Package 'spatialrisk'

May 9, 2026

Type Package

Title Spatial Concentration and Radius-Based Risk Calculations

Version 0.8.0

Author Martin Haringa [aut, cre]

Maintainer Martin Haringa <mtharinga@gmail.com>

BugReports https://github.com/mharinga/spatialrisk/issues

Description Provides methods for spatial concentration and radius-based risk calculations. The package focuses on efficient determination of the sum of observations within a given radius, identifying local concentration hotspots, and aggregating point data to polygon geometries. These methods are useful for applications such as insurance, urban analytics, environmental exposure analysis, and other spatial point pattern workflows. The underlying maximum covering problem is described by Church (1974) <doi:10.1007/BF01942293>.

License GPL (>= 2)

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

LazyData true

LinkingTo Rcpp, RcppProgress

Depends R (>= 4.1.0)

Imports data.table, dplyr, fs, lifecycle, Rcpp, RcppProgress, rlang, sf, terra, units

Encoding UTF-8

RoxygenNote 7.3.3

Suggests classInt, colourvalues, GenSA, geohashTools, ggplot2, knitr, leafem, leafgl, leaflet, mapview, mgcv, rmarkdown, testthat, tmap, vroom

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-05-04 12:50:02 UTC

Contents

choropleth . . . 2
choropleth_ggplot2 . . . 4
concentration . . . 5
concentration_hotspot . . . 7
convert_crs_df . . . 9
Groningen . . . 10
haversine . . . 11
highest_concentration . . . 12
insurance . . . 14
interpolate_spline . . . 15
knmi_historic_data . . . 16
knmi_stations . . . 17
neighborhood_gh_search . . . 18
nl_corop . . . 19
nl_gemeente . . . 20
nl_postcode2 . . . 20
nl_postcode3 . . . 21
nl_postcode4 . . . 22
nl_provincie . . . 23
plot.conc . . . 23
plot.hotspot . . . 24
plot.neighborhood . . . 25
plot_points . . . 27
points_in_circle . . . 28
points_to_polygon . . . 29
spatialrisk-deprecated . . . 31

Index **32**

`choropleth` *Create choropleth map*

Description

Creates a choropleth map from an 'sf' object, for example one produced by `summarise_points_by_polygon()`. Polygons are shaded according to values in a specified column, with clustering based on the Fisher–Jenks algorithm.

Usage

```
choropleth(
  data,
  value = "output",
  id = NULL,
  mode = c("plot", "view"),
  n = 7,
```

```
  legend_title = "Value",
  palette = "viridis",
  id_name = NULL,
  ...
)
```

Arguments

`data`	An object of class `sf`.
`value`	A string giving the name of the column used to shade the polygons.
`id`	Optional string giving the name of the column containing polygon IDs used for tooltips.
`mode`	A string indicating whether to create a static map (`"plot"`, default) or an interactive map (`"view"`).
`n`	Integer; number of clusters. Default is `7`.
`legend_title`	A string giving the legend title.
`palette`	A palette name or vector of colors. See `tmaptools::palette_explorer()` for available palettes. Prefix the name with `"-"` to reverse the order. Default is `"viridis"`.
`id_name`	Deprecated. Use `id` instead.
`...`	Additional arguments passed to `tmap::tm_polygons()`.

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 <- summarise_points_by_polygon(nl_provincie, insurance, "amount")
choropleth(test, value = "amount_sum")
choropleth(test, value = "amount_sum", id = "areaname", mode = "view")
```

`choropleth_ggplot2` *Choropleth map of an sf object with ggplot2*

Description

Deprecated. Use [choropleth()] instead.

Usage

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

Arguments

`sf_object`	An object of class `sf` containing polygon geometries.
`value`	Column in `sf_object` used to shade the polygons.
`n`	Integer. Number of clusters to use in Fisher classification.
`dig.lab`	Integer. Number of digits to display in legend labels.
`legend_title`	Character. Title for the legend.
`option`	Character string indicating the colormap option passed to `ggplot2`.
`direction`	Numeric. Order of colors in the scale.

Details

‘choropleth_ggplot2()‘ is deprecated. Use [choropleth()] instead.

Value

A `ggplot` object containing the choropleth map.

`concentration` *Sum values within a radius around target coordinates*

Description

Calculates the sum of all observations from a reference data set that fall within a given radius (in meters) of each target point.

Usage

```
concentration(
  sub,
  full,
  value,
  lon_sub = lon,
  lat_sub = lat,
  lon_full = lon,
  lat_full = lat,
  radius = 200,
  display_progress = TRUE,
  result_col = "radius_sum"
)

radius_sum(
  targets,
  reference,
  value,
  lon_targets = "lon",
  lat_targets = "lat",
  lon_reference = "lon",
  lat_reference = "lat",
  radius = 200,
  progress = TRUE,
  result_col = "radius_sum"
)
```

Arguments

`sub`	Deprecated. Use `targets` instead.
`full`	Deprecated. Use `reference` instead.
`value`	A string giving the name of the column in ‘reference‘ to be summed.
`lon_sub`	Deprecated. Use `lon_targets` instead.
`lat_sub`	Deprecated. Use `lat_targets` instead.
`lon_full`	Deprecated. Use `lon_reference` instead.
`lat_full`	Deprecated. Use `lat_reference` instead.

radius	Numeric. Radius of the circle in meters. Must be positive (default: 200).
display_progress	Deprecated. Use progress in radius_sum() instead.
result_col	A string giving the name of the output column. Default is '"radius_sum"'.
targets	A data.frame of target points for which sums are calculated. Must include at least columns for longitude and latitude.
reference	A data.frame containing reference points. Must include at least columns for longitude, latitude, and the value of interest to summarize.
lon_targets	A string with the name of the longitude column in 'targets'. Default is '"lon"'.
lat_targets	A string with the name of the latitude column in 'targets'. Default is '"lat"'.
lon_reference	A string with the name of the longitude column in 'reference'. Default is '"lon"'.
lat_reference	A string with the name of the latitude column in 'reference'. Default is '"lat"'.
progress	Logical. Whether to display a progress bar. Default is 'TRUE'.

Details

This function uses a C++ backend for efficient distance calculations (Haversine formula).

Value

A data.frame equal to 'targets' with an additional numeric column named by 'result_col' containing the summed values from 'reference'.

Author(s)

Martin Haringa

Examples

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

reference <- data.frame(lon = c(6.5614, 6.5620, 6.5630),
                        lat = c(53.2132, 53.2140, 53.2150),
                        amount = c(10, 20, 15))

radius_sum(targets, reference, value = "amount", radius = 100,
           progress = FALSE)
```

`concentration_hotspot` *Identify fixed-radius concentration hotspots*

Description

Identifies centre coordinates of fixed-radius circles with high local concentration. In insurance applications this can be used to find locations where the total insured value within a prescribed radius is largest.

Usage

```
concentration_hotspot(
  data,
  value,
  top_n = 1,
  radius = 200,
  cell_size = 100,
  grid_precision = 1,
  max_refinement_points = 1000,
  lon = "lon",
  lat = "lat",
  crs_metric = 3035,
  progress = TRUE,
  method = c("continuous", "grid", "observed")
)
```

Arguments

`data`	A data.frame containing point-level exposures. Must include columns for longitude, latitude, and the value of interest.
`value`	A string giving the name of the numeric column in `data` to aggregate within each radius.
`top_n`	Positive integer greater or equal to 1. Specifies how many non-overlapping hotspots are returned. Default is `1`.
`radius`	Numeric. Radius of the circle in meters. This is typically the regulatory or scenario radius. Default is `200`.
`cell_size`	Numeric. Size of the initial screening cells in meters. This is used by `method = "continuous"` and `method = "grid"`. Smaller values give a finer initial search but increase computation time. `method = "observed"` searches observed point locations and does not use this value as a search-grid resolution. Default is `100`.
`grid_precision`	Numeric. Approximate spacing in meters used for grid-based refinement. This is used by `method = "grid"` and by `method = "continuous"` only when the local subset is larger than `max_refinement_points` and the method falls back to grid refinement. It is not used by `method = "observed"`. Smaller values evaluate more candidate centres and increase search precision. Default is `1`.

`max_refinement_points`
: Positive integer. Maximum number of local points used for pair-intersection refinement. If the local subset contains more points, `method = "continuous"` automatically falls back to the grid refinement used by `method = "grid"`. Default is `1000`.

`lon`
: A string giving the longitude column in `data`. Default is `"lon"`.

`lat`
: A string giving the latitude column in `data`. Default is `"lat"`.

`crs_metric`
: Numeric. EPSG code for a projected CRS with meter units, used for distances, buffers, raster cells, and pair-intersection calculations. The default `3035` is ETRS89 / LAEA Europe and is a suitable default for Europe-wide applications. For other regions, choose a metric CRS appropriate to the study area, for example a local UTM zone, `5070` for the conterminous United States, or `3577` for Australia. For Asian portfolios there is no single universal choice; use a national projected CRS or the relevant UTM zone. Default is `3035`.

`progress`
: Logical. Whether to print progress messages for the main hotspot search steps. This is useful for larger portfolios and for `top_n > 1`. Default is `TRUE`.

`method`
: Hotspot search strategy. `"continuous"` is the default and searches for a centre that may lie between observed points. `"observed"` searches only observed point locations as candidate centres. `"grid"` uses the original grid-refinement workflow.

Details

The default `method = "continuous"` first uses terra rasterisation and focal sums to locate an approximate hotspot area. It then refines the local result by evaluating observed local points and the circle centres implied by local point pairs. The local refinement subset is chosen automatically around the terra-selected approximate centre, using a conservative margin based on `radius` and `cell_size`. If more than `max_refinement_points` local points are involved, it falls back to the grid refinement used by `method = "grid"`. In that fallback case, `grid_precision` controls the local refinement grid; otherwise the pair-intersection step does not use `grid_precision`. The pair-refined result is exact only within the terra-selected local search area. The `"observed"` method is fast and deterministic, but can miss a larger hotspot when the optimal centre lies between observed points. The `"grid"` method uses a grid-based search with local refinement; smaller `grid_precision` values generally increase precision and computation time.

Value

An object of class `hotspot`. The main components are `hotspots`, containing the selected centre coordinates and summed values, and `contributing_points`, containing the points inside the selected hotspot radii. The summed value column is named from `value`; for example, `value = "amount"` creates an `amount_sum` column. In `contributing_points`, `data_row` gives the row number of the contributing point in the original input data.

Author(s)

Martin Haringa

Examples

```
portfolio <- Groningen[1:200, c("lon", "lat", "amount")]

hotspot <- concentration_hotspot(
  portfolio,
  value = "amount",
  radius = 200,
  cell_size = 100,
  progress = FALSE,
  top_n = 2
)

hotspot$hotspots
head(hotspot$contributing_points)

observed_hotspot <- concentration_hotspot(
  portfolio,
  value = "amount",
  radius = 200,
  method = "observed",
  progress = FALSE
)

rbind(
  continuous = hotspot$hotspots[1, ],
  observed = observed_hotspot$hotspots
)
```

---

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

Groningen *Example addresses in Groningen*

Description

A sample of addresses in Groningen with point coordinates in EPSG:4326 and an example numeric amount column.

Usage

```
Groningen
```

Format

A data frame with 25,000 rows and 9 variables:

street Street name.

number House number.

letter House letter.

suffix House number suffix.

postal_code Postal code.

city City name.

lon Longitude.

lat Latitude.

amount Example numeric amount.

Details

The `amount` column is an example value used in package examples and tests.

Source

BAG, the Dutch registry for addresses and buildings (Basisregistratie Adressen en Gebouwen), adapted for package examples.

`haversine` *Haversine great-circle distance*

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

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

Value

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

References

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

Examples

```
haversine(53.24007, 6.520386, 53.24054, 6.520386)

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` *Example insurance portfolio*

Description

A sample insurance portfolio with postal codes, insured amounts, population at 4-digit postcode level, and point coordinates in EPSG:4326.

Usage

```
insurance
```

Format

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

postcode 6-digit postal code.

population_pc4 Population for the corresponding 4-digit postcode area.

amount Insured amount.

lon Longitude of the corresponding 6-digit postal code.

lat Latitude of the corresponding 6-digit postal code.

Details

This dataset is intended for examples and tests of spatial concentration workflows.

Author(s)

Martin Haringa

`interpolate_spline` *Interpolate values using spherical splines*

Description

Deprecated. Spline interpolation and smoothing on the sphere. This function is outside the main scope of **spatialrisk** and will be removed in a future release.

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.
`value`	Column with values in `observations`.
`lon_obs`	Column in `observations` with longitude.
`lat_obs`	Column in `observations` with latitude.
`lon_targets`	Column in `targets` with longitude.
`lat_targets`	Column in `targets` with latitude.
`k`	Basis dimension. For small data sets reduce `k` manually.

Value

Object equal to `targets` with an extra prediction column.

References

Splines on the sphere

knmi_historic_data *Retrieve historic weather data for the Netherlands*

Description

This function retrieves historic hourly weather data collected by official KNMI weather stations. See knmi_stations for the station metadata included in this package.

Usage

```
knmi_historic_data(
  startyear,
  endyear,
  stations = NULL,
  progress = interactive()
)
```

Arguments

startyear, endyear	Start and end year for the historic weather data. Both must be single whole years. KNMI hourly data is available from 1951.
stations	Optional station IDs to download. The default, NULL, downloads all stations in knmi_stations.
progress	Should a progress bar be shown? Defaults to interactive().

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 (EPSG:4326);
- lat = Latitude of station (EPSG:4326).

Details

The data is downloaded from KNMI when the function is called. This requires an internet connection and may take some time when many stations or years are requested.

Value

Data frame containing weather data and metadata for weather station locations.

Author(s)

Martin Haringa

Examples

```
## Not run:
knmi_historic_data(2015, 2019, stations = c(260, 280))

## End(Not run)
```

---

knmi_stations *KNMI weather stations*

Description

A data frame containing station IDs and location metadata for official KNMI weather stations in the Netherlands.

Usage

```
knmi_stations
```

Format

A data frame with 50 rows and 7 variables:

station Station ID.

city City where the station is located.

lon Longitude of the station in EPSG:4326.

lat Latitude of the station in EPSG:4326.

altitude Altitude of the station in meters.

X Projected X coordinate of the station in EPSG:32631.

Y Projected Y coordinate of the station in EPSG:32631.

Author(s)

Martin Haringa

Source

Royal Netherlands Meteorological Institute (KNMI), adapted for package examples.

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` *COROP regions in the Netherlands*

Description

An `sf` object with COROP region geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_corop
```

Format

A simple feature object with 40 rows and 5 variables:

corop_nr COROP number.

areaname COROP region name.

geometry COROP region geometry.

lon Longitude of the COROP centroid.

lat Latitude of the COROP centroid.

Details

COROP regions are regional areas used for analytical purposes by, among others, Statistics Netherlands.

Author(s)

Martin Haringa

Source

Statistics Netherlands (CBS), adapted for package examples.

nl_gemeente *Municipalities in the Netherlands*

Description

An `sf` object with municipal geometries for the Netherlands in 2021. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_gemeente
```

Format

A simple feature object with 380 rows and 6 variables:

id Municipality identifier.

code Municipality code.

areaname Municipality name.

lon Longitude of the municipality centroid.

lat Latitude of the municipality centroid.

geometry Municipality geometry.

Author(s)

Martin Haringa

Source

Statistics Netherlands (CBS), adapted for package examples.

nl_postcode2 *Two-digit postcode regions in the Netherlands*

Description

An `sf` object with 2-digit postcode region geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_postcode2
```

Format

A simple feature object with 90 rows and 4 variables:

areaname 2-digit postcode area.

geometry Postcode region geometry.

lon Longitude of the 2-digit postcode centroid.

lat Latitude of the 2-digit postcode centroid.

Details

Postal codes in the Netherlands are alphanumeric and consist of four digits followed by two uppercase letters. This object aggregates those codes to their first two digits.

Author(s)

Martin Haringa

Source

Adapted from Dutch postcode boundary data for package examples.

---

nl_postcode3 *Three-digit postcode regions in the Netherlands*

Description

An `sf` object with 3-digit postcode region geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_postcode3
```

Format

A simple feature object with 799 rows and 4 variables:

areaname 3-digit postcode area.

geometry Postcode region geometry.

lon Longitude of the 3-digit postcode centroid.

lat Latitude of the 3-digit postcode centroid.

Details

Postal codes in the Netherlands are alphanumeric and consist of four digits followed by two uppercase letters. This object aggregates those codes to their first three digits.

Author(s)

Martin Haringa

Source

Adapted from Dutch postcode boundary data for package examples.

---

nl_postcode4 *Four-digit postcode regions in the Netherlands*

Description

An `sf` object with 4-digit postcode region geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_postcode4
```

Format

A simple feature object with 4053 rows and 7 variables:

pc4 4-digit postcode.

areaname Name of corresponding 4-digit postcode area.

city City name.

biggest_20cities Whether the area is in one of the twenty largest cities in the Netherlands.

geometry Postcode region geometry.

lon Longitude of the 4-digit postcode centroid.

lat Latitude of the 4-digit postcode centroid.

Details

Postal codes in the Netherlands are alphanumeric and consist of four digits followed by two uppercase letters. This object aggregates those codes to their first four digits.

Author(s)

Martin Haringa

Source

Adapted from Dutch postcode boundary data for package examples.

`nl_provincie` *Provinces in the Netherlands*

Description

An `sf` object with province geometries for the Netherlands. Centroid coordinates are included in EPSG:4326.

Usage

```
nl_provincie
```

Format

A simple feature object with 12 rows and 4 variables:

areaname Province name.

geometry Province geometry.

lon Longitude of the province centroid.

lat Latitude of the province centroid.

Author(s)

Martin Haringa

Source

Statistics Netherlands (CBS), adapted for package examples.

`plot.conc` *Automatically create a plot for objects obtained from highest_concentration()*

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.hotspot *Plot concentration hotspot results*

Description

Visualise objects returned by `concentration_hotspot()`. The default plot shows hotspot centres, fixed-radius buffers, and the contributing points. For terra-based results, diagnostic raster layers can also be plotted.

Usage

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

Arguments

x	An object of class hotspot.
type	Plot type. "concentration" shows hotspot buffers and contributing points and works for all hotspot search methods. "focal", "rasterized", and "updated_focal" are diagnostic terra layers and are only available for terra-based results.
color1	Optional colour or colours for hotspot buffers and points. If NULL, colours are chosen with grDevices::hcl.colors().
max.rad	Maximum point radius passed to mapview::mapview(). Default is 20.
...	Additional arguments passed to mapview::mapview() for the contributing point layer when type = "concentration", or to the raster mapview call for diagnostic raster layers.

Details

The observed-points hotspot method does not create terra raster or focal objects. For observed-points results, use type = "concentration".

Value

A mapview object.

---

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",
  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

Deprecated.

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

Usage

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

map_points(
  data,
  value = NULL,
  lon = "lon",
  lat = "lat",
  crs = 4326,
  at = NULL,
  layer_name = NULL,
  ...
)
```

Arguments

df	Deprecated. Use data instead. 'plot_points()' was renamed to [map_points()].
value	A string giving the name of the column in data to be visualized. If NULL, points are mapped without a color variable.
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.

layer_name	Optional layer name passed to mapview.
...	Additional arguments passed to mapview::mapview().
data	A data.frame containing columns for longitude and latitude.

Value

An interactive mapview object.

Examples

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

## End(Not run)
```

---

points_in_circle *Find points within radius around one or more center coordinates*

Description

Deprecated.

'points_in_circle()' was renamed to [points_within_radius()].

This function selects rows from a data frame whose longitude/latitude coordinates fall within a given radius (in meters) from one or more specified center points. 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
)

points_within_radius(
  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 or vector, longitude(s) of the circle center(s).
lat_center	Numeric scalar or vector, latitude(s) of the circle center(s).
lon	A string with the name of the longitude column in ‘data‘.
lat	A string with the name of the latitude column in ‘data‘.
radius	Numeric, circle radius in meters. Default is 200.
sort	Logical, if ‘TRUE‘ results are sorted by distance within each center.

Value

A data frame subset of ‘data‘ with an extra column ‘distance_m‘ and if multiple centers are provided, also a column ‘center_index‘.

---

points_to_polygon *Summarise point values by polygon*

Description

Deprecated.

Spatially joins point data to polygon geometries and summarises a numeric point attribute for each polygon.

Usage

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

summarise_points_by_polygon(
  polygons,
  points,
  value,
  fun = sum,
  lon = "lon",
  lat = "lat",
  crs = 4326,
  output_col = NULL,
  na.rm = TRUE,
  outside = c("message", "warning", "ignore"),
  repair_geometry = TRUE
)
```

Arguments

sf_map	Deprecated. Use polygons instead.
df	Deprecated. Use points instead.
oper	Deprecated expression used to aggregate values.
crs	Coordinate reference system of the point coordinates. Default is 4326.
outside_print	Deprecated. Use outside instead. 'points_to_polygon()' was renamed to [summarise_points_by_polygon()].
polygons	An object of class sf containing polygon geometries.
points	A data.frame containing point coordinates and the value to summarise.
value	A string giving the name of the numeric column in points to summarise.
fun	A summary function, such as sum, mean, or length. Default is sum.
lon	A string with the name of the longitude column in points. Default is "lon".
lat	A string with the name of the latitude column in points. Default is "lat".
output_col	Optional string giving the name of the output column. If NULL, the name is created from value and fun, for example "amount_sum" or "amount_mean".
na.rm	Logical. Whether to remove missing values when fun supports an na.rm argument. Default is TRUE.
outside	What to do when points fall outside all polygons: "message" (default), "warning", or "ignore".
repair_geometry	Logical. Whether to try sf::st_buffer(x, 0) if transforming polygon geometries fails. Default is TRUE.

Value

An sf object equal to polygons with an additional summary column.

Examples

```
summarise_points_by_polygon(
  polygons = nl_postcode2,
  points = insurance,
  value = "amount",
  fun = sum
)
```

spatialrisk-deprecated *Deprecated aliases*

Description

These functions are deprecated compatibility wrappers. Use the replacement functions shown in the warning messages.

Deprecated.

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	Deprecated wrapper argument.
value	Deprecated wrapper argument.
top_n	Deprecated wrapper argument.
radius	Deprecated wrapper argument.
cell_size	Deprecated wrapper argument.
grid_precision	Deprecated wrapper argument.
lon	Deprecated wrapper argument.
lat	Deprecated wrapper argument.
crs_metric	Deprecated wrapper argument.
print_progress	Deprecated wrapper argument.
	‘find_highest_concentration()‘ was renamed to [concentration_hotspot()].

Index

∗ **datasets**
  Groningen, 10
  insurance, 14
  knmi_stations, 17
  nl_corop, 19
  nl_gemeente, 20
  nl_postcode2, 20
  nl_postcode3, 21
  nl_postcode4, 22
  nl_provincie, 23

choropleth, 2
choropleth_ggplot2, 4
concentration, 5, *12*
concentration_hotspot, 7
convert_crs_df, 9

find_highest_concentration
  *(*spatialrisk-deprecated*)*, 31

Groningen, 10

haversine, 11
highest_concentration, 12, *18*

insurance, 14
interpolate_spline, 15

knmi_historic_data, 16
knmi_stations, *16*, 17

map_points *(*plot_points*)*, 27

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

plot.conc, 23
plot.hotspot, 24
plot.neighborhood, 25
plot_points, 27
points_in_circle, 28
points_to_polygon, 29
points_within_radius
  *(*points_in_circle*)*, 28

radius_sum *(*concentration*)*, 5

spatialrisk-deprecated, 31
summarise_points_by_polygon
  *(*points_to_polygon*)*, 29