Package: certegis (via r-universe)

October 15, 2024

Title A Certe R Package for Geographic Information Science

Version 1.3.9

Description A Certe R package for geographic information science (GIS), using the 'sf' package and Dutch reference data. This package is part of the 'certedata' universe.

URL https://certe-medical-epidemiology.github.io/certegis, https://github.com/certe-medical-epidemiology/certegis

Depends R (>= 4.1.0)

Imports dplyr (>= 1.0.5), yaml (>= 2.2.0)

Suggests certeplot2, httr (>= 1.4.0), jsonlite (>= 1.7.1), sf (>= 0.9.5), testthat (>= 2.0.0)

License GPL-2

Encoding UTF-8

LazyData true

LazyDataCompression bzip2

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Config/testthat/edition 2

Repository https://certe-medical-epidemiology.r-universe.dev

RemoteUrl https://github.com/certe-medical-epidemiology/certegis

RemoteRef HEAD

RemoteSha 4b72533c5746ff2af75126387146c873c80984dd

Contents

`cases_within_radius` *Check Cases Within Radius*

Description

Based on the postcodes4_afstanden data set, this function determines the specified minimum number of cases within a certain radius.

Usage

```
cases_within_radius(
  data,
  radius_km = 10,
  minimum_cases = 10,
  column_count = NULL,
  ...
)
```

Arguments

`data`	data set containing a column 'postcode'
`radius_km`	radius in kilometres from each zip code. The search *diameter* is twice this number (since zip codes e.g. to the west and to the east are searched).
`minimum_cases`	minimum number of cases to search for
`column_count`	column name in `data` with the number of case counts, defaults to the first column with numeric values
`...`	ignored, allows for future extensions

Value

This function adds two columns (`"cases_within_radius" <dbl>` and `"minimum_met" <lgl>`) to the input data.

Examples

```
library(dplyr, warn.conflicts = FALSE)

postcodes_friesland <- geo_postcodes4 |>
  filter_geolocation(provincie == "Friesland") |>
  pull(postcode)

# example with Norovirus cases:
```

```
noro <- data.frame(postcode = postcodes_friesland,
                   n = floor(runif(length(postcodes_friesland),
                                   min = 0, max = 3)))
head(noro)

radial_check <- cases_within_radius(noro, radius_km = 10, minimum_cases = 10)
head(radial_check)


# dplyr group support:
mdro <- data.frame(type = rep(c("ESBL", "MRSA", "VRE"), 20),
                   pc4 = postcodes_friesland[1:20],
                   n = floor(runif(60, min = 0, max = 3)))
mdro |>
  group_by(type) |>
  cases_within_radius()


# plotting support:
if (require("certeplot2")) {

  radial_check |>
    add_map() |>
    filter_geolocation(provincie == "Friesland") |>
    plot2(category = cases_within_radius,
          category.title = "Cases",
          datalabels = FALSE,
          colour_fill = "viridis")

}
```

`cbs_geodata` *Data Sets with Geometries of Dutch Provinces, Municipalities and Zip Codes*

Description

Data Sets with Geometries of Dutch Provinces, Municipalities and Zip Codes

Usage

```
geo_gemeenten

geo_ggdregios

geo_nuts3

geo_postcodes2

geo_postcodes3
```

`geo_postcodes4`

`geo_postcodes6`

`geo_provincies`

Format

An object of class `sf` (inherits from `data.frame`) with 345 rows and 4 columns.

An object of class `sf` (inherits from `data.frame`) with 25 rows and 4 columns.

An object of class `sf` (inherits from `data.frame`) with 40 rows and 4 columns.

An object of class `sf` (inherits from `data.frame`) with 90 rows and 4 columns.

An object of class `sf` (inherits from `data.frame`) with 798 rows and 4 columns.

An object of class `sf` (inherits from `data.frame`) with 4068 rows and 4 columns.

An object of class `sf` (inherits from `data.frame`) with 58481 rows and 4 columns.

An object of class `sf` (inherits from `data.frame`) with 12 rows and 4 columns.

Details

These data.frames are of additional class `sf` and contain 3 variables:

- `...`
  name of the area, these are: geo_gemeenten$gemeente , geo_ggdregios$ggdregio , geo_nuts3$nuts3 , geo_postcodes2$postcode , geo_postcodes3$postcode , geo_postcodes4$postcode , geo_postcodes6$postcode , geo_provincies$provincie
- `inwoners`
  number of inhabitants in the area
- `oppervlakte_km2`
  area in square kilometres
- `geometry`
  multipolygonal object of the area

All data sets have the coordinate reference system (CRS) set to EPSG:28992 ('RD New'), following the sphere of Earth. They can be flattened to e.g. EPSG:4326 ('WGS 84') using `st_transform()`.

See the repository file to update these data sets.

NOTE: all data sets contains all areas of the whole country of the Netherlands, except for `geo_postcodes6` which was cropped to only cover the Certe region (using `crop_certe()`).

Source

The data in these data.frames are retrieved from, and publicly available at, Statistics Netherlands:

- Centraal Bureau voor de Statistiek (CBS), 'Gebiedsindelingen', GPKG 2022 v1, `https://www.cbs.nl`

- Centraal Bureau voor de Statistiek (CBS), 'Kerncijfers per postcode', ZIP 2020 v1, https://www.cbs.nl

Examples

```
if (require("certeplot2")) {

  geo_postcodes6 |>
    filter_geolocation(plaats == "Groningen") |>
    plot2(category = inwoners / oppervlakte_km2,
          datalabels = FALSE,
          title = "City of Groningen (PC6 level)")

}

if (require("certeplot2")) {

  geo_postcodes4 |>
    filter_geolocation(plaats == "Groningen") |>
    plot2(category = inwoners / oppervlakte_km2,
          datalabels = FALSE,
          title = "City of Groningen (PC4 level)")

}

if (require("sf")) {

  head(geo_gemeenten)

}
```

---

geocoding *Geocoding to Find Coordinates and Addresses*

Description

Geocoding is the process of retrieving geographic coordinates based on text, such as an address or the name of a place (Wikipedia page). On the other hand, **reverse geocoding** is the process of retrieving the name and address from geographic coordinates (Wikipedia page).

Usage

```
geocode(
  place,
  as_coordinates = FALSE,
  only_netherlands = TRUE,
  api_key = read_secret("gis.api_key"),
  api_requests_per_second = 1
```

```
)

reverse_geocode(
  sf_data,
  api_key = read_secret("gis.api_key"),
  api_requests_per_second = 1
)
```

Arguments

`place`	a (vector of) names or addresses of places
`as_coordinates`	a logical to indicate whether the result should be returned as coordinates (i.e., class `sfc_POINT`)
`only_netherlands`	a logical to indicate whether only Dutch places should be searched
`api_key`	free API key created at `https://geocode.maps.co`
`api_requests_per_second`	number of requests per second
`sf_data`	an 'sf' object or an 'sfc' object (i.e., a vector with geometric `sfc_POINT`s). Can also be a character vector, in which case `geocode()` will be called first.

Details

These functions use OpenStreetMap (OSM), by using the API of `https://geocode.maps.co`.

`geocode()` provides geocoding and returns an 'sf' data.frame at default. In case of multiple results, the distance from the main Certe building in Groningen is leading.

`reverse_geocode()` provides reversed geocoding and returns a data.frame with the columns "name", "address", "zipcode" and "city".

For both functions, the `https://geocode.maps.co` API will only be called on unique input values, to increase speed.

Source

Data © OpenStreetMap contributors, ODbL 1.0. `https://osm.org/copyright`

Examples

```
## Not run:

# geocoding: retrieve 'sf' data.frame based on place names
coord <- geocode("Van Swietenlaan 2, Groningen")
coord

# reverse geocoding: get the name and address
reverse_geocode(coord)
```

```
# places can be any text, and the results are prioritised based on
# the distance from the main Certe building, so:
reverse_geocode(c("Certe", "IKEA"))

hospitals <- geocode(c("Martini ziekenhuis",
                       "Medisch Centrum Leeuwarden",
                       "Tjongerschans Heerenveen",
                       "Scheper Emmen"))
hospitals

if (require("certeplot2")) {
  geo_gemeenten |>
    crop_certe() |>
    plot2(datalabels = FALSE) |>
    add_sf(hospitals, colour = "certeroze", datalabels = place)
}


## End(Not run)
```

GIS *Geodata Functions*

Description

These are functions to work with geographical data. To determine coordinates based on a location (or vice versa), use `geocode()` / `reverse_geocode()`.

Usage

```
get_map(maptype = "postcodes4")

add_map(data, maptype = NULL, by = NULL, crop_certe = TRUE)

is.sf(sf_data)

as.sf(data)

crop_certe(sf_data)

filter_geolocation(sf_data, ...)

filter_sf(sf_data, xmin = NULL, xmax = NULL, ymin = NULL, ymax = NULL)

convert_to_degrees_CRS4326(sf_data)

convert_to_metre_CRS28992(sf_data)
```

```
degrees_to_sf(longitudes, latitudes, crs = 28992)

latitude(sf_data)

longitude(sf_data)
```

Arguments

maptype	type of geometric data, must be one of: "gemeenten", "ggdregios", "nuts3", "postcodes2", "postcodes3", "postcodes4", "postcodes6", "provincies". For add_map(), this is determined automatically if left blank.
data	data set to join left to the geodata
by	column to join by
crop_certe	logical to keep only the Certe region
sf_data	a data set of class 'sf'
...	filters to set
xmin, xmax, ymin, ymax	coordination filters for sf_data, given in degrees following EPSG:4326 ('WGS 84')
longitudes	vector of longitudes
latitudes	vector of latitudes
crs	the coordinate reference system (CRS) to use as output

Details

All of these functions will check if the sf package is installed, and will load its namespace (but not attach the package).

crop_certe() cuts any geometry to the Certe region (more of less): the Northern three provinces of the Netherlands and municipalities of Noordoostpolder, Urk, and Steenwijkerland. This will be based on postcodes.

filter_geolocation() filters an sf object on qualitative values such as 'gemeente' and 'provincie'. The input data sf_data will be joined with postcodes and filtering can thus be done on any of these columns: postcode, inwoners, inwoners_man, inwoners_vrouw, plaats, gemeente, provincie, nuts3, ggdregio.

filter_sf() filters an sf object on coordinates, and is internally used by crop_certe().

convert_to_degrees_CRS4326() will transform SF data to WGS 84 – WGS84 - World Geodetic System 1984, used in GPS, CRS 4326.

convert_to_metre_CRS28992() will transform SF data to Amersfoort / RD New – Netherlands - Holland - Dutch, CRS 28992.

latitude() specifies the north-south position ('y axis') and longitude() specifies the east-west position ('x axis'). They return the numeric coordinate of the centre of a simple feature.

Value

An `sf` model. The column with geodata is always called `"geometry"`.

Examples

```
# Retrieving and joining maps ----------------------------------------

get_map() # defaults to the geo_postcodes4 data set

# adding a map applies a RIGHT JOIN to get all relevant geometric data
data.frame(postcode = 7753, number_of_cases = 3) |>
  add_map()


# Cropping to Certe region -------------------------------------------

# Note: provinces do not include Flevoland
geo_provincies |> crop_certe()

# but other geometries do, such as geo_gemeenten
if (require("certeplot2")) {
  geo_gemeenten |> crop_certe() |>    # cropped municipalities
    plot2(title = "Certe Region") |>
    add_sf(
      geo_provincies |> crop_certe(), # cropped provinces
      colour_fill = NA,
      colour = "black",
      linewidth = 0.5)
}


# Filtering geometries -----------------------------------------------

geo_gemeenten |>
  crop_certe() |>
  # notice that the `provincie` column is not even in `geo_gemeenten`
  filter_geolocation(provincie == "Flevoland")

geo_gemeenten |>
  crop_certe() |>
  filter_geolocation(inwoners_vrouw >= 50000)

if (require("certeplot2")) {
  geo_postcodes4 |>
    filter_geolocation(gemeente == "Tytsjerksteradiel") |>
    plot2(category = inwoners,
          datalabels = postcode)

}

# filter on a latitude of 52.5 degrees and higher
geo_provincies |> filter_sf(ymin = 52.5)
```

```
# Transforming Coordinate Reference System (CRS) -----------------------

geo_provincies |> convert_to_degrees_CRS4326()

geo_provincies |> convert_to_metre_CRS28992()


# Other functions ------------------------------------------------------

degrees_to_sf(4.5, 54)

if (require("certeplot2")) {
  geo_provincies |>
      crop_certe() |>
      plot2(category = NULL, colour_fill = NA) |>
      add_sf(degrees_to_sf(6.5, 53),
             datalabels = "Some Point!")
}

latitude(geo_provincies)
longitude(geo_provincies)
```

`inwoners_per_postcode_leeftijd`
Number of Inhabitants per Zip Code and Age

Description

Number of Inhabitants per Zip Code and Age

Usage

```
inwoners_per_postcode_leeftijd
```

Format

A data.frame with 99,260 observations and 5 variables:

- `postcode`
  zip code, contains PC2, PC3 and PC4
- `leeftijd`
  age group per 5 years: 0-4, 5-9, ..., 90-94, 95+
- `inwoners`
  total number of inhabitants
- `inwoners_man`
  total number of male inhabitants
- `inwoners_vrouw`
  total number of female inhabitants

Details

See the repository file to update this data set.

Source

The data in this data.frame are retrieved from, and publicly available at, Statistics Netherlands: *StatLine*, Centraal Bureau voor de Statistiek (CBS), 'Bevolking en leeftijd per postcode' (data set 83502NED), 1 januari 2021, https://opendata.cbs.nl.

Examples

```
head(inwoners_per_postcode_leeftijd)
str(inwoners_per_postcode_leeftijd)
```

---

postcodes — *Data Set with Dutch Zip Codes, Cities, Municipalities and Province*

Description

Data Set with Dutch Zip Codes, Cities, Municipalities and Province

Usage

```
postcodes
```

Format

A data.frame with 4,963 observations and 9 variables:

- `postcode`
  zip code, contains PC2, PC3 and PC4
- `inwoners`
  total number of inhabitants
- `inwoners_man`
  total number of male inhabitants
- `inwoners_vrouw`
  total number of female inhabitants
- `plaats`
  formal Dutch city name
- `gemeente`
  formal Dutch municipality name
- `provincie`
  formal Dutch province name
- `nuts3`
  Nomenclature of Territorial Units for Statistics, level 3 (in Dutch: COROP region, *Coordinatie Commissie Regionaal OnderzoeksProgramma*)

- `ggdregio`
  name of the regional GGD service (public healthcare service)

Details

See the repository file to update this data set.

Source

The data in this data.frame are retrieved from, and publicly available at, Statistics Netherlands: *StatLine*, Centraal Bureau voor de Statistiek (CBS), 'Bevolking per geslacht per postcode' (data set 83503NED), 1 januari 2021, https://opendata.cbs.nl.

Examples

```
head(postcodes)
str(postcodes)
```

`postcodes4_afstanden` *Distance from Zip Code to Zip Code*

Description

This data set was obtained by calculating the difference from the middle point of a zip code geometry to another zip code geometry (using the geo_postcodes4 data set and the `sf` package).

Usage

```
postcodes4_afstanden
```

Format

A data.frame with 562,330 observations and 3 variables:

- `postcode.x`
  zip code (PC4)
- `postcode.y`
  zip code (PC4)
- `afstand_km`
  distance in kilometres

Source

The data in this data.frame are retrieved from, and publicly available at, Statistics Netherlands:

- Centraal Bureau voor de Statistiek (CBS), 'Gebiedsindelingen', GPKG 2022 v1, https://www.cbs.nl

Examples

```
head(postcodes4_afstanden)
```

Index