Package ‘spatialrisk’

October 14, 2022

Type Package

Title Calculating Spatial Risk

Version 0.7.0

Maintainer Martin Haringa <mtharinga@gmail.com>

Description Methods for spatial risk calculations. It offers an efficient approach to determine the sum of all observations within a circle of a certain radius. This might be beneficial for insurers who are required (by a recent European Commission regulation) to determine the maximum value of insured fire risk policies of all buildings that are partly or fully located within a circle of a radius of 200m. See Church (1974) <doi:10.1007/BF01942293> for a description of the problem.

License GPL (>= 2)

URL https://github.com/mharinga/spatialrisk,
https://mharinga.github.io/spatialrisk/

LazyData true

LinkingTo Rcpp, RcppProgress

Imports classInt, colourvalues, data.table, dplyr, fs, GenSA, geohashTools, ggplot2, leafem, leafgl, leaflet, lubridate, methods, Rcpp, RcppProgress, sf, tmap, units, viridis

Depends R (>= 3.3)

Encoding UTF-8

RoxygenNote 7.1.2

Suggests automap, gstat, knitr, mgcv, rmarkdown, testthat, vroom

NeedsCompilation yes

Author Martin Haringa [aut, cre]

Repository CRAN

Date/Publication 2021-11-10 15:30:02 UTC
** choropleth .......................... 2
choropleth_ggplot2 ..................... 3
choropleth_sf ........................ 4
choropleth_tmap ....................... 5
concentration ........................ 6
Groningen ............................ 7
haversine ................................ 8
highest_concentration ............... 9
insurance ............................. 12
interpolate_krige ..................... 13
interpolate_spline ................... 14
knmi_historic_data .................... 16
knmi_stations ........................ 17
neighborhood_gh_search ............. 18
nl_corop ................................ 19
nl_gemeente .......................... 20
nl_postcode2 .......................... 21
nl_postcode3 .......................... 21
nl_postcode4 .......................... 22
nl_provincie .......................... 23
plot.concentration .................... 23
plot.neighborhood .................... 25
plot_points ........................... 26
points_in_circle ..................... 27
points_to_polygon .................... 28

** choropleth .......................... 2

Create choropleth map

** choropleth(  
    sf_object,  
    value = "output",  
    id_name = "areaname",  
    mode = "plot",  
    n = 7,

Description

Takes an object produced by points_to_polygon(), and creates the corresponding choropleth map. The given clustering is according to the Fisher-Jenks algorithm. This commonly used method for choropleths seeks to reduce the variance within classes and maximize the variance between classes.

Usage

choropleth(  
    sf_object,  
    value = "output",  
    id_name = "areaname",  
    mode = "plot",  
    n = 7,
choropleth_ggplot2

legend_title = "Clustering",
palette = "viridis"
)

Arguments

sf_object object of class sf
value column name to shade the polygons
id_name column name of ids to plot
mode choose between static (’plot’ is default) and interactive map (’view’)
n number of clusters (default is 7)
legend_title title of legend
palette palette name or a vector of colors. See tmaptools:palette_explorer() for the named palettes. Use a "." as prefix to reverse the palette. The default palette is "viridis".

Value

tmap

Author(s)

Martin Haringa

Examples

test <- points_to_polygon(nl_provincie, insurance, sum(amount, na.rm = TRUE))
choropleth(test)
choropleth(test, id_name = "areaname", mode = "view")

choropleth_ggplot2 Map object of class sf using ggplot2

Description

Takes an object produced by choropleth_sf(), and creates the corresponding choropleth map.

Usage

choropleth_ggplot2(
    sf_object,
    value = output,
    n = 7,
    dig.lab = 2,
    legend_title = "Class",
    option = "D",
    direction = 1
)
Arguments

- **sf_object**: object of class sf
- **value**: column to shade the polygons
- **n**: number of clusters (default is 7)
- **dig.lab**: number of digits in legend (default is 2)
- **legend_title**: title of legend
- **option**: a character string indicating the colormap option to use. Four options are available: "magma" (or "A"), "inferno" (or "B"), "plasma" (or "C"), "viridis" (or "D", the default option) and "cividis" (or "E").
- **direction**: Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.

Value

ggplot map

Author(s)

Martin Haringa

Examples

test <- points_to_polygon(nl_postcode2, insurance, sum(amount, na.rm = TRUE))
choropleth_ggplot2(test)

---

**choropleth_sf**

*Aggregate attributes of coordinates to area level (deprecated function; use 'points_to_polygon' instead)*

Description

A data.frame containing coordinates (in terms of longitude and latitude) is joined to the polygon level. Then arithmetic operations on the attributes of the coordinates are applied to obtain aggregated values for each polygon.

Usage

choropleth_sf(sf_map, df, oper, crs = 4326, outside_print = FALSE)

Arguments

- **sf_map**: object of class sf
- **df**: data.frame containing coordinates (column names should be 'lon' and 'lat')
- **oper**: an arithmetic operation on the polygon level
- **crs**: coordinate reference system: integer with the EPSG code, or character with proj4string
- **outside_print**: print points that are not within a polygon (default is FALSE).
**Value**

an object of class sf

**Author(s)**

Martin Haringa

---

**choropleth_tmap**  
*Map object of class sf using tmap (deprecated function; use 'choropleth' instead)*

---

**Description**

Takes an object produced by `choropleth_sf()`, and creates the corresponding choropleth map.

**Usage**

```r
choropleth_tmap(
  sf_object,
  value = "output",
  id_name = "areaname",
  mode = "plot",
  n = 7,
  legend_title = "Clustering",
  palette = "viridis"
)
```

**Arguments**

- **sf_object**: object of class sf
- **value**: column name to shade the polygons
- **id_name**: column name of ids to plot
- **mode**: choose between static ("plot" is default) and interactive map ("view")
- **n**: number of clusters (default is 7)
- **legend_title**: title of legend
- **palette**: palette name or a vector of colors. See `tmaptools::palette_explorer()` for the named palettes. Use a "-" as prefix to reverse the palette. The default palette is "viridis".

**Value**

tmap

**Author(s)**

Martin Haringa
concentration  Concentration risk

Description
The sum of all observations within a circle of a certain radius.

Usage
concentration(
  sub,
  full,
  value,
  lon_sub = lon,
  lat_sub = lat,
  lon_full = lon,
  lat_full = lat,
  radius = 200,
  display_progress = TRUE
)

Arguments
  sub  data.frame of locations to calculate concentration risk for (target points). sub
        should include at least columns for longitude and latitude.
  full  data.frame to find the locations within radius r from locations in sub (reference
        locations). full should include at least columns for longitude, latitude and value
        of interest to summarize.
  value  column name with value of interest to summarize in full.
  lon_sub  column name in sub with longitude (lon is default).
  lat_sub  column name in sub with latitude (lat is default).
  lon_full  column name in full with longitude in full (lon is default).
  lat_full  column name in full with latitude in full (lat is default).
  radius  radius (in meters) (default is 200m).
  display_progress  show progress bar (TRUE/FALSE). Defaults to TRUE.

Value
A data.frame equal to data.frame sub including an extra column concentration.

Author(s)
Martin Haringa
Examples

```r
df <- data.frame(location = c("p1", "p2"), lon = c(6.561561, 6.561398), lat = c(53.21369, 53.21326))
concentration(df, Groningen, value = amount, radius = 100)
```

---

Groningen  
Coordinates of houses in Groningen

Description

A dataset of postal codes and the corresponding spatial locations in terms of a latitude and a longitude.

Usage

Groningen

Format

A data frame with 25000 rows and 8 variables:

- **street**: Name of street
- **number**: Number of house
- **letter**: Letter of house
- **suffix**: Suffix to number of house
- **postal_code**: Postal code of house
- **city**: The name of the city
- **lon**: Longitude (in degrees)
- **lat**: Latitude (in degrees)
- **amount**: Random value

Source

The BAG is the Dutch registry for Buildings and addresses (Basisregistratie adressen en gebouwen).
haversine

Haversine great circle distance

Description

The shortest distance between two points (i.e., the 'great-circle-distance' or 'as the crow flies'), according to the 'haversine method'. This method assumes a spherical earth, ignoring ellipsoidal effects. Note that this version is implemented in C++. A quick benchmark to the version of geosphere showed it to be a non-insignificant speed enhancement. The algorithm converges in one-twentieth of the original time.

Usage

haversine(lat_from, lon_from, lat_to, lon_to, r = 6378137)

Arguments

- lat_from: Latitude of point.
- lon_from: Longitude of point.
- lat_to: Latitude of point.
- lon_to: Longitude of point.
- r: Radius of the earth; default = 6378137m

Details

The Haversine ('half-versed-sine') formula was published by R.W. Sinnott in 1984, although it has been known for much longer.

Value

Vector of distances in the same unit as r (default in meters).

Author(s)

Martin Haringa

References


Examples

haversine(53.24007, 6.520386, 53.24054, 6.520386)
highest_concentration  

**Description**

Find the centre coordinates of a circle with a fixed radius that maximizes the coverage of total fire risk insured. ‘highest_concentration()’ returns the coordinates (lon/lat) and the corresponding concentration. The concentration is defined as the sum of all observations within a circle of a certain radius. See `concentration` for determining concentration for pre-defined coordinates.

Find the centre coordinates of a circle with a fixed radius that maximizes the coverage of total fire risk insured. ‘highest_concentration()’ returns the coordinates (lon/lat) and the corresponding concentration. The concentration is defined as the sum of all observations within a circle of a certain radius. See `concentration` for determining concentration for pre-defined coordinates.

**Usage**

```r
highest_concentration(
  df,
  value,
  lon = lon,
  lat = lat,
  lowerbound = NULL,
  radius = 200,
  grid_distance = 25,
  gh_precision = 6,
  display_progress = TRUE
)
```

```r
highest_concentration(
  df,
  value,
  lon = lon,
  lat = lat,
  lowerbound = NULL,
  radius = 200,
  grid_distance = 25,
  gh_precision = 6,
  display_progress = TRUE
)
```

**Arguments**

- **df**  
  data.frame of locations, should at least include column for longitude, latitude and sum insured

- **value**  
  column name with value of interest to summarize (e.g. sum insured)

- **lon**  
  column name with longitude (defaults to ‘lon’)

- **lat**  
  column name with latitude (defaults to ‘lat’)

- **lowerbound**  
  lower bound (optional)

- **radius**  
  circle radius

- **grid_distance**  
  grid distance

- **gh_precision**  
  grid hierarchy precision

- **display_progress**  
  display progress information
### Details

A recently European Commission regulation requires insurance companies to determine the maximum value of insured fire risk policies of all buildings that are partly or fully located within circle of a radius of 200m (Commission Delegated Regulation (EU), 2015, Article 132). The problem can be stated as: "find the centre coordinates of a circle with a fixed radius that maximizes the coverage of total fire risk insured". This can be viewed as a particular instance of the Maximal Covering Location Problem (MCLP) with fixed radius. See Gomes (2018) for a solution to the maximum fire risk insured capital problem using a multi-start local search meta-heuristic. The computational performance of `highest_concentration()` is investigated to overcome the long times the MCLP algorithm is taking. `highest_concentration()` is written in C++, and for 500,000 buildings it needs about 5-10 seconds to determine the maximum value of insured fire risk policies that are partly or fully located within circle of a radius of 200m.

`highest_concentration()` uses Gustavo Niemeyer’s wonderful and elegant geohash coordinate system. Niemeyer’s Geohash method encodes latitude and longitude as binary string where each binary value derived from a decision as to where the point lies in a bisected region of latitude or longitudinal space. The first step is to convert all latitude/longitude coordinates into geohash-encoded strings.

The length of the geohash (`gh_precision`) controls the ’zoom level’:

- precision 5 is 4.89 x 4.89km;
- precision 6 is 1.22km x 0.61km;
- precision 7 is 153m x 153m;
- precision 8 is 39m x 19m.

For a circle with a radius of 200m the precision of the geohash should be set equal to 6 (default). Then the ‘value’ column is aggregated (sum) per geohash (with a buffer of size ‘radius’ around each geohash, since the coordinates of the highest concentration can be near the edge of the geohash). The geohashes with a aggregated value below the lowerbound are removed, where the lowerbound is equal to the maximum of the ‘value’ column. Then a grid is created, with a distance of ‘grid_distance’ between the points. See example section for a illustration of the algorithm. As a last step for each grid point the concentration is calculated.

A recently European Commission regulation requires insurance companies to determine the maximum value of insured fire risk policies of all buildings that are partly or fully located within circle of a radius of 200m (Commission Delegated Regulation (EU), 2015, Article 132). The problem can be stated as: "find the centre coordinates of a circle with a fixed radius that maximizes the coverage

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lat</td>
<td>column name with latitude (defaults to ‘lat’)</td>
</tr>
<tr>
<td>lowerbound</td>
<td>set lower bound for outcome (defaults to NULL)</td>
</tr>
<tr>
<td>radius</td>
<td>radius (in meters) (default is 200m)</td>
</tr>
<tr>
<td>grid_distance</td>
<td>distance (in meters) for precision of concentration risk (default is 25m).</td>
</tr>
<tr>
<td>gh_precision</td>
<td>positive integer to define geohash precision. See details.</td>
</tr>
<tr>
<td>display_progress</td>
<td>show progress bar (TRUE/FALSE). Defaults to TRUE.</td>
</tr>
</tbody>
</table>
of total fire risk insured”. This can be viewed as a particular instance of the Maximal Covering Location Problem (MCLP) with fixed radius. See Gomes (2018) for a solution to the maximum fire risk insured capital problem using a multi-start local search meta-heuristic. The computational performance of highest_concentration() is investigated to overcome the long times the MCLP algorithm is taking. highest_concentration() is written in C++, and for 500,000 buildings it needs about 5-10 seconds to determine the maximum value of insured fire risk policies that are partly or fully located within circle of a radius of 200m.

‘highest_concentration()’ uses Gustavo Niemeyer’s wonderful and elegant geohash coordinate system. Niemeyer’s Geohash method encodes latitude and longitude as binary string where each binary value derived from a decision as to where the point lies in a bisected region of latitude or longitudinal space. The first step is to convert all latitude/longitude coordinates into geohash-encoded strings.

The length of the geohash (‘gh_precision’) controls the ’zoom level’:

- precision 5 is 4.89 x 4.89km;
- precision 6 is 1.22km x 0.61km;
- precision 7 is 153m x 153m;
- precision 8 is 39m x 19m.

For a circle with a radius of 200m the precision of the geohash should be set equal to 6 (default). Then the ‘value’ column is aggregated (sum) per geohash (with a buffer of size ‘radius’ around each geohash, since the coordinates of the highest concentration can be near the edge of the geohash). The geohashes with a aggregated value below the lowerbound are removed, where the lowerbound is equal to the maximum of the ‘value’ column. Then a grid is created, with a distance of ‘grid_distance’ between the points. See example section for a illustration of the algorithm. As a last step for each grid point the concentration is calculated.

Value

data.frame with coordinates (lon/lat) with the highest concentrations

data.frame with coordinates (lon/lat) with the highest concentrations

Author(s)

Martin Haringa

Martin Haringa

References


Examples

```r
## Not run:
# Find highest concentration with a precision of a grid of 25 meters
hc1 <- highest_concentration(Groningen, amount, radius = 200, grid_distance = 25)

# Look for coordinates with even higher concentrations in the
# neighborhood of the coordinates with the highest concentration
hc1_nghb <- neighborhood_gh_search(hc1, max.call = 7000)
print(hc1_nghb)

# Create map with geohashes above the lowerbound
# The highest concentration lies in one of the geohashes
plot(hc1)

# Create map with highest concentration
plot(hc1_nghb)

## End(Not run)
## Not run:
# Find highest concentration with a precision of a grid of 25 meters
hc1 <- highest_concentration(Groningen, amount, radius = 200, grid_distance = 25)

# Look for coordinates with even higher concentrations in the
# neighborhood of the coordinates with the highest concentration
hc1_nghb <- neighborhood_gh_search(hc1, max.call = 7000)
print(hc1_nghb)

# Create map with geohashes above the lowerbound
# The highest concentration lies in one of the geohashes
plot(hc1)

# Create map with highest concentration
plot(hc1_nghb)

## End(Not run)
```

### insurance

#### Sum insured per postal code in the Netherlands

**Description**

A dataset of postal codes with their sum insured, population and the corresponding spatial locations in terms of a latitude and a longitude.
interpolate_krige

Usage

insurance

Format

A data frame with 29,990 rows and 5 variables:

- **postcode**: 6-digit postal code
- **population.pc4**: Population per 4-digit postal code
- **amount**: Sum insured
- **lon**: Longitude (in degrees) of the corresponding 6-digit postal code
- **lat**: Latitude (in degrees) of the corresponding 6-digit postal code

Description

Interpolation and smoothing on the sphere by means of ordinary kriging.

Usage

```r
interpolate_krige(
  observations,
  targets,
  value,
  lon_obs = lon,
  lat_obs = lat,
  lon_targets = lon,
  lat_targets = lat
)
```

Arguments

- **observations**: data.frame of observations.
- **targets**: data.frame of locations to calculate the interpolated and smoothed values for (target points).
- **value**: Column with values in observations.
- **lon_obs**: Column in observations with longitude (lon is default).
- **lat_obs**: Column in observations with latitude (lat is default).
- **lon_targets**: Column in targets with longitude (lon is default).
- **lat_targets**: Column in targets with latitude (lat is default).
Details

observations should include at least columns for longitude and latitude.

targets should include at least columns for longitude, latitude and value of interest to interpolate and smooth.

Kriging can be considered as linear regression with spatially correlated residuals. Kriging is most appropriate when it is known there is a spatially correlated distance or directional bias in the data. It is often used in soil science and geology.

See splines on the sphere for interpolation and smoothing on the sphere by means of splines.

Value

Object equal to object targets including extra columns for the predicted value and the variance.

Author(s)

Martin Haringa

References

gstat::krige

Examples

```r
## Not run:
target <- sf::st_drop_geometry(nl_postcode3)
obs <- insurance %>% dplyr::sample_n(1000)
pop_df <- interpolate_krige(obs, target, population_pc4)
pop_sf <- left_join(nl_postcode3, pop_df)
choropleth(pop_sf, value = "population_pc4_pred", n = 13)
choropleth(pop_sf, value = "population_pc4_var", n = 13)

## End(Not run)
```

interpolate_spline | Splines on the sphere

Description

Spline interpolation and smoothing on the sphere.
interpolate_spline

Usage
interpolate_spline(
  observations,
  targets,
  value,
  lon_obs = lon,
  lat_obs = lat,
  lon_targets = lon,
  lat_targets = lat,
  k = 50
)

Arguments
observations  data.frame of observations.
targets  data.frame of locations to calculate the interpolated and smoothed values for (target points).
value  Column with values in observations.
lon_obs  Column in observations with longitude (lon is default).
lon_targets  Column in targets with longitude (lon is default).
lon_obs  Column in observations with longitude (lon is default).
lon_targets  Column in targets with longitude (lon is default).
lat_obs  Column in observations with latitude (lat is default).
lon_targets  Column in targets with latitude (lat is default).
lat_obs  Column in observations with latitude (lat is default).
lon_targets  Column in targets with latitude (lat is default).
k  (default 50) is the basis dimension. For small data sets reduce k manually rather than using default.

Details
observations should include at least columns for longitude and latitude.
targets should include at least columns for longitude, latitude and value of interest to interpolate and smooth.
A smooth of the general type discussed in Duchon (1977) is used: the sphere is embedded in a 3D Euclidean space, but smoothing employs a penalty based on second derivatives (so that locally as the smoothing parameter tends to zero we recover a "normal" thin plate spline on the tangent space). This is an unpublished suggestion of Jean Duchon.
See ordinary kriging for interpolation and smoothing on the sphere by means of kriging.

Value
Object equal to object targets including an extra column with predicted values.

Author(s)
Martin Haringa

References
Splines on the sphere
Examples

```r
## Not run:
target <- sf::st_drop_geometry(nl_postcode3)
obs <- dplyr::sample_n(insurance, 1000)
pop_df <- interpolate_spline(obs, target, population_pc4, k = 20)
pop_sf <- left_join(nl_postcode3, pop_df)
choropleth(pop_sf, value = "population_pc4_pred", n = 13)

## End(Not run)
```

knmi_historic_data

Retrieve historic weather data for the Netherlands

Description

This function retrieves historic weather data collected by the official KNMI weather stations. See spatialrisk::knmi_stations for a list of the official KNMI weather stations.

Usage

```r
knmi_historic_data(startyear, endyear)
```

Arguments

- `startyear`: start year for historic weather data.
- `endyear`: end year for historic weather data.

Format

The returned data frame contains the following columns:

- `station` = ID of measurement station;
- `date` = Date;
- `FH` = Hourly mean wind speed (in 0.1 m/s)
- `FX` = Maximum wind gust (in 0.1 m/s) during the hourly division;
- `T` = Temperature (in 0.1 degrees Celsius) at 1.50 m at the time of observation;
- `DR` = Precipitation duration (in 0.1 hour) during the hourly division;
- `RH` = Hourly precipitation amount (in 0.1 mm) (-1 for <0.05 mm);
- `city` = City where the measurement station is located;
- `lon` = Longitude of station (crs = 4326);
- `lat` = Latitude of station (crs = 4326).

Value

Data frame containing weather data and meta data for weather station locations.
**knmi_stations**

**Author(s)**

Martin Haringa

**Examples**

```r
## Not run:
knmi_historic_data(2015, 2019)

## End(Not run)
```

---

### Description

A data frame containing the IDs and meta-data on the official KNMI weather stations.

### Usage

```r
knmi_stations
```

### Format

A data frame with 50 rows and 7 variables:

- **station**: ID of the station (209-391)
- **city**: City where the station is located
- **lon**: Longitude of station (crs = 4326)
- **lat**: Latitude of the station (crs = 4326)
- **altitude**: Altitude of the station (in meters)
- **X**: X coordinate of the station (crs = 32631)
- **Y**: Y coordinate of the station (crs = 32631)

**Author(s)**

Martin Haringa
neighborhood_gh_search

Search for coordinates with higher concentrations within geohash

Description

highest_concentration returns the highest concentration within a portfolio based on grid points. However, higher concentrations can be found within two grid points. 'neighborhood_gh_search()' looks for even higher concentrations in the neighborhood of the grid points with the highest concentrations. This optimization is done by means of Simulated Annealing.

Usage

neighborhood_gh_search(
  hc,  
  highest_geohash = 1,  
  max.call = 1000,  
  verbose = TRUE,  
  seed = 1  
)

neighborhood_gh_search(
  hc,  
  highest_geohash = 1,  
  max.call = 1000,  
  verbose = TRUE,  
  seed = 1  
)

Arguments

hc                  object of class ‘concentration’ obtained from ‘highest_concentration()’

highest_geohash    the number of geohashes the searching algorithm is applied to. Defaults to 1 (i.e. algorithm is only applied to the geohash with the highest concentration).

max.call            maximum number of calls to the concentration function (i.e. the number of coordinates in the neighborhood of the highest concentration). Defaults to 1000.

verbose             show messages from the algorithm (TRUE/FALSE). Defaults to FALSE.

seed                set seed
Value

data.frame
data.frame

Author(s)

Martin Haringa
Martin Haringa

Examples

## Not run:
# Find highest concentration with a precision of a grid of 25 meters
hc1 <- highest_concentration(Groningen, amount, radius = 200, grid_distance = 25)

# Increase the number of calls to the concentration function for more extensive search
hc1_nghb <- neighborhood_gh_search(hc1, max.call = 7000, highest_geohash = 1)
hc2_nghb <- neighborhood_gh_search(hc1, max.call = 7000, highest_geohash = 2)
plot(hc1_nghb)
plot(hc2_nghb)

## End(Not run)

## Not run:
# Find highest concentration with a precision of a grid of 25 meters
hc1 <- highest_concentration(Groningen, amount, radius = 200, grid_distance = 25)

# Increase the number of calls to the concentration function for more extensive search
hc1_nghb <- neighborhood_gh_search(hc1, max.call = 7000, highest_geohash = 1)
hc2_nghb <- neighborhood_gh_search(hc1, max.call = 7000, highest_geohash = 2)
plot(hc1_nghb)
plot(hc2_nghb)

## End(Not run)

---

nl_corop  

Object of class sf for COROP regions in the Netherlands

Description

An object of class sf (simple feature) for COROP regions in the Netherlands.

Usage

nl_corop
Format

A simple feature object with 40 rows and 5 variables:

- **corop_nr**  corop number
- **areaname**  corop name
- **geometry**  geometry object of COROP region
- **lon**  longitude of the corop centroid
- **lat**  latitude of the corop centroid

Details

A COROP region is a regional area within the Netherlands. These regions are used for analytical purposes by, among others, Statistics Netherlands. The Dutch abbreviation stands for Coördinatiecommissie Regionaal Onderzoeksprogramma, literally the Coordination Commission Regional Research Programme.

Author(s)

Martin Haringa

---

**nl_gemeente**  
*Object of class sf for municipalities in the Netherlands*

Description

An object of class sf (simple feature) for municipalities (Dutch: gemeentes) in the Netherlands in the year 2018.

Usage

nl_gemeente

Format

A simple feature object with 380 rows and 6 variables:

- **id**  id of gemeente
- **code**  code of gemeente
- **areaname**  name of gemeente
- **geometry**  geometry object of gemeente
- **lon**  longitude of the gemeente centroid
- **lat**  latitude of the gemeente centroid

Author(s)

Martin Haringa
nl_postcode2

Object of class sf for 2-digit postcode regions in the Netherlands

Description

An object of class sf (simple feature) for 2-digit postal codes (Dutch: postcode) regions in the Netherlands.

Usage

nl_postcode2

Format

A simple feature object with 90 rows and 4 variables:

- areaname 2-digit postal code
- geometry geometry object of postal code
- lon longitude of the 2-digit postal code centroid
- lat latitude of the 2-digit postal code centroid

Details

Postal codes in the Netherlands, known as postcodes, are alphanumeric, consisting of four digits followed by two uppercase letters. The first two digits indicate a city and a region, the second two digits and the two letters indicate a range of house numbers, usually on the same street.

Author(s)

Martin Haringa

nl_postcode3

Object of class sf for 3-digit postcode regions in the Netherlands

Description

An object of class sf (simple feature) for 3-digit postal codes (Dutch: postcode) regions in the Netherlands.

Usage

nl_postcode3
Format

A simple feature object with 799 rows and 3 variables:

- `areaname` 3-digit postal code
- `geometry` geometry object of postal code
- `lon` longitude of the 3-digit postal code centroid
- `lat` latitude of the 3-digit postal code centroid

Details

Postal codes in the Netherlands, known as postcodes, are alphanumeric, consisting of four digits followed by two uppercase letters. The first two digits indicate a city and a region, the second two digits and the two letters indicate a range of house numbers, usually on the same street.

Author(s)

Martin Haringa

---

**nl_postcode4**

Object of class `sf` for 4-digit postcode regions in the Netherlands

---

Description

An object of class `sf` (simple feature) for 4-digit postal codes (Dutch: postcode) regions in the Netherlands.

Usage

`nl_postcode4`

Format

A simple feature object with 4053 rows and 7 variables:

- `pc4` 4-digit postal code
- `areaname` name of corresponding 4-digit postal code
- `city` name of city
- `geometry` geometry object of postal code
- `lon` longitude of the 4-digit postal code centroid
- `lat` latitude of the 4-digit postal code centroid
Postal codes in the Netherlands, known as postcodes, are alphanumeric, consisting of four digits followed by two uppercase letters. The first two digits indicate a city and a region, the second two digits and the two letters indicate a range of house numbers, usually on the same street.

Author(s)

Martin Haringa

---

**nl_provincie**

Object of class sf for provinces in the Netherlands

---

**Description**

An object of class sf (simple feature) for provinces (Dutch: provincies) in the Netherlands.

**Usage**

nl_provincie

**Format**

A simple feature object with 12 rows and 4 variables:

- **areaname** province name
- **geometry** geometry object of province
- **lon** longitude of the province centroid
- **lat** latitude of the province centroid

Author(s)

Martin Haringa

---

**plot.concentration**

Automatically create a plot for objects obtained from highest_concentration()

---

**Description**

Takes an object produced by ‘highest_concentration()’, and creates an interactive map.

Takes an object produced by ‘highest_concentration()’, and creates an interactive map.
Usage

```r
## S3 method for class 'concentration'
plot(
  x,
  grid_points = TRUE,
  legend_title = NULL,
  palette = "viridis",
  legend_position = "bottomleft",
  ...
)
```

Arguments

- `x`: object of class ‘concentration’ obtained from ‘highest_concentration()’
- `grid_points`: show grid points (TRUE), or objects (FALSE)
- `legend_title`: title of legend
- `palette`: palette for grid points (defaults to "viridis")
- `legend_position`: legend position for grid points legend (defaults to "bottomleft")
- `...`: additional arguments affecting the interactive map produced

Value

Interactive view of geohashes with highest concentrations

Author(s)

Martin Haringa

Martin Haringa
plot.neighborhood

Automatically create a plot for objects obtained from neighborhood_gh_search()

Description

Takes an object produced by 'neighborhood_gh_search()'; and creates an interactive map.

Usage

## S3 method for class 'neighborhood'
plot(
x,  
buffer = 0,  
legend_title = NULL,  
palette = "viridis",  
legend_position = "bottomleft",  
palette_circle = "YlOrRd",  
legend_position_circle = "bottomright",  
legend_title_circle = "Highest concentration",  
...
)

## S3 method for class 'neighborhood'
plot(
x,  
buffer = 0,  
legend_title = NULL,  
palette = "viridis",  
legend_position = "bottomleft",  
palette_circle = "YlOrRd",  
legend_position_circle = "bottomright",  
legend_title_circle = "Highest concentration",  
...
)

Arguments

x object neighborhood object produced by 'neighborhood_gh_search()'
buffer numeric value, show objects within buffer (in meters) from circle (defaults to 0)
legend_title title of legend
palette palette for points (defaults to "viridis")
legend_position legend position for points legend (defaults to "bottomleft")
plot_points

palette_circle palette for circles (default to "YlOrRd")
legend_position_circle
    legend position for circles legend (defaults to "bottomright")
legend_title_circle
    title of legend for circles
...
    additional arguments affecting the interactive map produced

Value
Interactive view of highest concentration on map
Interactive view of highest concentration on map

Author(s)
Martin Haringa
Martin Haringa

plot_points Create map with points

Description
Create map for data.frame with points.

Usage
plot_points(df, value, lon = lon, lat = lat, palette = "viridis",
            legend_position = "bottomleft", crs = 4326)

Arguments
df data.frame with column for lon and lat
value column in df
lon column with lon
lat column with lat
palette color palette
legend_position position for legend (default is "bottomleft")
crs crs (default is 4326)
Value

leaflet map

Examples

## Not run:
plot_points(Groningen, value = amount)
## End(Not run)

## Not run:
plot_points(Groningen, value = amount)
## End(Not run)

--

points_in_circle Points in circle

Description

Find all observations in a data.frame within a circle of a certain radius.

Usage

points_in_circle(
  data, 
  lon_center, 
  lat_center, 
  lon = lon, 
  lat = lat, 
  radius = 200
)

Arguments
data data.frame with at least columns for longitude and latitude.
lon_center numeric value referencing to the longitude of the center of the circle
lat_center numeric value referencing to the latitude of the center of the circle
lon column name in data with longitudes (lon is default).
lat column name in data with latitudes (lat is default).
radius radius (in meters) (defaults to 200m).

Value
data.frame. Column distance_m gives the distance to the center of the circle (in meters).

Author(s)

Martin Haringa
points_to_polygon

Examples

points_in_circle(Groningen, lon_center = 6.571561, lat_center = 53.21326, radius = 60)

Description

A data.frame containing coordinates (in terms of longitude and latitude) is joined to the polygon level. Then arithmetic operations on the attributes of the coordinates are applied to obtain aggregated values for each polygon.

Usage

points_to_polygon(sf_map, df, oper, crs = 4326, outside_print = FALSE)

Arguments

- **sf_map**: object of class sf
- **df**: data.frame containing coordinates (column names should be 'lon' and 'lat')
- **oper**: an arithmetic operation on the polygon level
- **crs**: coordinate reference system: integer with the EPSG code, or character with proj4string
- **outside_print**: print points that are not within a polygon (default is FALSE).

Value

an object of class sf

Author(s)

Martin Haringa

Examples

```r
points_to_polygon(nl_postcode2, insurance, sum(amount, na.rm = TRUE))
## Not run:
shp_read <- sf::st_read("~/path/to/file.shp")
points_to_polygon(shp_read, insurance, sum(amount, na.rm = TRUE))
## End(Not run)
```
Index

* datasets
  Groningen, 7
  insurance, 12
  knmi_stations, 17
  nl_corop, 19
  nl_gemeente, 20
  nl_postcode2, 21
  nl_postcode3, 21
  nl_postcode4, 22
  nl_provincie, 23

choropleth, 2
choropleth_ggplot2, 3
choropleth_sf, 4
choropleth_tmap, 5
concentration, 6, 9

Groningen, 7
gstat::krige, 14

haversine, 8
highest_concentration, 9, 18

insurance, 12
interpolate_krige, 13
interpolate_spline, 14

knmi_historic_data, 16
knmi_stations, 17

neighborhood_gh_search, 18
nl_corop, 19
nl_gemeente, 20
nl_postcode2, 21
nl_postcode3, 21
nl_postcode4, 22
nl_provincie, 23

plot.concentration, 23
plot.neighborhood, 25
plot_points, 26

points_in_circle, 27
points_to_polygon, 28