Package: spatialrisk (via r-universe)

August 23, 2024

Type Package

Title Calculating Spatial Risk

Version 0.7.1.9000

Maintainer Martin Haringa <mtharinga@gmail.com>

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

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

Depends R (>= 3.3)

Encoding UTF-8

RoxygenNote 7.3.2

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

Repository https://mharinga.r-universe.dev

RemoteUrl https://github.com/mharinga/spatialrisk

RemoteRef HEAD

RemoteSha 5d35cb18bec6fb305b613d1f201d3696fe44989c

Contents

`choropleth` *Create choropleth map*

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,
  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 correspoding 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)
```

`concentration` *Concentration calculation*

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	data.frame of target points to calculate concentration risk for, including at least columns for longitude and latitude.
full	data.frame containing reference points, where the function finds locations within a radius from the target points. Should include at least columns for longitude, latitude, and the value of interest to summarize.
value	column name with value of interest to summarize in `full`.
lon_sub	column name in `sub` for longitude (default is `lon`).
lat_sub	column name in `sub` for latitude (default is `lat`).
lon_full	column name in `full` for longitude in `full` (default is `lon`).
lat_full	column name in `full` for latitude in `full` (default is `lat`).
radius	numeric. Radius of the circle in meters (default is 200).
display_progress	boolean indicating whether to show progress bar (TRUE/FALSE). Defaults to TRUE.

Value

A data.frame equal to `sub` including an additional column `concentration`.

Author(s)

Martin Haringa

Examples

```
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)
```

`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

Determines the central coordinates of a circle with a constant 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`	data.frame. Should include at least columns for longitude, latitude, and the value of interest to summarize.
`value`	column name with value of interest to summarize in `df`.
`top_n`	positive integer value greater or equal to 1 (default is 1).
`radius`	numeric. Radius of the circle in meters (default is 200).
`cell_size`	numeric. Size of cell in meters (default is 100).
`grid_precision`	numeric. Precision of grid in meters (default is 1).
`lon`	column name in `df` with longitude (default is "lon"). Should be in EPSG:4326.
`lat`	column name in `df` with latitude (default is "lat"). Should be in EPSG:4326.
`crs_metric`	numeric. The metric Coordinate Reference System (CRS) is used solely in the background calculations. For European coordinates, EPSG:3035 (default) is recommended. For the United States, EPSG:6317 can be utilized. For Asia and the Pacific regions, EPSG:8859 is recommended.
`print_progress`	print progress iteration steps.

Details

A recent regulation by the European Commission mandates insurance companies to report the maximum value of insured fire risk policies for all buildings partially or fully situated within a circle with a radius of 200 meters (see Article 132 - fire risk sub-module - of the Delegated Regulation). This article captures the risk of catastrophic fire or explosion, including as a result of terrorist attacks. The sub-module is based on the scenario that the insurance or reinsurance undertaking incurs a loss equal to the capital insured for each building located partly or fully within a radius of 200 meters.

This problem resembles a Maximal Covering Location Problem (MCLP) with a fixed radius, belonging to the category of facility location problems. The main aim is to select the best locations for a predetermined number of facilities to achieve maximum coverage of demand points within a specified radius of each facility. In essence, the objective is to identify optimal facility locations to cover as many demand points as feasible, while ensuring that each demand point falls within the designated distance (radius) of at least one facility.

Value

A list with two elements:

1. A data.frame containing the `top_n` concentrations as specified by `top_n`.
2. A data.frame containing the rows from `df` that correspond to the `top_n` 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

```
x <- find_highest_concentration(Groningen, "amount")
plot(x)

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 adresses (Basisregistratie adressen en gebouwen).

`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`	Latitude of the starting point.
`lon_from`	Longitude of the starting point.
`lat_to`	Latitude of the destination point.
`lon_to`	Longitude of the destination point.
`r`	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.

Value

Vector of 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

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

`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.

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` *Splines on the sphere*

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` *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

```
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

`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 Coordinatiecommissie 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.concentration *Automatically create a plot for objects obtained from* `find_highest_concentration()`

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_points` *Create map with points*

Description

Create map for a data.frame containing points.

Usage

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

Arguments

`df`	data.frame containing columns for longitude and latitude.
`value`	column in `df` to be visualized.
`lon`	column in `df` containing longitude values.
`lat`	column in `df` containing latitude values.
`crs`	crs code for the coordinate reference system (default is 4326).
`at`	the breakpoints used for visualisation.

Examples

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

## End(Not run)
```

`points_in_circle` *Filter observations within circle*

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(
  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(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` *Map points to polygons*

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