Package ‘stplanr’

September 17, 2019

Type Package
Title Sustainable Transport Planning
Version 0.3.1
Maintainer Robin Lovelace <robin0x@gmail.com>
Description Tools for transport planning with an emphasis on spatial transport data and non-motorized modes. Enables common transport planning tasks including: downloading and cleaning transport datasets; creating geographic “desire lines” from origin-destination (OD) data; route assignment, locally and via interfaces to routing services such as <http://cyclestreets.net/>; calculation of route segment attributes such as bearing and aggregate flow; and ‘travel watershed’ analysis. See Lovelace and Ellison (2018) <doi:10.32614/RJ-2018-053>.
License MIT + file LICENSE
BugReports https://github.com/ropensci/stplanr/issues
LazyData yes
Depends R (>= 3.0.2)
Imports sp (>= 1.3.1), curl (>= 3.2), readr (>= 1.1.1), dplyr (>= 0.7.6), httr (>= 1.3.1), jsonlite (>= 1.5), stringi (>= 1.2.4), stringr (>= 1.3.1), lubridate (>= 1.7.4), maptools (>= 0.9.3), raster (>= 2.6.7), rgeos (>= 0.3.28), openxlsx (>= 4.1.0), methods, R.utils (>= 2.7.0), geosphere (>= 1.5.7), Rcpp (>= 0.12.1), igraph (>= 1.2.2), nabor (>= 0.5.0), rlang (>= 0.2.2), lwgeom (>= 0.1.4), sf (>= 0.6.3), magrittr
LinkingTo RcppArmadillo (>= 0.9.100.5.0), Rcpp (>= 0.12.18)
Suggests testthat (>= 2.0.0), knitr (>= 1.20), rmarkdown (>= 1.10), dodgr (>= 0.0.3), stats19, cyclestreets, pbapply, leaflet, rgdal, pct, tmap, bench
VignetteBuilder knitr
URL https://github.com/ropensci/stplanr

https://docs.ropensci.org/stplanr/

SystemRequirements GNU make
RoxygenNote 6.1.1
Encoding UTF-8
NeedsCompilation yes
Author Robin Lovelace [aut, cre] (<https://orcid.org/0000-0001-5679-6536>),
  Richard Ellison [aut],
  Malcolm Morgan [aut] (<https://orcid.org/0000-0002-9488-9183>),
  Barry Rowlingson [ctb] (Author of overline),
  Nick Bearman [ctb] (Co-author of gclip),
  Nikolai Berkoff [ctb] (Co-author of line2route),
  Scott Chamberlain [rev] (Scott reviewed the package for rOpenSci, see
  https://github.com/ropensci/onboarding/issues/10),
  Mark Padgham [ctb]
Repository CRAN
Date/Publication 2019-09-17 21:50:02 UTC

R topics documented:

- stplanr-package ........................................... 5
- angle_diff ................................................. 5
- as_sf_fun ................................................ 6
- bbox_scale ............................................... 7
- buff_geo ................................................ 7
- calc_catchment ......................................... 8
- calc_catchment_sum .................................... 10
- calc_network_catchment ................................. 13
- ca_local .................................................. 15
- cents ..................................................... 16
- crs_select_aeq ......................................... 17
- decode_gl ............................................... 18
- destination_zones ....................................... 19
- dist_google ............................................. 20
- dl_stats19 ............................................... 21
- find_network_nodes ..................................... 22
- flow ...................................................... 23
- flowlines ................................................ 24
- flow_dests .............................................. 25
- format_stats19_ac ..................................... 25
- format_stats19_ca ..................................... 26
- format_stats19_ve ..................................... 27
- gclip ..................................................... 27
- geo_bb ................................................... 28
- geo_bb_matrix .......................................... 29
- geo_buffer ............................................... 30
- geo_code ................................................. 30
- geo_length .............................................. 31
R topics documented:

geo_projected .................................................. 32
geo_select_aeq .................................................. 32
gsection .......................................................... 33
gtf2sldf .......................................................... 35
islines ............................................................. 36
is_linepoint ....................................................... 36
line2df ............................................................. 37
line2route ........................................................ 38
line2routeRetry ............................................... 39
lineLabels ......................................................... 40
line_bearing ...................................................... 41
line_length ....................................................... 41
line_match ......................................................... 42
line_midpoint .................................................... 43
line_sample ....................................................... 43
line_segment ...................................................... 44
line_to_points .................................................. 45
line_via ........................................................... 45
locate2spdf ....................................................... 46
l_poly .............................................................. 47
mapshape ......................................................... 48
mapshape_available ........................................... 49
mats2line ........................................................ 50
nearest2spdf ..................................................... 50
nearest_cyclestreets ....................................... 51
nearest_google ................................................ 52
nearest_osm ..................................................... 53
n_sample_length ................................................ 54
n_vertices ......................................................... 55
od2line ............................................................ 56
od2odf ............................................................. 57
odmatrix_to_od ................................................. 58
od_aggregate ..................................................... 59
od_aggregate_from .......................................... 60
od_aggregate_to ............................................... 61
od_coords ......................................................... 62
od_coords2line ................................................ 63
od_data_sample ................................................. 64
od_dist ............................................................ 64
od_id ............................................................... 65
od_id_order ...................................................... 66
od_oneway ........................................................ 67
od_radiation ..................................................... 68
od_to_odmatrix ............................................... 69
onewaygeo ......................................................... 70
onewayid .......................................................... 71
osm_net_example ............................................... 73
overline .................................................. 73
overline2 .............................................. 74
overline_intersection .................................. 76
plot.sfNetwork,ANY-method ............................ 77
plot.SpatialLinesNetwork,ANY-method ............... 78
points2flow ............................................. 78
points2line ............................................ 79
points2odf ............................................. 80
quadrant ................................................. 80
read_stats19_ac ....................................... 81
read_stats19_ca ....................................... 82
read_stats19_ve ....................................... 83
read_table_builder .................................. 83
reproject .............................................. 84
route .................................................. 85
routes_fast .......................................... 86
routes_slow .......................................... 86
route_cyclestreet .................................. 87
route_dodgr .......................................... 88
route_graphhopper ................................ 89
route_local .......................................... 91
route_network ...................................... 92
route_osrm ......................................... 92
route_transportapi_public ......................... 93
sfNetwork-class .................................. 95
sln2points ........................................... 95
SpatialLinesNetwork ................................. 96
SpatialLinesNetwork-class ......................... 97
sp_aggregate ....................................... 97
stplanr-deprecated ................................ 99
summary.sfNetwork-method ....................... 99
summary.SpatialLinesNetwork-method .......... 100
sum_network_links ................................ 100
sum_network_routes ................................ 101
table2matrix ....................................... 102
toptailgs ........................................... 103
toptail_buff ....................................... 104
update_line_geometry ................................ 105
viaroute .......................................... 105
viaroute2sldf ...................................... 107
weightfield ......................................... 108
writeGeoJSON ...................................... 109
zones ................................................. 109

Index 110
Description
The stplanr package provides functions to access and analyse data for transportation research, including origin-destination analysis, route allocation and modelling travel patterns.

Interesting functions
- `overline()` - Aggregate overlaying route lines and data intelligently
- `calc_catchment()` - Create a 'catchment area' to show the areas serving a destination
- `route_cyclestreet()` - Finds the fastest routes for cyclists between two places.

Author(s)
Robin Lovelace <rob00x@gmail.com>

See Also
https://github.com/ropensci/stplanr

angle_diff

Description
This function was designed to find lines that are close to parallel and perpendicular to some predefined route. It can return results that are absolute (contain information on the direction of turn, i.e. + or - values for clockwise/anticlockwise), bidirectional (which mean values greater than +/- 90 are impossible).

Usage
angle_diff(l, angle, bidirectional = FALSE, absolute = TRUE)

Arguments
- `l` A spatial lines object
- `angle` an angle in degrees relative to North, with 90 being East and -90 being West. (direction of rotation is ignored).
- `bidirectional` Should the result be returned in a bidirectional format? Default is FALSE. If TRUE, the same line in the opposite direction would have the same bearing
- `absolute` If TRUE (the default) only positive values can be returned
Details

Building on the convention used in `bearing()` and in many applications, North is defined as 0, East as 90 and West as -90.

See Also

Other lines: `geo_toptail, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_sample, line_segment, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptail_buff, toptailgs, update_line_geometry`

Examples

```r
# Find all routes going North-South
a <- angle_diff(flowlines, angle = 0, bidirectional = TRUE, absolute = TRUE)
plot(flowlines)
plot(flowlines[a < 15, ], add = TRUE, lwd = 3, col = "red")
# East-West
plot(flowlines[a > 75, ], add = TRUE, lwd = 3, col = "green")
angle_diff(flowlines_sf[2, ], angle = 0)
```

---

**as_sf_fun**

Convert functions support sf/sp

Description

Convert functions support sf/sp

Usage

```r
as_sf_fun(input, FUN, ...)
```

Arguments

- `input`: Input object - an sf or sp object
- `FUN`: A function that works on sp/sf data
- `...`: Arguments passed to FUN
**bbox_scale**

*Scale a bounding box*

**Description**

Takes a bounding box as an input and outputs a bounding box of a different size, centred at the same point.

**Usage**

```
bbox_scale(bb, scale_factor)
```

**Arguments**

- `bb`: the bounding box or spatial object that will be used to crop
- `scale_factor`: Numeric vector determining how much the bounding box will grow or shrink. Two numbers refer to extending the bounding box in x and y dimensions, respectively. If the value is 1, the output size will be the same as the input.

**See Also**

Other geo: `buff_geo`, `crs_select_aeq`, `decode.gl`, `gclip`, `geo_bb_matrix`, `geo_bb`, `mapshape_available`, `mapshape`, `quadrant`, `reproject`

**Examples**

```
bb <- matrix(c(-1.55, 53.80, -1.50, 53.83), nrow = 2)
bb1 <- bbox_scale(bb, scale_factor = 1.05)
bb2 <- bbox_scale(bb, scale_factor = c(2, 1.05))
bb3 <- bbox_scale(bb, 0.1)
plot(x = bb2[,1], y = bb2[,2])
points(bb1[,1], bb1[,2])
points(bb3[,1], bb3[,2])
points(bb[,1], bb[,2], col = "red")
```

---

**buff_geo**

*Create a buffer of n metres for non-projected 'geographical' spatial data*

**Description**

Solves the problem that buffers will not be circular when used on non-projected data.

**Usage**

```
buff_geo(shp, width, ...)
```
Arguments

shp A spatial object with a geographic CRS (e.g. WGS84) around which a buffer should be drawn

width The distance (in metres) of the buffer (when buffering sp objects)

... Arguments passed to the buffer (see ?rgeos::gBuffer or ?sf::st_buffer for details)

See Also

Other geo: bbox_scale, crs_select_aeq, decode_gl, gclip, geo_bb_matrix, geo_bb, mapshape_available, mapshape, quadrant, reproject

Examples

```r
r <- routes_fast[1:3, ]
buff <- buff_geo(r, width = 100)
plot(buff)
plot(r, add = TRUE)
# Test it works the same on projected data
shp <- sp::spTransform(r, sp::CRS("+init=epsg:27700"))
buff2 <- buff_geo(shp, 50) # test if it works the same on projected data
plot(buff2)
plot(r, add = TRUE) # note they do not show
buff3 <- sp::spTransform(buff2, sp::CRS("+init=epsg:4326"))
plot(buff)
plot(buff3, add = TRUE, col = "black")
```

calc_catchment

Calculate catchment area and associated summary statistics.

Description

Calculate catchment area and associated summary statistics.

Usage

```r
calc_catchment(polygonlayer, targetlayer, calccols, distance = 500, 
projection = paste0("+proj=aea +lat_1=90 +lat_2=-18.416667 ", 
"+lat_0=0 +lon_0=10 +x_0=0 +y_0=0 +ellps=GRS80", 
" +towgs84=0,0,0,0,0,0 +units=m +no_defs"), 
retainAreaProportion = FALSE, dissolve = FALSE, quadsegs = NULL)
```

Arguments

polygonlayer A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.
**calc_catchment**

**targetlayer**
A SpatialPolygonsDataFrame, SpatialLinesDataFrame, SpatialPointsDataFrame, SpatialPolygons, SpatialLines or SpatialPoints object containing the specifications of the facility for which the catchment area is being calculated. If the object contains more than one facility (e.g., multiple cycle paths) the aggregate catchment area will be calculated.

**calccols**
A vector of column names containing the variables in the polygonlayer to be used in the calculation of the summary statistics for the catchment area. If dissolve = FALSE, all other variables in the original SpatialPolygonsDataFrame for zones that fall partly or entirely within the catchment area will be included in the returned SpatialPolygonsDataFrame but will not be adjusted for the proportion within the catchment area.

**distance**
Defines the size of the catchment area as the distance around the targetlayer in the units of the projection (default = 500 metres)

**projection**
The proj4string used to define the projection to be used for calculating the catchment areas or a character string 'austalbers' to use the Australian Albers Equal Area projection. Ignored if the polygonlayer is projected in which case the targetlayer will be converted to the projection used by the polygonlayer. In all cases the resulting object will be reprojected to the original coordinate system and projection of the polygon layer. Default is an Albers Equal Area projection but for more reliable results should use a local projection (e.g., Australian Albers Equal Area project).

**retainAreaProportion**
Boolean value. If TRUE retains a variable in the resulting SpatialPolygonsDataFrame containing the proportion of the original area within the catchment area (Default = FALSE).

**dissolve**
Boolean value. If TRUE collapses the underlying zones within the catchment area into a single region with statistics for the whole catchment area.

**quadsegs**
Number of line segments to use to approximate a quarter circle. Parameter passed to buffer functions, default is 5 for sp and 30 for sf.

**Details**
Calculates the catchment area of a facility (e.g., cycle path) using straight-line distance as well as summary statistics from variables available in a SpatialPolygonsDataFrame with census tracts or other zones. Assumes that the frequency of the variable is evenly distributed throughout the zone. Returns a SpatialPolygonsDataFrame.

**See Also**
Other rnet: SpatialLinesNetwork, calc_catchment_sum, calc_moving_catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overline2, overline.plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, sln2points, sum_network_links, sum_network_routes

**Examples**
```r
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"))
```
unzip(file.path(data_dir, "testcycleway.zip"))
sa1income <- as(sf::read_sf("smallsa1.shp"), "Spatial")
testcycleway <- as(sf::read_sf("testcycleway.shp"), "Spatial")
cway_catch <- calc_catchment(
  polygonlayer = sa1income,
  targetlayer = testcycleway,
  calccols = c("Total"),
  distance = 800,
  projection = "austalbers",
  dissolve = TRUE
)
plot(sa1income)
plot(cway_catch, add = TRUE, col = "green")
plot(testcycleway, col = "red", add = TRUE)
plot(sa1income)
plot(testcycleway)
f = list.files(".", "testcycleway\smallsa1")
file.remove(f)
cway_catch <- calc_catchment(
  polygonlayer = sa1income,
  targetlayer = testcycleway,
  calccols = c("Total"),
  distance = 800,
  projection = "austalbers",
  dissolve = TRUE
)
plot(sa1income$geometry)
plot(testcycleway$geometry, col = "red", add = TRUE)
plot(cway_catch["Total"], add = TRUE)

## End(Not run)

calc_catchment_sum

**Calculate summary statistics for catchment area.**

**Description**

Calculate summary statistics for catchment area.

**Usage**

```r
calc_catchment_sum(polygonlayer, targetlayer, calccols, distance = 500,
  projection = paste0("+proj=aea +lat_1=90 +lat_2=-18.416667",
  " +lat_0=0 +lon_0=10 +x_0=0 +y_0=0",
  " +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"),
  retainAreaProportion = FALSE, quadsegs = NA)
```
Arguments

- **polygonlayer**: A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.

- **targetlayer**: A SpatialPolygonsDataFrame, SpatialLinesDataFrame, SpatialPointsDataFrame, SpatialPolygons, SpatialLines or SpatialPoints object containing the specifications of the facility for which the catchment area is being calculated. If the object contains more than one facility (e.g., multiple cycle paths) the aggregate catchment area will be calculated.

- **calccols**: A vector of column names containing the variables in the polygonlayer to be used in the calculation of the summary statistics for the catchment area.

- **distance**: Defines the size of the catchment area as the distance around the targetlayer in the units of the projection (default = 500 metres)

- **projection**: The proj4string used to define the projection to be used for calculating the catchment areas or a character string 'austalbers' to use the Australian Albers Equal Area projection. Ignored if the polygonlayer is projected in which case the targetlayer will be converted to the projection used by the polygonlayer. In all cases the resulting object will be reprojected to the original coordinate system and projection of the polygon layer. Default is an Albers Equal Area projection but for more reliable results should use a local projection (e.g., Australian Albers Equal Area project).

- **retainAreaProportion**: Boolean value. If TRUE retains a variable in the resulting SpatialPolygonsDataFrame containing the proportion of the original area within the catchment area (Default = FALSE).

- **quadsegs**: Number of line segments to use to approximate a quarter circle. Parameter passed to buffer functions, default is 5 for sp and 30 for sf.

Details

Calculates the summary statistics for a catchment area of a facility (e.g., cycle path) using straight-line distance from variables available in a SpatialPolygonsDataFrame with census tracts or other zones. Assumes that the frequency of the variable is evenly distributed throughout the zone. Returns either a single value if calccols is of length = 1, or a named vector otherwise.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment, calc_moving_catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overline2, overline, plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, sln2points, sum_network_links, sum_network_routes

Examples

```r
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"))
unzip(file.path(data_dir, "testcycleway.zip"))
```
calc_moving_catchment

Calculate summary statistics for all features independently.

description

Calculate summary statistics for all features independently.

Usage

calc_moving_catchment(polygonlayer, targetlayer, calccols,
  distance = 500, projection = "worldalbers",
  retainAreaProportion = FALSE)

Arguments

  polygonlayer  A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.
  targetlayer A SpatialPolygonsDataFrame, SpatialLinesDataFrame or SpatialPointsDataFrame object containing the specifications of the facilities and zones for which the catchment areas are being calculated.
  calccols A vector of column names containing the variables in the polygonlayer to be used in the calculation of the summary statistics for the catchment areas.
  distance Defines the size of the catchment areas as the distance around the targetlayer in the units of the projection (default = 500 metres)
calc_network_catchment

projection

The proj4string used to define the projection to be used for calculating the catchment areas or a character string ‘austalbers’ to use the Australian Albers Equal Area projection. Ignored if the polygon layer is projected in which case the target layer will be converted to the projection used by the polygon layer. In all cases the resulting object will be reprojected to the original coordinate system and projection of the polygon layer. Default is an Albers Equal Area projection but for more reliable results should use a local projection (e.g., Australian Albers Equal Area project).

retainAreaProportion

Boolean value. If TRUE retains a variable in the resulting SpatialPolygonsDataFrame containing the proportion of the original area within the catchment area (Default = FALSE).

Details

Calculates the summary statistics for a catchment area of multiple facilities or zones using straight-line distance from variables available in a SpatialPolygonsDataFrame with census tracts or other zones. Assumes that the frequency of the variable is evenly distributed throughout the zone. Returns the original source dataframe with additional columns with summary variables.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum, calc_catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overline2, overline.plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, sln2points, sum_network_links, sum_network_routes

Examples

```r
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsal.zip"))
unzip(file.path(data_dir, "testcycleway.zip"))
salincome <- readOGR(".", "smallsal")
testcycleway <- readOGR(".", "testcycleway")
calc_moving_catchment(
  polygonlayer = salincome,
  targetlayer = testcycleway,
  calccols = c("Total"),
  distance = 800,
  projection = "austalbers"
)

## End(Not run)
```
Description

Calculate catchment area and associated summary statistics using network.

Usage

calc_network_catchment(sln, polygonlayer, targetlayer, calcccols,
maximpedance = 1000, distance = 100,
projection = paste0("+proj=aea +lat_1=90 +lat_2=-18.416667",
" +lat_0=0 +lon_0=10 +x_0=0 +y_0=0",
" +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"),
retainAreaProportion = FALSE, dissolve = FALSE)

Arguments

sln The SpatialLinesNetwork to use.
polygonlayer A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.
targetlayer A SpatialPolygonsDataFrame, SpatialLinesDataFrame or SpatialPointsDataFrame object containing the specifications of the facilities and zones for which the catchment areas are being calculated.
calccols A vector of column names containing the variables in the polygonlayer to be used in the calculation of the summary statistics for the catchment area. If dissolve = FALSE, all other variables in the original SpatialPolygonsDataFrame for zones that fall partly or entirely within the catchment area will be included in the returned SpatialPolygonsDataFrame but will not be adjusted for the proportion within the catchment area.
maximpedance The maximum value of the network’s weight attribute in the units of the weight (default = 1000).
distance Defines the additional catchment area around the network in the units of the projection. (default = 100 metres)
projection The proj4string used to define the projection to be used for calculating the catchment areas or a character string ‘austalbers’ to use the Australian Albers Equal Area projection. Ignored if the polygonlayer is projected in which case the targetlayer will be converted to the projection used by the polygonlayer. In all cases the resulting object will be reprojected to the original coordinate system and projection of the polygon layer. Default is an Albers Equal Area projection but for more reliable results should use a local projection (e.g., Australian Albers Equal Area project).
retainAreaProportion Boolean value. If TRUE retains a variable in the resulting SpatialPolygonsDataFrame containing the proportion of the original area within the catchment area (Default = FALSE).
dissolve Boolean value. If TRUE collapses the underlying zones within the catchment area into a single region with statistics for the whole catchment area.
Details

Calculates the catchment area of a facility (e.g., cycle path) using network distance (or other weight variable) as well as summary statistics from variables available in a SpatialPolygonsDataFrame with census tracts or other zones. Assumes that the frequency of the variable is evenly distributed throughout the zone. Returns a SpatialPolygonsDataFrame.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum, calc_catchment, calc_moving_catchment, find_network_nodes, gsection, islines, lineLabels, overline2, overline, plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, sln2points, sum_network_links, sum_network_routes

Examples

```r
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"), exdir = tempdir())
unzip(file.path(data_dir, "testcycleway.zip"), exdir = tempdir())
unzip(file.path(data_dir, "sydroads.zip"), exdir = tempdir())
salincome <- readOGR(tempdir(), "smallsa1")
testcycleway <- readOGR(tempdir(), "testcycleway")
sydroads <- readOGR(tempdir(), "roads")
sydnetwork <- SpatialLinesNetwork(sydroads)
calc_network_catchment(
  sln = sydnetwork,
  polygonlayer = salincome,
  targetlayer = testcycleway,
  calccols = c("Total"),
  maximpedance = 800,
  distance = 200,
  projection = "austalbers",
  dissolve = TRUE
)
## End(Not run)
```

**ca_local**  
*SpatialPointsDataFrame representing road traffic deaths*

Description

This dataset represents the type of data downloaded and cleaned using stplanr functions. It represents a very small sample (with most variables stripped) of open data from the UK’s Stats19 dataset.

Usage

data(ca_local)
Format

A SpatialPointsDataFrame with 11 rows and 2 columns

Examples

```r
## Not run:
# Generate data
ac <- read_stats19_ac()
ca <- read_stats19_ca()
ve <- read_stats19_ve()
library(dplyr)
ca_ac <- inner_join(ca, ac)
ca_cycle <- ca_ac %>%
  filter(Casualty_Severity == "Fatal" & !is.na(Latitude)) %>%
  select(Age = Age_of_Casualty, Mode = Casualty_Type, Longitude, Latitude)
ca_sp <- sp::SpatialPointsDataFrame(coords = ca_cycle[3:4], data = ca_cycle[1:2])
data("route_network")
proj4string(ca_sp) <- proj4string(route_network)
bb <- bb2poly(route_network)
ca_local <- ca_sp[bb, ]

## End(Not run)
```

cents

Spatial points representing home locations

Description

These points represent population-weighted centroids of Medium Super Output Area (MSOA) zones within a 1 mile radius of of my home when I was writing this package.

Usage

data(cents)

Format

A spatial dataset with 8 rows and 5 variables

Details

- geo_code the official code of the zone
- MSOA11NM name zone name
- percent_fem the percent female
- avslope average gradient of the zone

Cents was generated from the data repository pct-data: https://github.com/npct/pct-data. This data was accessed from within the pct repo: https://github.com/npct/pct, using the following code:
**Examples**

```r
## Not run:
cents
plot(cents)

## End(Not run)
```

---

**crs_select_aeq**

*Select a custom projected CRS for the area of interest*

**Description**

This function takes a spatial object with a geographic (WGS84) CRS and returns a custom projected CRS focussed on the centroid of the object. This function is especially useful for using units of metres in all directions for data collected anywhere in the world.

**Usage**

```r
crs_select_aeq(shp)
```

**Arguments**

- `shp` A spatial object with a geographic (WGS84) coordinate system

**Details**

The function is based on this stackexchange answer: [http://gis.stackexchange.com/questions/121489](http://gis.stackexchange.com/questions/121489)

**See Also**

Other geo: `bbox_scale`, `buff_geo`, `decode_gl`, `gclip`, `geo_bb_matrix`, `geo_bb`, `mapshape_available`, `mapshape`, `quadrant`, `reproject`

**Examples**

```r
data("routes_fast")
new_crs <- crs_select_aeq(routes_fast)
plot(routes_fast)
rf_projected <- sp::spTransform(routes_fast, new_crs)
plot(rf_projected)
sp::bbox(rf_projected)
line_length <- rgeos::gLength(rf_projected, byid = TRUE)
plot(line_length, rf_projected$length)
cor(line_length, rf_projected$length)
```
**decode_gl**

*Decode Google polyline compressed string*

### Description

Decode Google polyline compressed string

### Usage

```
decode_gl(polyline, precision = 6, forceline = TRUE)
```

### Arguments

- **polyline**
  A character string or vector of character strings containing the encoded polyline to be decoded.

- **precision**
  An integer indicating the number of decimals in the initial encoded coordinates. Default is 6 (for OSRM default).

- **forceline**
  Boolean value indicating if the returned coordinates should be a line (i.e., minimum two points). Default is TRUE.

### Details

An implementation of the Google Maps Encoded Polyline Algorithm for decoding strings. Returns a dataframe if polyline is of length 1 and a list of dataframes otherwise.

See the more recent googlePolylines package, which is faster: [https://github.com/SymbolixAU/googlePolylines](https://github.com/SymbolixAU/googlePolylines)

### See Also

Other geo: `bbox_scale, buff_geo, crs_select_aeq, gclip, geo_bb_matrix, geo_bb, mapshape_available, mapshape, quadrant, reproject`

### Examples

```r
## Not run:
decode_gl("_p~iF~ps|U_ulLnnqC_mqNvxq`@", precision = 5)

## End(Not run)
```
destination_zones

Example destinations data

Description

This dataset represents trip destinations on a different geographic level than the origins stored in the object cents.

Usage

data(destination_zones)

Format

A spatial dataset with 87 features

See Also

Other example data: flow_dests, flowlines, flow, route_network, routes_fast, routes_slow

Examples

## Not run:
# This is how the dataset was constructed - see
# http://cowz.geodata.soton.ac.uk/download/
download.file(
  "http://cowz.geodata.soton.ac.uk/download/files/COWZ_EW_2011_BFC.zip",
  "COWZ_EW_2011_BFC.zip"
)
unzip("COWZ_EW_2011_BFC.zip")
wz <- raster::shapefile("COWZ_EW_2011_BFC.shp")
to_remove <- list.files(pattern = "COWZ", full.names = TRUE, recursive = TRUE)
file.remove(to_remove)
proj4string(wz)
wz <- sp::spTransform(wz, proj4string(zones))
destination_zones <- wz[zones, ]
plot(destination_zones)
devtools::use_data(destination_zones)
head(destination_zones@data)
destinations <- rgeos::gCentroid(destinations, byid = TRUE)
destinations <- sp::SpatialPointsDataFrame(destinations, destination_zones@data)
devtools::use_data(destinations, overwrite = TRUE)
destinations_sf <- sf::st_as_sf(destinations)
devtools::use_data(destinations_sf)

## End(Not run)
dist_google

Return travel network distances and time using the Google Maps API

Description

Return travel network distances and time using the Google Maps API

Usage

```
dist_google(from, to, google_api = Sys.getenv("GOOGLEDIST"),
g_units = "metric", mode = c("bicycling", "walking", "driving",
"transit"), arrival_time = "")
```

Arguments

- `from`: Two-column matrix or data frame of coordinates representing latitude and longitude of origins.
- `to`: Two-column matrix or data frame of coordinates representing latitude and longitude of destinations.
- `google_api`: String value containing the Google API key to use.
- `g_units`: Text string, either metric (default) or imperial.
- `mode`: Text string specifying the mode of transport. Can be bicycling (default), walking, driving or transit.
- `arrival_time`: Time of arrival in date format.

Details

Absence authorization, the google API is limited to a maximum of 100 simultaneous queries, and so will, for example, only returns values for up to 10 origins times 10 destinations.

Details

Estimate travel times accounting for the road network - see https://developers.google.com/maps/documentation/distance-matrix/ Note: Currently returns the json object returned by the Google Maps API and uses the same origins and destinations.

See Also

Other od: od2line, od2df, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate
dl_stats19

**Description**

Download Stats19 data

**Usage**

dl_stats19(zip_url = paste0("http://data.dft.gov.uk.s3.amazonaws.com/",  
data_dir = tempdir())

## Examples

```r
## Not run:
# Distances from one origin to one destination
from <- c(-46.3, -23.4)
to <- c(-46.4, -23.4)
dist_google(from = from, to = to, mode = "walking") # not supported on last test
dist_google(from = from, to = to, mode = "driving")
dist_google(from = c(0, 52), to = c(0, 53))
data("cents")
# Distances from between all origins and destinations
dists_cycle <- dist_google(from = cents, to = cents)
dists_drive <- dist_google(cents, cents, mode = "driving")
dists_trans <- dist_google(cents, cents, mode = "transit")
dists_trans_am <- dist_google(cents, cents,  
  mode = "transit",  
  arrival_time = strptime("2016-05-27 09:00:00",  
    format = "%Y-%m-%d %H:%M:%S", tz = "BST"  
  )
)
# Find out how much longer (or shorter) cycling takes than walking
summary(dists_cycle$duration / dists_trans$duration)
# Difference between travelling now and for 9am arrival
summary(dists_trans_am$duration / dists_trans$duration)
odf <- points2odf(cents)
odf <- cbind(odf, dists)
head(odf)
flow <- points2flow(cents)
# show the results for duration (thicker line = shorter)
plot(flow, lwd = mean(odf$duration) / odf$duration)
  dist_google(c("Hereford"), c("Weobley", "Leominster", "Kington"))
  dist_google(c("Hereford"), c("Weobley", "Leominster", "Kington"),  
    mode = "transit", arrival_time = strptime("2016-05-27 17:30:00",  
      format = "%Y-%m-%d %H:%M:%S", tz = "BST"  
    )
  )

## End(Not run)
```
find_network_nodes

Arguments

zip_url The url where the data is stored
data_dir Directory to which to download the file

Details

This convenience function downloads and unzips UK road traffic casualty data. It results in unzipped .csv data in R’s temporary directory.

Ensure you have a fast internet connection and at least 100 Mb space

Examples

## Not run:

dl_stats19()

# Load all stats19 datasets
ac <- read_stats19_ac()
ca <- read_stats19_ca()
ve <- read_stats19_ve()
# now you can analyse the UK’s stats19 data in a single table

## End(Not run)

find_network_nodes

Find graph node ID of closest node to given coordinates

Description

Find graph node ID of closest node to given coordinates

Usage

find_network_nodes(sln, x, y = NULL, maxdist = 1000)

Arguments

sln SpatialLinesNetwork to search.
x Either the x (longitude) coordinate value, a vector of x values, a dataframe or matrix with (at least) two columns, the first for coordinate for x (longitude) values and a second for y (latitude) values, or a named vector of length two with values of ‘lat’ and ‘lon’. The output of geo_code() either as a single result or as multiple (using rbind() ) can also be used.
y Either the y (latitude) coordinate value or a vector of y values.
maxdist The maximum distance within which to match the nodes to coordinates. If the SpatialLinesNetwork is projected then distance should be in the same units as the projection. If longlat, then distance is in metres. Default is 1000.
**Value**

An integer value with the ID of the node closest to \((x, y)\) with a value of `NA` the closest node is further than `maxdist` from \((x, y)\). If \(x\) is a vector, returns a vector of Node IDs.

**Details**

Finds the node ID of the closest point to a single coordinate pair (or a set of coordinates) from a `SpatialLinesNetwork`.

**See Also**

Other rnet: `SpatialLinesNetwork`, `calc_catchment_sum`, `calc_catchment`, `calc_moving_catchment`, `calc_network_catchment`, `gsection`, `islines`, `lineLabels`, `overline2`, `overline`, `plot`, `SpatialLinesNetwork`, `ANY-method`, `plot`, `sfNetwork`, `ANY-method`, `sln2points`, `sum_network_links`, `sum_network_routes`

**Examples**

```r
data(routes_fast)
rlen <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rlen)
find_network_nodes(sln, -1.516734, 53.828)
```

---

**flow**

*data frame of commuter flows*

---

**Description**

This dataset represents commuter flows (work travel) between origin and destination zones (see `cents()`). The data is from the UK and is available as open data: [http://wicid.ukdataservice.ac.uk/](http://wicid.ukdataservice.ac.uk/).

**Usage**

```r
data(flow)
```

**Format**

A data frame with 49 rows and 15 columns

**Details**

The variables are as follows:

- Area.of.residence. id of origin zone
- Area.of.workplace id of destination zone
- All. Travel to work flows by all modes
- \([, 4:15]\). Flows for different modes
• id. unique id of flow

Although these variable names are unique to UK data, the data structure is generalisable and typical of flow data from any source. The key variables are the origin and destination ids, which link to the cents georeferenced spatial objects.

See Also

Other example data: destination_zones, flow_dests, flowlines, route_network, routes_fast, routes_slow

Examples

```r
## Not run:
# This is how the dataset was constructed - see
# https://github.com/npct/pct - if download to ~/repos
flow <- readRDS("~/repos/pct/pct-data/national/flow.Rds")
data(cents)
o <- flow$Area.of.residence %in% cents$geo_code[-1]
d <- flow$Area.of.workplace %in% cents$geo_code[-1]
flow <- flow[o & d, ] # subset flows with o and d in study area
library(devtools)
flow$id <- paste(flow$Area.of.residence, flow$Area.of.workplace)
use_data(flow, overwrite = TRUE)

# Convert flows to spatial lines dataset
flowlines <- od2line(flow = flow, zones = cents)
# use_data(flowlines, overwrite = TRUE)

# Convert flows to routes
routes_fast <- line2route(l = flowlines, plan = "fastest")
routes_slow <- line2route(l = flowlines, plan = "quietest")
use_data(routes_fast)
use_data(routes_slow)
routes_fast_sf <- sf::st_as_sf(routes_fast)
routes_slow_sf <- sf::st_as_sf(routes_slow)

## End(Not run)
```

---

**flowlines**

spatial lines dataset of commuter flows

**Description**

Flow data after conversion to a spatial format with `od2line()` (see `flow()`).

**Format**

A spatial lines dataset with 49 rows and 15 columns
**flow_dests**

**See Also**

Other example data: destination_zones, flow_dests, flow, route_network, routes_fast, routes_slow

---

**flow_dests**

data frame of invented commuter flows with destinations in a different layer than the origins

---

**Description**

data frame of invented commuter flows with destinations in a different layer than the origins

**Usage**

data(flow_dests)

**Format**

A data frame with 49 rows and 15 columns

**See Also**

Other example data: destination_zones, flowlines, flow, route_network, routes_fast, routes_slow

**Examples**

```r
## Not run:
# This is how the dataset was constructed
flow_dests <- flow
flow_dests$Area.of.workplace <- sample(x = destinations$WZ11CD, size = nrow(flow))
flow_dests <- dplyr::rename(flow_dests, WZ11CD = Area.of.workplace)
devtools::use_data(flow_dests)
## End(Not run)
```

---

**format_stats19_ac**

*Format UK 'Stats19' road traffic casualty data*

**Description**

Format UK 'Stats19' road traffic casualty data

**Usage**

format_stats19_ac(ac)
Arguments

ac     Dataframe representing the raw Stats19 data read-in with `read_csv()`.

Details

This is a helper function to format raw stats19 data

Examples

```r
## Not run:
ac <- format_stats19_ac(ac)

## End(Not run)
```

---

`format_stats19_ca` Format UK ‘Stats19’ road traffic casualty data

Description

Format UK ‘Stats19’ road traffic casualty data

Usage

`format_stats19_ca(ca)`

Arguments

ca     Dataframe representing the raw Stats19 data read-in with `read_csv()`.

Details

This is a helper function to format raw stats19 data

Examples

```r
## Not run:
ca <- format_stats19_ca(ca)

## End(Not run)
```
format_stats19_ve  Format UK 'Stats19' road traffic casualty data

Description
Format UK 'Stats19' road traffic casualty data

Usage
format_stats19_ve(ve)

Arguments
ve  Dataframe representing the raw Stats19 data read-in with read_csv().

Details
This is a helper function to format raw stats19 data

Examples
## Not run:
ve <- format_stats19_ve(ve)
## End(Not run)

gclip  Crops spatial object x to the bounding box of spatial object (or matrix) b

Description
This function is a cross between the spatial subsetting functions such as sp::over(), rgeos::gIntersects() etc, and the cropping functions of raster::crop() and rgeos::gIntersection(). The output is the subset of spatial object a with an outline described by a square bounding box. The utility of such a function is illustrated in the following question: http://gis.stackexchange.com/questions/46954/clip-spatial-object-to-bounding-box-in-r/.

Usage
gclip(shp, bb)

Arguments
shp  The spatial object a to be cropped
bb  the bounding box or spatial object that will be used to crop shp
geo_bb

Flexible function to generate bounding boxes

Description

Takes a geographic object or bounding box as an input and outputs a bounding box, represented as a bounding box, corner points or rectangular polygon.

Usage

```r
geo_bb(shp, scale_factor = 1, distance = 0, output = c("polygon", "points", "bb"))
```

Arguments

- `shp`: Spatial object (from sf or sp packages)
- `scale_factor`: Numeric vector determining how much the bounding box will grow or shrink. Two numbers refer to extending the bounding box in x and y dimensions, respectively. If the value is 1, the output size will be the same as the input.
- `distance`: Distance in metres to extend the bounding box by
- `output`: Type of object returned (polygon by default)

See Also

- `bb_scale`

Other geo: `bbox_scale, buff_geo, crs_select_aeq, decode_gl, geo_bb_matrix, mapshape_available, mapshape, quadrant, reproject`
geo_bb_matrix

Create matrix representing the spatial bounds of an object

Description

Converts a range of spatial data formats into a matrix representing the bounding box

Usage

geo_bb_matrix(shp)

Arguments

shp Spatial object (from sf or sp packages)

See Also

Other geo: bbox_scale, buff_geo, crs_select_aeq, decode_g1, gclip, geo_bb, mapshape_available, mapshape, quadrant, reproject

Examples

geo_bb_matrix(routes_fast)
geo_bb_matrix(routes_fast_sf)
geo_bb_matrix(cents[1, ])
geo_bb_matrix(c(-2, 54))
geo_bb_matrix(sf::st_coordinates(cents_sf))
geo_buffer

Perform a buffer operation on a temporary projected CRS

Description

This function solves the problem that buffers will not be circular when used on non-projected data.

Usage

geo_buffer(shp, dist = NULL, width = NULL, ...)

Arguments

shp A spatial object with a geographic CRS (e.g. WGS84) around which a buffer should be drawn

dist The distance (in metres) of the buffer (when buffering simple features)

width The distance (in metres) of the buffer (when buffering sp objects)

... Arguments passed to the buffer (see ?rgeos::gBuffer or ?sf::st_buffer for details)

See Also

buff_geo

Examples

buff_sp <- geo_buffer(routes_fast, width = 100)
class(buff_sp)
plot(buff_sp, col = "red")
routes_fast_sf <- sf::st_as_sf(routes_fast)
buff_sf <- geo_buffer(routes_fast_sf, dist = 50)
plot(buff_sf$geometry, add = TRUE)

geo_code

Convert text strings into points on the map

Description

Generate a lat/long pair from data using Google’s geolocation API.

Usage

geo_code(address, service = "nominatim",
base_url = "https://maps.google.com/maps/api/geocode/json",
return_all = FALSE, pat = NULL)
Arguments

- **address**: Text string representing the address you want to geocode
- **service**: Which service to use? Nominatim by default
- **base_url**: The base url to query
- **return_all**: Should the request return all information returned by Google Maps? The default is FALSE: to return only two numbers: the longitude and latitude, in that order
- **pat**: The API key used. By default this is set to NULL and this is usually acquired automatically through a helper, api_pat().

See Also

Other nodes: `locate2spdf`, `nearest_google`, `nearest_osm`

Examples

```r
## Not run:
geo_code(address = "Hereford")
geo_code("LS7 3HB")
geo_code("hereford", return_all = TRUE)
# needs api key in .Renviron
geo_code("hereford", service = "google", pat = Sys.getenv("GOOGLE"), return_all = TRUE)

## End(Not run)
```

---

**geo_length**

*Calculate line length of line with geographic or projected CRS*

Description

Takes a line (represented in sf or sp classes) and returns a numeric value representing distance in meters.

Usage

```r
geo_length(shp)
```

Arguments

- **shp**: A spatial line object

See Also

`buff_geo`

Examples

```r
geo_length(routes_fast)
geo_length(routes_fast_sf)
```
### geo_projected

**Perform GIS functions on a temporary, projected version of a spatial object**

**Description**

This function performs operations on projected data.

**Usage**

```r
geo_projected(shp, fun, crs, silent, ...)
```

**Arguments**

- **shp**: A spatial object with a geographic (WGS84) coordinate system
- **fun**: A function to perform on the projected object (e.g. the `rgeos` or `sf` packages)
- **crs**: An optional coordinate reference system (if not provided it is set automatically by `crs_select_aeq()`)
- **silent**: A binary value for printing the CRS details (default: TRUE)
- **...**: Arguments to pass to `fun`, e.g. `byid = TRUE` if the function is `rgeos::gLength()`

**Examples**

```r
shp <- routes_fast_sf[2:4, ]
plot(geo_projected(shp, sf::st_buffer, dist = 100)$geometry)
shp <- routes_fast[2:4, ]
geo_projected(shp, fun = rgeos::gBuffer, width = 100, byid = TRUE)
rlength <- geo_projected(routes_fast, fun = rgeos::gLength, byid = TRUE)
plot(routes_fast$length, rlength)
```

### geo_select_aeq

**Select a custom projected CRS for the area of interest**

**Description**

This function takes a spatial object with a geographic (WGS84) CRS and returns a custom projected CRS focussed on the centroid of the object. This function is especially useful for using units of metres in all directions for data collected anywhere in the world.

**Usage**

```r
geo_select_aeq(shp)
```

**Arguments**

- **shp**: A spatial object with a geographic (WGS84) coordinate system
Details

The function is based on this stackexchange answer: http://gis.stackexchange.com/questions/121489

Examples

```r
sp::bbox(routes_fast)
new_crs <- crs_select_aeq(routes_fast)
rf_projected <- sp::spTransform(routes_fast, new_crs)
sp::bbox(rf_projected)
line_length <- rgeos::gLength(rf_projected, byid = TRUE)
plot(line_length, rf_projected$length)
geo_select_aeq(zones_sf)
```

---

**geo_toptail**

*Clip the first and last n metres of SpatialLines*

Description

Takes lines and removes the start and end point, to a distance determined by the user.

Usage

```r
geo_toptail(l, toptail_dist, ...)
```

Arguments

- `l` A SpatialLines object
- `toptail_dist` The distance (in metres) to top and tail the line by. Can either be a single value or a vector of the same length as the SpatialLines object.
- `...` Arguments passed to rgeos::gBuffer()

Details

Note: `toptailgs()` is around 10 times faster, but only works on data with geographic CRS’s due to its reliance on the geosphere package.

See Also

Other lines: `angle_diff, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_sample, line_segment, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptail_buff, toptailgs, update_line_geometry`
Examples

```r
l <- routes_fast[2:4,]
l_toptail <- geo_toptail(l, toptail_dist = 300)
plot(l)
plot(l_toptail, col = "red", add = TRUE, lwd = 3)
plot(cents, col = "blue", add = TRUE, pch = 15)

# Note the behaviour when the buffer size removes lines
r_toptail <- geo_toptail(l, toptail_dist = 900)
plot(r_toptail, lwd = 9, add = TRUE) # short route removed
geo_toptail(routes_fast_sf[2:4,], 300)
```

---

### gsection

**Function to split overlapping SpatialLines into segments**

### Description

Divides SpatialLinesDataFrame objects into separate Lines. Each new Lines object is the aggregate of a single number of aggregated lines.

### Usage

```r
gsection(sl, buff_dist = 0)
```

### Arguments

- `sl` SpatialLinesDataFrame with overlapping Lines to split by number of overlapping features.
- `buff_dist` A number specifying the distance in meters of the buffer to be used to crop lines before running the operation. If the distance is zero (the default) touching but non-overlapping lines may be aggregated.

### See Also

Other rnet: `SpatialLinesNetwork, calc_catchment_sum, calc_catchment, calc_moving_catchment, calc_network_catchment, find_network_nodes, islines, lineLabels, overline2, overline, plot,SpatialLinesNetwork, ANY-method, plot,sfNetwork, ANY-method, sln2points, sum_network_links, sum_network_routes`

### Examples

```r
sl <- routes_fast[2:4,]
rsec <- gsection(sl)
rsec_buff <- gsection(sl, buff_dist = 1)
plot(sl[1,], lwd = 9, col = 1:nrow(sl))
plot(rsec, col = 5 + (1:length(rsec)), add = TRUE, lwd = 3)
plot(rsec_buff, col = 5 + (1:length(rsec_buff)), add = TRUE, lwd = 3)
## Not run:
sl <- routes_fast_sf[2:4,]
```
Import GTFS shapes and route data to SpatialLinesDataFrame.

Description

Takes a string with the file path of the zip file with the GTFS feed, imports the shapes (geometry), route and agency data and returns a SpatialLinesDataFrame for the GTFS feed.

Usage

gtfs2sldf(gtfszip = "")

Arguments

gtfszip String with the file path of the GTFS feed zip file

Examples

f <- system.file("extdata", "beartransit-ca-us.zip", package = "stplanr")
# update file to latest version
# see https://code.google.com/p/googletransitdatafeed/wiki/PublicFeeds
u <- "http://data.trilliumtransit.com/gtfs/beartransit-ca-us/beartransit-ca-us.zip"
# download.file(u, f)
gtfs <- gtfs2sldf(gtfszip = f)
plot(gtfs, col = gtfs$route_long_name)
plot(gtfs[gtfs$route_long_name == "Central Campus", ])
## Not run:
# An example of a larger gtfs feed
download.file(
  "http://www.yrt.ca/google/google_transit.zip",
  paste0(tempdir(), "/gtfsfeed.zip")
)
yrtgtfs <- gtfs2sldf(paste0(tempdir(), "/gtfsfeed.zip"))
sp::plot(yrtgtfs, col = paste0("#", yrtgtfs$route_color))
## End(Not run)
is_lines

Do the intersections between two geometries create lines?

Description

This is a function required in `overline()`. It identifies whether sets of lines overlap (beyond shared points) or not.

Usage

`islines(g1, g2)`

Arguments

- `g1`: A spatial object
- `g2`: A spatial object

See Also

Other rnet: `SpatialLinesNetwork`, `calc_catchment_sum`, `calc_catchment`, `calc_moving_catchment`, `calc_network_catchment`, `find_network_nodes`, `gsection`, `lineLabels`, `overline2`, `overline`, `plot`, `SpatialLinesNetwork`, `ANY-method`, `plot`, `sfNetwork`, `ANY-method`, `sln2points`, `sum_network_links`, `sum_network_routes`

Examples

```r
## Not run:
rnet <- overline(routes_fast[c(2, 3, 22), ], attrib = "length")
plot(rnet)
lines(routes_fast[22, ], col = "red") # line without overlaps
islines(routes_fast[2, ], routes_fast[3, ])
islines(routes_fast[2, ], routes_fast[22, ])
# sf implementation
islines(routes_fast_sf[2, ], routes_fast_sf[3, ])
islines(routes_fast_sf[2, ], routes_fast_sf[22, ])
## End(Not run)
```

is_linepoint

Identify lines that are points

Description

OD matrices often contain 'intrazonal' flows, where the origin is the same point as the destination. This function can help identify such intrazonal OD pairs, using 2 criteria: the total number of vertices (2 or fewer) and whether the origin and destination are the same.
Usage
    is_linepoint(l)

Arguments
    l  A spatial lines object

Details
    Returns a boolean vector. TRUE means that the associated line is in fact a point (has no distance). This can be useful for removing data that will not be plotted.

See Also
    Other lines: angle_diff, geo_toptail, line2df, line_bearing, line_match, line_midpoint, line_sample, line_segment, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptail_buff, toptailgs, update_line_geometry

Examples
    data(flowlines)
    islp <- is_linepoint(flowlines)
    nrow(flowlines)
    sum(islp)
    # Remove invisible 'linepoints'
    nrow(flowlines[!islp, ])

---

**line2df**  
*Convert geographic line objects to a data frame with from and to co-ords*

Description
    This function returns a data frame with fx and fy and tx and ty variables representing the beginning and end points of spatial line features respectively.

Usage
    line2df(l)

Arguments
    l  A spatial lines object

See Also
    Other lines: angle_diff, geo_toptail, is_linepoint, line_bearing, line_match, line_midpoint, line_sample, line_segment, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptail_buff, toptailgs, update_line_geometry
Examples

data(flowlines)
line2df(flowlines[5,]) # beginning and end of a single straight line
line2df(flowlines) # on multiple lines
line2df(routes_fast[5:6,]) # beginning and end of routes
line2df(routes_fast_sf[5:6,]) # beginning and end of routes

line2route

Convert straight OD data (desire lines) into routes

Description

Convert straight OD data (desire lines) into routes

Usage

line2route(l, route_fun = stplanr::route_cyclestreet, n_print = 10,
list_output = FALSE, l_id = NA, time_delay = 0, ...)

Arguments

l
A spatial (linestring) object

route_fun
A routing function to be used for converting the straight lines to routes od2line()

n_print
A number specifying how frequently progress updates should be shown

list_output
If FALSE (default) assumes spatial (linestring) object output. Set to TRUE to save output as a list.

l_id
Character string naming the id field from the input lines data, typically the origin and destination ids pasted together. If absent, the row name of the straight lines will be used.

time_delay
Number or seconds to wait between each query

Arguments passed to the routing function, e.g. route_cyclestreet()

Details

See route_cyclestreet() and other route functions for details.

A parallel implementation of this was available until version 0.1.8.

See Also

Other routes: line2routeRetry, nearest2spdf, route_cyclestreet, route_dodgr, route_graphhopper,
route_local, route_osrm, route_transportapi_public, route, viaroute2s1df, viaroute
Examples

```r
## Not run:
l <- flowlines[2:5, ]
r <- line2route(l)
rf <- line2route(l = 1, "route_cyclestreet", plan = "fastest")
rq <- line2route(l = 1, plan = "quietest", silent = TRUE)
plot(r)
plot(rf, col = "red", add = TRUE)
plot(rq, col = "green", add = TRUE)
plot(l, add = T)
line2route(flowlines_sf[2:3, ], route_osrm)
# Plot for a single line to compare 'fastest' and 'quietest' route
n <- 2
plot(l[n, ])
lines(rf[n, ], col = "red")
lines(rq[n, ], col = "green")
# Example with list output
l <- l[1:3, ]
r_list <- line2route(l = 1, list_output = TRUE)
line2route(l[1, ], route_graphhopper)
## End(Not run)
```

### line2routeRetry

*Convert straight spatial (linestring) object from flow data into routes retrying on connection (or other) intermittent failures*

#### Description

Convert straight spatial (linestring) object from flow data into routes retrying on connection (or other) intermittent failures

#### Usage

```
line2routeRetry(lines, pattern = "^Error: ", n_retry = 3, ...)
```

#### Arguments

- **lines**: A spatial (linestring) object
- **pattern**: A regex that the error messages must not match to be retried, default "^Error: " i.e. do not retry errors starting with "Error: "
- **n_retry**: Number of times to retry
- **...**: Arguments passed to the routing function, e.g. `route_cyclestreet()`

#### Details

See `line2route()` for the version that is not retried on errors.
See Also

Other routes: `line2route`, `nearest2spdf`, `route_cyclestreet`, `route_dodgr`, `route_graphhopper`, `route_local`, `route_osrm`, `route_transportapi_public`, `route`, `viaroute2sldf`, `viaroute`

Examples

```r
## Not run:
data(flowlines)
rf_list <- line2routeRetry(flowlines[1:2], pattern = "nonexistenceerror", silent = F)
## End(Not run)
```

---

**lineLabels**

*Label SpatialLinesDataFrame objects*

**Description**

This function adds labels to lines plotted using base graphics. Largely for illustrative purposes, not designed for publication-quality graphics.

**Usage**

```r
lineLabels(sl, attrib)
```

**Arguments**

- `sl` A SpatialLinesDataFrame with overlapping elements
- `attrib` A text string corresponding to a named variable in `sl`

**Author(s)**

Barry Rowlingson

**See Also**

**line_bearing**

*Find the bearing of straight lines*

**Description**

This is a simple wrapper around the geosphere function `bearing()` to return the bearing (in degrees relative to north) of lines.

**Usage**

```r
line_bearing(l, bidirectional = FALSE)
```

**Arguments**

- `l`: A spatial lines object
- `bidirectional`: Should the result be returned in a bidirectional format? Default is FALSE. If TRUE, the same line in the opposite direction would have the same bearing.

**Details**

Returns a boolean vector. TRUE means that the associated line is in fact a point (has no distance). This can be useful for removing data that will not be plotted.

**See Also**

Other lines: `angle_diff`, `geo_toptail`, `is_linepoint`, `line2df`, `line_match`, `line_midpoint`, `line_sample`, `line_segment`, `line_via`, `mats2line`, `n_sample_length`, `n_vertices`, `onewaygeo`, `onewayid`, `points2line`, `toptail_buff`, `toptailgs`, `update_line_geometry`

**Examples**

```r
data(flowlines)
b1 <- line_bearing(flowlines)
b2 <- line_bearing(flowlines, bidirectional = TRUE)
plot(b1, b2)
line_bearing(flowlines_sf[1:9, ])
```

---

**line_length**

*Calculate length of lines in geographic CRS*

**Description**

Calculate length of lines in geographic CRS

**Usage**

```r
line_length(l, byid = TRUE)
```
Arguments

1. A spatial lines object
2. Logical determining whether the length is returned per object (default is true)

**line_match**

*Match two sets of lines based on similarity*

Description

This function is a wrapper around gDistance that matches lines based on the Hausdorff distance.

Usage

```r
line_match(l1, l2, threshold = 0.01, return_sp = FALSE)
```

Arguments

1. A spatial object
2. A spatial object
3. The threshold for a match - distances greater than this will not count as matches
4. Should the function return a spatial result (FALSE by default)

See Also

Other lines: `angle_diff`, `geo_toptail`, `is_linepoint`, `line2df`, `line_bearing`, `line_midpoint`, `line_sample`, `line_segment`, `line_via`, `mats2line`, `n_sample_length`, `n_vertices`, `onewaygeo`, `onewayid`, `points2line`, `toptail_buff`, `toptailgs`, `update_line_geometry`

Examples

```r
x1 <- 2:4
x2 <- 3:5
match(x1, x2) # how the base function works
l1 <- flowlines[2:4, ]
l2 <- routes_fast[3:5, ]
(lmatches <- line_match(l1, l2)) # how the stplanr version works
l2matched <- l2[lmatches[!is.na(lmatches)], ]
plot(l1)
plot(l2, add = TRUE)
plot(l2matched, add = TRUE, col = "red") # showing matched routes
l2matched2 <- line_match(l1, l2, return_sp = TRUE)
identical(l2matched, l2matched2)
# decreasing the match likelihood via the threshold
line_match(l1, l2, threshold = 0.003)
```
**line_midpoint**

Find the mid-point of lines

**Description**

This is a wrapper around `SpatialLinesMidPoints()` that allows it to find the midpoint of lines that are not projected, which have a lat/long CRS.

**Usage**

```r
line_midpoint(l)
```

**Arguments**

- `l` A spatial lines object

**See Also**

Other lines: `angle_diff`, `geo_toptail`, `is_linepoint`, `line2df`, `line_bearing`, `line_match`, `line_sample`, `line_segment`, `line_via`, `mats2line`, `n_sample_length`, `n_vertices`, `onewaygeo`, `onewayid`, `points2line`, `toptail_buff`, `toptailgs`, `update_line_geometry`

**Examples**

```r
data(routes_fast)
line_midpoint(routes_fast[2:5, ])
```

---

**line_sample**

Sample n points along lines with density proportional to a weight

**Description**

Sample n points along lines with density proportional to a weight

**Usage**

```r
line_sample(l, n, weights)
```

**Arguments**

- `l` The SpatialLines object along which to create sample points
- `n` The total number of points to sample
- `weights` The relative probabilities of lines being samples
line_segment

See Also

Other lines: angle_diff, geo_toptail, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_segment, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptail_buff, toptailgs, update_line_geometry

Examples

```r
l <- flowlines[2:5, ]
n <- 100
l_lengths <- line_length(l)
weights <- l$All
p <- line_sample(l, 50, weights)
plot(p)
p <- line_sample(l, 50, weights = 1:length(l))
plot(p)
```

**line_segment**  Divide SpatialLines dataset into regular segments

**Description**

Divide SpatialLines dataset into regular segments

**Usage**

```r
line_segment(l, n_segments, segment_length = NA)
```

**Arguments**

- `l`: A spatial lines object
- `n_segments`: The number of segments to divide the line into
- `segment_length`: The approximate length of segments in the output (overrides `n_segments` if set)

**See Also**

Other lines: angle_diff, geo_toptail, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_sample, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptail_buff, toptailgs, update_line_geometry

**Examples**

```r
data(routes_fast)
l <- routes_fast[2, ]
library(sp)
l_seg2 <- line_segment(l = l, n_segments = 2)
plot(l_seg2, col = l_seg2$group, lwd = 50)
```
line_to_points

Convert a spatial (linestring) object to points. The number of points will be double the number of lines with `line2points`. A closely related function, `line2pointsn` returns all the points that were line vertices. The points corresponding with a given line, i, will be (2*i):(2*i+1).

Description

Convert a spatial (linestring) object to points. The number of points will be double the number of lines with `line2points`. A closely related function, `line2pointsn` returns all the points that were line vertices. The points corresponding with a given line, i, will be (2*i):(2*i+1).

Usage

```r
describe(line_to_points(l, ids = rep(1:nrow(l), each = 2)))
```

Arguments

- `l` An `sf` object or a `SpatialLinesDataFrame` from the older sp package.
- `ids` Vector of ids (by default 1:nrow(l))

Examples

```r
l <- routes_fast_sf[2:4,]
lpoints <- line2points(l)
lpoints2 <- line2pointsn(l)
plot(sf::st_geometry(lpoints), pch = lpoints$id, cex = lpoints$id, col = "black")
plot(lpoints2$geometry, add = TRUE)
# in sp data forms (may be depreciated)
l <- routes_fast[2:4,]
lpoints <- line2points(l)
lpoints2 <- line2pointsn(l)
plot(lpoints, pch = lpoints$id, cex = lpoints$id)
points(lpoints2)
```

line_via

Add geometry columns representing a route via intermediary points

Description

Takes an origin (A) and destination (B), represented by the linestring l, and generates 3 extra geometries based on points p:
Usage

line_via(l, p)

Arguments

l A spatial lines object
p A spatial points object

Details

1. From A to P1 (P1 being the nearest point to A)
2. From P1 to P2 (P2 being the nearest point to B)
3. From P2 to B

See Also

Other lines: angle_diff, geo_toptail, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_sample, line_segment, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptail_buff, toptailgs, update_line_geometry

Examples

```r
l <- flowlines_sf[2:4, ]
p <- destinations_sf
lv <- line_via(l, p)
## Not run:
library(mapview)
mapview(lv) +
  mapview(lv$leg_orig, col = "red")
## End(Not run)
library(sf)
plot(lv[3], lwd = 9, reset = FALSE)
plot(lv$leg_orig, col = "red", lwd = 5, add = TRUE)
plot(lv$leg_via, col = "black", add = TRUE)
plot(lv$leg_dest, col = "green", lwd = 5, add = TRUE)
```
locate2spdf(lat, lng = lng, osrmurl = "http://router.project-osrm.org", return_sf = FALSE)

Arguments

lat      Numeric vector containing latitude coordinate for each coordinate to map. Also accepts dataframe with latitude in the first column and longitude in the second column.
lng      Numeric vector containing longitude coordinate for each coordinate to map.
osrmurl Base URL of the OSRM service
return_sf Boolean value if this function should return an sf object, if FALSE returns sp object (default FALSE).

Details

Retrieve coordinates of the node(s) on the network mapped from coordinates passed to functions using OSRM API v4 only. For API v5, use nearest_osm.

See Also

Other nodes: geo_code, nearest_google, nearest_osm

Examples

## Not run:
locate2spdf(
  lat = c(50.3, 50.2),
  lng = c(13.2, 13.1)
)
## End(Not run)

l_poly

Line polygon

Description

This dataset represents road width for testing.

Usage

data(l_poly)

Format

A SpatialPolygon
Examples

```r
## Not run:
l <- routes_fast[13,]
l_poly <- buff_geo(l, 8)
plot(l_poly)
plot(routes_fast, add = TRUE)
# allocate road width to relevant line
devtools::use_data(l_poly)

## End(Not run)
```

mapshape  

Simplify geometry of spatial objects with the mapshaper library

Description

Simplify geometry of spatial objects with the mapshaper library

Usage

```r
mapshape(shp, percent = 10, ms_options = "", dsn = "mapshape", silent = FALSE)
```

Arguments

- `shp`: A spatial object to be simplified.
- `percent`: A number between 1 and 100 stating how aggressively to simplify the object (1 is a very aggressive simplification).
- `ms_options`: Text string of options passed to mapshaper such as `no-topology` (a flag) and `snap-interval=1` (a key value pair). See the mapshaper documentation for details: [https://github.com/mbloch/mapshaper/wiki/Command-Reference](https://github.com/mbloch/mapshaper/wiki/Command-Reference).
- `dsn`: The name of the temporary file to write to (deleted after use).
- `silent`: Logical determining whether the function call is printed to screen.

Details

Note: more advance R/mapshaper tools are provided by the rmapshaper package: [https://github.com/ateucher/rmapshaper](https://github.com/ateucher/rmapshaper).

Calls the JavaScript command-line GIS application mapshaper ([https://github.com/mbloch/mapshaper](https://github.com/mbloch/mapshaper)) from the system to simplify geographic features, and then tidies up. mapshaper must be installed and available to `system()`. mapshape writes new a file to disk. Thanks to Richard and Adrian Ellison for demonstrating this in R.
mapshape_available

See Also

Other geo: bbox_scale, buff_geo, crs_select_aeq, decode_gl, gclip, geo_bb_matrix, geo_bb, mapshape_available, quadrant, reproject

Examples

```r
# Not run:
shp <- routes_fast[2,]
plot(shp)

rfs10 <- mapshape(shp)
rfs5 <- mapshape(shp, percent = 5)
rfs1 <- mapshape(shp, percent = 1)
plot(rfs10, add = TRUE, col = "red")
plot(rfs5, add = TRUE, col = "blue")
plot(rfs1, add = TRUE, col = "grey")

# snap the lines to the nearest interval
rfs_int <- mapshape(shp, ms_options = "snap-interval=0.001")
plot(shp)
plot(rfs_int, add = TRUE)

mapshape(routes_fast_sf[2,])

# End(Not run)
```

mapshape_available Does the computer have mapshaper available?

Description

This helper function for mapshape() determines whether or not the JavaScript library mapshaper is available.

Usage

mapshape_available()

See Also

Other geo: bbox_scale, buff_geo, crs_select_aeq, decode_gl, gclip, geo_bb_matrix, geo_bb, mapshape, quadrant, reproject

Examples

mapshape_available()
mats2line

Convert 2 matrices to lines

Description
Converting 2 matrices to lines

Usage
mats2line(mat1, mat2)

Arguments
mat1 Matrix representing origins
mat2 Matrix representing destinations

See Also
Other lines: angle_diff, geo_toptail, is_linepoint, line2df, line_bearing, line_match,
line_midpoint, line_sample, line_segment, line_via, n_sample_length, n_vertices, onewaygeo,
onewayid, points2line, toptail_buff, toptailgs, update_line_geometry

Examples
{
m1 <- matrix(c(1, 2, 1, 2), ncol = 2)
m2 <- matrix(c(9, 9, 9, 1), ncol = 2)
l <- mats2line(m1, m2)
class(l)
lsf <- sf::st_sf(l, crs = 4326)
class(lsf)
plot(lsf)
# mapview::mapview(lsf)
}

nearest2spdf

Return SpatialPointsDataFrame with nearest street from OSRM nearest service

Description
Return SpatialPointsDataFrame with nearest street from OSRM nearest service

Usage
nearest2spdf(lat, lng, osrmurl = "http://router.project-osrm.org", return_sf = FALSE)
Arguments

- **lat**: Numeric vector containing latitude coordinate for each coordinate to map. Also accepts dataframe with latitude in the first column and longitude in the second column.
- **lng**: Numeric vector containing longitude coordinate for each coordinate to map.
- **osrmurl**: Base URL of the OSRM service
- **return_sf**: Boolean value if this function should return an sf object, if FALSE returns sp object (default FALSE).

Details

Retrieve coordinates and name of the node(s) on the network mapped from coordinates passed to functions using OSRM API v4 only. For API v5, use nearest_osm.

See Also

Other routes: line2routeRetry, line2route, route_cyclestreet, route_dodgr, route_graphhopper, route_local, route_osrm, route_transportapi_public, route, viaroute2sldf, viaroute

Examples

```r
## Not run:
nearest2spdf(
  lat = c(50.3, 50.2),
  lng = c(13.2, 13.1)
)
## End(Not run)
```

### Description

Generate nearest point on the route network of a point using the CycleStreets.net

### Usage

```r
nearest_cyclestreets(shp = NULL, lat, lng,
  pat = api_pat("cyclestreet"))
```
nearest_google

Generate nearest point on the route network of a point using the Google Maps API

Usage

nearest_google(lat, lng, google_api)

Arguments

- **lat**: Numeric vector containing latitude coordinate for each coordinate to map. Also accepts dataframe with latitude in the first column and longitude in the second column.
- **lng**: Numeric vector containing longitude coordinate for each coordinate to map.
- **google_api**: String value containing the Google API key to use.

Details

Retrieve coordinates of the node(s) on the network mapped from coordinates passed to functions.

Examples

```r
## Not run:
nearest_cyclestreets(53, 0.02, pat = Sys.getenv("CYCLESTREET"))
nearest_cyclestreets(cents[, 1], pat = Sys.getenv("CYCLESTREET"))
nearest_cyclestreets(cents_sf[, 1], pat = Sys.getenv("CYCLESTREET"))
## End(Not run)
```
nearest_osm

See Also

Other nodes: geo_code, locate2spdf, nearest_osm

Examples

```r
## Not run:
nearest_google(lat = 50.333, lng = 3.222, google_api = "api_key_here")

## End(Not run)
```

---

**nearest_osm**

Generate nearest point on the route network of a point from OSRM locate service

**Description**

Generate nearest point on the route network of a point from OSRM locate service

**Usage**

```r
nearest_osm(lat, lng, number = 1, api = 5, profile = "driving",
            protocol = "v1", osrmurl = "http://router.project-osrm.org",
            return_sf = FALSE)
```

**Arguments**

- `lat` Numeric vector containing latitude coordinate for each coordinate to map. Also accepts dataframe with latitude in the first column and longitude in the second column.
- `lng` Numeric vector containing longitude coordinate for each coordinate to map.
- `number` Number of locations to return (API v5 only)
- `api` An integer value containing the OSRM API version (either 4 or 5). Default is 5.
- `profile` OSRM profile to use (for API v5), defaults to "driving".
- `protocol` The protocol to use for the API (for v5), defaults to "v1".
- `osrmurl` Base URL of the OSRM service
- `return_sf` Boolean value if this function should return an sf object, if FALSE returns sp object (default FALSE).

**Details**

Retrieve coordinates of the node(s) on the network mapped from coordinates passed to functions.

**See Also**

Other nodes: geo_code, locate2spdf, nearest_google
Examples

```r
## Not run:
nearest_osm(
  lat = 50.3,
  lng = 13.2
)
## End(Not run)
```

```r
n_sample_length  Sample integer number from given continuous vector of line lengths and probabilities, with total n
```

Description

Sample integer number from given continuous vector of line lengths and probabilities, with total n

Usage

```r
n_sample_length(n, l_lengths, weights)
```

Arguments

- `n`: Sum of integer values returned
- `l_lengths`: Numeric vector of line lengths
- `weights`: Relative probabilities of samples on lines

See Also

Other lines: `angle_diff`, `geo_toptail`, `is_linepoint`, `line2df`, `line_bearing`, `line_match`, `line_midpoint`, `line_sample`, `line_segment`, `line_via`, `mats2line`, `n_vertices`, `onewaygeo`, `onewayid`, `points2line`, `toptail_buff`, `toptailgs`, `update_line_geometry`

Examples

```r
n <- 10
l_lengths <- 1:5
weights <- 9:5
(res <- n_sample_length(n, l_lengths, weights))
sum(res)
n <- 100
l_lengths <- c(12, 22, 15, 14)
weights <- c(38, 10, 44, 34)
(res <- n_sample_length(n, l_lengths, weights))
sum(res)
# more examples:
n_sample_length(5, 1:5, c(0.1, 0.9, 0, 0, 0))
n_sample_length(5, 1:5, c(0.5, 0.3, 0.1, 0, 0))
```
n_vertices

l <- flowlines[2:6, ]
l_lengths <- line_length(l)
n <- n_sample_length(10, l_lengths, weights = l$All)

---

n_vertices

Retrieve the number of vertices from a SpatialLines or SpatialPolygons object

Description

Returns a vector of the same length as the number of lines, with the number of vertices per line or polygon.

Usage

n_vertices(l)

Arguments

l

A SpatialLines or SpatialPolygons object

Details

See http://gis.stackexchange.com/questions/58147/ for more information.

See Also

Other lines: angle_diff, geo_toptail, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_sample, line_segment, line_via, mats2line, n_sample_length, onewaygeo, onewayid, points2line, toptail_buff, toptailgs, update_line_geometry

Examples

n_vertices(routes_fast)
n_vertices(routes_fast_sf)
**od2line**  
_Convert origin-destination data to spatial lines_

**Description**

Origin-destination (‘OD’) flow data is often provided in the form of 1 line per flow with zone codes of origin and destination centroids. This can be tricky to plot and link-up with geographical data. This function makes the task easier.

**Usage**

```r
od2line(flow, zones, destinations = NULL, zone_code = names(zones)[1],
        origin_code = names(flow)[1], dest_code = names(flow)[2],
        zone_code_d = NA, silent = FALSE)

od2line2(flow, zones)
```

**Arguments**

- `flow`: A data frame representing origin-destination data. The first two columns of this data frame should correspond to the first column of the data in the zones. Thus in `cents()`, the first column is `geo_code`. This corresponds to the first two columns of `flow()`.
- `zones`: A spatial object representing origins (and destinations if no separate destinations object is provided) of travel.
- `destinations`: A spatial object representing destinations of travel flows.
- `zone_code`: Name of the variable in `zones` containing the ids of the zone. By default this is the first column names in the zones.
- `origin_code`: Name of the variable in `flow` containing the ids of the zone of origin. By default this is the first column name in the flow input dataset.
- `dest_code`: Name of the variable in `flow` containing the ids of the zone of destination. By default this is the second column name in the flow input dataset or the first column name in the destinations if that is set.
- `zone_code_d`: Name of the variable in `destinations` containing the ids of the zone. By default this is the first column names in the destinations.
- `silent`: TRUE by default, setting it to TRUE will show you the matching columns

**Details**

Origin-destination (OD) data is often provided in the form of 1 line per OD pair, with zone codes of the trip origin in the first column and the zone codes of the destination in the second column (see the vignette("stplanr-od")) for details. `od2line()` creates a spatial (linestring) object representing movement from the origin to the destination for each OD pair. It takes data frame containing origin and destination cones (`flow`) that match the first column in a a spatial (polygon or point) object (`zones`).
See Also

Other od: dist_google, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate

Examples

l <- od2line(flow = flow, zones = cents)
plot(cents)
plot(l, lwd = l$All / mean(l$All), add = TRUE)
# When destinations are different
head(flow_dests[1:5])  # check data
head(destinations[1:5])
flowlines_dests <- od2line(flow_dests, cents, destinations = destinations)
plot(flowlines_dests)
l <- od2line(flow, zones_sf)
plot(l["All"], lwd = l$All/mean(l$All))

---

od2odf

**Extract coordinates from OD data**

Description

Extract coordinates from OD data

Usage

od2odf(flow, zones)

Arguments

- **flow**: A data frame representing origin-destination data. The first two columns of this data frame should correspond to the first column of the data in the zones. Thus in `cents()`, the first column is geo_code. This corresponds to the first two columns of `flow()`.

- **zones**: A spatial object representing origins (and destinations if no separate destinations object is provided) of travel.

Details

Origin-destination (OD) data is often provided in the form of 1 line per OD pair, with zone codes of the trip origin in the first column and the zone codes of the destination in the second column (see the vignette("stplanr-od")) for details. od2odf() creates an 'origin-destination data frame', based on a data frame containing origin and destination cones (flow) that match the first column in a a spatial (polygon or point) object (zones).

The function returns a data frame with coordinates for the origin and destination.
See Also

Other od: dist_google, od2line, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate

Examples

data(flow)
data(zones)
od2odf(flow[1:2, ], zones)

odmatrix_to_od

Convert origin-destination data from wide to long format

Description

This function takes a matrix representing travel between origins (with origin codes in the rownames of the matrix) and destinations (with destination codes in the colnames of the matrix) and returns a data frame representing origin-destination pairs.

Usage

odmatrix_to_od(odmatrix)

Arguments

odmatrix A matrix with row and columns representing origin and destination zone codes and cells representing the flow between these zones.

Details

The function returns a data frame with rows ordered by origin and then destination zone code values and with names orig, dest and flow.

See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate

Examples

odmatrix <- od_to_odmatrix(flow)
odmatrix_to_od(odmatrix)
flow[1:9, 1:3]
odmatrix_to_od(od_to_odmatrix(flow[1:9, 1:3]))
od_aggregate

Aggregate OD data between polygon geometries

Description

Aggregate OD data between polygon geometries

Usage

od_aggregate(flow, zones, aggzones, aggzone_points = NULL, 
cols = FALSE, aggcols = FALSE, FUN = sum, 
prop_by_area = ifelse(identical(FUN, mean) == FALSE, TRUE, FALSE), 
digits = getOption("digits"))

Arguments

flow A data frame representing origin-destination data. The first two columns of this 
data frame should correspond to the first column of the data in the zones. Thus 
in cents(), the first column is geo_code. This corresponds to the first two 
columns of flow().
zones A spatial object representing origins (and destinations if no separate destinations 
object is provided) of travel.
aggzones A SpatialPolygonsDataFrame containing the new boundaries to aggregate to.
aggzone_points Points representing origins of OD flows (typically population-weighted cen-
troids)
cols A character vector containing the names of columns on which to apply FUN. By 
default, all numeric columns are aggregated.
aggcols A character vector containing the names of columns in aggzones to retain in 
the aggregated data.frame. By default, only the first column is retained. These 
columns are renamed with a prefix of "o_" and "d_".
FUN Function to use on aggregation. Default is sum.
prop_by_area Boolean value indicating if the values should be proportionally adjusted based 
on area. Default is TRUE unless FUN = mean.
digits The number of digits to use when proportionally adjusting values based on area. 
Default is the value of getOption("digits").

Value

data.frame containing the aggregated od flows.

Details

Origin-destination (‘OD’) flow data is often provided in the form of 1 line per flow with zone codes 
of origin and destination centroids. This function aggregates OD flows between polygon geometries 
allocating the original flows to larger zones based on area.
See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate

Examples

zones$quadrant <- c(1, 2, 1, 4, 5, 6, 7, 1)
aggzones <- rgeos::gUnaryUnion(zones, id = zones@data$quadrant)
aggzones <- sp::SpatialPolygonsDataFrame(aggzones, data.frame(region = c(1:6)), match.ID = FALSE)
sp::proj4string(aggzones) <- sp::proj4string(zones)
aggzones_sf <- sf::st_as_sf(aggzones)
aggzones_sf <- sf::st_set_crs(aggzones_sf, sf::st_crs(zones_sf))
od_agg <- od_aggregate(flow, zones_sf, aggzones_sf)
colSums(od_agg[3:9]) == colSums(flow[3:9])
od_sf_agg <- od2line(od_agg, aggzones_sf)
plot(flowlines, lwd = flowlines$Bicycle)
plot(od_sf_agg$geometry, lwd = od_sf_agg$Bicycle, add = TRUE, col = "red")

od_aggregate_from

Summary statistics of trips originating from zones in OD data

Description

This function takes a data frame of OD data and returns a data frame reporting summary statistics for each unique zone of origin.

Usage

od_aggregate_from(flow, attrib = NULL, FUN = sum, ..., col = 1)

Arguments

flow A data frame representing origin-destination data. The first two columns of this data frame should correspond to the first column of the data in the zones. Thus in cents(), the first column is geo_code. This corresponds to the first two columns of flow().
attrib A character vector corresponding to the variables in sl$ on which the function(s) will operate.
FUN A function to summarise OD data by
... Additional arguments passed to FUN
col The column that the OD dataset is grouped by (1 by default, the first column usually represents the origin)
od_aggregate_to

Details

It has some default settings: the default summary statistic is \( \text{sum()} \) and the first column in the OD data is assumed to represent the zone of origin. By default, if `attrib` is not set, it summarises all numeric columns.

See Also

Other od: `dist_google, od2line, od2odf, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate`

Examples

```r
od_aggregate_from(flow)
```

---

### od_aggregate_to

**Summary statistics of trips arriving at destination zones in OD data**

**Description**

This function takes a data frame of OD data and returns a data frame reporting summary statistics for each unique zone of destination.

**Usage**

```r
od_aggregate_to(flow, attrib = NULL, FUN = sum, ..., col = 2)
```

**Arguments**

- `flow`: A data frame representing origin-destination data. The first two columns of this data frame should correspond to the first column of the data in the zones. Thus in `cents()`, the first column is `geo_code`. This corresponds to the first two columns of `flow()`.
- `attrib`: A character vector corresponding to the variables in `sl$` on which the function(s) will operate.
- `FUN`: A function to summarise OD data by
- `...`: Additional arguments passed to `FUN`
- `col`: The column that the OD dataset is grouped by (1 by default, the first column usually represents the origin)

**Details**

It has some default settings: it assumes the destination ID column is the 2nd and the default summary statistic is \( \text{sum()} \). By default, if `attrib` is not set, it summarises all numeric columns.
See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate

Examples

od_aggregate_to(flow)

---

od_coords

Create matrices representing origin-destination coordinates

Description

This function takes a wide range of input data types (spatial lines, points or text strings) and returns a matrix of coordinates representing origin (fx, fy) and destination (tx, ty) points.

Usage

od_coords(from = NULL, to = NULL, l = NULL)

Arguments

from An object representing origins (if lines are provided as the first argument, from is assigned to l)
to An object representing destinations
l Only needed if from and to are empty, in which case this should be a spatial object representing desire lines

See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate

Examples

od_coords(from = c(0, 52), to = c(1, 53)) # lon/lat coordinates
od_coords(from = cents[1, ], to = cents[2, ]) # Spatial points
od_coords(cents_sf[1:3, ], cents_sf[2:4, ]) # sf points
# od_coords("Hereford", "Leeds") # geocode locations
od_coords(flowlines[1:3, ])
od_coords(flowlines_sf[1:3, ])

od_coords2line

Convert origin-destination coordinates into desire lines

Description

Convert origin-destination coordinates into desire lines

Usage

od_coords2line(odc, crs = 4326, remove_duplicates = TRUE)

Arguments

odc      A data frame or matrix representing the coordinates of origin-destination data. The first two columns represent the coordinates of the origin (typically longitude and latitude) points; the third and fourth columns represent the coordinates of the destination (in the same CRS). Each row represents travel from origin to destination.

crs      A number representing the coordinate reference system of the result, 4326 by default.

remove_duplicates

Should rows with duplicated rows be removed? TRUE by default.

See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate

Examples

odf <- od_coords(l = flowlines_sf)
oelines <- od_coords2line(odf)
oelines <- od_coords2line(odf, crs = 4326)
plot(odelines)
x_coords = 1:3
n = 50
d = data.frame(lapply(1:4, function(x) sample(x_coords, n, replace = TRUE)))
names(d) = c("fx", "fy", "tx", "ty")
l = od_coords2line(d)
plot(l)
nrow(l)
l_with_duplicates = od_coords2line(d, remove_duplicates = FALSE)
plot(l_with_duplicates)
nrow(l_with_duplicates)
Example of origin-destination data from UK Census

Description

See data-raw/generate-data.Rmd for details on how this was created.

Format

A data frame (tibble) object

Examples

od_data_sample

Quickly calculate Euclidean distances of od pairs

Description

It is common to want to know the Euclidean distance between origins and destinations in OD data. You can calculate this by first converting OD data to SpatialLines data, e.g. with od2line(). However this can be slow and overkill if you just want to know the distance. This function is a few orders of magnitude faster.

Usage

od_dist(flow, zones)

Arguments

flow

A data frame representing origin-destination data. The first two columns of this data frame should correspond to the first column of the data in the zones. Thus in cents(), the first column is geo_code. This corresponds to the first two columns of flow().

zones

A spatial object representing origins (and destinations if no separate destinations object is provided) of travel.

Details

Note: this function assumes that the zones or centroids in cents have a geographic (lat/lon) CRS.

See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate
Examples

```r
data(flow)
data(cents)
od_dist(flow, cents)
```

---

**od_id**

*Combine two ID values to create a single ID number*

**Description**

Combine two ID values to create a single ID number

**Usage**

```r
od_id_szudzik(x, y, ordermatters = FALSE)

od_id_max_min(x, y)

od_id_character(x, y)
```

**Arguments**

- `x`: a vector of numeric, character, or factor values
- `y`: a vector of numeric, character, or factor values
- `ordermatters`: logical, does the order of values matter to pairing, default = FALSE

**Details**

In OD data it is common to have many 'oneway' flows from "A to B" and "B to A". It can be useful to group these and have a single ID that represents pairs of IDs with or without directionality, so they contain 'twoway' or bi-directional values.

od_id* functions take two vectors of equal length and return a vector of IDs, which are unique for each combination but the same for twoway flows.

- the Szudzik pairing function, on two vectors of equal length. It returns a vector of ID numbers.

This function superseeds od_id_order as it is faster on large datasets

**See Also**

- `od_oneway`

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate
Examples

```r
(d <- od_data_sample[2:9, 1:2])
(id <- od_id_character(d[[1]], d[[2]]))
duplicated(id)
od_id_szudzik(d[[1]], d[[2]])
od_id_max_min(d[[1]], d[[2]])
n <- 100
ids <- as.character(runif(n, 1e4, 1e7 - 1))
# benchmark of methods:
x <- data.frame(id1 = rep(ids, times = n),
id2 = rep(ids, each = n),
val = 1,
stringsAsFactors = FALSE)
bench::mark(check = FALSE, iterations = 10,
od_id_order(x),
od_id_character(x$id1, x$id2),
od_id_szudzik(x$id1, x$id2),
od_id_max_min(x$id1, x$id2))
```

---

### od_id_order

Generate ordered ids of OD pairs so lowest is always first

This function is slow on large datasets, see `szudzik_pairing` for faster alternative

#### Description

Generate ordered ids of OD pairs so lowest is always first. This function is slow on large datasets, see `szudzik_pairing` for faster alternative.

#### Usage

```r
od_id_order(x, id1 = names(x)[1], id2 = names(x)[2])
```

#### Arguments

- `x` A data frame or `SpatialLinesDataFrame`, representing an OD matrix.
- `id1` Optional (it is assumed to be the first column) text string referring to the name of the variable containing the unique id of the origin.
- `id2` Optional (it is assumed to be the second column) text string referring to the name of the variable containing the unique id of the destination.

#### Examples

```r
x <- data.frame(id1 = c(1, 1, 2, 2, 3), id2 = c(1, 2, 3, 1, 4))
od_id_order(x) # 4th line switches id1 and id2 so stplanr.key is in order
```
od_oneway

Aggregate od pairs they become non-directional

Description

For example, sum total travel in both directions.

Usage

od_oneway(x, attrib = names(x[-c(1:2)])[vapply(x[-c(1:2)], is.numeric, TRUE)], id1 = names(x)[1], id2 = names(x)[2], stplanr.key = NULL)

Arguments

x  A data frame or SpatialLinesDataFrame, representing an OD matrix
attrib  A vector of column numbers or names, representing variables to be aggregated. By default, all numeric variables are selected. aggregate
id1  Optional (it is assumed to be the first column) text string referring to the name of the variable containing the unique id of the origin
id2  Optional (it is assumed to be the second column) text string referring to the name of the variable containing the unique id of the destination
stplanr.key  Optional key of unique OD pairs regardless of the order, e.g., as generated by od_id_max_min() or od_id_szudzik()

Details

Flow data often contains movement in two directions: from point A to point B and then from B to A. This can be problematic for transport planning, because the magnitude of flow along a route can be masked by flows the other direction. If only the largest flow in either direction is captured in an analysis, for example, the true extent of travel will by heavily under-estimated for OD pairs which have similar amounts of travel in both directions. Flows in both direction are often represented by overlapping lines with identical geometries (see flowlines()) which can be confusing for users and are difficult to plot.

Value

oneway outputs a data frame (or sf data frame) with rows containing results for the user-selected attribute values that have been aggregated.

See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate
Examples

```r
(od_min = od_data_sample[c(1, 2, 9), 1:6])
(od_oneway = od_oneway(od_min))
nrow(od_oneway) < nrow(od_min) # result has fewer rows
sum(od_min$all) == sum(od_oneway$all) # but the same total flow
od_oneway(od_min, attrib = "all")
attrib <- which(vapply(flow, is.numeric, TRUE))
flow_oneway <- od_oneway(flow, attrib = attrib)
colSums(flow_oneway[attrib]) == colSums(flow[attrib]) # test if the colSums are equal
# Demonstrate the results from oneway and onewaygeo are identical
flow_oneway_geo <- onewaygeo(flowlines, attrib = attrib)
flow_oneway_sf <- od_oneway(flowlines_sf)
par(mfrow = c(1, 2))
plot(flow_oneway_geo, lwd = flow_oneway_geo$All / mean(flow_oneway_geo$All))
plot(flow_oneway_sf$geometry, lwd = flow_oneway_sf$All / mean(flow_oneway_sf$All))
par(mfrow = c(1, 1))
od_max_min <- od_oneway(od_min, stplanr.key = od_id_character(od_min[[1]], od_min[[2]]))
cor(od_max_min$all, od_oneway$all)
# benchmark performance
bench::mark(check = FALSE, iterations = 3,
onewayid(flowlines_sf, attrib),
od_oneway(flowlines_sf)
)
```

---

**od_radiation**

*Function that estimates flow between points or zones using the radiation model*

---

**Description**

This is an implementation of the radiation model proposed in a paper by Simini et al. (2012).

**Usage**

```r
od_radiation(p, pop_var = "population", proportion = 1)
```

**Arguments**

- **p**: A SpatialPoints dataframe, the first column of which contains a unique ID
- **pop_var**: A character string representing the variable that corresponds to the population of the zone or point
- **proportion**: A number representing the proportion of the population who commute (1, the default, means 100 percent of the population commute to work)

**References**

See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf, sp_aggregate

Examples

# load some points data
data(cents)
# plot the points to check they make sense
plot(cents)
class(cents)
# Create test population to model flows
set.seed(2050)
cents$population <- runif(n = nrow(cents), min = 100, max = 1000)
# estimate
flowlines_radiation <- od_radiation(cents, pop_var = "population")
flowlines_radiation$flow
sum(flowlines_radiation$flow, na.rm = TRUE) # the total flow in the system
sum(cents$population) # the total inter-zonal flow
plot(flowlines_radiation, lwd = flowlines_radiation$flow / 100)
points(cents, cex = cents$population / 100)

od_to_odmatrix

Convert origin-destination data from long to wide format

Description

This function takes a data frame representing travel between origins (with origin codes in name_orig, typically the 1st column) and destinations (with destination codes in name_dest, typically the second column) and returns a matrix with cell values (from attrib, the third column by default) representing travel between origins and destinations.

Usage

od_to_odmatrix(flow, attrib = 3, name_orig = 1, name_dest = 2)

Arguments

flow A data frame representing flows between origin and destinations
attrib A number or character string representing the column containing the attribute
data of interest from the flow data frame
name_orig A number or character string representing the zone of origin
name_dest A number or character string representing the zone of destination
See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, odmatrix_to_od, points2flow, points2odf, sp_aggregate

Examples

od_to_odmatrix(flow)
od_to_odmatrix(flow[1:9, ])
od_to_odmatrix(flow[1:9, ], attrib = "Bicycle")

---

onewaygeo  Aggregate flows so they become non-directional (by geometry - the slow way)

Description

Flow data often contains movement in two directions: from point A to point B and then from B to A. This can be problematic for transport planning, because the magnitude of flow along a route can be masked by flows the other direction. If only the largest flow in either direction is captured in an analysis, for example, the true extent of travel will be heavily underestimated for OD pairs which have similar amounts of travel in both directions. Flows in both directions are often represented by overlapping lines with identical geometries (see flowlines()) which can be confusing for users and are difficult to plot.

Usage

onewaygeo(x, attrib)

Arguments

x  A SpatialLinesDataFrame
attrib  A text string containing the name of the line’s attribute to aggregate or a numeric vector of the columns to be aggregated

Details

This function aggregates directional flows into non-directional flows, potentially halving the number of lines objects and reducing the number of overlapping lines to zero.

Value

onewaygeo outputs a SpatialLinesDataFrame with single lines and user-selected attribute values that have been aggregated. Only lines with a distance (i.e. not intra-zone flows) are included.
onewayid

Aggregate ods so they become non-directional

Description

For example, sum total travel in both directions.

Usage

onewayid(x, attrib, id1 = names(x)[1], id2 = names(x)[2], stplanr.key = od_id_order(x, id1, id2))

## S3 method for class 'data.frame'
onewayid(x, attrib, id1 = names(x)[1], id2 = names(x)[2], stplanr.key = od_id_order(x, id1, id2))

## S3 method for class 'SpatialLines'
onewayid(x, attrib, id1 = names(x)[1], id2 = names(x)[2], stplanr.key = od_id_order(x, id1, id2))

Arguments

x A data frame or SpatialLinesDataFrame, representing an OD matrix
attrib A vector of column numbers or names for deciding which attribute(s) of class numeric to aggregate

Examples

plot(flowlines[1:30,], lwd = flowlines$On.foot[1:30])
singlines <- onewaygeo(flowlines[1:30,], attrib = which(names(flowlines) == "On.foot"))
plot(singlines, lwd = singlines$On.foot / 2, col = "red", add = TRUE)
## Not run:
plot(flowlines, lwd = flowlines$All / 10)
singlelines <- onewaygeo(flowlines, attrib = 3:14)
plot(singlelines, lwd = singlelines$All / 20, col = "red", add = TRUE)
sum(singlelines$All) == sum(flowlines$All)
nrow(singlelines)
singlelines_sf <- onewaygeo(flowlines_sf, attrib = 3:14)
sum(singlelines_sf$All) == sum(flowlines_sf$All)
summary(singlelines$All == singlelines_sf$All)

## End(Not run)
id1  Optional (it is assumed to be the first column) text string referring to the name of the variable containing the unique id of the origin

id2  Optional (it is assumed to be the second column) text string referring to the name of the variable containing the unique id of the destination

stplanr.key  A key of unique OD pairs regardless of the order, autogenerate by `od_id_order()`

Details

Flow data often contains movement in two directions: from point A to point B and then from B to A. This can be problematic for transport planning, because the magnitude of flow along a route can be masked by flows the other direction. If only the largest flow in either direction is captured in an analysis, for example, the true extent of travel will be heavily under-estimated for OD pairs which have similar amounts of travel in both directions. Flows in both direction are often represented by overlapping lines with identical geometries (see `flowlines()`) which can be confusing for users and are difficult to plot.

Value

`onewayid` outputs a data.frame with rows containing results for the user-selected attribute values that have been aggregated.

See Also

Other lines: `angle_diff`, `geo_toptail`, `is_linepoint`, `line2df`, `line Bearing`, `line_match`, `line_midpoint`, `line_sample`, `line_segment`, `line_via`, `mats2line`, `n_sample_length`, `n_vertices`, `onewaygeo`, `points2line`, `toptail_buff`, `toptailgs`, `update_line_geometry`

Examples

```r
flow_oneway <- onewayid(flow, attrib = 3)
nrow(flow_oneway) < nrow(flow) # result has fewer rows
sum(flow$All) == sum(flow_oneway$All) # but the same total flow
# using names instead of index for attribute
onewayid(flow, attrib = "All")
# using many attributes to aggregate
attrib <- which(vapply(flow, is.numeric, TRUE))
flow_oneway <- onewayid(flow, attrib = attrib)
colSums(flow_oneway[attrib]) == colSums(flow[attrib]) # test if the colSums are equal
 Demonstrate the results from onewayid and onewaygeo are identical
flow_oneway_geo <- onewaygeo(flowlines, attrib = attrib)
plot(flow_oneway$All, flow_oneway_geo$All)
flow_oneway_sf <- onewayid(flowlines_sf, 3)
plot(flow_oneway_geo, lwd = flow_oneway_geo$All / mean(flow_oneway_geo$All))
plot(flow_oneway_sf$geometry, lwd = flow_oneway_sf$All / mean(flow_oneway_sf$All))
# with spatial data
data(flowlines)
fo <- onewayid(flowlines, attrib = "All")
head(fo@data)
plot(fo)
sum(fo$All) == sum(flowlines$All)
# test results for one line
```
n <- 3
plot(fo[n, ], lwd = 20, add = TRUE)
f_over_n <- rgeos::gEquals(fo[n, ], flowlines["All"], byid = TRUE)
sum(flowlines$All[f_over_n]) == sum(fo$All[n]) # check aggregation worked
plot(flowlines[which(f_over_n)[1], ], add = TRUE, col = "white", lwd = 10)
plot(flowlines[which(f_over_n)[2], ], add = TRUE, lwd = 5)

osm_net_example

Example of OpenStreetMap road network

Description
Example of OpenStreetMap road network

Format
An sf object

Examples

overline

Convert series of overlapping lines into a route network

Description
This function takes a series of Lines stored in a SpatialLinesDataFrame and converts these into a single route network.

Usage
overline(sl, attrib, fun = sum, na.zero = FALSE, buff_dist = 0)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sl</td>
<td>A SpatialLinesDataFrame with overlapping elements</td>
</tr>
<tr>
<td>attrib</td>
<td>A character vector corresponding to the variables in sl$ on which the function(s) will operate.</td>
</tr>
<tr>
<td>fun</td>
<td>The function(s) used to aggregate the grouped values (default: sum). If length of fun is smaller than attrib then the functions are repeated for subsequent attributes.</td>
</tr>
<tr>
<td>na.zero</td>
<td>Sets whether aggregated values with a value of zero are removed.</td>
</tr>
<tr>
<td>buff_dist</td>
<td>A number specifying the distance in meters of the buffer to be used to crop lines before running the operation. If the distance is zero (the default) touching but non-overlapping lines may be aggregated.</td>
</tr>
</tbody>
</table>
Author(s)

Barry Rowlingson

References


See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum, calc_catchment, calc_moving_catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overline2, plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, sln2points, sum_network_links, sum_network_routes

Examples

```r
sl <- routes_fast[2:4, ]
rnet1 <- overline(sl = sl, attrib = "length")
rnet2 <- overline(sl = sl, attrib = "length", buff_dist = 1)
plot(rnet1, lwd = rnet1$length / mean(rnet1$length))
plot(rnet2, lwd = rnet2$length / mean(rnet2$length))
## Not run:
# sf methods
sl <- routes_fast_sf[2:4, ]
rnet_sf <- overline(sl = sl, attrib = "length", buff_dist = 10)
plot(rnet_sf, lwd = rnet_sf$length / mean(rnet_sf$length))
## End(Not run)
```

---

**overline2**

Convert series of overlapping lines into a route network (new method)

Description

This function is intended as a replacement for overline() and is significantly faster especially on large datasets. However, it also uses more memory.

Usage

```
overline2(x, attrib, ncores = 1, simplify = TRUE,
         regionalise = 1e+05)
```
overline2

Arguments

x An SF data.frame of LINESTRINGS
attrib character, column names in x to be summed
ncores integer, how many cores to use in parallel processing, default = 1
simplify logical, if TRUE group final segments back into lines, default = TRUE
regionalise integer, during simplification regionalisation is used if the number of segments exceeds this value

Details

The overline2 function breaks each line into many straight segments and then looks for duplicated segments. Attributes are summed for all duplicated segments, and if simplify is TRUE the segments with identical attributes are recombined into linestrings.

Regionalisation breaks the dataset into a 10 x 10 grid and then performed the simplification across each grid. This significantly reduces computation time for large datasets, but slightly increases the final file size. For smaller datasets it increases computation time slightly but reduces memory usage and so may also be useful.

A known limitation of this method is that overlapping segments of different lengths are not aggregated. This can occur when lines stop halfway down a road. Typically these errors are small, but some artefacts may remain within the resulting data.

For very large datasets nrow(x) > 1000000, memory usage can be significant. In these cases is is possible to overline subsets of the dataset, rbind the results together, and then overline again, to produce a final result.

Multicore support is only enabled for the regionalised simplification stage as it does not help with other stages.

Value

An SF data.frame of LINESTRINGS

Author(s)

Malcolm Morgan

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum, calc_catchment, calc-moving-catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overline, plot, SpatialLinesNetwork, ANY-method.plot, sfNetwork, ANY-method, sln2points, sum_network_links, sum_network_routes

Examples

sl = routes_fast_sf[routes_fast_sf$length > 0, ]
s1$bicycle = 1
system.time({rnet1 = overline2(sl, “bicycle”)})

system.time({rnet2 = overline2(sl, “bicycle”, ncores = 4)})
overline_intersection

Convert series of overlapping lines into a route network

Description

This function takes overlapping LINESTRINGs stored in an sf object and returns a route network composed of non-overlapping geometries and aggregated values.

Usage

overline_intersection(sl, attrib, fun = sum, na.zero = FALSE, buff_dist = 0)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sl</td>
<td>An sf LINESTRING object with overlapping elements</td>
</tr>
<tr>
<td>attrib</td>
<td>A character vector corresponding to the variables in sl$ on which the function(s) will operate.</td>
</tr>
<tr>
<td>fun</td>
<td>The function(s) used to aggregate the grouped values (default: sum). If length of fun is smaller than attrib then the functions are repeated for subsequent attributes.</td>
</tr>
<tr>
<td>na.zero</td>
<td>Sets whether aggregated values with a value of zero are removed.</td>
</tr>
<tr>
<td>buff_dist</td>
<td>A number specifying the distance in meters of the buffer to be used to crop lines before running the operation. If the distance is zero (the default) touching but non-overlapping lines may be aggregated.</td>
</tr>
</tbody>
</table>
Examples

```r
routes_fast_sf$value = 1
sl <- routes_fast_sf[4:6,]
attrib = c("value", "length")
rnet = overline_intersection(sl = sl, attrib)
plot(rnet, lwd = rnet$value)
# A larger example
sl <- routes_fast_sf[4:7,]
rnet = overline_intersection(sl = sl, attrib = c("value", "length"))
plot(rnet, lwd = rnet$value)

rnet_sf <- overline(routes_fast_sf[4:7,], attrib = c("value", "length"), buff_dist = 10)
plot(rnet_sf, lwd = rnet_sf$value)

# An even larger example (not shown, takes time to run)
# rnet = overline_intersection(routes_fast_sf, attrib = c("value", "length"))
# rnet_sf <- overline(routes_fast_sf[4:7,], attrib = c("value", "length"), buff_dist = 10)
# plot(rnet$geometry, lwd = rnet$value * 2, col = "grey")
# plot(rnet_sf$geometry, lwd = rnet_sf$value, add = TRUE)
```

Description

Plot an sfNetwork

Usage

```r
## S4 method for signature 'sfNetwork,ANY'
plot(x, component = "sl", ...)
```

Arguments

- `x` The sfNetwork to plot
- `component` The component of the network to plot. Valid values are "sl" for the geographic
  (sf) representation or "graph" for the graph representation.
- `...` Arguments to pass to relevant plot function.

See Also

Other rnet: `SpatialLinesNetwork`, `calc_catchment_sum`, `calc_catchment`, `calc_moving_catchment`,
`calc_network_catchment`, `find_network_nodes`, `gsection`, `islines`, `lineLabels`, `overline2`,
`overline`, `plot`, `SpatialLinesNetwork`, `ANY-method`, `sln2points`, `sum_network_links`,
`sum_network_routes`

Examples

```r
sln_sf <- SpatialLinesNetwork(route_network_sf)
plot(sln_sf)
```
plot(SpatialLinesNetwork, ANY-method)

*Plot a SpatialLinesNetwork*

**Description**

Plot a SpatialLinesNetwork

**Usage**

```r
## S4 method for signature 'SpatialLinesNetwork,ANY'
plot(x, component = "sl", ...)
```

**Arguments**

- `x`: The SpatialLinesNetwork to plot
- `component`: The component of the network to plot. Valid values are "sl" for the geographic (SpatialLines) representation or "graph" for the graph representation.
- `...`: Arguments to pass to relevant plot function.

**See Also**

Other rnet: SpatialLinesNetwork, calc_catchment_sum, calc_catchment, calc_moving_catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overline2, overline.plot, sfNetwork, ANY-method, sln2points, sum_network_links, sum_network_routes

**Examples**

```r
sln <- SpatialLinesNetwork(route_network)
plot(sln)
plot(sln, component = "graph")
```

---

**points2flow**

*Convert a series of points into geographical flows*

**Description**

Takes a series of geographical points and converts them into a spatial (linestring) object representing the potential flows, or 'spatial interaction', between every combination of points.

**Usage**

points2flow(p)
points2line

Arguments

p  A spatial (point) object

See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2odf, sp_aggregate

Examples

data(cents)
plot(cents)
flow <- points2flow(cents)
plot(flow, add = TRUE)
flow_sf <- points2flow(cents_sf)
plot(flow_sf)

points2line

Convert a series of points, or a matrix of coordinates, into a line

Description

This is a simple wrapper around spLines() that makes the creation of SpatialLines objects easy and intuitive

Usage

points2line(p)

Arguments

p  A spatial (points) object or matrix representing the coordinates of points.

See Also

Other lines: angle_diff, geo_toptail, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_sample, line_segment, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, toptail_buff, toptailgs, update_line_geometry

Examples

p <- matrix(1:4, ncol = 2)
library(sp)
l <- points2line(p)
plot(l)
l <- points2line(cents)
plot(l)
p <- line2points(routes_fast)
points2odf  

Convert a series of points into a dataframe of origins and destinations

Description

Takes a series of geographical points and converts them into a data.frame representing the potential flows, or 'spatial interaction', between every combination of points.

Usage

points2odf(p)

Arguments

p  A spatial points object

See Also

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, sp_aggregate

Examples

data(cents)
df <- points2odf(cents)
cents_centroids <- rgeos::gCentroid(cents, byid = TRUE)
df2 <- points2odf(cents_centroids)
df3 <- points2odf(cents_sf)

quadrant  

Split a spatial object into quadrants

Description

Split a spatial object (initially tested on SpatialPolygons) into quadrants.

Usage

quadrant(sp_obj, number_out = FALSE)
**Arguments**

- **sp_obj** Spatial object
- **number_out** Should the output be numbers from 1:4 (FALSE by default)

**Details**

Returns a character vector of NE, SE, SW, NW corresponding to north-east, south-east quadrants respectively. If number_out is TRUE, returns numbers from 1:4, respectively.

**See Also**

Other geo: `bbox_scale`, `buff_geo`, `crs_select_aeq`, `decode_gl`, `gclip`, `geo_bb_matrix`, `geo_bb`, `mapshape_available`, `mapshape`, `reproject`

**Examples**

```r
data(zones)
sp_obj <- zones
(quads <- quadrant(sp_obj))
plot(sp_obj, col = factor(quads))
points(rgeos::gCentroid(sp_obj), col = "white")
# edge cases (e.g. when using rasters) lead to NAs
sp_obj <- raster::rasterToPolygons(raster::raster(ncol = 3, nrow = 3))
(quads <- quadrant(sp_obj))
plot(sp_obj, col = factor(quads))
```

---

**Description**

Import and format UK 'Stats19' road traffic casualty data

**Usage**

```r
read_stats19_ac(data_dir = tempdir(), filename = "Accidents0514.csv")
```

**Arguments**

- **data_dir** Character string representing where the data is stored. If empty, R will attempt to download and unzip the data for you.
- **filename** Character string of the filename of the .csv to read in - default values are those downloaded from the UK Department for Transport (DfT).

**Details**

This is a wrapper function to access and load stats19 data in a user-friendly way. The function returns a data frame, in which each record is a reported incident in the stats19 dataset.

Ensure you have a fast internet connection and at least 100 Mb space.
Examples

```r
## Not run:
ac <- read_stats19_ac()

## End(Not run)
```

---

### read_stats19_ca

Import and format UK 'Stats19' road traffic casualty data

---

**Description**

Import and format UK 'Stats19' road traffic casualty data

**Usage**

```r
read_stats19_ca(data_dir = tempdir(), filename = "Casualties0514.csv")
```

**Arguments**

- `data_dir` Character string representing where the data is stored. If empty, R will attempt to download and unzip the data for you.
- `filename` Character string of the filename of the .csv to read in - default values are those downloaded from the UK Department for Transport (DfT).

**Details**

This is a wrapper function to access and load stats 19 data in a user-friendly way. The function returns a data frame, in which each record is a reported incident in the stats19 dataset.

Ensure you have a fast internet connection and at least 100 Mb space.

**Examples**

```r
## Not run:
ca <- read_stats19_ca()

## End(Not run)
```
**read_stats19_ve**

Import and format UK 'Stats19' road traffic casualty data

**Description**

Import and format UK 'Stats19' road traffic casualty data

**Usage**

```
read_stats19_ve(data_dir = tempdir(), filename = "Vehicles0514.csv")
```

**Arguments**

- **data_dir**: Character string representing where the data is stored. If empty, R will attempt to download and unzip the data for you.
- **filename**: Character string of the filename of the .csv to read in - default values are those downloaded from the UK Department for Transport (DfT).

**Details**

This is a wrapper function to access and load stats 19 data in a user-friendly way. The function returns a data frame, in which each record is a reported incident in the stats19 dataset.

Ensure you have a fast internet connection and at least 100 Mb space.

**Examples**

```
## Not run:
ve <- read_stats19_ve()

## End(Not run)
```

---

**read_table_builder**

Import and format Australian Bureau of Statistics (ABS) TableBuilder files

**Description**

Import and format Australian Bureau of Statistics (ABS) TableBuilder files

**Usage**

```
read_table_builder(dataset, filetype = "csv", sheet = 1, removeTotal = TRUE)
```
Arguments

dataset Either a dataframe containing the original data from TableBuilder or a character string containing the path of the unzipped TableBuilder file.

filetype A character string containing the filetype. Valid values are ‘csv’, ‘legacycsv’ and ‘xlsx’ (default = ‘csv’). Required even when dataset is a dataframe. Use ‘legacycsv’ for csv files derived from earlier versions of TableBuilder for which csv outputs were csv versions of the xlsx files. Current csv output from TableBuilder follow a more standard csv format.

sheet An integer value containing the index of the sheet in the xlsx file (default = 1).

removeTotal A boolean value. If TRUE removes the rows and columns with totals (default = TRUE).

Details

The Australian Bureau of Statistics (ABS) provides customised tables for census and other datasets in a format that is difficult to use in R because it contains rows with additional information. This function imports the original (unzipped) TableBuilder files in .csv or .xlsx format before creating an R dataframe with the data.

Examples

data_dir <- system.file("extdata", package = "stplanr")
t1 <- read_table_builder(file.path(data_dir, "SA1Population.csv"))
t2 <- read_table_builder(file.path(data_dir, "SA1Population.xlsx"),
filetype = "xlsx", sheet = 1, removeTotal = TRUE)
salpop <- read.csv(file.path(data_dir, "SA1Population.csv"), header = FALSE)
t3 <- read_table_builder(salpop)

reproject

Reproject lat/long spatial object so that they are in units of 1m

description

Many GIS functions (e.g. finding the area)

Usage

reproject(shp, crs = crs_select_aeq(shp))

Arguments

shp A spatial object with a geographic (WGS84) coordinate system

crs An optional coordinate reference system (if not provided it is set automatically by crs_select_aeq()).
route

See Also

Other geo: bbox_scale, buff_geo, crs_select_aeq, decode_gl, gclip, geo_bb_matrix, geo_bb, mapshape_available, mapshape, quadrant

Examples

data(routes_fast)
rf_aeq <- reproject(routes_fast[1:3, ])
rf_osgb <- reproject(routes_fast[1:3, ], 27700)

route

Plan routes on the transport network

Description

Takes origins and destinations, finds the optimal routes between them and returns the result as a spatial (sf or sp) object. The definition of optimal depends on the routing function used

Usage

route(from = NULL, to = NULL, l = NULL,
route_fun = stplanr::route_cyclestreet, n_print = 10,
list_output = FALSE, ...)

Arguments

from An object representing origins (if lines are provided as the first argument, from is assigned to l)
to An object representing destinations
l Only needed if from and to are empty, in which case this should be a spatial object representing desire lines
route_fun A routing function to be used for converting the straight lines to routes od2line()
n_print A number specifying how frequently progress updates should be shown
list_output If FALSE (default) assumes spatial (linestring) object output. Set to TRUE to save output as a list.
... Arguments passed to the routing function, e.g. route_cyclestreet()

See Also

Other routes: line2routeRetry, line2route, nearest2spdf, route_cyclestreet, route_dodgr, route_graphhopper, route_local, route_osrm, route_transportapi_public, viaroute2sldf, viaroute
Examples

```r
## Not run:
from <- "Leek, UK"
to <- "Hereford, UK"
route_leek_to_hereford <- route(from, to)
route(cents_sf[1:3, ], cents_sf[2:4, ]) # sf points
route(flowlines_sf[2:4, ])) # lines

## End(Not run)
```

## routes_fast

A `spatstat.solist` object with 84 points and 1 column

**Description**

A spatial lines dataset of commuter flows on the travel network representing the 'fastest' between `cents()` objects with `od2line()` (see `flow()`).

**Usage**

```r
data(routes_fast)
```

**Format**

A spatial lines dataset with 49 rows and 15 columns

**See Also**

Other example data: `destination_zones`, `flow_dests`, `flowlines`, `flow`, `route_network`, `routes_slow`

## routes_slow

A `spatstat.solist` object with 84 points and 1 column

**Description**

A spatial lines dataset of commuter flows on the travel network representing the 'quietest' between `cents()` objects with `od2line()` (see `flow()`).

**Usage**

```r
data(routes_slow)
```

**Format**

A spatial lines dataset with 49 rows and 15 columns

**See Also**

Other example data: `destination_zones`, `flow_dests`, `flowlines`, `flow`, `route_network`, `routes_fast`
route_cyclestreet

Plan a single route with CycleStreets.net

Description

Provides an R interface to the CycleStreets.net cycle planning API, a route planner made by cyclists for cyclists. The function returns a SpatialLinesDataFrame object representing the an estimate of the fastest, quietest or most balance route. Currently only works for the United Kingdom and part of continental Europe, though other areas may be requested by contacting CycleStreets. See https://www.cyclestreets.net/api/ for more information.

Usage

route_cyclestreet(from, to, plan = "fastest", silent = TRUE, pat = NULL, base_url = "https://www.cyclestreets.net", reporterrors = TRUE, save_raw = "FALSE")

Arguments

from
Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth.

to
Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth. This represents the destination of the trip.

plan
Text strong of either "fastest" (default), "quietest" or "balanced"

silent
Logical (default is FALSE). TRUE hides request sent.

pat
The API key used. By default this is set to NULL and this is usually aquired automatically through a helper, api_pat().

base_url
The base url from which to construct API requests (with default set to main server)

reporterrors
Boolean value (TRUE/FALSE) indicating if cyclestreets (TRUE by default). should report errors (FALSE by default).

save_raw
Boolean value which returns raw list from the json if TRUE (FALSE by default).

Details

This function uses the online routing service CycleStreets.net to find routes suitable for cyclists between origins and destinations. Requires an internet connection, a CycleStreets.net API key and origins and destinations within the UK (and various areas beyond) to run.

Note that if from and to are supplied as character strings (instead of lon/lat pairs), Google’s geocoding services are used via geo_code().

You need to have an api key for this code to run. Loading a locally saved copy of the api key text string before running the function, for example, will ensure it is available on any computer:
route_dodgr

Route on local data using the dodgr package

Description
Route on local data using the dodgr package

Usage
route_dodgr(from = NULL, to = NULL, l = NULL, net = NULL)

Arguments
from An object representing origins (if lines are provided as the first argument, from is assigned to l)
to An object representing destinations
route_graphhopper

1. Only needed if from and to are empty, in which case this should be a spatial object representing desire lines

net sf object representing the route network

See Also

Other routes: line2routeRetry, line2route, nearest2spdf, route_cyclestreet, route_graphhopper, route_local, route_osrm, route_transportapi_public, route, viaroute2sldf, viaroute

Examples

```r
# from <- geo_code("pedallers arms leeds")
from <- c(-1.5327, 53.8006)
# to <- geo_code("gzing")
to <- c(-1.5279, 53.8044)
# next 4 lines recreate `\texttt{stplanr::osm\_net\_example}`
# pts <- rbind(from, to)
# colnames(pts) <- c("x", "y")
# net <- dodgr::dodgr\_streetnet(pts = rbind(from, to), expand = 0.1)
# osm\_net\_example <- net[c("highway", "name", "lanes", "maxspeed")]
r <- route_dodgr(from, to, net = osm\_net\_example)
plot(osm\_net\_example\$geometry)
plot(r\$geometry, add = TRUE, col = "red", lwd = 5)
```

route_graphhopper

**Plan a route with the graphhopper routing engine**

Description

Provides an R interface to the graphhopper routing engine, an open source route planning service.

Usage

```r
route_graphhopper(from, to, l = NULL, vehicle = "bike",
silent = TRUE, pat = NULL, base_url = "https://graphhopper.com")
```

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth.</td>
</tr>
<tr>
<td>to</td>
<td>Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth. This represents the destination of the trip.</td>
</tr>
<tr>
<td>l</td>
<td>Only needed if from and to are empty, in which case this should be a spatial object representing desire lines</td>
</tr>
<tr>
<td>vehicle</td>
<td>A text string representing the vehicle. Can be bike (default), car or foot. See <a href="https://graphhopper.com/api/1/docs/supported-vehicle-profiles/">https://graphhopper.com/api/1/docs/supported-vehicle-profiles/</a> for further details.</td>
</tr>
</tbody>
</table>
silent Logical (default is FALSE). TRUE hides request sent.

pat The API key used. By default this is set to NULL and this is usually acquired automatically through a helper, api_pat().

base_url The base url from which to construct API requests (with default set to main server)

Details

The function returns a SpatialLinesDataFrame object. See https://github.com/graphhopper for more information.

To test graphhopper is working for you, try something like this, but with your own API key: To use this function you will need to obtain an API key from https://graphhopper.com/#directions-api. It is assumed that you have set your api key as a system environment for security reasons (so you avoid typing the API key in your code). Do this by adding the following to your .Renviron file (see ?.Renviron or the 'api-packages' vignette at https://cran.r-project.org/package=httr for more on this):

```r
GRAPHHOPPER='FALSE-Key-eccbf612-214e-437d-8b73-06bdf9e6877d'.
```

(Note: key not real, use your own key.)

```r
obj <- jsonlite::fromJSON(url)
```

Where `url` is an example api request from https://github.com/graphhopper/directions-api/blob/master/routing.md.

See Also

route_cyclestreet

Other routes: line2routeRetry, line2route, nearest2spdf, route_cyclestreet, route_dodgr, route_local, route_osrm, route_transportapi_public, route, viaroute2sldf, viaroute

Examples

```r
## Not run:
from <- c(-0.12, 51.5)
to <- c(-0.14, 51.5)

r1 <- route_graphhopper(from = from, to = to, silent = FALSE)
r2 <- route_graphhopper("London Eye", "Westminster", vehicle = "foot")
r3 <- route_graphhopper("London Eye", "Westminster", vehicle = "car")

plot(r1)
plot(r2, add = TRUE, col = "blue") # compare routes
plot(r3, add = TRUE, col = "red")

## End(Not run)
```
**route_local**

**Plan a route with local data**

**Description**

This function returns the shortest path between locations in, or near to, segments on a `SpatialLinesNetwork`.

**Usage**

```r
route_local(sln, from, to, l = NULL)
```

**Arguments**

- `sln`: The `SpatialLinesNetwork` to use.
- `from`: Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth.
- `to`: Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth. This represents the destination of the trip.
- `l`: Only needed if `from` and `to` are empty, in which case this should be a spatial object representing desire lines

**See Also**

Other routes: `line2routeRetry`, `line2route`, `nearest2spdf`, `route_cyclestreet`, `route_dodgr`, `route_graphhopper`, `route_osrm`, `route_transportapi_public`, `route`, `viaroute2sldf`, `viaroute`

**Examples**

```r
from <- c(-1.535181, 53.82534)
to <- c(-1.52446, 53.80949)
sln <- SpatialLinesNetwork(route_network_sf)
r <- route_local(sln, from, to)
plot(sln)
plot(r$geometry, add = TRUE, col = "red", lwd = 5)
plot(cents[c(3, 4),], add = TRUE)
r2 <- route_local(sln = sln, cents_sf[3, ], cents_sf[4, ])
plot(r2$geometry, add = TRUE, col = "blue", lwd = 3)
```
route_network

A spatial lines dataset representing a route network

Description

The flow of commuters using different segments of the road network represented in the `flowlines()` and `routes_fast()` datasets

Usage

data(route_network)

Format

A spatial lines dataset 80 rows and 1 column

See Also

Other example data: destination_zones, flow_dests, flowlines, flow, routes_fast, routes_slow

Examples

```r
## Not run:
# Generate route network
data <- overline(routes_fast, “All”, fun = sum)
data_sf <- sf::st_as_sf(data)
## End(Not run)
```

route_osrm

Plan a route with OSRM

Description

This is a wrapper around viaroute that returns a single route between A and B.

Usage

`route_osrm(from, to, l = NULL, alt = FALSE, ..., singleline = TRUE)`
**route_transportapi_public**

**Description**

Provides an R interface to the TransportAPI.com public transport API. The function returns a SpatialLinesDataFrame object representing the public route. Currently only works for the United Kingdom. See [https://developer.transportapi.com/documentation](https://developer.transportapi.com/documentation) for more information.

**Arguments**

- **from**: Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth.
- **to**: Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth. This represents the destination of the trip.
- **l**: Only needed if from and to are empty, in which case this should be a spatial object representing desire lines.
- **alt**: Boolean value to return alternative routes (default = TRUE).
- **...**: Arguments passed to viaroute()
- **singleline**: Should a single line be returned? Default is TRUE.

**Note**

The public-facing OSRM routing service (the default) only provides routing for cars by default. For details, see [https://github.com/Project-OSRM/osrm-backend/issues/4530](https://github.com/Project-OSRM/osrm-backend/issues/4530).

**See Also**

Other routes: `line2routeRetry`, `line2route`, `nearest2spdf`, `route_cyclestreet`, `route_dodgr`, `route_graphhopper`, `route_local`, `route_transportapi_public`, `route`, `viaroute2sldf`, `viaroute`.

**Examples**

```r
## Not run:
from <- c(-1.55, 53.80) # geo_code("leeds")
to <- c(-1.76, 53.80) # geo_code("bradford uk")
r <- route_osrm(from, to)
plot(r)
r_many <- line2route(flowlines_sf[2:9, ], route_osrm, time_delay = 1)
plot(cents)
plot(r_many$geometry)
```

---

**Plan a single route with TransportAPI.com**

---
route_transportapi_public

Usage

route_transportapi_public(from, to, silent = FALSE, 
  region = "southeast", modes = NA, not_modes = NA)

Arguments

from Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth.
to Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth. This represents the destination of the trip.
silent Logical (default is FALSE). TRUE hides request sent.
region String for the active region to use for journey plans. Possible values are 'southeast' (default) or 'tfl'.
modes Vector of character strings containing modes to use. Default is to use all modes.
not_modes Vector of character strings containing modes not to use. Not used if modes is set.

Details

This function uses the online routing service TransportAPI.com to find public routes between origins and destinations. It does not require any key to access the API.

Note that if from and to are supplied as character strings (instead of lon/lat pairs), Google’s geocoding services are used via geo_code.

Note: there is now a dedicated transportAPI package: https://github.com/ITSLeeds/transportAPI

See Also

line2route

Other routes: line2routeRetry, line2route, nearest2spdf, route_cyclestreet, route_dodgr, route_graphhopper, route_local, route_osrm, route_viaroute2sldf, viaroute

Examples

## Not run:
# Plan the 'public' route from Hereford to Leeds
rqh <- route_transportapi_public(from = "Hereford", to = "Leeds")
plot(rqh)

## End(Not run)

# Aim plan public transport routes with transportAPI
**sfNetwork-class**

An S4 class representing a (typically) transport network

**Description**

This class uses a combination of a sf layer and an igraph object to represent transport networks that can be used for routing and other network analyses.

**Slots**

s1 A sf line layer with the geometry and other attributes for each link the in network.
g The graph network corresponding to s1.
 nb A list containing vectors of the nodes connected to each node in the network.
 weightfield A character vector containing the variable (column) name from the SpatialLines-DataFrame to be used for weighting the network.

**sln2points**

Generate spatial points representing nodes on a SpatialLinesNetwork or sfNetwork.

**Description**

Generate spatial points representing nodes on a SpatialLinesNetwork or sfNetwork.

**Usage**

sln2points(sln)

**Arguments**

sln The SpatialLinesNetwork to use.

**See Also**

Other rnet: SpatialLinesNetwork, calc_catchment_sum, calc_catchment, calc_moving_catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overline2, overline.plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, sum_network_links, sum_network_routes

**Examples**

data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
(sln_nodes <- sln2points(sln))
plot(sln)
plot(sln_nodes, add = TRUE)
SpatialLinesNetwork Create object of class SpatialLinesNetwork or sfNetwork

Description

Creates a new SpatialLinesNetwork (for SpatialLines) or sfNetwork (for sf) object that can be used for routing analysis within R.

Usage

SpatialLinesNetwork(sl, uselonglat = FALSE, tolerance = 0)

Arguments

sl A SpatialLines or SpatialLinesDataFrame containing the lines to use to create the network.
uselonglat A boolean value indicating if the data should be assumed to be using WGS84 latitude/longitude coordinates. If FALSE or not set, uses the coordinate system specified by the SpatialLines object.
tolerance A numeric value indicating the tolerance (in the units of the coordinate system) to use as a tolerance with which to match nodes.

Details

This function is used to create a new SpatialLinesNetwork from an existing SpatialLines or SpatialLinesDataFrame object. A typical use case is to represent a transport network for routing and other network analysis functions. This function and the corresponding SpatialLinesNetwork class is an implementation of the SpatialLinesNetwork developed by Edzer Pebesma and presented on RPubs. The original implementation has been rewritten to better support large (i.e., detailed city-size) networks and to provide additional methods useful for conducting transport research following on from the initial examples provided by Janoska(2013).

References


See Also

Other rnet: calc_catchment_sum, calc_catchment, calc_moving_catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overlineZ, overline.plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, sln2points, sum_network_links, sum_network_routes
SpatialLinesNetwork-class

An S4 class representing a (typically) transport network

Description

This class uses a combination of a SpatialLinesDataFrame and an igraph object to represent transport networks that can be used for routing and other network analyses.

Slots

- sl: A SpatialLinesDataFrame with the geometry and other attributes for each link in the network.
- g: The graph network corresponding to sl.
- nb: A list containing vectors of the nodes connected to each node in the network.
- weightfield: A character vector containing the variable (column) name from the SpatialLinesDataFrame to be used for weighting the network.

sp_aggregate

Aggregate SpatialPolygonsDataFrame to new geometry.

Description

Aggregate SpatialPolygonsDataFrame to new geometry.

Usage

sp_aggregate(zones, aggzones, cols = FALSE, FUN = sum,
prop_by_area = ifelse(identical(FUN, mean) == FALSE, TRUE, FALSE),
digits = getOption("digits"))
**Arguments**

- **zones**
  A spatial object representing origins (and destinations if no separate destinations object is provided) of travel.

- **aggzones**
  A SpatialPolygonsDataFrame containing the new boundaries to aggregate to.

- **cols**
  A character vector containing the names of columns on which to apply FUN. By default, all numeric columns are aggregated.

- **FUN**
  Function to use on aggregation. Default is sum.

- **prop_by_area**
  Boolean value indicating if the values should be proportionally adjusted based on area. Default is TRUE unless FUN = mean.

- **digits**
  The number of digits to use when proportionally adjusting values based on area. Default is the value ofgetOption("digits").

**Value**

SpatialPolygonsDataFrame

**Details**

This function performs aggregation on a SpatialPolygonsDataFrame to a different geometry specified by another SpatialPolygons object.

**See Also**

Other od: dist_google, od2line, od2odf, od_aggregate_from, od_aggregate_to, od_aggregate_from, od_aggregate_to, od_aggregate, od_coords2line, od_coords, od_dist, od_id, od_oneway, od_radiation, od_to_odmatrix, odmatrix_to_od, points2flow, points2odf

**Examples**

```r
## Not run:
zones$data$region <- 1
zones$data[, c(2, 5), c("region")]) <- 2
aggzones <- sp::SpatialPolygonsDataFrame(rgeos::gUnaryUnion(zones,
  id = zones$data$region
), data.frame(region = c(1, 2)))
zones$data$region <- NULL
zones$data$exdata <- 5
library(sp)
sp_aggregate(zones, aggzones)

## End(Not run)
```
Deprecated functions in stplanr

Description

These functions are depreciated and will be removed:

Details

- `dl_stats19()`: see the `stats19` package

summary, sfNetwork-method

Print a summary of a sfNetwork

Description

Print a summary of a sfNetwork

Usage

```r
## S4 method for signature 'sfNetwork'
summary(object, ...)
```

Arguments

- `object`: The sfNetwork
- `...`: Arguments to pass to relevant summary function.

Examples

```r
data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
summary(sln)
```
### summary,SpatialLinesNetwork-method

*Print a summary of a SpatialLinesNetwork*

**Description**

Print a summary of a SpatialLinesNetwork

**Usage**

```r
## S4 method for signature 'SpatialLinesNetwork'
summary(object, ...)
```

**Arguments**

- `object` The SpatialLinesNetwork
- `...` Arguments to pass to relevant summary function.

**Examples**

```r
data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
summary(sln)
```

### sum_network_links

*Summarise links from shortest paths data*

**Description**

Summarise links from shortest paths data

**Usage**

```r
sum_network_links(sln, routedata)
```

**Arguments**

- `sln` The SpatialLinesNetwork or sfNetwork to use.
- `routedata` A dataframe where the first column contains the Node ID(s) of the start of the routes, the second column indicates the Node ID(s) of the end of the routes, and any additional columns are summarised by link. If there are no additional columns, then overlapping routes are counted.
sum_network_routes

Details

Find the shortest path on the network between specified nodes and returns a SpatialLinesDataFrame or sf containing the path(s) and summary statistics of each one.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum, calc_catchment, calc_moving_catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overline2, overline, plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, sln2points, sum_network_routes

Examples

```r
sln_sf <- SpatialLinesNetwork(route_network_sf)
plot(sln_sf)

nodes_df <- data.frame(
  start = rep(c(1, 2, 3, 4, 5), each = 4),
  end = rep(c(50, 51, 52, 33), times = 5)
)

weightfield(sln_sf) # field used to determine shortest path

library(sf)

shortpath_sf <- sum_network_links(sln_sf, nodes_df)

plot(shortpath_sf["count"], lwd = shortpath_sf$count, add = TRUE)
```

---

sum_network_routes Summarise shortest path between nodes on network

Description

Summarise shortest path between nodes on network

Usage

`sum_network_routes(sln, start, end, sumvars, combinations = FALSE)`

Arguments

- `sln` The SpatialLinesNetwork to use.
- `start` Node ID(s) of route starts.
- `end` Node ID(s) of route ends.
- `sumvars` Character vector of variables for which to calculate summary statistics.
- `combinations` Boolean value indicating if all combinations of start and ends should be calculated. If TRUE then every start Node ID will be routed to every end Node ID. This is faster than passing every combination to start and end. Default is FALSE.
Details

Find the shortest path on the network between specified nodes and returns a SpatialLinesDataFrame containing the path(s) and summary statistics of each one.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum, calc_catchment, calc_moving_catchment, calc_network_catchment, find_network_nodes, gsection, islines, lineLabels, overline2, overline.plot, SpatialLinesNetwork, ANY-method.plot, sfNetwork, ANY-method.sln2points, sum_network_links

Examples

sln <- SpatialLinesNetwork(route_network)
weightfield(sln) # field used to determine shortest path
shortpath <- sum_network_routes(sln, start = 1, end = 50, sumvars = "length")
plot(shortpath, col = "red", lwd = 4)
plot(sln, add = TRUE)

Description

Return Matrix containing travel times between origins and destinations

Usage

table2matrix(lat, lng, destlat = NA, destlng = NA, api = 5,
profile = "driving", protocol = "v1",
osrmurl = "http://router.project-osrm.org")

Arguments

lat Numeric vector containing latitude coordinate for each coordinate to calculate travel times. Also accepts dataframe with latitude in the first column and longitude in the second column.

lng Numeric vector containing longitude coordinate for each coordinate to calculate travel times.

destlat Numeric vector containing destination latitude coordinate for each coordinate to calculate travel times. Also accepts dataframe with latitude in the first column and longitude in the second column. Default is value of lat.

destlng Numeric vector containing longitude coordinate for each destination coordinate to calculate travel times. Default is value of lng.

api An integer value containing the OSRM API version (either 4 or 5). Default is 5.
toptailgs

profile OSRM profile to use (for API v5), defaults to "driving".
protocol The protocol to use for the API (for v5), defaults to "v1".
osrmurl Base URL of the OSRM service

Details
Return a matrix containing travel times between origins and destinations

Examples

## Not run:
table2matrix(seq(from = 50, to = 52, by = 0.1), seq(from = 12, to = 14, by = 0.1))

## End(Not run)

---

toptailgs  

Clip the first and last n metres of SpatialLines

Description

Takes lines and removes the start and end point, to a distance determined by the user. Uses the geosphere::distHaversine function and requires coordinates in WGS84 (lng/lat).

Usage

toptailgs(l, toptail_dist, tail_dist = NULL)

Arguments

l A SpatialLines object
toptail_dist The distance (in metres) to top the line by. Can be either a single value or a vector of the same length as the SpatialLines object. If tail_dist is missing, is used as the tail distance.
tail_dist The distance (in metres) to tail the line by. Can be either a single value or a vector of the same length as the SpatialLines object.

See Also

Other lines: angle_diff, geo_toptail, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_sample, line_segment, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptail_buff, update_line_geometry
Examples

data("routes_fast")
rf <- routes_fast[2:3, ]
r_toptail <- toptailgs(rf, toptail_dist = 300)
plot(rf, lwd = 3)
plot(r_toptail, col = "red", add = TRUE)
plot(cents, add = TRUE)

toptyl_buff

Clip the beginning and ends SpatialLines to the edge of SpatialPolygon borders

Description

Takes lines and removes the start and end point, to a distance determined by the nearest polygon border.

Usage

toptyl_buff(l, buff, ...)

Arguments

l A SpatialLines object
buff A SpatialPolygons object to act as the buffer
... Arguments passed to rgeos::gBuffer()

See Also

Other lines: angle_diff, geo_toptyl, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_sample, line_segment, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptylgs, update_line_geometry

Examples

r_toptyl <- toptyl_buff(routes_fast, zones)
sel <- row.names(routes_fast) %in% row.names(r_toptyl)
rf_cross_poly <- routes_fast[sel, ]
plot(zones)
plot(routes_fast, col = "blue", lwd = 4, add = TRUE)
# note adjacent lines removed
plot(rf_cross_poly, add = TRUE, lwd = 2)
plot(r_toptyl, col = "red", add = TRUE)
update_line_geometry  

Update line geometry

Description

Take two SpatialLines objects and update the geometry of the former with that of the latter, retaining the data of the former.

Usage

update_line_geometry(l, nl)

Arguments

l  
A SpatialLines object, whose geometry is to be modified

nl  
A SpatialLines object of the same length as l to provide the new geometry

See Also

Other lines: angle_diff, geo_toptail, is_linepoint, line2df, line_bearing, line_match, line_midpoint, line_sample, line_segment, line_via, mats2line, n_sample_length, n_vertices, onewaygeo, onewayid, points2line, toptail_buff, toptailgs

Examples

data(flowlines)
l <- flowlines[2:5, ]
nl <- routes_fast
nrow(l)
nrow(nl)
l <- l[!is_linepoint(l), ]
names(l)
names(nl)
l_newgeom <- update_line_geometry(l, nl)
plot(l, lwd = l$All / mean(l$All))
plot(l_newgeom, lwd = l$All / mean(l$All))
names(l_newgeom)

viaroute  

Query OSRM service and return json string result

Description

Query OSRM service and return json string result
Usage

viaroute(startlat = NULL, startlng = NULL, endlat = NULL, endlng = NULL, viapoints = NULL, api = 5, profile = "driving", protocol = "v1", osrmurl = "http://router.project-osrm.org", zoom = 18, instructions = TRUE, alt = TRUE, geometry = TRUE, uturns = "default")

Arguments

startlat  A single value or vector containing latitude(s) of the start of routes.
startlng  A single value or vector containing longitude(s) of the end of routes.
endlat    A single value or vector containing latitude(s) of the end of routes.
endlng    A single value or vector containing longitude(s) of the end of routes.
viapoints A list of dataframes containing latitude (first column), longitude (second) column for points to use for each route. Optionally a third column containing a boolean value indicating if u-turns are allowed at each viapoint.
api       An integer value containing the OSRM API version (either 4 or 5). Default is 5.
profile   OSRM profile to use (for API v5), defaults to "driving".
protocol  The protocol to use for the API (for v5), defaults to "v1".
osrmurl   URL for OSRM service, e.g. an osrm instance running on localhost. By default this is "http://router.project-osrm.org".
zoom      Zoom level for route geometry (0 to 18) for API v4 (default = 18). Higher values are more detailed.
instructions Boolean value to return instructions (default = TRUE).
alt        Boolean value to return alternative routes (default = TRUE).
geometry   Boolean value to return route geometries (default = TRUE).
uturns    Boolean value to allow u-turns at via points (default = TRUE).

Details

Constructs the query URL used with the OSRM HTTP API and returns a string or vector of strings with the json-encoded results. Can be used in conjunction with the viaroute2sldf function.

See also https://cran.r-project.org/web/packages/osrm/index.html

See Also

Other routes: line2routeRetry, line2route, nearest2spdf, route_cyclestreet, route_dodgr, route_graphhopper, route_local, route_osrm, route_transportapi_public, route, viaroute2sldf

Examples

## Not run:
exroutes <- viaroute(50, 0, 51, 1)
r <- viaroute2sldf(exroutes)
plot(r)
lngs <- c(-33.5, -33.6, -33.7)
lats <- c(150, 150.1, 150.2)
exroutes <- viaroute(viapoints = list(data.frame(x = lngs, y = lats)))
r <- viaroute2sldf(exroutes)

# End(Not run)

viaroute2sldf

Convert json result of OSRM routing query to SpatialLinesDataFrame

Description

Convert json result of OSRM routing query to SpatialLinesDataFrame

Usage

viaroute2sldf(osrmresult, return_sf = FALSE)

Arguments

osrmresult Single character string or character vector containing encoded json result(s) of OSRM routing queries.
return_sf Boolean value if this function should return an sf object, if FALSE returns sp object (default FALSE).

Details

Converts the result of a (successful) OSRM routing query and returns a SpatialLinesDataFrame containing the route, route summary and instructions.

See Also

Other routes: line2routeRetry, line2route, nearest2spdf, route_cyclestreet, route_dodgr, route_graphhopper, route_local, route_osrm, route_transportapi_public, route, viaroute

Examples

## Not run:
viaroute2sldf(
  viaroute(
    startlat = 52.503033,
    startlng = 13.420526,
    endlat = 52.516582,
    endlng = 13.429290
  )
)

# End(Not run)
weightfield

Get or set weight field in SpatialLinesNetwork

Description

Get or set value of weight field in SpatialLinesNetwork

Usage

weightfield(x)

weightfield(x, varname) <- value

weightfield(x, varname) <- value

## S4 method for signature 'SpatialLinesNetwork'
weightfield(x)

## S4 method for signature 'sfNetwork'
weightfield(x)

## S4 replacement method for signature 'SpatialLinesNetwork,ANY'
weightfield(x) <- value

## S4 replacement method for signature 'sfNetwork,ANY'
weightfield(x) <- value

## S4 replacement method for signature 'SpatialLinesNetwork,character'
weightfield(x, varname) <- value

## S4 replacement method for signature 'sfNetwork,character'
weightfield(x, varname) <- value

Arguments

x SpatialLinesNetwork to use
varname The name of the variable to set/use.
value Either the name of the variable to use as the weight field or a dataframe or vector containing the weights to use if varname is passed to the replacement function. If the dataframe contains multiple columns, the column with the same name as varname is used, otherwise the first column is used.
writeGeoJSON

Details

These functions manipulate the value of weightfield in a SpatialLinesNetwork. When changing the value of weightfield, the weights of the graph network are updated with the values of the corresponding variables.

Examples

data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
weightfield(sln) <- "length"
weightfield(sln, "randomnum") <- sample(1:10, size = nrow(sln@sl), replace = TRUE)

writeGeoJSON shp filename

Argument

shp The spatial object a to be cropped
filename File name of the output geojson

zones

Spatial polygons of home locations for flow analysis.

Description

These correspond to the cents() data.

Details

- geo_code. the official code of the zone

Examples

zones
zones_sf
plot(zones_sf)
Index

*Topic datasets
  ca_local, 15
cents, 16
destination_zones, 19
flow, 23
flow_dests, 25
flowlines, 24
l_poly, 57
od_data_sample, 64
osm_net_example, 73
route_network, 92
routes_fast, 86
routes_slow, 86
zones, 109

*Topic package
  stplanr-package, 5

angle_diff, 5, 33, 37, 41–44, 46, 50, 54, 55, 71, 72, 79, 103–105

as_sf_fun, 6
as_sp_fun (as_sf_fun), 6

bb2poly (geo_bb), 28
bbox_scale, 7, 8, 17, 18, 28, 29, 49, 81, 85
bearing(), 6, 41
buff_geo, 7, 7, 17, 18, 28, 29, 49, 81, 85

calca_local, 15
calc_catchment, 8, 11, 15, 15, 23, 34, 36, 40, 74, 75, 77, 78, 95, 96, 101, 102
calc_catchment(), 5
calc_catchment_sum, 9, 10, 13, 15, 23, 34, 36, 40, 74, 75, 77, 78, 95, 96, 101, 102
calc_moving_catchment, 9, 11, 12, 15, 23, 34, 36, 40, 74, 75, 77, 78, 95, 96, 101, 102
calc_network_catchment, 9, 11, 13, 13, 23, 34, 36, 40, 74, 75, 77, 78, 95, 96, 101, 102
cents, 16
cents(), 23, 56, 57, 59–61, 64, 86, 109
cents_sf (cents), 16
crs_select_aeq, 7, 8, 17, 25, 29, 49, 81, 85
crs_select_aeq(), 32, 84
decode_gl, 7, 8, 17, 18, 28, 29, 49, 81, 85
destination_zones, 19, 24, 25, 86, 92
destinations (destination_zones), 19
destinations_sf (destination_zones), 19
dist_google, 20, 57, 58, 60–65, 67, 69, 70, 79, 80, 98
dl_stats19, 21
dl_stats19(), 99

find_network_nodes, 9, 11, 13, 15, 22, 34, 36, 40, 74, 75, 77, 78, 95, 96, 101, 102
flow, 19, 23, 25, 86, 92
flow(), 24, 56, 57, 59–61, 64, 86
flow_dests, 19, 24, 25, 28, 86, 92
flowlines, 19, 24, 25, 28, 86, 92
flowlines(), 67, 70, 72, 92
flowlines sf (flowlines), 24
format_stats19_ac, 25
format_stats19_ca, 26
format_stats19_ve, 27
gclip, 7, 8, 17, 18, 27, 28, 29, 49, 81, 85
gcd_bb, 7, 8, 17, 18, 28, 29, 49, 81, 85
gcd_bb_matrix, 7, 8, 17, 18, 28, 29, 49, 81, 85
geo_buffer, 30
geo_code, 30, 47, 53
geo_length, 31
geo_projected, 32
geo_select_aeq, 32
geo_toptail, 6, 33, 37, 41–44, 46, 50, 54, 55, 71, 72, 79, 103–105
gProjected (geo_projected), 32
INDEX

line2route, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route(), 39
line2routeRetry, 38, 39, 51, 85, 88–91, 93, 94, 106, 107
line2route retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry(), 39
line2routeRetry(), 38, 39, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry(), 39
line2route_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry(), 39
line2route_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry(), 39
line2route_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry(), 39
line2route_retry_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry(), 39
line2route_retry_retry_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry_retry, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry_retry(), 39
line2route_retry_retry_retry_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry_retry_retry, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry_retry_retry(), 39
line2route_retry_retry_retry_retry_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry_retry_retry retries, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry_retry_retry_retry, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
line2route_retry_retry_retry_retry_retry_retry_retry_retry_retry(),
overline2, 9, 11, 13, 15, 23, 34, 36, 40, 74, 74, 77, 78, 95, 96, 101, 102
overline_intersection, 76
plot, sfNetwork, ANY-method, 77
plot, SpatialLinesNetwork, ANY-method, 78
points2flow, 20, 57, 58, 60–65, 67, 69, 70, 78, 80, 98
points2line, 6, 33, 37, 41–44, 46, 50, 54, 55, 71, 72, 79, 103–105
points2odf, 20, 57, 58, 60–65, 67, 69, 70, 79, 80, 98
quadrant, 7, 8, 17, 18, 28, 29, 49, 80, 85
read_stats19_ac, 81
read_stats19_ca, 82
read_stats19_ve, 83
read_table_builder, 83
reproject, 7, 8, 17, 18, 28, 29, 49, 81, 84
rgeos::gLength(), 32
rgeos::gSimplify(), 48
route, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
route_cyclestreet, 38, 40, 51, 85, 87, 89–91, 93, 94, 106, 107
route_cyclestreet(), 5, 38, 39, 85
route_cyclestreets (route_cyclestreet), 87
route_dodgr, 38, 40, 51, 85, 88, 89, 90, 91, 93, 94, 106, 107
route_graphhopper, 38, 40, 51, 85, 88, 89, 90, 91, 93, 94, 106, 107
route_local, 38, 40, 51, 85, 88–90, 91, 93, 94, 106, 107
route_network, 19, 24, 25, 86, 92
route_network_sf (route_network), 92
route_osrm, 38, 40, 51, 85, 88–91, 92, 94, 106, 107
route_transportapi_public, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
routes_fast, 19, 24, 25, 86, 89, 92
routes_fast(), 92
routes_fast_sf (routes_fast), 86
routes_slow, 19, 24, 25, 86, 89, 92
routes_slow_sf (routes_slow), 86
sfNetwork-class, 95
sln2points, 9, 11, 13, 15, 23, 34, 36, 40, 74, 75, 77, 78, 95, 96, 101, 102
sp_aggregate, 20, 57, 58, 60–65, 67, 69, 70, 79, 80, 97
SpatialLinesMidPoints(), 43
SpatialLinesNetwork, 9, 11, 13, 15, 23, 34, 36, 40, 74, 75, 77, 78, 95, 96, 101, 102
SpatialLinesNetwork-class, 97
stLines(), 79
stplanr (stplanr-package), 5
stplanr-deprecated, 99
stplanr-package, 99
sum_network_links, 9, 11, 13, 15, 23, 34, 36, 40, 74, 75, 77, 78, 95, 96, 100, 102
sum_network_routes, 9, 11, 13, 15, 23, 34, 36, 40, 74, 75, 77, 78, 95, 96, 101, 102
summary.sfNetwork-method, 99
summary.SpatialLinesNetwork-method, 100
system(), 48
table2matrix, 102
to tail (geo_to tail), 33
to tail_buff, 6, 33, 37, 41–44, 46, 50, 54, 55, 71, 72, 79, 103, 104, 105
to tail ogs, 6, 33, 37, 41–44, 46, 50, 54, 55, 71, 72, 79, 103, 104, 105
to tail ogs(), 33
update_line_geometry, 6, 33, 37, 41–44, 46, 50, 54, 55, 71, 72, 79, 103, 104, 105
viaroute, 38, 40, 51, 85, 88–91, 93, 94, 105, 107
viaroute2sldf, 38, 40, 51, 85, 88–91, 93, 94, 106, 107
weightfield, 108
weightfield.sfNetwork-method
(weightfield), 108
weightfield.SpatialLinesNetwork-method
(weightfield), 108
weightfield<- (weightfield), 108
weightfield<-, sfNetwork, ANY-method
(weightfield), 108
weightfield<-, sfNetwork, character-method
(weightfield), 108
weightfield<-, SpatialLinesNetwork, ANY-method
(weightfield), 108
weightfield<-, SpatialLinesNetwork, character-method (weightfield), 108
writeGeoJSON, 109
zones, 109
zones_sf(zones), 109