Package 'scrubr'

August 29, 2016

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 0.1.1

License MIT + file LICENSE

URL https://github.com/ropenscilabs/scrubr

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

LazyData TRUE

VignetteBuilder knitr

Imports methods, stats, utils, Matrix, magrittr, qlcMatrix, lazyeval

Suggests testthat, knitr, rgbif, sp, rworldmap, maps

RoxygenNote 5.0.1

NeedsCompilation no

Author Scott Chamberlain [aut, cre]

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2016-03-08 00:23:05

R topics documented:

scrubr-package *scrubr - Clean biological occurrence data*

Description

scrubr - 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_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)
```

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 bade data points as an attribute you can access. Default: TRUE
field	(character) Name of filed in input data.frame x with country names
country	(character) A single country name

Details

Explanation of the functions:

- coord_impossible - Impossible coordinates
- coord_incomplete - Incomplete coordinaes
- coord_pol_centroids - Points at political centroids
- coord_unlikely - Unlikely coordinates
- coord_within - Check if points are within user input political boundaries

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 `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 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)) {
   library("rgbif")
   wkt <- 'POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))'
   res <- rgbif::occ_data(geometry = wkt, limit=100)$data
} else {
   res <- sample_data_4
}

## By specific country name
NROW(res)
df_within <- dframe(res) %>% coord_within(country = "Egypt")
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)
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")
```

`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
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)
```

scrubr_datasets *scrubr datasets*

Description

- sample_data_1
- sample_data_2
- sample_data_3
- sample_data_4

taxonomy *Taxonomy based cleaning*

Description

Taxonomy based cleaning

Usage

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

Arguments

x	(data.frame) A data.frame
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)) {
   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