %>% date_standardize("%d%b%Y")
dframe(df) %>% date_standardize("%Y")
```

```
dframe(df) %>% date_standardize("%y")

# drop records without dates
NROW(df)
NROW(dframe(df) %>% date_missing())

# Create date field from other fields
df <- sample_data_2
## NSE
dframe(df) %>% date_create(year, month, day)
## SE
date_create_(dframe(df), "year", "month", "day")
```

---

dedup *Deduplicate records*

Description

Deduplicate records

Usage

```
dedup(x, how = "one", tolerance = 0.9)
```

Arguments

x	(data.frame) A data.frame, tibble, or data.table
how	(character) How to deal with duplicates. The default of "one" keeps one record of each group of duplicates, and drops the others, putting them into the dups attribute. "all" drops all duplicates, in case e.g., you don't want to deal with any records that are duplicated, as e.g., it may be hard to tell which one to remove.
tolerance	(numeric) Score (0 to 1) at which to determine a match. You'll want to inspect outputs closely to tweak this value based on your data, as results can vary.

Value

Returns a data.frame, optionally with attributes

Examples

```
df <- sample_data_1
smalldf <- df[1:20, ]
smalldf <- rbind(smalldf, smalldf[10,])
smalldf[21, "key"] <- 1088954555
NROW(smalldf)
dp <- dframe(smalldf) %>% dedup()
NROW(dp)
attr(dp, "dups")
```

```
# Another example - more than one set of duplicates
df <- sample_data_1
twodups <- df[1:10, ]
twodups <- rbind(twodups, twodups[c(9, 10), ])
rownames(twodups) <- NULL
NROW(twodups)
dp <- dframe(twodups) %>% dedup()
NROW(dp)
attr(dp, "dups")
```

---

dframe *Compact data.frame*

Description

Compact data.frame

Usage

```
dframe(x)
```

Arguments

x	Input data.frame

Examples

```
dframe(sample_data_1)
dframe(mtcars)
dframe(iris)
```

---

eco_region *Filter points within ecoregions*

Description

Filter points within ecoregions

Usage

```
eco_region(x, dataset = "meow", region, lat = NULL, lon = NULL, drop = TRUE)

regions_meow()

regions_fao()
```

Arguments

x	(data.frame) A data.frame
dataset	(character) the dataset to use. one of: "meow" (Marine Ecoregions of the World), "fao" (). See Details.
region	(character) the region name. has the form a:b where a is a variable name (column in the sf object) and b is the value you want to filter to within that variable. See Details.
lat, lon	(character) name of the latitude and longitude column to use
drop	(logical) Drop bad data points or not. Either way, we parse out bad data points as an attribute you can access. Default: TRUE #param ignore.na (logical) To consider NA values as a bad point or not. Default: FALSE

Details

see scrubr_cache for managing the cache of data

Value

Returns a data.frame, with attributes

dataset options

- Marine Ecoregions of the World (meow):
  - data from: https://opendata.arcgis.com/datasets/ed2be4cf8b7a451f84fd093c2e7660e3_0.geojson
- Food and Agriculture Organization (fao):
  - data from: http://www.fao.org/geonetwork/srv/en/main.home?uuid=ac02a460-da52-11dc-9d70-0017f293bd28

region options

- within meow:
  - ECOREGION: many options, see regions_meow()
  - ECO_CODE: many options, see regions_meow()
  - and you can use others as well; run regions_meow() to get the data used within eco_region() and see what variables/columns can be used
- within fao:
  - OCEAN: Atlantic, Pacific, Indian, Arctic
  - SUBOCEAN: 1 through 11 (inclusive)
  - F_AREA (fishing area): 18, 21, 27, 31, 34, 37, 41, 47, 48, 51, 57, 58, 61, 67, 71, 77, 81, 87, 88
  - and you can use others as well; run regions_fao() to get the data used within eco_region() and see what variables/columns can be used

Examples

```
## Not run:
if (requireNamespace("mapview") && requireNamespace("sf") && interactive()) {
## Marine Ecoregions of the World
wkt <- 'POLYGON((-119.8 12.2, -105.1 11.5, -106.1 21.6, -119.8 20.9, -119.8 12.2))'
res <- rgbif::occ_data(geometry = wkt, limit=300)$data
res2 <- sf::st_as_sf(res, coords = c("decimalLongitude", "decimalLatitude"))
res2 <- sf::st_set_crs(res2, 4326)
mapview::mapview(res2)
tmp <- eco_region(dframe(res), dataset = "meow",
   region = "ECOREGION:Mexican Tropical Pacific")
tmp2 <- sf::st_as_sf(tmp, coords = c("decimalLongitude", "decimalLatitude"))
tmp2 <- sf::st_set_crs(tmp2, 4326)
mapview::mapview(tmp2)

## FAO
wkt <- 'POLYGON((72.2 38.5,-173.6 38.5,-173.6 -41.5,72.2 -41.5,72.2 38.5))'
manta_ray <- rgbif::name_backbone("Mobula alfredi")$usageKey
res <- rgbif::occ_data(manta_ray, geometry = wkt, limit=300, hasCoordinate = TRUE)
dat <- sf::st_as_sf(res$data, coords = c("decimalLongitude", "decimalLatitude"))
dat <- sf::st_set_crs(dat, 4326)
mapview::mapview(dat)
tmp <- eco_region(dframe(res$data), dataset = "fao", region = "OCEAN:Indian")
tmp <- tmp[!is.na(tmp$decimalLongitude), ]
tmp2 <- sf::st_as_sf(tmp, coords = c("decimalLongitude", "decimalLatitude"))
tmp2 <- sf::st_set_crs(tmp2, 4326)
mapview::mapview(tmp2)
}
## End(Not run)
```

`fix_names` *Change taxonomic names to be the same for each taxon*

Description

That is, this function attempts to take all the names that are synonyms, for whatever reason (e.g., some names have authorities on them), and collapses them to the same string - making data easier to deal with for making maps, etc. OR - you can think of this as a tool for

Usage

```
fix_names(x, how = "shortest", replace = NULL)
```

Arguments

`x`	(data.frame) A data.frame. the target taxonomic name column should be 'name'
`how`	One of a few different methods: • shortest - Takes the shortest name string that is likely to be the prettiest to display name, and replaces alll names with that one, better for maps, etc.

- supplied - If this method, supply a vector of names to replace the names with.

`replace` A data.frame of names to replace names in the occurrence data.frames with. Only used if how="supplied". The data.frame should have two columns: the first is the names to match in the input `x` data.frame, and the second column is the name to replace with. The column names don't matter.

Value

a data.frame

Examples

```
## Not run:
df <- sample_data_7

# method: shortest
fix_names(df, how="shortest")$name

# method: supplied
(replace_df <- data.frame(
 one = unique(df$name),
 two = c('P. contorta', 'P.c. var. contorta',
         'P.c. subsp bolanderi', 'P.c. var. murrayana'),
 stringsAsFactors = FALSE))
fix_names(df, how="supplied", replace = replace_df)$name

## End(Not run)
```

`scrubr_datasets` *scrubr datasets*

Description

- sample_data_1
- sample_data_2
- sample_data_3
- sample_data_4
- sample_data_6

taxonomy *Taxonomy based cleaning*

Description

Taxonomy based cleaning

Usage

```
tax_no_epithet(x, name = NULL, drop = TRUE)
```

Arguments

x	(data.frame) A data.frame1
name	(character) Taxonomic name field Optional. See Details.
drop	(logical) Drop bad data points or not. Either way, we parse out bade data points as an attribute you can access. Default: TRUE

Value

Returns a data.frame, with attributes

Examples

```
if (requireNamespace("rgbif", quietly = TRUE) && interactive()) {
   library("rgbif")
   res <- rgbif::occ_data(limit = 200)$data
} else {
   res <- sample_data_3
}

# Remove records where names don't have genus + epithet
## so removes those with only genus and those with no name (NA or NULL)
NROW(res)
df <- dframe(res) %>% tax_no_epithet(name = "name")
NROW(df)
attr(df, "name_var")
attr(df, "tax_no_epithet")
```

Index