

Package: pavement (via r-universe)

October 16, 2026

Title Analyzing Spatial Events on Roadways

Version 0.3.1

Description Pavement is a package designed to analyze spatial events occurring on roadways. It provides a comprehensive toolkit for working with spatial data, empowering users to understand patterns and trends in road-related phenomena.

License MIT + file LICENSE

URL <https://github.com/NONONOexe/pavement>,
<https://nononoexe.github.io/pavement/>

BugReports <https://github.com/NONONOexe/pavement/issues>

Depends R (>= 3.5)

Imports graphics, grDevices, igraph, lubridate, lwgeom, magrittr,
osmdata, plotly, rgl, rlang, sf, stats, utils

Suggests here, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

BuildManual TRUE

LinkingTo cpp11

VignetteBuilder knitr

Config/pak/sysreqs libabsl-dev cmake libfreetype6-dev libgdal-dev gdal-bin libgeos-dev libglpk-dev libglu1-mesa-dev make texlive libicu-dev libpng-dev libuv1-dev libxml2-dev libgl1-mesa-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev zlib1g-dev

Repository <https://nononoexe.r-universe.dev>

Date/Publication 2025-11-20 05:38:38 UTC

RemoteUrl <https://github.com/NONONOexe/pavement>

RemoteRef HEAD

RemoteSha 7c018e5ff5acb84a45117641d23c45dd848b7426

Contents

calculate_local_moran	3
compute_epanechnikov	4
compute_gaussian	4
convert_bbox	5
convolute_segmented_network	6
convolute_spatiotemporal_network	7
create_bbox	9
create_coordinates	10
create_graph	11
create_line_graph	11
create_linestring	12
create_osm_roads	13
create_points	14
create_polygon	15
decompose_linestring	15
durations	16
extract_road_endpoints	17
extract_road_intersections	17
extract_road_network_links	18
extract_road_network_nodes	18
extract_segmented_network_links	19
extract_segmented_network_nodes	20
fetch_roads	21
filter_and_cast_geometries	22
filter_points_within_tolerance	24
find_duration	25
generate_ids	26
get_connected_links	26
jdg2011_datum	27
osm_highway_values	28
plot.road_network	28
plot.spatiotemporal_network	29
plot_local_moran	30
remove_points_near_endpoints	31
road_network	32
run_consistency_test	33
sample_accidents	34
sample_accidents_multiyear	35
sample_points_along_linestring	36
sample_roads	36
segmented_network	37
set_events	38
spatiotemporal_events	39
spatiotemporal_network	40
split_linestrings	41
test_spatio_properties	42

<i>calculate_local_moran</i>	3
transform_coordinates	42
Index	44

calculate_local_moran	<i>Calculate spatio-temporal local Moran's I</i>
-----------------------	--

Description

Computes the Local Moran's I statistic for each spatiotemporal unit (spatiotemporal segment) using a memory-efficient, iterative approach. This function can handle both `segmented_network` and `spatiotemporal_network` objects.

Usage

```
calculate_local_moran(network_object, dist_threshold = 1, time_threshold = 2)
```

Arguments

network_object A `segmented_network` or `spatiotemporal_network` object with events assigned via `set_events()`.

dist_threshold The spatial distance threshold to define neighbors.

time_threshold The temporal distance threshold (in hours) to define neighbors.

Value

The input network object with a new component `$moran_results`. This is a data frame containing the Local Moran's I statistic (I), z-scores (z), spatially lagged z-scores (`lagged_z`), and cluster classification (`classification`) for each spatiotemporal segment.

Examples

```
# First, create a network and assign events
network_with_events <- sample_roads |>
  create_road_network() |>
  create_spatiotemporal_network(spatial_length = 0.5) |>
  set_events(sample_accidents)

# Then, calculate Local Moran's I
moran_result <- calculate_local_moran(
  network_with_events,
  dist_threshold = 1,
  time_threshold = 2
)

# View the results
head(moran_result$moran_results)

# Plot the results
plot_local_moran(moran_result, snapshot_time = 12)
plot_local_moran(moran_result, plot_3d = TRUE)
```

compute_epanechnikov *Compute the Epanechnikov function*

Description

The Epanechnikov function is one of the quadratic kernels used in kernel density estimation. It is defined as follows:

$$K(x) = \frac{3}{4}(1 - x^2) \quad \text{if } |x| \leq 1.$$

Usage

```
compute_epanechnikov(x)
```

Arguments

x A numeric vector.

Value

A numeric vector of weights of the kernel for each input points.

Examples

```
x <- seq(-3, 3, 0.1)
y <- compute_epanechnikov(x)
plot(x, y, xlim = c(-3, 3), ylim = c(0, 1), type = "l")
```

compute_gaussian *Compute the Gaussian function*

Description

The Gaussian function is a bell-shaped function defined as follows:

$$K(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right).$$

Usage

```
compute_gaussian(x, sigma = 1)
```

Arguments

x A numeric vector.
sigma The standard deviation of the Gaussian function.

Value

A numeric vector of weights of the kernel for each input points.

Examples

```
x <- seq(-3, 3, 0.1)
y <- compute_gaussian(x, sigma = 1)
plot(x, y, xlim = c(-3, 3), ylim = c(0, 1), type = "l")
```

convert_bbox

Convert a bounding box to coordinates or a polygon

Description

This function converts a bounding box to either coordinates or a polygon.

Usage

```
convert_bbox(bbox, output = c("coordinates", "polygon"), crs = NULL)
```

```
convert_bbox_to_coordinates(bbox)
```

```
convert_bbox_to_polygon(bbox, crs = NULL)
```

Arguments

bbox	A numeric matrix with two columns and two rows. bbox represents the minimum and maximum coordinates. The bounding box can be created using the create_bbox function.
output	A string specifying the output type. Either "coordinates" or "polygon".
crs	A string specifying the coordinate reference system. This is only used when converting to a polygon.

Value

A coordinates object or a sfc polygon object.

Examples

```
bbox <- create_bbox(center_lon = 136.9024,
                    center_lat = 35.1649,
                    width      = 0.10,
                    height     = 0.05)
convert_bbox_to_coordinates(bbox)
convert_bbox_to_polygon(bbox, crs = 4326)
```

convolute_segmented_network

Convolute segmented road network

Description

This function convolves a segmented network using a specified kernel function, typically for traffic modeling or network analysis. It computes weights densities based on the distance between links in the network and the number of events assigned to each link. Optionally, it can adjust for branching in the network.

Usage

```
convolute_segmented_network(
  segmented_network,
  kernel = compute_epanechnikov,
  bandwidth = 3,
  use_esd = TRUE,
  correct_boundary_effects = TRUE,
  ...
)
```

Arguments

segmented_network	A segmented_network object assigned with events.
kernel	A kernel function to use for convolution (default is Epanechnikov kernel).
bandwidth	Numeric value representing the bandwidth for the kernel function (default is 3).
use_esd	If TRUE, considers branching in the kernel using the Equal Split Discontinuous kernel (ESD). ESD follows the method described in Okabe et al., accounting for road intersections and ensuring that kernel weights are correctly distributed across accross branches (default is TRUE).
correct_boundary_effects	If TRUE, corrects for boundary effects by normalizing the kernel weights to account for kernel values outside the network (default is TRUE).
...	Additional arguments passed to the kernel function.

Value

The segmented network with updated link densities.

References

Okabe, A., Satoh, T., & Sugihara, K. (2009). A kernel density estimation method for networks, its computational method and a GIS-based tool. *International Journal of Geographical Information Science*, 23(1), 7-32. doi:[10.1080/13658810802475491](https://doi.org/10.1080/13658810802475491)

Examples

```
# Create a road network
road_network <- create_road_network(sample_roads)

# Assign sample accidents data
road_network <- set_events(road_network, sample_accidents)

# Segment the road network
segmented_network <- create_segmented_network(
  road_network,
  segment_length = 0.5
)

# Check the segmented road network after assigning events
segmented_network

# Apply the convolution to calculate link densities using
# the kernel function
convoluted_network <- convolute_segmented_network(segmented_network)

# Check the convoluted network with the computed densities
convoluted_network

# Plot the convoluted network showing the density distribution
plot(convoluted_network, mode = "density")
```

`convolute_spatiotemporal_network`*Convolute a spatiotemporal network to perform TNKDE*

Description

This function performs Temporal Network Kernel Density Estimation (TNKDE) on a spatiotemporal network object. It calculates density values across the network's segments for a series of time points.

Usage

```
convolute_spatiotemporal_network(
  segmented_network,
  kernel_space = compute_epanechnikov,
  kernel_time = compute_epanechnikov,
  bandwidth_space = 3,
  bandwidth_time = 2,
  time_points = 0:23,
  use_esd = TRUE,
  correct_boundary_effects = TRUE
)
```

Arguments

<code>segmented_network</code>	A spatiotemporal network object, typically the output of <code>create_spatiotemporal_network()</code> followed by <code>set_events()</code> .
<code>kernel_space</code>	The spatial kernel function (default: <code>compute_epanechnikov</code>).
<code>kernel_time</code>	The temporal kernel function (default: <code>compute_epanechnikov</code>).
<code>bandwidth_space</code>	The spatial bandwidth (in the network's distance units).
<code>bandwidth_time</code>	The temporal bandwidth (in hours).
<code>time_points</code>	A numeric vector of time points (e.g., hours 0:23) at which to estimate the density.
<code>use_esd</code>	If TRUE (default), uses the Equal Split Discontinuous (ESD) kernel for spatial smoothing to account for network intersections.
<code>correct_boundary_effects</code>	If TRUE (default), corrects for boundary effects by normalizing the kernel weights from each source to sum to 1.

Value

The input network object with the `$segments` component updated to an `sf` data frame of spatial segments, now containing new columns (`density_t_0`, `density_t_1`, etc.) with the calculated density values. The total density over all segments and time points is normalized to sum to 1.

Examples

```
## Not run:
# Create a network and assign events
tnkde_input <- sample_roads |>
  create_road_network() |>
  create_spatiotemporal_network(
    spatial_length = 0.5,
    temporal_length = "1 hour"
  ) |>
  set_events(sample_accidents, time_column = "time")

# Run the TNKDE calculation
tnkde_result <- convolute_spatiotemporal_network(
  tnkde_input,
  bandwidth_space = 2,
  bandwidth_time = 1.5,
  time_points = 0:23
)

# View the results
print(tnkde_result$segments)

# Plot the results
# Plot a 2D snapshot for a specific time
plot(tnkde_result, mode = "density", snapshot_time = 19)
```

```
# Plot the full 3D space-time cube
plot(tnkde_result, mode = "density", plot_3d = TRUE)

## End(Not run)
```

create_bbox	<i>Create a bounding box</i>
-------------	------------------------------

Description

This function creates a bounding box from either cardinal coordinates (north, south, east and west) or center coordinates and dimensions (center longitude and latitude, and size of width and height).

Usage

```
create_bbox(
  north = NULL,
  south = NULL,
  east = NULL,
  west = NULL,
  center_lon = NULL,
  center_lat = NULL,
  width = NULL,
  height = NULL
)
```

Arguments

north	The northernmost latitude.
south	The southernmost latitude.
east	The easternmost longitude.
west	The westernmost longitude.
center_lon	The center longitude.
center_lat	The center latitude.
width	The width of the bounding box.
height	The height of the bounding box.

Value

A 2x2 matrix representing the bounding box.

Examples

```
# Create a bounding box from cardinal coordinates
create_bbox(
  north = 35.1899,
  south = 35.1399,
  east  = 136.9524,
  west  = 136.8524
)

# Create a bounding box from center coordinates and dimensions
create_bbox(
  center_lon = 136.9024,
  center_lat = 35.1649,
  width      = 0.10,
  height     = 0.05
)
```

create_coordinates	<i>Create a collection of coordinates.</i>
--------------------	--

Description

This function creates a collection of coordinates from a sequence x and y coordinates.

Usage

```
create_coordinates(...)
```

Arguments

... A sequence of x and y coordinates.

Value

A coordinates object which is a matrix with x and y columns.

Examples

```
# Create a `coordinates` object from a sequence of x and y coordinates
create_coordinates(1, 2, 3, 4)

# Create a `coordinates` object from an with a vector of x and y
# coordinates
create_coordinates(c(1, 2, 3, 4))
```

create_graph	Create a graph from nodes and links
--------------	-------------------------------------

Description

This function creates a graph object from a set of nodes and links.

Usage

```
create_graph(nodes, links, directed = FALSE)
```

Arguments

nodes	A sf object representing nodes.
links	A sf object representing links between nodes.
directed	A logical indicating whether the graph is directed.

Value

A igraph object.

Examples

```
# Create nodes and links
nodes <- extract_road_network_nodes(sample_roads)
links <- extract_road_network_links(sample_roads, nodes)

# Create the graph
graph <- create_graph(nodes, links)
graph

# Plot the graph
plot(graph)
```

create_line_graph	Create a line graph from a network
-------------------	------------------------------------

Description

This function creates a line graph where nodes represent the midpoints of links in the input network.

Usage

```
create_line_graph(network)
```

Arguments

`network` A `road_network` or `segmented_network` object.

Value

A `igraph` object representing the midpoint graph.

Examples

```
# Create a road network
network <- create_road_network(sample_roads)

# Plot the road network
plot(network$graph)

# Create a midpoint graph from a road network
graph <- create_line_graph(network)

# Plot the midpoint graph
plot(graph)
```

`create_linestring` *Create a linestring geometry*

Description

This function creates a simple feature linestring object from a series of x, y coordinates.

Usage

```
create_linestring(...)

## S3 method for class 'numeric'
create_linestring(..., crs = NULL)

## S3 method for class 'coordinates'
create_linestring(coordinates, crs = NULL, ...)
```

Arguments

`...` A series of x, y coordinates.

`crs` The coordinate reference system.

`coordinates` A `coordinates` object.

Value

A simple feature linestring object.

Examples

```
# Create a linestring from individual coordinates
linestring_1 <- create_linestring(0, 1, 1, 0, 2, 1)
linestring_1
plot(linestring_1)

# Create a linestring from a numeric vector
linestring_2 <- create_linestring(
  c(0, 1, 1, 1, 1, 0, 0, 0, 0, 0.5, 1, 0.5)
)
linestring_2
plot(linestring_2)
```

create_osm_roads	<i>Create processed OpenStreetMap road data for a given area</i>
------------------	--

Description

This is a convenient wrapper function that fetches road data from OpenStreetMap for a specified bounding box, crops it, and transforms it to a Cartesian coordinate system for analysis.

Usage

```
create_osm_roads(north, south, east, west)
```

Arguments

north	The northern latitude of the bounding box (e.g., 35.1886).
south	The southern latitude of the bounding box (e.g., 35.1478).
east	The eastern longitude of the bounding box (e.g., 136.9449).
west	The western longitude of the bounding box (e.g., 136.8736).

Value

An sf object containing the processed road data, ready for use in functions like `create_road_network()`.

Examples

```
## Not run:
# Get processed road data for a specific area in Nagoya
nagoya_roads <- create_osm_roads(
  north = 35.1886,
  south = 35.1478,
  east  = 136.9449,
  west  = 136.8736
)
plot(nagoya_roads$geometry)

## End(Not run)
```

create_points	<i>Create points geometry</i>
---------------	-------------------------------

Description

This function creates a simple feature collection of points from a series of x, y coordinates.

Usage

```
create_points(...)

## S3 method for class 'numeric'
create_points(..., crs = NULL, as_multipoint = FALSE)

## S3 method for class 'coordinates'
create_points(coordinates, crs = NULL, as_multipoint = FALSE, ...)
```

Arguments

...	A series of x, y coordinates.
crs	The coordinate reference system.
as_multipoint	Logical; if TRUE, create a single MULTIPOINT geometry instead of multiple POINT geometries.
coordinates	A coordinates object.

Value

A simple feature geometry list column (sfc).

Examples

```
# Create multiple points from individual coordinates
points <- create_points(0, 0, 0, 2, 2, 2, 2, 1, 0, 1)
points
plot(points)

# Create points from a numeric vector
create_points(c(0, 0, 0, 2, 2, 2, 2, 1, 0, 1))

# Create a `MULTIPOINT` instead of separate `POINT`s
create_points(0, 0, 0, 2, 2, 2, 2, 1, 0, 1, as_multipoint = TRUE)
```

create_polygon	<i>Create a polygon geometry</i>
----------------	----------------------------------

Description

This function creates a polygon from a bounding box or a series of x, y coordinates.

Usage

```
create_polygon(...)

## S3 method for class 'numeric'
create_polygon(..., crs = NULL)

## S3 method for class 'coordinates'
create_polygon(coordinates, crs = NULL, ...)
```

Arguments

...	A series of x, y coordinates.
crs	The coordinate reference system.
coordinates	A coordinates object.

Value

A polygon object.

Examples

```
polygon <- create_polygon(
  136.9009, 35.16377,
  136.9159, 35.16377,
  136.9159, 35.17377,
  136.9009, 35.17377,
  crs = 4326
)
```

decompose_linestring	<i>Decompose a linestring into a list of line segments</i>
----------------------	--

Description

This function decomposes a linestring into a list of line segments, where each segment is a linestring connecting two consecutive points of the input linestring.

Usage

```
decompose_linestring(linestring)
```

Arguments

`linestring` A linestring object.

Value

A list of linestring objects, each representing a segment of the input linestring.

Examples

```
# Create a linestring object
linestring <- create_linestring(0, -1, 0, 1, 2, 1, 2, 0, 0, 0)
plot(linestring)

# Decompose the linestring into line segments
segments <- decompose_linestring(linestring)
plot(segments, col = c("#E69F00", "#56B4E9", "#009E73", "#F0E442"))
```

durations

Create durations

Description

This function creates a data frame of durations.

Usage

```
create_durations(duration_length, ...)
```

Arguments

`duration_length` The length of each duration.

`...` Additional arguments passed to or from other methods.

Value

A durations object.

Examples

```
# Create durations of 1 hour
create_durations("1 hour")

# Create durations of 30 minutes
create_durations("30 minutes")
```

`extract_road_endpoints`*Extract road endpoints*

Description

This function extracts the endpoints of a set of roads. It identifies and counts the roads that intersect at each endpoint.

Usage

```
extract_road_endpoints(roads)
```

Arguments

`roads` A linestring object representing roads. It should be have a column named `road_id`.

Value

A points object representing road endpoints. Each endpoint has the following columns:

- `parent_road`: A list of road IDs that intersect at the endpoint.
- `num_overlaps`: The number of roads that intersect at the endpoint.

`extract_road_intersections`*Extract road intersections*

Description

This function identifies and extracts intersections between roads. It considers only intersections where multiple roads on the same layer.

Usage

```
extract_road_intersections(roads)
```

Arguments

`roads` A linestring object representing roads. It should be have a column named `layer` and a column named `road_id`.

Value

A data frame with the following columns:

- `parent_road`: A list of road IDs that intersect at the intersection.
- `num_overlaps`: The number of roads that intersect at the intersection.

Examples

```
# Extract road intersections
intersections <- extract_road_intersections(sample_roads)
intersections

# Plot the intersections
plot(sample_roads$geometry)
plot(intersections$geometry, pch = 16, col = "#E69F00", add = TRUE)
```

extract_road_network_links

Extract road network links

Description

This function extracts the road network links from a set of road geometries. It splits the road geometries at the nodes and creates a new sf object representing the road network links.

Usage

```
extract_road_network_links(roads, nodes)
```

Arguments

roads	A linestring object representing the roads.
nodes	A point object representing the road network nodes.

Value

A linestring object representing the road network links.

See Also

To create the nodes, use the [extract_road_network_nodes\(\)](#)

extract_road_network_nodes

Extract road network nodes

Description

This function extracts nodes from roads represented as linestring objects downloaded from OpenStreetMap. Nodes are defined as intersections between roads and endpoints of road segments.

Usage

```
extract_road_network_nodes(roads)
```

Arguments

roads A linestring object representing roads.

Value

A points object representing road network nodes.

extract_segmented_network_links

Extract segmented network links from a road network

Description

This function extracts the segmented network links from a road network based on a specified set of segmented network nodes. It splits the linestrings of the road network links at the points of the segmented network nodes to create the segmented network links.

Usage

```
extract_segmented_network_links(
  road_network,
  segmented_network_nodes,
  tolerance = 1e-05
)
```

Arguments

road_network A road_network object created with create_road_network()
 segmented_network_nodes A data frame representing the segmented network nodes.
 tolerance A numeric value representing the maximum distance allowed between a linestring of a road link and a point of a segmented network node. If the distance exceeds this value, the point will not be used to split the linestring.

Value

A data frame representing the segmented network links.

Examples

```
# Create a road network
road_network <- create_road_network(sample_roads)

# Extract segmented network nodes by length of 1
segmented_network_nodes <- extract_segmented_network_nodes(road_network, 1)

# Extract segmented network links
```

```
segmented_network_links <- extract_segmented_network_links(  
  road_network,  
  segmented_network_nodes  
)  
segmented_network_links  
  
# Plot the segmented network nodes and links  
plot(segmented_network_links$geometry, col = "#E69F00")  
plot(segmented_network_nodes$geometry, pch = 16, add = TRUE)
```

extract_segmented_network_nodes

Extract nodes of segmented network from a road network

Description

This function extracts nodes at regular intervals along each link in a road network

Usage

```
extract_segmented_network_nodes(road_network, segment_length)
```

Arguments

road_network A road_network object created with create_network().
segment_length The length of each segment to sample along the links.

Value

An sf object with the sampled points.

Examples

```
# Create a road network  
road_network <- create_road_network(sample_roads)  
  
# Extract nodes with a segment length of 1  
segmented_network_nodes <- extract_segmented_network_nodes(road_network, 1)  
segmented_network_nodes  
  
# Plot the segmented network nodes  
plot(road_network)  
plot(segmented_network_nodes$geometry, add = TRUE, pch = 16, col = "#E69F00")
```

fetch_roads

Fetch road data from OpenStreetMap

Description

fetch_roads() fetches road data from OpenStreetMap. It retrieves roads within a specified bounding box or within an area defined by a center point and a radius. Additionally, roads can be cropped to fit within the specified boundaries.

Usage

```
fetch_roads(x, ...)

## S3 method for class 'matrix'
fetch_roads(x, crop = FALSE, ...)

## S3 method for class 'sfc_POINT'
fetch_roads(x, radius = 15, crop = FALSE, circle_crop = FALSE, ...)

## S3 method for class 'POINT'
fetch_roads(x, radius = 15, crop = FALSE, circle_crop = FALSE, ...)

## S3 method for class 'numeric'
fetch_roads(x, y, radius = 15, crop = FALSE, circle_crop = FALSE, ...)

## S3 method for class 'character'
fetch_roads(x, crop = FALSE, circle_crop = FALSE, ...)
```

Arguments

x	An object defining the area or location of interest. This can be: • A matrix representing a bounding box. • An sfc_POINT or POINT object representing a central point. • Numeric values representing longitude (x) and latitude (y). • A character string containing a pre-defined Overpass query.
...	Additional arguments passed to the method.
crop	A logical value indicating whether to crop road data to the specified area (default: FALSE).
radius	Numeric value representing the search radius in meters (default: 15).
circle_crop	A logical value indicating whether to crop roads within a circular area instead of a bounding box (default: FALSE). This is only effective if crop = TRUE.
y	Numeric value representing latitude (required if x is numeric and represents longitude).

Value

An sf object of class LINESTRING, containing road data with the followings columns:

id Unique road identifier
highway Road classification (e.g. "residential", "primary")
name Road name
layer Layer information for roads at different elevations
oneway Logical indicating whether the road is one-way
osm_id OpenStreetMap unique identifier

Examples

```
## Not run:
# Define a bounding box
bbox <- create_bbox(north = 35.17377,
                    south = 35.16377,
                    east  = 136.91590,
                    west  = 136.90090)

# Download road data with in the bounding box
roads <- fetch_roads(bbox)

# Plot the roads
plot(roads$geometry)

# Download and crop road data strictly to the bounding box
roads_cropped <- fetch_roads(bbox, crop = TRUE)
plot(roads_cropped$geometry)

# Download roads using a center point and radius
center_lon <- 136.8817
center_lat <- 35.1709
radius_m   <- 500

roads_radius <- fetch_roads(x      = center_lon,
                           y      = center_lat,
                           radius = radius_m)
plot(roads_radius$geometry)

## End(Not run)
```

filter_and_cast_geometries

Filter and cast geometries

Description

This function filters a given geometries by specific type and casts it to the specified type.

Usage

```
filter_and_cast_geometries(
  geometries,
  geometry_type = c("POINT", "LINESTRING", "POLYGON")
)

get_points(geometries)

get_linestrings(geometries)

get_polygons(geometries)
```

Arguments

geometries A sf object representing geometries.

geometry_type The type of geometry to filter and cast to. Must be one of "POINT", "LINESTRING", or "POLYGON".

Value

A sf object containing only the specified type of geometry.

Examples

```
plot(1, type = "n", xlab = "", ylab = "", axes = FALSE, xlim = c(0, 12), ylim = c(0, 10))
text(0, c(2, 6), c("Single", "Multiple"), srt = 90, cex = 1)
text(c(2, 6, 10), 8, c("Point", "Linestring", "Polygon"), cex = 1)

geometries <- sf::st_sf(geometry = c(
  # POINT
  create_points(2, 2),
  # LINESTRING
  create_linestring(5, 1, 7, 3),
  # POLYGON
  create_polygon(9, 1, 9, 3, 11, 3, 11, 1),
  # MULTIPOINT
  sf::st_union(create_points(1, 7, 2, 6, 3, 5)),
  # MULTILINESTRING
  sf::st_union(
    create_linestring(5, 6, 6, 7),
    create_linestring(6, 5, 7, 6)
  ),
  # MULTIPOLYGON
  sf::st_union(
    create_polygon(9, 7, 10, 7, 10, 6, 9, 6),
    create_polygon(10, 5, 11, 6, 11, 5)
  )
))
plot(geometries, pch = 16, lwd = 2, add = TRUE)
```

```
# Get points from geometries
plot(get_points(geometries), pch = 16)

# Get linestrings from geometries
plot(get_linestrings(geometries), lwd = 2)

# Get polygons from geometries
plot(get_polygons(geometries), lwd = 2)
```

```
filter_points_within_tolerance
```

Filter points within a tolerance distance

Description

This function filters points that are within a specified tolerance distance from a reference linestring.

Usage

```
filter_points_within_tolerance(points, linestring, tolerance = 0.01)
```

Arguments

<code>points</code>	A sfc object containing points to filter.
<code>linestring</code>	A linestring object.
<code>tolerance</code>	A numeric value representing the maximum allowable distance between points and the linestring. Points within this distance from the linestring will be included in the filtered set.

Value

A sfc object containing the filtered points.

Examples

```
# Create a points
points <- create_points(
  0.000, 1.000, 0.500, 0.600, 1.000, 0.010, 1.500, 0.501, 2.000, 0.990
)

# Create a linestring
linestring <- create_linestring(0, 1, 1, 0, 2, 1)

# Plot the points
plot(linestring, col = "gray")
plot(points, add = TRUE)

# Filter points within a tolerance distance (default: 0.01)
filtered_points <- filter_points_within_tolerance(points, linestring)
```

```
# Plot the filtered points
plot(linestring, col = "gray")
plot(filtered_points, add = TRUE)
```

find_duration	<i>Find the index of the duration that contains the event time</i>
---------------	--

Description

This function find the index of the duration that contains the event time.

Usage

```
find_duration(events, durations, ...)
```

Arguments

events	A spatiotemporal event object.
durations	A duration object.
...	Additional arguments passed to or from other methods.

Value

A vector of indices corresponding to the durations that contain the event times.

Examples

```
# Create a spatiotemporal event object
accidents <- create_spatiotemporal_events(sample_accidents)

# Create a duration object
durations <- create_durations("1 hour")

# Find the duration that contains each accident
find_duration(accidents, durations)
```

generate_ids	<i>Generate unique IDs based on parent IDs</i>
--------------	--

Description

This function generates unique IDs based on parent IDs. It first extracts the first ID from each parent ID, then ranks these IDs and uses the ranks to format the IDs.

Usage

```
generate_ids(parent_list, id_format)
```

Arguments

parent_list	A list of parent IDs.
id_format	A format string for the IDs.

Value

A vector of unique IDs.

Examples

```
parent_list <- c("rd_0001", "rd_0002", "rd_0003")
id_format <- "jn_%06x"
generate_ids(parent_list, id_format)
```

get_connected_links	<i>Get connected links from a specified node</i>
---------------------	--

Description

This function retrieves all links connected to a specified node in a road network.

Usage

```
get_connected_links(network, node_ids)
```

Arguments

network	A road_network or segmented_network object.
node_ids	A vector of node IDs to find connected links for.

Value

A vector of link IDs connected to the specified node.

Examples

```
# Create a road network
road_network <- create_road_network(sample_roads)
target_node <- road_network$nodes[3,]$id
target_node

# Get connected links
connected_links <- get_connected_links(road_network, target_node)
connected_links

# Plot the target node and the connected links
is_connected <- road_network$links$id %in% connected_links
connected_links_geom <- road_network$links$geometry[is_connected]
plot(road_network, col = "gray")
plot(connected_links_geom, add = TRUE, col = "#E69F00", lwd = 2)
plot(road_network$nodes[3,]$geometry, add = TRUE, pch = 19)
```

jdg2011_datum

Japan Geodetic Datum 2011 (JDG2011) Reference Points

Description

A dataset representing reference points for Japan Geodetic Datum 2011 (JDG2011). This dataset contains system numbers, longitude, latitude, and corresponding coordinate reference system (CRS) codes for various geodetic reference points in Japan.

Usage

```
jdg2011_datum
```

Format

A data.frame with 19 rows and 4 columns:

system_no System number (Roman numeral I-XIX) representing geodetic reference zones

lon Longitude in decimal degrees

lat Latitude in decimal degrees

crs Coordinate reference system (CRS) code for reference point

Examples

```
# Show the dataset
jdg2011_datum
```

osm_highway_values	<i>Values for the 'highway' tag in OpenStreetMap data</i>
--------------------	---

Description

osm_highway_values contains a character vector of values for the 'highway' tag in OpenStreetMap data.

Usage

```
osm_highway_values
```

Format

A character vector of length 12.

Examples

```
osm_highway_values
```

plot.road_network	<i>Plot a road network</i>
-------------------	----------------------------

Description

This function plots a road network.

Usage

```
## S3 method for class 'road_network'
plot(x, y, mode = c("default", "event", "graph"), ...)
```

Arguments

x	A road network object.
y	This argument is not used.
mode	The plotting mode. "event" mode shows the location events assigned to the road network. "graph" mode shows the direction of links when the network is directed. This utilizes the plot.igraph function.
...	Additional arguments passed to or from other methods.

Examples

```
# Create the road network
road_network <- create_road_network(sample_roads, events = sample_accidents)

# Plot the road network
plot(road_network)

# Plot the road network with events
plot(road_network, mode = "event")

# Plot the road network as a graph
plot(road_network, mode = "graph")
```

```
plot.spatiotemporal_network
```

Plot spatiotemporal network or TNKDE results

Description

Creates a plot to visualize the events, counts, or **TNKDE density** on a spatiotemporal network. The default is a 2D snapshot plot. The 3D view is optimized for visualizing TNKDE results across time.

Usage

```
## S3 method for class 'spatiotemporal_network'
plot(
  x,
  y,
  mode = c("event", "count", "density"),
  snapshot_time = 12,
  plot_3d = FALSE,
  time_range = 0:23,
  ...
)
```

Arguments

<code>x</code>	The spatiotemporal_network object (often the output of convolute_spatiotemporal_network).
<code>y</code>	(Not used).
<code>mode</code>	Character string specifying the type of visualization: "event" (individual points), "count" (event count per segment), or "density" (TNKDE density).
<code>snapshot_time</code>	Numeric. The hour (0-23) for which to create the static 2D snapshot plot.
<code>plot_3d</code>	Logical. If TRUE, an interactive 3D plot using rgl is created.
<code>time_range</code>	Numeric vector (e.g., 0:23) specifying the time points to include in the 3D plot.
<code>...</code>	Additional arguments passed to helper functions.

Value

NULL (invisibly). Opens an interactive rgl window for 3D plots, or draws a base R plot for 2D.

Examples

```
# Run the TNKDE calculation
tnkde_result <- sample_roads |>
  create_road_network() |>
  create_spatiotemporal_network(
    spatial_length = 0.5,
    temporal_length = "1 hour"
  ) |>
  set_events(sample_accidents, time_column = "time") |>
  convolute_spatiotemporal_network(
    bandwidth_space = 2,
    bandwidth_time = 1.5,
    time_points = 0:23
  )

# Plotting examples:

# A. 2D Snapshot: smoothed density
# Use Case: Visualize the primary analysis result at 19:00.
plot(tnkde_result, mode = "density", snapshot_time = 19)

# B. 2D Snapshot: raw event counts per segment
# Use Case: See raw event counts at a specific hour (e.g., 7:00).
plot(tnkde_result, mode = "count", snapshot_time = 7)

# C. 3D Plot: full density cube
# This is the primary visualization for spatiotemporal analysis.
# plot(tnkde_result, mode = "density", plot_3d = TRUE)
```

plot_local_moran

Plot spatio-temporal local Moran's I results

Description

Creates a plot to visualize the results of `calculate_local_moran`. The default is a 2D snapshot plot. If `plot_3d` is TRUE, it creates an interactive 3D plot showing significant clusters in a space-time cube.

Usage

```
plot_local_moran(moran_results_object, plot_3d = FALSE, snapshot_time = 12)
```

Arguments

- `moran_results_object` The output object from `calculate_local_moran`.
- `plot_3d` Logical. If TRUE, an interactive 3D plot is created. If FALSE (default), a static 2D snapshot plot is created.
- `snapshot_time` Numeric. The hour (0-23) for which to create the 2D plot.

Value

A plotly object for the 3D plot, or NULL for the 2D plot.

Examples

```
## Not run:
# First, run the Local Moran's I calculation
moran_result <- sample_roads |>
  create_road_network() |>
  create_spatiotemporal_network(spatial_length = 0.5) |>
  set_events(sample_accidents) |>
  calculate_local_moran(dist_threshold = 1, time_threshold = 2)

# --- Plotting Examples ---

# 1. Plot a 2D snapshot for a specific time (e.g., 19:00)
plot_local_moran(moran_result, snapshot_time = 19)

# 2. Plot the full 3D space-time cube
plot_local_moran(moran_result, plot_3d = TRUE)

## End(Not run)
```

`remove_points_near_endpoints`

Remove points near endpoints of a linestring

Description

This function removes points from a set of points that are within a specified tolerance distance of the start or end point of a linestring.

Usage

```
remove_points_near_endpoints(points, linestring, tolerance = 0.01)
```

Arguments

points	A sfc object containing points to filter.
linestring	A linestring object.
tolerance	A numeric value representing the maximum distance from the line's ends to consider points for filtering. Points within this distance will be excluded.

Value

A sfc object containing the filtered points.

Examples

```
# Create a points
points <- create_points(
  0.000, 1.000, 0.500, 0.600, 1.000, 0.010, 1.500, 0.501, 2.000, 0.990
)

# Create a linestring
linestring <- create_linestring(0, 1, 1, 0, 2, 1)

# Plot the points
plot(linestring, col = "gray")
plot(points, add = TRUE)

# Filter points within a tolerance distance (default: 0.01)
filtered_points <- remove_points_near_endpoints(points, linestring)

# Plot the filtered points
plot(linestring, col = "gray")
plot(filtered_points, add = TRUE)
```

road_network

Create a road network from roads

Description

This function constructs a road network from a set of roads. The road network is represented nodes and links between nodes. Nodes are defined as intersections between roads and endpoints of road segments.

Usage

```
create_road_network(roads, directed = FALSE, events = NULL, ...)
```

Arguments

roads	A linestring object representing roads.
directed	Logical indicating whether the road network is directed.
events	A sf object representing events.
...	Additional arguments passed to or from other methods.

Value

A road network object.

Examples

```
# Create the road network
road_network <- create_road_network(sample_roads)

# Print the road network summary
road_network

# Plot the road network
plot(road_network)
```

run_consistency_test *Run the full spatio-temporal consistency test*

Description

run_consistency_test() evaluates the temporal consistency of Local Moran's I cluster classifications across multiple years. The input accident data can be either:

- A single data.frame containing a year column with multiple years of data.
- A list of sf data frames, each representing a separate year.

The function calculates Local Moran's I for each year and then tests the consistency of classifications over time.

Usage

```
run_consistency_test(
  network_object,
  accident_data,
  dist_threshold = 1,
  time_threshold = 2,
  p_value_threshold = 0.05,
  time_column = "time",
  year_column = "year"
)
```

Arguments

- network_object** A base network object, either `segmented_network` or `spatiotemporal_network`.
- accident_data** Either:
- A `data.frame` containing accident records for multiple years (must include a year column).
 - A list of `sf` data frames, one per year.
- dist_threshold** The spatial distance threshold for `calculate_local_moran`.
- time_threshold** The temporal distance threshold for `calculate_local_moran`.
- p_value_threshold**
The p-value cutoff for the final consistency test.
- time_column** The name of the column in the accident data that contains the time information (e.g., "occurrence_hour"). Defaults to "time".
- year_column** The name of the column indicating the year in `accident_data` (if `data.frame`). Defaults to "year".

Value

A data frame with the final consistency test results.

sample_accidents	<i>Sample accidents data</i>
------------------	------------------------------

Description

A sample dataset containing information about 10 accidents. The dataset is for demonstration and testing and does not represent real-world events.

Usage

```
sample_accidents
```

Format

A `sf` object with 10 rows and 4 columns:

id Accident ID

time Time of the accident (in hours)

weather Weather condition at the time of the accident

severity Severity of the accident

geometry Point geometry of the accident location

Examples

```
# Show information about the accidents
sample_accidents

# Plot the locations of the accidents
plot(sample_roads$geometry)
plot(sample_accidents$geometry, pch = 4, col = "red", add = TRUE)
```

```
sample_accidents_multiyear
```

Sample accident data for five years

Description

Synthetic traffic accident data over five years.

Usage

```
sample_accidents_multiyear
```

Format

An sf object with 50 rows and 6 variables:

id Unique accident identifier (character).

time Time of the accident (integer, hour of day).

weather Weather condition (factor: "Sunny", "Cloudy", "Rainy", "Foggy", "Snowy").

severity Severity of accident (factor: "Minor", "Serious", "Fatal").

geometry Spatial point location (sfc_POINT).

year Year of occurrence (integer, 1–5).

Examples

```
sample_accidents_multiyear
```

sample_points_along_linestring
Sample points along a linestring

Description

This function samples points at regular intervals along a linestring object.

Usage

```
sample_points_along_linestring(linestrings, segment_length)
```

Arguments

linestrings A sfc object containing linestrings.
segment_length The desired length of each segment between sampled points.

Value

A sfc object containing the sampled points, grouped as MULTIPOINTs for each input linestring.

Examples

```
library(sf)

# Create a linestrings
linestrings <- c(
  create_linestring(0, 1, 2, 1),
  create_linestring(1, 1.3, 1, 0, 2, 0.5)
)

# Sample points along the linestrings
sampled_points <- sample_points_along_linestring(linestrings, 0.5)

# Plot the sampled points
plot(linestrings)
plot(sampled_points, add = TRUE, pch = 16, col = c("#E69F00", "#56B4E9"))
```

sample_roads *Sample roads data*

Description

A sample dataset representing simple roads. It intended for demonstration and testing and does not represent real-world roads.

Usage

```
sample_roads
```

Format

A sf linestring object with 6 rows and 2 columns:

id Road ID

layer Layer of the road (indicates a road on an elevated structure such as a bridge, tunnel, or highway)

geometry Linestring geometry of the road

Examples

```
# Show information about the roads
sample_roads

# Shape of the roads
plot(sample_roads$geometry)
```

segmented_network	<i>Create a segmented road network</i>
-------------------	--

Description

This function creates a segmented road network by dividing each link of the input road network into segments of a specified length.

Usage

```
create_segmented_network(road_network, segment_length = 1, events = NULL, ...)

## S3 method for class 'road_network'
create_segmented_network(road_network, segment_length = 1, events = NULL, ...)
```

Arguments

road_network A road_network object representing the input road network.

segment_length A numeric value specifying the length of each segment.

events A sf object representing events.

... Additional arguments passed to or from other methods.

Value

A segmented_network object.

Examples

```
# Create a road network
road_network <- create_road_network(sample_roads)

# Create a segmented road network
segmented_network <- create_segmented_network(
  road_network,
  segment_length = 0.5
)
segmented_network

# Plot the segmented road network
plot(segmented_network)
```

set_events	<i>Set events on a space.</i>
------------	-------------------------------

Description

This function sets events on a space.

Usage

```
set_events(x, events, ...)
```

Arguments

x	A space object.
events	A sf or spatiotemporal events object.
...	Additional arguments passed to or from other methods.

Value

The input object with the events added.

Examples

```
# Create the road network
road_network <- create_road_network(sample_roads)

# Set accidents on the road network
road_network <- set_events(road_network, sample_accidents)
```

spatiotemporal_events *Create a spatiotemporal event collection*

Description

This function creates a spatiotemporal event collection object from a given `sf` object, allowing spatial and temporal data to be handled together.

Usage

```
create_spatiotemporal_events(x, ...)

## S3 method for class 'sf'
create_spatiotemporal_events(
  x,
  time_column_name = "time",
  time_format = "%H",
  ...
)
```

Arguments

<code>x</code>	A <code>sf</code> object representing the spatial data.
<code>...</code>	Additional arguments passed to or from other methods.
<code>time_column_name</code>	A character string specifying the column name in <code>x</code> that contains time-related data. Defaults to <code>"time"</code>
<code>time_format</code>	A character string specifying the format of the time data in the <code>time_column_name</code> . For example, you can use formats like <code>"%Y-%m-%d"</code> for dates or <code>"%H:%M:%S"</code> for time. Defaults to <code>"%H"</code> .

Value

An `sf` object with class `spatiotemporal_events` added, representing a spatiotemporal event collection.

Examples

```
# Simple feature collection object representing accidents
sample_accidents

# Create a spatiotemporal event collection
create_spatiotemporal_events(sample_accidents)
```

`spatiotemporal_network`*Create spatiotemporal network*

Description

This function creates a spatiotemporal network from a road network.

Usage

```
create_spatiotemporal_network(  
  road_network,  
  spatial_length = 1,  
  temporal_length = "1 hour",  
  events = NULL,  
  ...  
)
```

Arguments

<code>road_network</code>	A <code>road_network</code> object.
<code>spatial_length</code>	The spatial length of segment.
<code>temporal_length</code>	The temporal length of segment.
<code>events</code>	A event collection object.
<code>...</code>	Additional arguments passed to or from other methods.

Value

A `spatiotemporal_network` object.

Examples

```
# Create a road network  
road_network <- create_road_network(sample_roads)  
  
# Create a spatiotemporal road network  
spatiotemporal_network <- create_spatiotemporal_network(  
  road_network,  
  spatial_length = 0.5,  
  temporal_length = "4 hour"  
)  
spatiotemporal_network  
  
# Plot the spatiotemporal road network  
plot(spatiotemporal_network)
```

split_linestrings	<i>Split multiple linestrings into segments</i>
-------------------	---

Description

Split multiple linestrings into segments

Usage

```
split_linestrings(linestrings, split_points, tolerance = 0.01)
```

Arguments

linestrings	An sfc object containing LINESTRING geometries, or a single linestring geometry.
split_points	An sfc object containing MULTIPOINT geometries, (one per linestring), or a set of POINT/MULTIPOINT geometries for a single LINESTRING.
tolerance	A numeric value representing the maximum distance allowed between a linestring and its split points. Points beyond this tolerance will be ignored.

Value

If the input is a single LINESTRING, returns an sfc of LINESTRINGs. If the input is multiple LINESTRINGs, returns a list of sfc objects, one for each input LINESTRING.

Examples

```
# Single `LINESTRING`
line <- create_linestring(0, 0, 0, 2, 2, 2, 2, 1, 0, 1)
points <- create_points(0, 1, 1, 2, 2, 1, as_multipoint = TRUE)
plot(line)
plot(points, add = TRUE)

segments <- split_linestrings(line, points)[[1]]
plot(segments, col = c("#E69F00", "#56B4E9", "#009E73", "#F0E442"), lwd = 2)

# Multiple `LINESTRING`s
lines <- c(create_linestring(0.0, 0.0, 0.0, 2.0, 2.0, 2.0, 2.0, 1.0, 0.0,
  1.0),
  create_linestring(2.5, 1.0, 3.5, 1.0, 3.5, 0.0, 2.5, 0.0, 2.5,
  0.5, 3.5, 0.5))
points <- c(create_points(0.0, 1.0, 1.0, 2.0, 2.0, 1.0,
  as_multipoint = TRUE),
  create_points(2.5, 0.5,
  as_multipoint = TRUE))
plot(lines)
plot(points, add = TRUE)
```

```
segments <- do.call(c, split_linestrings(lines, points))
plot(segments,
      col = c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2",
              "#D55E00"),
      lwd = 2)
```

test_spatio_properties

Test for temporal consistency of spatio-temporal properties

Description

This function evaluates the temporal consistency of Local Moran's I cluster classifications for each spatio-temporal unit (spatiotemporal segment) across multiple years.

Usage

```
test_spatio_properties(list_of_moran_results, p_value_threshold = 0.05)
```

Arguments

list_of_moran_results

A list where each element is the output of calculate_local_moran for a specific year.

p_value_threshold

The p-value cutoff to determine consistency (default: 0.05).

Value

A data frame summarizing the consistency test for each spatiotemporal segment.

transform_coordinates *Transform to Cartesian or geographic coordinates*

Description

This function transforms a spatial object to Cartesian coordinates (plane rectangular coordinate system) or geographic coordinates (WGS84; EPSG:4326). The Cartesian system currently supports only Aichi Prefecture's JDG2011 system (EPSG:6675).

Usage

```
transform_coordinates(  
  spatial_object,  
  target = c("cartesian", "geographic"),  
  quiet = FALSE  
)  
  
transform_to_cartesian(spatial_object, quiet = FALSE)  
  
transform_to_geographic(spatial_object, quiet = FALSE)
```

Arguments

spatial_object	A spatial object.
target	A character string specifying the target coordinate system: "cartesian" (EPSG:6675) or "geographic" (EPSG:4326).
quiet	Logical. If TRUE, suppresses the warning when CRS is missing.

Details

If the CRS is missing (NA), a warning is issued, and the original object is returned. The warning can be suppressed with the `quiet` argument.

Value

A spatial object transformed to the specified coordinate system.

Examples

```
# Create points  
points <- create_points(136.9024, 35.1649, crs = 4326)  
points  
  
# Transform to Cartesian coordinates  
transformed <- transform_to_cartesian(points)  
transformed  
  
# Transform to geographic coordinates  
transformed <- transform_to_geographic(points)  
transformed
```

Index

* datasets

- jdg2011_datum, [27](#)
- osm_highway_values, [28](#)
- sample_accidents, [34](#)
- sample_accidents_multiyear, [35](#)
- sample_roads, [36](#)

- calculate_local_moran, [3](#)
- compute_epanechnikov, [4](#)
- compute_gaussian, [4](#)
- convert_bbox, [5](#)
- convert_bbox_to_coordinates
(convert_bbox), [5](#)
- convert_bbox_to_polygon (convert_bbox),
[5](#)
- convolute_segmented_network, [6](#)
- convolute_spatiotemporal_network, [7](#)
- create_bbox, [9](#)
- create_coordinates, [10](#)
- create_durations (durations), [16](#)
- create_graph, [11](#)
- create_line_graph, [11](#)
- create_linestring, [12](#)
- create_osm_roads, [13](#)
- create_points, [14](#)
- create_polygon, [15](#)
- create_road_network (road_network), [32](#)
- create_segmented_network
(segmented_network), [37](#)
- create_spatiotemporal_events
(spatiotemporal_events), [39](#)
- create_spatiotemporal_network
(spatiotemporal_network), [40](#)

- decompose_linestring, [15](#)
- durations, [16](#)

- extract_road_endpoints, [17](#)
- extract_road_intersections, [17](#)
- extract_road_network_links, [18](#)
- extract_road_network_nodes, [18](#)
- extract_road_network_nodes(), [18](#)
- extract_segmented_network_links, [19](#)
- extract_segmented_network_nodes, [20](#)

- fetch_roads, [21](#)
- filter_and_cast_geometries, [22](#)
- filter_points_within_tolerance, [24](#)
- find_duration, [25](#)

- generate_ids, [26](#)
- get_connected_links, [26](#)
- get_linestrings
(filter_and_cast_geometries),
[22](#)
- get_points
(filter_and_cast_geometries),
[22](#)
- get_polygons
(filter_and_cast_geometries),
[22](#)

- jdg2011_datum, [27](#)

- osm_highway_values, [28](#)

- plot.road_network, [28](#)
- plot.spatiotemporal_network, [29](#)
- plot_local_moran, [30](#)

- remove_points_near_endpoints, [31](#)
- road_network, [32](#)
- run_consistency_test, [33](#)

- sample_accidents, [34](#)
- sample_accidents_multiyear, [35](#)
- sample_points_along_linestring, [36](#)
- sample_roads, [36](#)
- segmented_network, [37](#)
- set_events, [38](#)
- spatiotemporal_events, [39](#)

spatiotemporal_network, [40](#)
split_linestrings, [41](#)

test_spatio_properties, [42](#)
transform_coordinates, [42](#)
transform_to_cartesian
 (transform_coordinates), [42](#)
transform_to_geographic
 (transform_coordinates), [42](#)