Package ‘stplanr’

January 7, 2019

Type Package

Title Sustainable Transport Planning

Version 0.2.7

Maintainer Robin Lovelace <rob00x@gmail.com>

Description Functionality and data access tools for transport planning, including origin-destination analysis, route allocation and modelling travel patterns.

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), rgdal (>= 1.3.4), 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)

VignetteBuilder knitr

URL https://github.com/ropensci/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],
Barry Rowlingson [ctb] (Author of overline),

Nick Bearman [ctb] (Co-author of gclip),
Malcolm Morgan [ctb] (Co-author of angle_diff),
Nikolai Berkoff [ctb] (Co-author of line2route),
Scott Chamberlin [rev] (Scott reviewed the package for rOpenSci, see
https://github.com/ropensci/onboarding/issues/10)

Repository CRAN

Date/Publication 2019-01-07 16:40:10 UTC

R topics documented:

stplanr-package 4
angle_diff 5
as_sf_fun 6
bbox_scale 6
buff_geo 7
calc_catchment 8
calc_catchment_sum 10
calc_moving_catchment 11
calc_network_catchment 13
ca_local 14
cents 15
crs_select_aeq 16
decode_gl 17
destination_zones 18
dist_google 19
dl_stats19 20
find_network_nodes 21
flow 22
flowlines 23
flow_dests 23
format_stats19_ac 24
format_stats19_ca 25
format_stats19_ve 25
gclip 26
geo_bb 27
geo_bb_matrix 28
geo_buffer 28
geo_code 29
geo_length 30
geo_projected 30
geo_select_aeq 31
geo_toptail 32
gsection 32
gtfs2sldf 33
islines 34
is_linepoint 35
line2df 35

line2route 36
line2routeRetry 37
lineLabels 38
line_bearing 38
line_length 39
line_match 39
line_midpoint 40
line_sample 40
line_segment 41
line_to_points 42
line_via 42
locate2spdf 43
l_poly 44
mapshape 45
mapshape_available 46
mats2line 46
nearest2spdf 47
nearest_cyclestreets 48
nearest_google 49
nearest_osm 49
n_sample_length 50
n_vertices 51
od2line 52
od2odf 53
od_aggregate 54
od_coords 55
od_dist 56
od_id_order 57
od_radiation 57
onewaygeo 58
onewayid 59
overline 61
plot,sfNetwork,ANY-method 62
plot,SpatialLinesNetwork,ANY-method 63
points2flow 63
points2line 64
points2odf 65
quadrant 65
read_stats19_ac 66
read_stats19_ca 67
read_stats19_ve 67
read_table_builder 68
reproject 69
route 70
routes_fast 71
routes_slow 71
route_cyclestreet 71
route_graphhopper 73

route_local . . . 74
route_network . . . 75
route_osrm . . . 76
route_transportapi_public . . . 77
sfNetwork-class . . . 78
sln2points . . . 78
SpatialLinesNetwork . . . 79
SpatialLinesNetwork-class . . . 80
sp_aggregate . . . 81
summary,sfNetwork-method . . . 82
summary,SpatialLinesNetwork-method . . . 82
sum_network_links . . . 83
sum_network_routes . . . 84
table2matrix . . . 84
toptailgs . . . 85
toptail_buff . . . 86
update_line_geometry . . . 87
viaroute . . . 87
viaroute2sldf . . . 89
weightfield . . . 89
writeGeoJSON . . . 91
zones . . . 91

Index **92**

`stplanr-package` **stplanr: Sustainable Transport Planning with R**

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` *Calculate the angular difference between lines and a predefined bearing*

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 oposite 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 definied as 0, East as 90 and West as -90.

Author(s)

Robin Lovelace and Malcolm Morgan

Examples

```
data(flowlines)
# 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

```
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 shp
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.

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)

Details

Returns a

Examples

```
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

```
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=0,0,0,0,0,0,0 +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.

`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.

Examples

```
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"))
unzip(file.path(data_dir, "testcycleway.zip"))
sa1income <- rgdal::readOGR(".", "smallsa1")
testcycleway <- rgdal::readOGR(".", "testcycleway")
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)
sa1income <- sf::read_sf("smallsa1.shp")
testcycleway <- sf::read_sf("testcycleway.shp")
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

```
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.

Examples

```
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"))
unzip(file.path(data_dir, "testcycleway.zip"))
sa1income <- readOGR(".", "smallsa1")
testcycleway <- readOGR(".", "testcycleway")
calc_catchment_sum(
  polygonlayer = sa1income,
  targetlayer = testcycleway,
  calccols = c("Total"),
  distance = 800,
  projection = "austalbers"
)

calc_catchment_sum(
  polygonlayer = sa1income,
  targetlayer = testcycleway,
  calccols = c("Total"),
  distance = 800,
  projection = "austalbers"
)

## End(Not run)
```

`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)
- `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).

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.

Examples

```
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"))
unzip(file.path(data_dir, "testcycleway.zip"))
sa1income <- readOGR(".", "smallsa1")
testcycleway <- readOGR(".", "testcycleway")
calc_moving_catchment(
  polygonlayer = sa1income,
  targetlayer = testcycleway,
  calccols = c("Total"),
  distance = 800,
  projection = "austalbers"
)

## End(Not run)
```

calc_network_catchment

Calculate catchment area and associated summary statistics using network.

Description

Calculate catchment area and associated summary statistics using network.

Usage

```
calc_network_catchment(sln, polygonlayer, targetlayer, calccols,
  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.

Examples

```
## 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())
sa1income <- readOGR(tempdir(), "smallsa1")
testcycleway <- readOGR(tempdir(), "testcycleway")
sydroads <- readOGR(tempdir(), "roads")
sydnetwork <- SpatialLinesNetwork(sydroads)
calc_network_catchment(
  sln = sydnetwork,
  polygonlayer = sa1income,
  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

```
## 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

```
## Not run:
cents <- rgdal::readOGR(dsn = "/home/robin/npct/pct-bigdata/cents.geojson", layer = "OGRGeoJSON")
# library(geojsonio) # load with the ropensci package geojsonio if rgdal fails
# cents <- geojsonio::geojson_read(x = "~/repos/pct/pct-data/national/cents.geojson")
crs <- sp::CRS("+init=epsg:4326")
crsuk <- sp::CRS("+init=epsg:27700")
cents <- sp::spTransform(x = cents, CRSobj = crsuk)
home <- geo_code("LS7 3HB")
home <- sp::SpatialPoints(matrix(home, ncol = 2), proj4string = crs)
home <- sp::spTransform(x = home, CRSobj = crsuk)
buf <- rgeos::gBuffer(home, width = 2000)
# Check it saved the points OK
cents <- cents[buf, ]
plot(buf)
points(cents)
cents <- sp::spTransform(x = cents, CRSobj = crs)
cents$geo_code <- as.character(cents$geo_code)
library(devtools)
# use_data(cents, overwrite = TRUE)
cents_sf <- sf::st_as_sf(cents)
devtools::use_data(cents_sf)

## 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

```
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

Examples

```
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.

Examples

```
## 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 `cents`.

Usage

```
data(destination_zones)
```

Format

A spatial dataset with 87 features

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

Absent 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.

Examples

```
## 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)
```

`dl_stats19` *Download Stats19 data*

Description

Download Stats19 data

Usage

```
dl_stats19(zip_url = paste0("http://data.dft.gov.uk.s3.amazonaws.com/",
  "road-accidents-safety-data/Stats19_Data_2005-2014.zip"),
  data_dir = tempdir())
```

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.

Examples

```
data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
SLN <- SpatialLinesNetwork(rnet)
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/.

Usage

```
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.

Examples

```
## 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` *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

Examples

```
## 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

```
## 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

```
## 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 funtions 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

Examples

```
data(cents)
cb <- rgeos::gBuffer(cents[8, ], width = 0.012, byid = TRUE)
plot(cents)
plot(cb, add = TRUE)
clipped <- gclip(cents, cb)
plot(clipped, add = TRUE)
clipped$avslope # gclip also returns the data attribute
points(clipped)
points(cents[cb, ], col = "red") # note difference
gclip(cents_sf, cb)
```

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

```
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

Examples

```
plot(geo_bb(routes_fast, distance = 100), col = "red")
plot(geo_bb(routes_fast, scale_factor = 0.8), col = "green", add = TRUE)
plot(geo_bb(sp::bbox(routes_fast)), add = TRUE) # works on bb also
plot(geo_bb(routes_fast, output = "points"), add = TRUE)
# Simple features implementation:
bb_sf1 <- geo_bb(routes_fast_sf, scale_factor = c(2, 1.5))
bb_sf2 <- geo_bb(routes_fast_sf, c(2, 1.5), distance = 100)
plot(bb_sf1)
plot(bb_sf2, add = TRUE)
plot(routes_fast, add = TRUE)
plot(geo_bb(routes_fast_sf, output = "points"), add = TRUE)
bb_matrix <- geo_bb(routes_fast, scale_factor = c(2, 1.1), output = "bb")
plot(routes_fast_sf[2:5, ], bbox = bb_matrix)
```

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)

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 aquired automatically through a helper, api_pat().

Examples

```
## 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

```
geo_length(shp)
```

Arguments

`shp`	A spatial line object

See Also

buff_geo

Examples

```
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

```
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 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

```
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

```
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

```
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

```
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.

Examples

```
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

```
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.

Examples

```
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, ]
rsec <- gsection(sl)
rsec <- gsection(sl, buff_dist = 100) # 4 features: issue

## End(Not run)
```

---

gtfs2sldf *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)
```

---

islines *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

Examples

```
## 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.

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 coords*

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

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 = "route_cyclestreet", n_print = 10,
  list_output = FALSE, l_id = NA, time_delay = 0, ...)
```

Arguments

`l`	A SpatialLinesDataFrame
`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 SpatialLinesDataFrame 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 github.com/ropensci/stplanr for details.

Examples

```
## Not run:
l <- flowlines[2:5, ]
r <- line2route(l, "route_osrm")
rf <- line2route(l = l, "route_cyclestreet", plan = "fastest")
rq <- line2route(l = l, 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, ]
rf_list <- line2route(l = l, list_output = TRUE)
line2route(l[1, ], route_graphhopper)

## End(Not run)
```

`line2routeRetry` *Convert straight SpatialLinesDataFrame from flow data into routes retrying on connection (or other) intermittent failures*

Description

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

Usage

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

Arguments

`lines`	A SpatialLinesDataFrame
`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.

Examples

```
## Not run:
data(flowlines)
rf_list <- line2routeRetry(flowlines[1:2, ], pattern = "nonexistanceerror", 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

```
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

`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

```
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 oposite 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.

Examples

```
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

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

Arguments

`l`	A spatial lines object
`byid`	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

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

Arguments

`l1`	A spatial object
`l2`	A spatial object
`threshold`	The threshold for a match - distances greater than this will not count as matches
`return_sp`	Should the function return a spatial result (FALSE by default)

Examples

```
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

```
line_midpoint(l)
```

Arguments

l	A spatial lines object

Examples

```
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

```
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

Examples

```
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

```
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 (overides n_segments if set)

Examples

```
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 SpatialLinesDataFrame 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 SpatialLinesDataFrame 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

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

line2pointsn(l)
```

Arguments

l	A SpatialLinesDataFrame
ids	Vector of ids (by default `1:nrow(l)`)

Examples

```
l <- routes_fast[2:4, ]
lpoints <- line_to_points(l)
lpoints2 <- line2pointsn(l)
plot(lpoints, pch = lpoints$id, cex = lpoints$id)
points(lpoints2, add = TRUE)
line_to_points(routes_fast_sf[2:4, ])
```

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

Examples

```
{
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 — *Return SpatialPointsDataFrame with located points from OSRM locate service*

Description

Return SpatialPointsDataFrame with located points from OSRM locate service

Usage

```
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.

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

```
## 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

```
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
dsn	The name of the temporary file to write to (deleted after use)
silent	Logical determining whether the function call is printed to screen `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. The percent argument refers to the percentage of removable points to retain. So `percent = 1` is a very aggressive simplication, saving a huge amount of hard-disk space. `rgeos::gSimplify()`

Details

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

Calls the JavaScript command-line GIS application 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.

Examples

```
## 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()
```

Examples

```
mapshape_available()
```

---

mats2line *Convert 2 matrices to lines*

Description

Convert 2 matrices to lines

Usage

```
mats2line(mat1, mat2)
```

Arguments

`mat1`	Matrix representing origins
`mat2`	Matrix representing origins

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.

Examples

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

## End(Not run)
```

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

Description

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

Usage

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

Arguments

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

Details

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

Examples

```
## 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_google	*Generate nearest point on the route network of a point using the Google Maps API*

Description

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

```
## 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

```
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.

Examples

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

## End(Not run)
```

---

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

```
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

Examples

```
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))
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 SpatalPolygons object

Details

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

Examples

```
n_vertices(routes_fast)
n_vertices(routes_fast_sf)
```

`od2line` *Convert flow data to SpatialLinesDataFrame*

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

```
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 = TRUE)

od2line2(flow, zones)
```

Arguments

`flow`	A data frame representing the flow between two points or zones. 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 SpatialPolygonsDataFrame or SpatialPointsDataFrame representing origins (and destinations if no separate destinations object is provided) of travel flows.
`destinations`	A SpatialPolygonsDataFrame or SpatialPointsDataFrame 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

The function expects zone codes to be in the 1st column of the zones/destinations datasets and the 1st and 2nd columns of the flow data, respectively.

`od2line2()` is a faster implementation (around 6 times faster on large datasets) that returns a `SpatialLines` object, omitting the data and working only when there is no destinations dataset (i.e. when the geography of origins is the same as that of destinations).

Examples

```
data(flow) # load example data - see ?flow for mor information
data(cents)
newflowlines <- od2line(flow = flow, zones = cents)
newflowlines2 <- od2line2(flow = flow, zones = cents)
plot(cents)
lines(newflowlines, lwd = 3)
lines(newflowlines2, col = "white")
nfl_sldf <- sp::SpatialLinesDataFrame(newflowlines, flow, match.ID = FALSE)
identical(nfl_sldf, newflowlines)
# When destinations are different
data(destinations)
head(flow_dests[1:5]) # check data
head(destinations@data[1:5])
flowlines_dests <- od2line(flow_dests, cents, destinations = destinations, silent = FALSE)
plot(flowlines_dests)
nfl_sf <- od2line(flow, zones_sf)
```

---

od2odf *Extract coordinates from OD data*

Description

Extract coordinates from OD data

Usage

```
od2odf(flow, zones)
```

Arguments

`flow`	A data frame representing the flow between two points or zones. 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 SpatialPolygonsDataFrame or SpatialPointsDataFrame representing origins and destinations of travel 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 can be tricky to plot and link-up with geographical data. This function makes the task easier.

References

Rae, A. (2009). From spatial interaction data to spatial interaction information? Geovisualisation and spatial structures of migration from the 2001 UK census. Computers, Environment and Urban Systems, 33(3). doi:10.1016/j.compenvurbsys.2009.01.007

Examples

```
data(flow)
data(zones)
od2odf(flow, zones)
```

`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 the flow between two points or zones. 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 SpatialPolygonsDataFrame or SpatialPointsDataFrame representing the original centroids or boundaries of the travel flow data. Note that in the case of a SpatialPointsDataFrame, the entirety of the flow will be allocated to the polygon in which the point is located rather than being distributed by area.
`aggzones`	A SpatialPolygonsDataFrame containing the new boundaries to aggregate to.
`aggzone_points`	Points representing origins of OD flows (typically population-weighted centroids)
`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.

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_coords — *General function to create a matrix representing origins and destinations*

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

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_dist` *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 the flow between two points or zones. 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 SpatialPolygonsDataFrame or SpatialPointsDataFrame representing origins (and destinations if no separate destinations object is provided) of travel flows.

Details

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

Examples

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

od_id_order — *Generate ordered ids of OD pairs so lowest is always first*

Description

Generate ordered ids of OD pairs so lowest is always first

Usage

```
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

```
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_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

```
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

Simini, F., Gonzalez, M.C., Maritan, A., Barabasi, A.L., 2012. A universal model for mobility and migration patterns. Nature. doi:10.1038/nature10856

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)
```

---

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 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.

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

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)
```

`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
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, autogenerated 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 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

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

Examples

```
data(flow)
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)
onewayid(flowlines_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)
```

---

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

`sl`	A SpatialLinesDataFrame with overlapping elements
`attrib`	A character vector corresponding to the variables in `sl$` on which the function(s) will operate.
`fun`	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.
`na.zero`	Sets whether aggregated values with a value of zero are removed.
`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.

Author(s)

Barry Rowlingson

References

Rowlingson, B (2015). Overlaying lines and aggregating their values for overlapping segments. Reproducible question from http://gis.stackexchange.com. See http://gis.stackexchange.com/questions/139681/overlaying-lines-and-aggregating-their-values-for-overlapping-segments.

Examples

```
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, ]
overline(sl = sl, attrib = "length", buff_dist = 10)
rnet_sf <- overline(routes_fast_sf, attrib = "length", buff_dist = 10)
plot(rnet_sf$geometry, lwd = rnet_sf$length / mean(rnet_sf$length))

## End(Not run)
```

`plot,sfNetwork,ANY-method`
Plot an sfNetwork

Description

Plot an sfNetwork

Usage

```
## 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.

Examples

```
SLN_sf <- SpatialLinesNetwork(route_network_sf)
plot(SLN_sf)
```

plot,SpatialLinesNetwork,ANY-method *Plot a SpatialLinesNetwork*

Description

Plot a SpatialLinesNetwork

Usage

```
## 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.

Examples

```
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 SpatialLinesDataFrame representing the potential flows, or 'spatial interaction', between every combination of points.

Usage

```
points2flow(p)
```

Arguments

`p`	SpatialPointsDataFrame

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 SpatialPoints obect or matrix representing the coordinates of points.

Examples

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

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

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.

Examples

```
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))
```

`read_stats19_ac` *Import and format UK 'Stats19' road traffic casualty data*

Description

Import and format UK 'Stats19' road traffic casualty data

Usage

```
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 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:
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

```
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

```
## 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
)
sa1pop <- read.csv(file.path(data_dir, "SA1Population.csv"), header = FALSE)
t3 <- read_table_builder(sa1pop)
```

---

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()).

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 = 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 SpatialLinesDataFrame output. Set to TRUE to save output as a list.
...	Arguments passed to the routing function, e.g. `route_cyclestreet()`

Examples

```
## 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 *spatial lines dataset of commuter flows on the travel network*

Description

spatial lines dataset of commuter flows on the travel network

Usage

```
data(routes_fast)
```

Format

A spatial lines dataset with 49 rows and 15 columns

routes_slow *spatial lines dataset of commuter flows on the travel network*

Description

spatial lines dataset of commuter flows on the travel network

Usage

```
data(routes_slow)
```

Format

A spatial lines dataset 49 rows and 15 columns

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:

`mytoken <- readLines("~/Dropbox/dotfiles/cyclestreets-api-key-rl") Sys.setenv(CYCLESTREET = mytoken)`

if you want the API key to be available in future sessions, set it using the .Renviron file e.g. on Linux machines in bash via:

`echo "CYCLESTREET=f3fe3d078ac34737" >> ~/.Renviron`

Read more about the .Renviron here: `?.Renviron`

See Also

line2route

Examples

```
## Not run:
from <- c(-1.55, 53.80) # geo_code("leeds")
to <- c(-1.76, 53.80) # geo_code("bradford uk")
json_output <- route_cyclestreet(from = from, to = to, plan = "quietest", save_raw = TRUE)
str(json_output) # what does cyclestreets give you?
rf_lb <- route_cyclestreet(from, to, plan = "fastest")
rf_lb@data
plot(rf_lb)
(rf_lb$length / (1000 * 1.61)) / # distance in miles
  (rf_lb$time / (60 * 60)) # time in hours - average speed here: ~8mph
# Plan a 'balanced' route from Pedaller's Arms to the University of Leeds
rb_pa <- route_cyclestreet("Pedaller's Arms, Leeds", "University of Leeds, UK", "balanced")

## End(Not run)
```

---

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

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

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
vehicle	A text string representing the vehicle. Can be bike (default), car or foot. See https://graphhopper.com/api/1/docs/supported-vehicle-profiles/ for further details.
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)

Details

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

To test graphopper 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):

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

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

`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

Examples

```
## 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, segements on a `SpatialLinesNetwork`.

Usage

```
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

Examples

```
bb = sf::st_bbox(routes_fast)
set.seed(5)
from = c(runif(1, bb$xmin, bb$xmax), runif(1, bb$ymin, bb$ymax))
to = c(runif(1, bb$xmin, bb$xmax), runif(1, bb$ymin, bb$ymax))
sln = SpatialLinesNetwork(routes_fast)
r <- route_local(sln, from, to)
plot(r)
 ## Not run:
# todo: next code chunk is not currently working:
r2 = route_local(sln = sln, cents_sf[5, ], cents_sf[6, ])
plot(r2)
r_many <- line2route(flowlines_sf[2:9, ], route_local, sln = sln)
plot(cents)
plot(sln@sl, add = TRUE)
plot(r_many, add = TRUE)

## End(Not run)
```

`route_network` *spatial lines dataset representing a route network*

Description

spatial lines dataset representing a route network

Usage

```
data(route_network)
```

Format

A spatial lines dataset 80 rows and 1 column

Examples

```
## Not run:
# Generate route network
route_network <- overline(routes_fast, "All", fun = sum)
route_network_sf <- sf::st_as_sf(route_network)

## 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)
```

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.

Examples

```
## 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)

## End(Not run)
```

route_transportapi_public
Plan a single route with TransportAPI.com

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/documentationfor more information.

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`.

See Also

line2route

Examples

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

## 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

`sl` A sf line layer with the geometry and other attributes for each link the in 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.

---

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.

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

Pebesma, E. (2013). Spatial Networks, URL:http://rpubs.com/edzer/6767.

Janoska, Z. (2013). Find shortest path in spatial network, URL:http://rpubs.com/janoskaz/10396.

Examples

```
SLN <- SpatialLinesNetwork(route_network)
class(SLN)
weightfield(SLN) # field used to determine shortest path
plot(SLN)
## Not run:
points(sln2points(SLN)[1, ], cex = 5)
points(sln2points(SLN)[50, ], cex = 5)
shortpath <- sum_network_routes(SLN, 1, 50, sumvars = "length")
plot(shortpath, col = "red", lwd = 4, add = TRUE)
points(sln2points(SLN)[35, ], cex = 5)
shortpath <- sum_network_routes(SLN, 1, 35, sumvars = "length")
plot(shortpath, col = "red", lwd = 4, add = TRUE)
library(sf)
SLN_sf <- SpatialLinesNetwork(route_network_sf)
plot(SLN_sf)
shortpath <- sum_network_routes(SLN_sf, 1, 50, sumvars = "length")
plot(shortpath$geometry, col = "red", lwd = 4, add = TRUE)

## End(Not run)
```

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 the in 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 SpatialPolygonsDataFrame or SpatialPointsDataFrame representing the original centroids or boundaries. Note that in the case of a SpatialPointsDataFrame, the original value will be allocated to the polygon in which the point is located rather than being distributed by area.
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 of getOption("digits").

Value

SpatialPolygonsDataFrame

Details

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

Examples

```
## 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)
```

---

summary,sfNetwork-method

Print a summary of a sfNetwork

Description

Print a summary of a sfNetwork

Usage

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

Arguments

object	The sfNetwork
...	Arguments to pass to relevant summary function.

Examples

```
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

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

Arguments

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

Examples

```
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

```
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 colums, then overlapping routes are counted.

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.

Examples

```
data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
SLN <- SpatialLinesNetwork(rnet)
weightfield(SLN) # field used to determine shortest path
shortpath <- sum_network_links(
  SLN,
  data.frame(
    start = rep(c(1, 2, 3, 4, 5), each = 4),
    end = rep(c(50, 51, 52, 33), times = 5)
  )
)
plot(shortpath, lwd = shortpath$count)
```

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.

Examples

```
data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
SLN <- SpatialLinesNetwork(rnet)
weightfield(SLN) # field used to determine shortest path
shortpath <- sum_network_routes(SLN, 1, 50, sumvars = "length")
plot(shortpath, col = "red", lwd = 4)
plot(SLN, add = TRUE)
```

table2matrix — *Return Matrix containing travel times between origins and destinations*

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.
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.

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)
```

`toptail_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

```
toptail_buff(l, buff, ...)
```

Arguments

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

Examples

```
r_toptail <- toptail_buff(routes_fast, zones)
sel <- row.names(routes_fast) %in% row.names(r_toptail)
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_toptail, 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

Examples

```
data(flowlines)
l <- flowlines[2:5, ]
nl <- routes_fast
nrow(l)
nrow(nl)
l <- l[!is_linepoint(l), ]
names(l)
names(routes_fast)
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 sservice, 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 uturns 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.

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)
plot(r)

## 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.

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.

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 *Write to geojson easily*

Description

Provides a user-friendly wrapper for rgdal::writeOGR(). Note, `geojson_write` from the geojsonio package provides the same functionality https://github.com/ropensci/geojsonio.

Usage

```
writeGeoJSON(shp, filename)
```

Arguments

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

```
## Not run:
zones <- rgdal::readOGR(dsn = "/home/robin/npct/pct-bigdata/msoas.geojson", layer = "OGRGeoJSON")
proj4string(zones) <- proj4string(cents)
zones <- zones[cents, ]
plot(zones)
points(cents)
zones_sf <- sf::st_as_sf(zones)

## End(Not run)
```

Index

∗Topic **datasets**
ca_local, 14
cents, 15
destination_zones, 18
flow, 22
flow_dests, 23
flowlines, 23
l_poly, 44
route_network, 75
routes_fast, 71
routes_slow, 71
zones, 91
∗Topic **package**
stplanr-package, 4

angle_diff, 5
as_sf_fun, 6
as_sp_fun *(*as_sf_fun*)*, 6

bb2poly *(*geo_bb*)*, 27
bbox_scale, 6
bearing(), *5*, *38*
buff_geo, 7

ca_local, 14
calc_catchment, 8
calc_catchment(), *4*
calc_catchment_sum, 10
calc_moving_catchment, 11
calc_network_catchment, 13
cents, 15
cents(), *22*, *52–54*, *56*, *91*
cents_sf *(*cents*)*, 15
crs_select_aeq, 16
crs_select_aeq(), *30*, *69*

decode_gl, 17
destination_zones, 18
destinations *(*destination_zones*)*, 18
destinations_sf *(*destination_zones*)*, 18
dist_google, 19
dl_stats19, 20

find_network_nodes, 21
flow, 22
flow(), *23*, *52–54*, *56*
flow_dests, 23
flowlines, 23
flowlines(), *58*, *60*
flowlines_sf *(*flowlines*)*, 23
format_stats19_ac, 24
format_stats19_ca, 25
format_stats19_ve, 25

gclip, 26
geo_bb, 27
geo_bb_matrix, 28
geo_buffer, 28
geo_code, 29
geo_length, 30
geo_projected, 30
geo_select_aeq, 31
geo_toptail, 32
gprojected *(*geo_projected*)*, 30
gsection, 32
gtfs2sldf, 33

is_linepoint, 35
islines, 34

l_poly, 44
line2df, 35
line2points *(*line_to_points*)*, 42
line2pointsn *(*line_to_points*)*, 42
line2route, 36
line2route(), *37*
line2routeRetry, 37
line_bearing, 38
line_length, 39
line_match, 39

line_midpoint, 40
line_sample, 40
line_segment, 41
line_to_points, 42
line_via, 42
lineLabels, 38
locate2spdf, 43

mapshape, 45
mapshape(), *46*
mapshape_available, 46
mats2line, 46

n_sample_length, 50
n_vertices, 51
nearest2spdf, 47
nearest_cyclestreets, 48
nearest_google, 49
nearest_osm, 49

od2line, 52
od2line(), *23*, *36*, *56*, *70*
od2line2 *(*od2line*)*, 52
od2line2(), *52*
od2odf, 53
od_aggregate, 54
od_coords, 55
od_dist, 56
od_id_order, 57
od_id_order(), *60*
od_radiation, 57
onewaygeo, 58
onewayid, 59
overline, 61
overline(), *4*, *34*

plot,sfNetwork,ANY-method, 62
plot,SpatialLinesNetwork,ANY-method, 63
points2flow, 63
points2line, 64
points2odf, 65

quadrant, 65

read_stats19_ac, 66
read_stats19_ca, 67
read_stats19_ve, 67
read_table_builder, 68
reproject, 69
rgeos::gLength(), *30*
rgeos::gSimplify(), *45*
route, 70
route_cyclestreet, 71
route_cyclestreet(), *4*, *36*, *37*, *70*
route_cyclestreets *(*route_cyclestreet*)*, 71
route_graphhopper, 73
route_local, 74
route_network, 75
route_network_sf *(*route_network*)*, 75
route_osrm, 76
route_transportapi_public, 77
routes_fast, 71
routes_fast_sf *(*routes_fast*)*, 71
routes_slow, 71
routes_slow_sf *(*routes_slow*)*, 71

sfNetwork-class, 78
sln2points, 78
sp_aggregate, 81
SpatialLinesMidPoints(), *40*
SpatialLinesNetwork, 79
SpatialLinesNetwork-class, 80
spLines(), *64*
stplanr *(*stplanr-package*)*, 4
stplanr-package, 4
sum_network_links, 83
sum_network_routes, 84
summary,sfNetwork-method, 82
summary,SpatialLinesNetwork-method, 82
system(), *45*

table2matrix, 84
toptail *(*geo_toptail*)*, 32
toptail_buff, 86
toptailgs, 85
toptailgs(), *32*

update_line_geometry, 87

viaroute, 87
viaroute2sldf, 89

weightfield, 89
weightfield,sfNetwork-method *(*weightfield*)*, 89
weightfield,SpatialLinesNetwork-method *(*weightfield*)*, 89

weightfield<- *(*weightfield*)*, 89
weightfield<-,sfNetwork,ANY-method
*(*weightfield*)*, 89
weightfield<-,sfNetwork,character-method
*(*weightfield*)*, 89
weightfield<-,SpatialLinesNetwork,ANY-method
*(*weightfield*)*, 89
weightfield<-,SpatialLinesNetwork,character-method
*(*weightfield*)*, 89
writeGeoJSON, 91

zones, 91
zones_sf *(*zones*)*, 91