

Package ‘spatialrisk’

September 14, 2025

Type Package

Title Calculating Spatial Risk

Version 0.7.3

Maintainer Martin Haringa <mtharinga@gmail.com>

BugReports <https://github.com/mtharinga/spatialrisk/issues>

Description Provides methods for spatial risk calculations, focusing on efficient determination of the sum of observations within a circle of a given radius. These methods are particularly relevant for applications such as insurance, where recent European Commission regulations require the calculation of the maximum insured value of fire risk policies for all buildings that are partly or fully located within a 200 m radius. The underlying problem is described by Church (1974) <[doi:10.1007/BF01942293](https://doi.org/10.1007/BF01942293)>.

License GPL (>= 2)

URL <https://github.com/mtharinga/spatialrisk>,
<https://mtharinga.github.io/spatialrisk/>

LazyData true

LinkingTo Rcpp, RcppProgress

Imports classInt, data.table, dplyr, fs, ggplot2, lifecycle, mapview,
Rcpp, RcppProgress, rlang, sf, terra, tmap, units, viridis

Depends R (>= 4.1.0)

Encoding UTF-8

RoxygenNote 7.3.3

Suggests colourvalues, GenSA, geohashTools, knitr, leafem, leafgl,
leaflet, mgcv, rmarkdown, testthat, vroom

NeedsCompilation yes

Author Martin Haringa [aut, cre]

Repository CRAN

Date/Publication 2025-09-14 09:20:02 UTC

Contents

choropleth	2
choropleth_ggplot2	3
concentration	5
convert_crs_df	6
find_highest_concentration	7
Groningen	9
haversine	10
highest_concentration	11
insurance	13
interpolate_spline	13
knmi_historic_data	15
knmi_stations	16
neighborhood_gh_search	16
nl_corop	17
nl_gemeente	18
nl_postcode2	19
nl_postcode3	19
nl_postcode4	20
nl_provincie	21
plot.conc	22
plot.concentration	23
plot.neighborhood	24
plot_points	25
points_in_circle	26
points_in_circle_vec	27
points_to_polygon	28
Index	29

choropleth	<i>Create choropleth map</i>
------------	------------------------------

Description

Creates a choropleth map from an ‘sf’ object, typically produced by points_to_polygon(). Polygons are shaded according to values in a specified column, with clustering based on the Fisher–Jenks algorithm. This method minimizes within-class variance and maximizes between-class variance, making it a common choice for choropleth maps.

Usage

```
choropleth(  
  sf_object,  
  value = "output",  
  id_name = "areaname",  
  mode = "plot",
```

```

    n = 7,
    legend_title = "Clustering",
    palette = "viridis"
  )

```

Arguments

<code>sf_object</code>	An object of class <code>sf</code> .
<code>value</code>	A string giving the name of the column used to shade the polygons.
<code>id_name</code>	A string giving the name of the column containing polygon IDs (used for tooltips in interactive mode).
<code>mode</code>	A string indicating whether to create a static map ("plot", default) or an interactive map ("view").
<code>n</code>	Integer; number of clusters. Default is 7.
<code>legend_title</code>	A string giving the legend title.
<code>palette</code>	A palette name or vector of colors. See <code>tmertools::palette_explorer()</code> for available palettes. Prefix the name with "-" to reverse the order. Default is "viridis".

Details

The function uses the Fisher–Jenks algorithm (`style = "fisher"`) to classify values into `n` groups.

Value

A `tmap` object (static or interactive, depending on `mode`).

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")

```

<code>choropleth_ggplot2</code>	<i>Choropleth map of an <code>sf</code> object with <code>ggplot2</code></i>
---------------------------------	--

Description

Creates a choropleth map from an object of class `sf`. If the chosen variable can be classified into discrete intervals using Fisher's natural breaks, the polygons are shaded by cluster. Otherwise, the variable is visualized on a continuous scale.

Usage

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

Arguments

<code>sf_object</code>	An object of class <code>sf</code> containing polygon geometries.
<code>value</code>	Column in <code>sf_object</code> used to shade the polygons (default = <code>output</code>).
<code>n</code>	Integer. Number of clusters to use in Fisher classification (default = 7).
<code>dig.lab</code>	Integer. Number of digits to display in legend labels (default = 2).
<code>legend_title</code>	Character. Title for the legend (default = "Class").
<code>option</code>	Character string indicating the colormap option passed to <code>viridis</code> . Options are: <ul style="list-style-type: none"> • "magma" (or "A") • "inferno" (or "B") • "plasma" (or "C") • "viridis" (or "D", default) • "cividis" (or "E")
<code>direction</code>	Numeric. Order of colors in the scale. If 1 (default), colors go from darkest to lightest. If -1, the order is reversed.

Details

The function first attempts to classify the chosen variable into `n` clusters using Fisher's natural breaks (`classInt::classIntervals()`). If this fails (e.g. due to insufficient unique values), the function falls back to a continuous scale.

Value

A ggplot object containing the choropleth map.

Author(s)

Martin Haringa

Examples

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

```
)  
choropleth_ggplot2(test, value = output)
```

concentration	<i>Concentration calculation</i>
---------------	----------------------------------

Description

Calculates the concentration, which is 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	A data.frame of target points for which concentration risk is calculated. Must include at least columns for longitude and latitude.
full	A data.frame containing reference points. Must include at least columns for longitude, latitude, and the value of interest to summarize.
value	Column name in full containing the values to be summed.
lon_sub	Column name in sub for longitude (default: lon).
lat_sub	Column name in sub for latitude (default: lat).
lon_full	Column name in full for longitude (default: lon).
lat_full	Column name in full for latitude (default: lat).
radius	Numeric. Radius of the circle in meters. Must be positive (default: 200).
display_progress	Logical. Whether to display a progress bar (TRUE/FALSE). Default is TRUE.

Details

This function uses a C++ backend for efficient distance calculations (Haversine formula). For each point in sub, it finds all points in full within the specified radius and sums their value.

Value

A data.frame equal to sub with an additional numeric column concentration containing the summed values from full.

Author(s)

Martin Haringa

Examples

```
# Target points
sub <- data.frame(location = c("p1", "p2"),
                  lon = c(6.561561, 6.561398),
                  lat = c(53.21369, 53.21326))

# Reference points with values
full <- data.frame(lon = c(6.5614, 6.5620, 6.5630),
                  lat = c(53.2132, 53.2140, 53.2150),
                  amount = c(10, 20, 15))

# Calculate concentration within 100 meters
concentration(sub, full, value = amount, radius = 100)
```

convert_crs_df

Convert Coordinate Reference System (CRS)

Description

Convert Coordinate Reference System (CRS) of a data.frame from one CRS to another.

Usage

```
convert_crs_df(
  df,
  crs_from = 3035,
  crs_to = 4326,
  lon_from = "x",
  lat_from = "y",
  lon_to = "lon",
  lat_to = "lat"
)
```

Arguments

df	data.frame to be converted.
crs_from	CRS code of the original coordinate system (default: 3035).
crs_to	CRS code of the target coordinate system (default: 4326).

lon_from	column name of longitude values in df (default: "x").
lat_from	column name of latitude values in df (default: "y").
lon_to	column name for longitude values in the converted data frame (default: "lon").
lat_to	column name for latitude values in the converted data frame (default: "lat").

Value

data.frame with converted coordinates

Author(s)

Martin Haringa

find_highest_concentration
Find highest concentration

Description

Identifies the central coordinates of a circle with a fixed radius that maximizes the coverage of demand points.

Usage

```
find_highest_concentration(
  df,
  value,
  top_n = 1,
  radius = 200,
  cell_size = 100,
  grid_precision = 1,
  lon = "lon",
  lat = "lat",
  crs_metric = 3035,
  print_progress = TRUE
)
```

Arguments

df	A data.frame containing demand points. Must include at least columns for longitude, latitude, and the value of interest.
value	Column name in df with the value of interest to summarize.
top_n	Positive integer greater or equal to 1 (default is 1). Specifies how many highest concentration circles are returned. If top_n > 1, then after each iteration the points belonging to the highest concentration are removed from df. This prevents the subsequent concentrations from being located in the same area, which would otherwise repeatedly select overlapping points with the largest values.

<code>radius</code>	Numeric. Radius of the circle in meters (default = 200).
<code>cell_size</code>	Numeric. Size of the grid cell in meters (default is 100). Defines the resolution of the initial raster grid. The choice of cell size depends on the size of the study area. For example, for a country the size of the Netherlands, cells of 100 × 100 meters are typically sufficient. For larger areas such as Germany, a cell size of 200 × 200 meters may be more appropriate. The choice of <code>cell_size</code> does not affect the final result, only the computational speed.
<code>grid_precision</code>	Numeric. Precision of the search grid in meters (default is 1). Determines the spacing of sub-points within each raster cell. For example, with <code>cell_size</code> = 100 and <code>grid_precision</code> = 1, 10,000 sub-points (100 × 100) are evaluated per cell. Larger values reduce the number of sub-points (and runtime), but also reduce spatial accuracy.
<code>lon</code>	Column name in <code>df</code> for longitude (default = "lon"). Must be in EPSG:4326.
<code>lat</code>	Column name in <code>df</code> for latitude (default = "lat"). Must be in EPSG:4326.
<code>crs_metric</code>	Numeric. Metric Coordinate Reference System (CRS) used in background calculations. For Europe use EPSG:3035 (default). For the United States use EPSG:6317 . For Asia-Pacific use EPSG:8859 .
<code>print_progress</code>	Logical. Whether to print progress messages (TRUE/FALSE).

Details

A recent regulation by the European Commission mandates insurers to report the maximum insured value of fire risk policies for all buildings partly or fully within a circle of radius 200 meters (see Article 132 - fire risk sub-module - of the Delegated Regulation). This captures the risk of catastrophic fire or explosion, including terrorist attacks.

The problem resembles a Maximal Covering Location Problem (MCLP) with a fixed radius, a classic facility location problem. The goal is to select the best locations to maximize coverage of demand points, ensuring each demand point lies within the radius of at least one selected facility.

Value

A list with two elements:

1. A `data.frame` with the `top_n` highest concentrations.
2. A `data.frame` with the subset of `df` corresponding to those concentrations.

Author(s)

Martin Haringa

References

Commission Delegated Regulation (EU) (2015). Solvency II Delegated Act 2015/35. Official Journal of the European Union, 58:124.

Examples

```
# Find single highest concentration
x <- find_highest_concentration(Groningen, value = "amount")
plot(x)

# Find top 2 concentrations with smaller grid cells
y <- find_highest_concentration(Groningen, "amount",
                               top_n = 2, cell_size = 50)
plot(y)
```

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	<i>Haversine great circle distance</i>
-----------	--

Description

Calculates the shortest distance between two points on the Earth's surface using the Haversine formula, also known as the great-circle distance or "as the crow flies".

Usage

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

Arguments

lat_from	Numeric. Latitude(s) of the starting point(s) in decimal degrees (EPSG:4326).
lon_from	Numeric. Longitude(s) of the starting point(s) in decimal degrees (EPSG:4326).
lat_to	Numeric. Latitude(s) of the destination point(s) in decimal degrees (EPSG:4326).
lon_to	Numeric. Longitude(s) of the destination point(s) in decimal degrees (EPSG:4326).
r	Numeric. Radius of the Earth in meters (default = 6378137).

Details

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

This function is fully vectorized: if multiple coordinates are supplied, it returns a distance for each pair of points.

Value

A numeric vector with distances in the same unit as *r* (default in meters).

Author(s)

Martin Haringa

References

Sinnott, R.W, 1984. Virtues of the Haversine. Sky and Telescope 68(2): 159.

Examples

```
# Single pair
haversine(53.24007, 6.520386, 53.24054, 6.520386)

# Vectorized usage
lat_from <- c(53.24, 52.37)
lon_from <- c(6.52, 4.90)
```

```
lat_to <- c(48.85, 51.92)
lon_to <- c(2.35, 4.48)
haversine(lat_from, lon_from, lat_to, lon_to)
```

highest_concentration *Highest concentration risk*

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.

Usage

```
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	set lowerbound.
radius	radius (in meters) (default is 200m).
grid_distance	distance (in meters) for precision of concentration risk (default is 25m). ‘neighborhood_search()’ can be used to search for coordinates with even higher concentrations in the neighborhood of the highest concentrations.
gh_precision	set precision for geohash.
display_progress	show progress bar (TRUE/FALSE). Defaults to TRUE.

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.

Value

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

Author(s)

Martin Haringa

References

Commission Delegated Regulation (EU) (2015). Solvency II Delegated Act 2015/35. Official Journal of the European Union, 58:124.

Gomes M.I., Afonso L.B., Chibeles-Martins N., Fradinho J.M. (2018). Multi-start Local Search Procedure for the Maximum Fire Risk Insured Capital Problem. In: Lee J., Rinaldi G., Mahjoub A. (eds) Combinatorial Optimization. ISCO 2018. Lecture Notes in Computer Science, vol 10856. Springer, Cham. <doi:10.1007/978-3-319-96151-4_19>

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)

# 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	<i>Sum insured per postal code in the Netherlands</i>
-----------	---

Description

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

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

Author(s)

Martin Haringa

interpolate_spline	<i>Splines on the sphere</i>
--------------------	------------------------------

Description

Spline interpolation and smoothing on the sphere.

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).
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).
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.

Value

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

Author(s)

Martin Haringa

References

[Splines on the sphere](#)

Examples

```
## 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 <- dplyr::left_join(nl_postcode3, pop_df)
choropleth(pop_sf, value = "population_pc4_pred", n = 13)

## End(Not run)
```

knmi_historic_data	<i>Retrieve historic weather data for the Netherlands</i>
--------------------	---

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

```
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;
- 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.

Author(s)

Martin Haringa

Examples

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

 knmi_stations

KNMI stations

Description

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

Usage

```
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
)
```


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

Author(s)

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 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 2021.

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
lon longitude of the gemeente centroid
lat latitude of the gemeente centroid
geometry geometry object of gemeente

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

biggest_20cities pc4 is in one of the following twenty (biggest) cities in the Netherlands: Amsterdam, Rotterdam, 's-Gravenhage, Utrecht, Eindhoven, Tilburg, Groningen, Almere, Breda, Nijmegen, Enschede, Apeldoorn, Haarlem, Amersfoort, Arnhem, 's-Hertogenbosch, Zoetermeer, Zwolle, Maastricht, Leiden.

geometry geometry object of postal code

lon longitude of the 4-digit postal code centroid

lat latitude of the 4-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_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.conc	<i>Automatically create a plot for objects obtained from highest_concentration()</i>
-----------	--

Description

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

Usage

```
## S3 method for class 'conc'
plot(
  x,
  grid_points = TRUE,
  legend_title = NULL,
  palette = "viridis",
  legend_position = "bottomleft",
  providers = c("CartoDB.Positron", "nlmaps.luchtfoto"),
  ...
)
```

Arguments

x	object of class ‘conc’ 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")
providers	providers to show. See ‘leaflet::providers’ for a list.
...	additional arguments affecting the interactive map produced

Value

Interactive view of geohashes with highest concentrations

Author(s)

Martin Haringa

plot.concentration	<i>Automatically create a plot for objects obtained from find_highest_concentration()</i>
--------------------	---

Description

Automatically create a plot for objects obtained from find_highest_concentration().

Usage

```
## S3 method for class 'concentration'
plot(
  x,
  type = c("concentration", "focal", "rasterized", "updated_focal"),
  color1 = NULL,
  max.rad = 20,
  ...
)
```

Arguments

x	x object of class concentration obtained from highest_concentration()
type	is one of "concentration" (default), "rasterized", "focal", "updated_focal". See details for more information.
color1	color when one concentration is plotted (default is "#4B0055").
max.rad	maximal radius for size of circles in plot (default is 20).
...	additional arguments.

Details

More info for type:

1. "concentration": this is..
2. "focal": this is..
3. "rasterized": this is..
4. "updated_focal": this is..

Author(s)

Martin Haringa

Examples

```
x <- find_highest_concentration(Groningen, "amount")
plot(x, "concentration")
plot(x, "rasterized")
plot(x, "focal")
plot(x, "updated_focal")
```

plot.neighborhood	<i>Automatically create a plot for objects obtained from neighborhood_gh_search()</i>
-------------------	---

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",
  providers = c("CartoDB.Positron", "nlmaps.luchtfoto"),
  ...
)
```

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")
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
providers	providers to show. See ‘leaflet::providers’ for a list.
...	additional arguments affecting the interactive map produced

Value

Interactive view of highest concentration on map

Author(s)

Martin Haringa

plot_points	Create interactive point map
-------------	------------------------------

Description

Creates an interactive map for a data.frame containing point coordinates, colored by a selected variable.

Usage

```
plot_points(df, value, lon = "lon", lat = "lat", crs = 4326, at = NULL)
```

Arguments

df	A data.frame containing columns for longitude and latitude.
value	A string giving the name of the column in df to be visualized.
lon	A string with the name of the column containing longitude values. Default is "lon".
lat	A string with the name of the column containing latitude values. Default is "lat".
crs	Integer; EPSG code for the coordinate reference system. Default is 4326.
at	Optional numeric vector; breakpoints used for visualization.

Value

An interactive mapview object.

Examples

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

points_in_circle	<i>Find points within a circle around a center coordinate</i>
------------------	---

Description

This function selects rows from a data frame whose longitude/latitude coordinates fall within a given radius (in meters) from a specified center point. It also calculates the distance of each point to the center.

Usage

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

Arguments

data	A data frame containing at least longitude and latitude columns.
lon_center	Numeric scalar, longitude of the circle center.
lat_center	Numeric scalar, latitude of the circle center.
lon	Name of the longitude column in data.
lat	Name of the latitude column in data.
radius	Numeric, circle radius in meters. Default is 200.
sort	Logical, if TRUE (default) results are sorted by distance from the center (closest first). If FALSE, the order of data is preserved.

Value

A data frame subset of data with an extra column `distance_m` giving the distance to the center point.

Author(s)

Martin Haringa

Examples

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

points_in_circle_vec *Filter observations within circle (vectorized)*

Description

Filter all observations in a data.frame that fall within a circle of a specified radius drawn around a given latitude and longitude point.

Usage

```
points_in_circle_vec(  
  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. Representing the longitude of the circle's center.
lat_center	numeric. Representing the latitude of the circle's center.
lon	column name in data containing longitudes (default is lon).
lat	column name in data containing latitudes (default is lat).
radius	radius of the circle in meters (default is 200m).

Value

A subset of the input data.frame containing only the observations that fall within the specified circle.

Author(s)

Martin Haringa

Examples

```
points_in_circle_vec(Groningen, lon_center = c(6.571561, 6.56561),  
  lat_center = c(53.21326, 53.20326), radius = 60)
```

points_to_polygon	<i>Map points to polygons</i>
-------------------	-------------------------------

Description

Join a data.frame containing coordinates (longitude and latitude) to polygon geometries. Arithmetic operations are then applied to the attributes of the joined coordinates 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 representing the polygon geometries.
df	data.frame containing coordinates (column names should be 'lon' and 'lat')
oper	arithmetic operation to be applied on the polygon level.
crs	coordinate reference system (default is 4326).
outside_print	logical indicating whether to print points that are not within a polygon (default is FALSE).

Value

An object of class sf

Author(s)

Martin Haringa

Examples

```
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, [9](#)
- insurance, [13](#)
- knmi_stations, [16](#)
- nl_corop, [17](#)
- nl_gemeente, [18](#)
- nl_postcode2, [19](#)
- nl_postcode3, [19](#)
- nl_postcode4, [20](#)
- nl_provincie, [21](#)

choropleth, [2](#)

choropleth_ggplot2, [3](#)

concentration, [5](#), [11](#)

convert_crs_df, [6](#)

find_highest_concentration, [7](#)

Groningen, [9](#)

haversine, [10](#)

highest_concentration, [11](#), [16](#)

insurance, [13](#)

interpolate_spline, [13](#)

knmi_historic_data, [15](#)

knmi_stations, [16](#)

neighborhood_gh_search, [16](#)

nl_corop, [17](#)

nl_gemeente, [18](#)

nl_postcode2, [19](#)

nl_postcode3, [19](#)

nl_postcode4, [20](#)

nl_provincie, [21](#)

plot.conc, [22](#)

plot.concentration, [23](#)

plot.neighborhood, [24](#)

plot_points, [25](#)

points_in_circle, [26](#)

points_in_circle_vec, [27](#)

points_to_polygon, [28](#)