Package 'osrm'

July 12, 2019

Type Package

Title Interface Between R and the OpenStreetMap-Based Routing Service OSRM

Version 3.3.1

Description An interface between R and the OSRM API. OSRM is a routing service based on OpenStreetMap data. See <http://project-osrm.org/> for more information. This package allows to compute distances (travel time and kilometric distance) between points and travel time matrices.

License GPL-3

LazyData TRUE

Imports jsonlite, RCurl, utils, stats, lwgeom, isoband, methods, gepaf, sp, sf

Depends R (>= 3.3.0)

Suggests cartography

URL https://github.com/rCarto/osrm

BugReports https://github.com/rCarto/osrm/issues

Encoding UTF-8

RoxygenNote 6.1.1

NeedsCompilation no

Author Timothée Giraud [cre, aut] (<https://orcid.org/0000-0002-1932-3323>), Robin Cura [ctb], Matthieu Viry [ctb]

Maintainer Timothée Giraud <timothee.giraud@cnrs.fr>

Repository CRAN

Date/Publication 2019-07-12 09:02:45 UTC

R topics documented:

`apotheke.df` *Coordinates of 100 Random Pharmacies in Berlin*

Description

A data.frame of coordinates of 100 random pharmacies in Berlin. The projection is WGS 84.

Source

© OpenStreetMap contributors - https://www.openstreetmap.org/copyright/en.

`apotheke.sf` *sf POINT of 100 Random Pharmacies in Berlin*

Description

100 random pharmacies in Berlin. The projection is WGS 84 / UTM zone 34N.

Source

© OpenStreetMap contributors - https://www.openstreetmap.org/copyright/en.

`apotheke.sp` *SpatialPointsDataFrame of 100 Random Pharmacies in Berlin*

Description

100 random pharmacies in Berlin. The projection is WGS 84 / UTM zone 34N.

Source

© OpenStreetMap contributors - https://www.openstreetmap.org/copyright/en.

com *Communes Coordinates*

Description

Coordinates of a set of communes in France. Coordinates are in WGS84.

Source

UMS RIATE

Examples

```
## Not run:
# Load data
data("com")

### osrmTable ###
# Inputs are data frames
# Travel time matrix
distCom <- osrmTable(loc = com[1:50, c("comm_id","lon","lat")])
# First 5 rows and columns
distCom$durations[1:5,1:5]

# Travel time matrix with different sets of origins and destinations
distCom2 <- osrmTable(src = com[1:10,c("comm_id","lon","lat")],
                      dst = com[11:20,c("comm_id","lon","lat")])
# First 5 rows and columns
distCom2$durations[1:5,1:5]

# Inputs are SpatialPointsDataFrames
distCom <- osrmTable(loc = src)
# First 5 rows and columns
distCom$durations[1:5,1:5]

# Travel time matrix with different sets of origins and destinations
distCom2 <- osrmTable(src = src, dst = dst)
# First 5 rows and columns
distCom2$durations[1:5,1:5]


### osrmRoute ###
# Travel path between points
route <- osrmRoute(src = com[1, c("comm_id", "lon","lat")],
                   dst = com[15, c("comm_id", "lon","lat")])
# Display the path
plot(com[c(1,15),3:4], asp =1, col = "red", pch = 20, cex = 1.5)
points(route[,1:2], type = "l", lty = 2)
text(com[c(1,15),3:4], labels = com[c(1,15),2], pos = 2)
```

```
# Travel path between points - output a SpatialLinesDataFrame
route2 <- osrmRoute(src=c("Bethune", 2.64781, 50.5199),
                    dst = c("Renescure", 2.369521, 50.72761),
                    sp = TRUE, overview = "full")

# Display the path
plot(com[c(1,4),3:4], asp =1, col = "red", pch = 20, cex = 1.5)
plot(route2, lty = 1,lwd = 4, add = TRUE)
plot(route2, lty = 1, lwd = 1, col = "white", add=TRUE)
text(com[c(1,4),3:4], labels = com[c(1,4),2], pos = 2)

# Input is SpatialPointsDataFrames
route3 <- osrmRoute(src = src[1,], dst = dst[1,], sp = TRUE)
route3@data



### osrmTrip ###
# Get a trip with a id lat lon data.frame
trips <- osrmTrip(loc = com[1:9, c(1,3,4)])

# Display the trip
plot(trips[[1]]$trip , col = 1:5)
points(com[1:10, 3:4], pch = 20, col = "red", cex = 0.5)

# Map
if(require("cartography")){
  osm <- getTiles(x = trips[[1]]$trip, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm)
  plot(trips[[1]]$trip, col = 1:5, add = TRUE)
  points(com[1:9, 3:4], pch = 20, col = "red", cex = 2)
}

# Get a trip with a SpatialPointsDataFrame
trips <- osrmTrip(loc = src)

# Map
if(require("cartography")){
  osm <- getTiles(x = trips[[1]]$trip, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm)
  plot(src, pch = 20, col = "red", cex = 2, add = TRUE)
  plot(trips[[1]]$trip, col = 1:5, add = TRUE, lwd=2)
}


### osrmIsochrone
# Get isochones with lon/lat coordinates, default breaks
iso <- osrmIsochrone(loc = c(6.026875, 48.93447))
plot(iso, col = paste0(rep("grey", nrow(iso)), c(seq(80,20,length.out = nrow(iso)))))

# Map
if(require("cartography")){
  osm <- getTiles(x = iso, crop = TRUE, type = "osmgrayscale")
```

```
  tilesLayer(x = osm)
  breaks <- sort(c(unique(iso$min), max(iso$max)))
  cartography::choroLayer(spdf = iso,
                          var = "center", breaks = breaks,
                          border = NA,
                          legend.pos = "topleft",legend.frame = TRUE,
                          legend.title.txt = "Isochrones\n(min)",
                          add = TRUE)
}

# Get isochones with a SpatialPointsDataFrame, custom breaks
iso2 <- osrmIsochrone(loc = src[1,], breaks = seq(from = 0, to = 40, by = 5))

# Map
if(require("cartography")){
  osm2 <- getTiles(x = iso2, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm2)
  breaks2 <- sort(c(unique(iso2$min), max(iso2$max)))
  cartography::choroLayer(spdf = iso2,
                          var = "center", breaks = breaks2,
                          border = NA,
                          legend.pos = "topleft",legend.frame = TRUE,
                          legend.title.txt = "Isochrones\n(min)",
                          add = TRUE)
}


## End(Not run)
```

---

dst *SpatialPointsDataFrame of 10 Communes in France*

Description

10 communes in France. The projection is RGF93 / Lambert-93.

Source

UMS RIATE

Examples

```
## Not run:
# Load data
data("com")

### osrmTable ###
# Inputs are data frames
# Travel time matrix
distCom <- osrmTable(loc = com[1:50, c("comm_id","lon","lat")])
```

```
# First 5 rows and columns
distCom$durations[1:5,1:5]

# Travel time matrix with different sets of origins and destinations
distCom2 <- osrmTable(src = com[1:10,c("comm_id","lon","lat")],
                      dst = com[11:20,c("comm_id","lon","lat")])
# First 5 rows and columns
distCom2$durations[1:5,1:5]

# Inputs are SpatialPointsDataFrames
distCom <- osrmTable(loc = src)
# First 5 rows and columns
distCom$durations[1:5,1:5]

# Travel time matrix with different sets of origins and destinations
distCom2 <- osrmTable(src = src, dst = dst)
# First 5 rows and columns
distCom2$durations[1:5,1:5]


### osrmRoute ###
# Travel path between points
route <- osrmRoute(src = com[1, c("comm_id", "lon","lat")],
                   dst = com[15, c("comm_id", "lon","lat")])
# Display the path
plot(com[c(1,15),3:4], asp =1, col = "red", pch = 20, cex = 1.5)
points(route[,1:2], type = "l", lty = 2)
text(com[c(1,15),3:4], labels = com[c(1,15),2], pos = 2)

# Travel path between points - output a SpatialLinesDataFrame
route2 <- osrmRoute(src=c("Bethune", 2.64781, 50.5199),
                    dst = c("Renescure", 2.369521, 50.72761),
                    sp = TRUE, overview = "full")

# Display the path
plot(com[c(1,4),3:4], asp =1, col = "red", pch = 20, cex = 1.5)
plot(route2, lty = 1,lwd = 4, add = TRUE)
plot(route2, lty = 1, lwd = 1, col = "white", add=TRUE)
text(com[c(1,4),3:4], labels = com[c(1,4),2], pos = 2)

# Input is SpatialPointsDataFrames
route3 <- osrmRoute(src = src[1,], dst = dst[1,], sp = TRUE)
route3@data



### osrmTrip ###
# Get a trip with a id lat lon data.frame
trips <- osrmTrip(loc = com[1:9, c(1,3,4)])

# Display the trip
plot(trips[[1]]$trip , col = 1:5)
points(com[1:10, 3:4], pch = 20, col = "red", cex = 0.5)
```

```
# Map
if(require("cartography")){
  osm <- getTiles(x = trips[[1]]$trip, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm)
  plot(trips[[1]]$trip, col = 1:5, add = TRUE)
  points(com[1:9, 3:4], pch = 20, col = "red", cex = 2)
}

# Get a trip with a SpatialPointsDataFrame
trips <- osrmTrip(loc = src)

# Map
if(require("cartography")){
  osm <- getTiles(x = trips[[1]]$trip, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm)
  plot(src, pch = 20, col = "red", cex = 2, add = TRUE)
  plot(trips[[1]]$trip, col = 1:5, add = TRUE, lwd=2)
}


### osrmIsochrone
# Get isochones with lon/lat coordinates, default breaks
iso <- osrmIsochrone(loc = c(6.026875, 48.93447))
plot(iso, col = paste0(rep("grey", nrow(iso)), c(seq(80,20,length.out = nrow(iso)))))

# Map
if(require("cartography")){
  osm <- getTiles(x = iso, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm)
  breaks <- sort(c(unique(iso$min), max(iso$max)))
  cartography::choroLayer(spdf = iso,
                          var = "center", breaks = breaks,
                          border = NA,
                          legend.pos = "topleft",legend.frame = TRUE,
                          legend.title.txt = "Isochrones\n(min)",
                          add = TRUE)
}

# Get isochones with a SpatialPointsDataFrame, custom breaks
iso2 <- osrmIsochrone(loc = src[1,], breaks = seq(from = 0, to = 40, by = 5))

# Map
if(require("cartography")){
  osm2 <- getTiles(x = iso2, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm2)
  breaks2 <- sort(c(unique(iso2$min), max(iso2$max)))
  cartography::choroLayer(spdf = iso2,
                          var = "center", breaks = breaks2,
                          border = NA,
                          legend.pos = "topleft",legend.frame = TRUE,
                          legend.title.txt = "Isochrones\n(min)",
                          add = TRUE)
```

```
}

## End(Not run)
```

`osrm` *Shortest Paths and Travel Time from OpenStreetMap via an OSRM API*

Description

An interface between R and the OSRM API.
OSRM is a routing service based on OpenStreetMap data. See <http://project-osrm.org/> for more information. This package allows to compute distances (travel time and kilometric distance) between points and travel time matrices.

- `osrmTable` Get travel time matrices between points.
- `osrmRoute` Get the shortest path between two points.
- `osrmTrip` Get the travel geometry between multiple unordered points.
- `osrmIsochrone` Get a SpatialPolygonsDataFrame of isochrones.

Note

This package relies on the usage of a running OSRM service (tested with version 5.22.0 of the OSRM API).
To change the OSRM server, change the `osrm.server` option:
`options(osrm.server = "http://address.of.the.server/")`.

To change the profile ("driving" is set by default and it is the only profile available on the demo server), use the `osrm.profile` option:
`options(osrm.profile = "name.of.the.profile")`

`osrmIsochrone` *Get Polygons of Isochrones*

Description

Based on `osrmTable`, this function buids polygons of isochrones.

Usage

```
osrmIsochrone(loc, breaks = seq(from = 0, to = 60, length.out = 7),
  exclude = NULL, res = 30, returnclass = "sp")
```

Arguments

loc	a numeric vector of longitude and latitude (WGS84), an sf object, a SpatialPointsDataFrame or a SpatialPolygonsDataFrame of the origine point.
breaks	a numeric vector of isochrone values (in minutes).
exclude	pass an optional "exclude" request option to the OSRM API.
res	number of points used to compute isochrones, one side of the square grid, the total number of points will be res*res.
returnclass	class of the returned polygons. Either "sp" of "sf".

Value

A SpatialPolygonsDateFrame or an sf MULTIPOLYGON of isochrones is returned. The data frame of the output contains four fields: id (id of each polygon), min and max (minimum and maximum breaks of the polygon), center (central values of classes).

See Also

osrmTable

Examples

```
## Not run:
# Load data
library(sf)
data("berlin")
# Get isochones with lon/lat coordinates
iso <- osrmIsochrone(loc = c(13.43,52.47), breaks = seq(0,14,2),
                     returnclass="sf")
plot(st_geometry(iso), col = c('grey80','grey60','grey50',
                               'grey40','grey30','grey20'))
# Map
if(require("cartography")){
  breaks <- sort(c(unique(iso$min), max(iso$max)))
  cartography::choroLayer(x = iso,
                          var = "center", breaks = breaks,
                          col = rev(carto.pal("green.pal",6)),
                          border = NA,
                          legend.pos = "topleft",legend.frame = TRUE,
                          legend.title.txt = "Isochrones\n(min)")
}

# Get isochones with an sf POINT
iso2 <- osrmIsochrone(loc = apotheke.sf[10,], returnclass="sf",
                      breaks = seq(from = 0, to = 16, by = 2))
# Map
if(require("cartography")){
  breaks2 <- sort(c(unique(iso2$min), max(iso2$max)))
  cartography::choroLayer(x = iso2, var = "center",
                          breaks = breaks2, border = NA,
                          legend.pos = "topleft",legend.frame = TRUE,
```

```
                          legend.title.txt = "Isochrones\n(min)")
}

## End(Not run)
```

`osrmRoute` *Get the Shortest Path Between Two Points*

Description

Build and send an OSRM API query to get the travel geometry between two points. This function interfaces the *route* OSRM service.

Usage

```
osrmRoute(src, dst, overview = "simplified", exclude = NULL, sp,
  returnclass)
```

Arguments

`src`	a numeric vector of identifier, longitude and latitude (WGS84), a SpatialPointsDataFrame, a SpatialPolygonsDataFrame or an sf object of the origine point.
`dst`	a numeric vector of identifier, longitude and latitude (WGS84), a SpatialPointsDataFrame, a SpatialPolygonsDataFrame or an sf object of the destination point.
`overview`	"full", "simplified" or FALSE. Add geometry either full (detailed), simplified according to highest zoom level it could be display on, or not at all.
`exclude`	pass an optional "exclude" request option to the OSRM API.
`sp`	deprecated, if sp is TRUE the function returns a SpatialLinesDataFrame.
`returnclass`	if returnclass="sf" an sf LINESTRING is returned. If returnclass="sp" a SpatialLineDataFrame is returned.

Value

If returnclass is not set, a data frame is returned. It contains the longitudes and latitudes of the travel path between the two points.
If returnclass is set to "sp", a SpatialLinesDataFrame is returned. If returnclass is set to "sf", an sf LINESTRING is returned. It contains 4 fields : identifiers of origine and destination, travel time in minutes and travel distance in kilometers.
If overview is FALSE, a named numeric vector is returned. It contains travel time (in minutes) and travel distance (in kilometers).

Examples

```
## Not run:
# Load data
data("berlin")
library(sf)
# Travel path between points
route1 <- osrmRoute(src = apotheke.sf[1, ], dst = apotheke.df[16, ],
                    returnclass="sf")
# Travel path between points excluding motorways
route2 <- osrmRoute(src = apotheke.sf[1, ], dst = apotheke.df[16, ],
                    returnclass="sf", exclude = "motorway")
# Display paths
plot(st_geometry(route1))
plot(st_geometry(route2), col = "red", add = TRUE)
plot(st_geometry(apotheke.sf[c(1,16),]), col = "red", pch = 20, add = TRUE)

# Return only duration and distance
route3 <- osrmRoute(src = apotheke.sf[1, ], dst = apotheke.df[16, ],
                    overview = FALSE)
route3

## End(Not run)
```

---

osrmTable *Get Travel Time Matrices Between Points*

Description

Build and send OSRM API queries to get travel time matrices between points. This function interfaces the *table* OSRM service.

Usage

```
osrmTable(loc, src = NULL, dst = NULL, exclude = NULL,
  gepaf = FALSE, measure = "duration")
```

Arguments

loc	a data frame containing 3 fields: points identifiers, longitudes and latitudes (WGS84). It can also be a SpatialPointsDataFrame, a SpatialPolygonsDataFrame or an sf object. If so, row names are used as identifiers. If loc parameter is used, all pair-wise distances are computed.
src	a data frame containing origin points identifiers, longitudes and latitudes (WGS84). It can also be a SpatialPointsDataFrame, a SpatialPolygonsDataFrame or an sf object. If so, row names are used as identifiers. If dst and src parameters are used, only pairs between scr/dst are computed.
dst	a data frame containing destination points identifiers, longitudes and latitudes (WGS84). It can also be a SpatialPointsDataFrame a SpatialPolygonsDataFrame or an sf object. If so, row names are used as identifiers.

exclude	pass an optional "exclude" request option to the OSRM API.
gepaf	a boolean indicating if coordinates are sent encoded with the google encoded algorithm format (TRUE) or not (FALSE). Must be FALSE if using the public OSRM API.
measure	a character indicating what measures are calculated. It can be "duration" (in minutes), "distance" (meters), or both c('duration', 'distance'). The demo server only allows "duration".

Details

If loc, src or dst are data frames we assume that the 3 first columns of the data frame are: identifiers, longitudes and latitudes.

Value

A list containing 3 data frames is returned. durations is the matrix of travel times (in minutes), sources and destinations are the coordinates of the origin and destination points actually used to compute the travel times (WGS84).

Note

If you want to get a large number of distances make sure to set the "max-table-size" argument (Max. locations supported in table) of the OSRM server accordingly.

See Also

osrmIsochrone

Examples

```
## Not run:
# Load data
data("berlin")

# Inputs are data frames
# Travel time matrix
distA <- osrmTable(loc = apotheke.df[1:50, c("id","lon","lat")])
# First 5 rows and columns
distA$durations[1:5,1:5]

# Travel time matrix with different sets of origins and destinations
distA2 <- osrmTable(src = apotheke.df[1:10,c("id","lon","lat")],
                    dst = apotheke.df[11:20,c("id","lon","lat")])
# First 5 rows and columns
distA2$durations[1:5,1:5]

# Inputs are sf points
distA3 <- osrmTable(loc = apotheke.sf[1:10,])
# First 5 rows and columns
distA3$durations[1:5,1:5]
```

```
# Travel time matrix with different sets of origins and destinations
distA4 <- osrmTable(src = apotheke.sf[1:10,], dst = apotheke.sf[11:20,])
# First 5 rows and columns
distA4$durations[1:5,1:5]

## End(Not run)
```

`osrmTrip` *Get the Travel Geometry Between Multiple Unordered Points*

Description

Build and send an OSRM API query to get the shortest travel geometry between multiple points. This function interfaces the *trip* OSRM service.

Usage

```
osrmTrip(loc, exclude = NULL, overview = "simplified",
  returnclass = "sp")
```

Arguments

`loc`	a SpatialPointsDataFrame or an sf object of the waypoints, or a data.frame with points as rows and 3 columns: identifier, longitudes and latitudes (WGS84 decimal degrees).
`exclude`	pass an optional "exclude" request option to the OSRM API.
`overview`	"full", "simplified". Add geometry either full (detailed) or simplified according to highest zoom level it could be display on.
`returnclass`	if returnclass="sf" an sf LINESTRING is returned. If returnclass="sp" a SpatialLineDataFrame is returned.

Details

As stated in the OSRM API, if input coordinates can not be joined by a single trip (e.g. the coordinates are on several disconnecte islands) multiple trips for each connected component are returned.

Value

A list of connected components. Each component contains:

trip A SpatialLinesDataFrame or sf LINESTRING (loc's CRS if there is one, WGS84 if not) containing a line for each step of the trip.

summary A list with 2 components: duration (in minutes) and distance (in kilometers).

See Also

osrmRoute

Examples

```
## Not run:
# Load data
data("berlin")
library(sf)
# Get a trip with a set of points (sf POINT)
trips <- osrmTrip(loc = apotheke.sf, returnclass = "sf")
mytrip <- trips[[1]]$trip
# Display the trip
plot(st_geometry(mytrip), col = "black", lwd = 4)
plot(st_geometry(mytrip), col = c("red", "white"), lwd = 1, add = TRUE)
plot(st_geometry(apotheke.sf), pch = 21, bg = "red", cex = 1, add = TRUE)

## End(Not run)
```

---

src *SpatialPointsDataFrame of 10 Communes in France*

Description

8 communes in France. The projection is RGF93 / Lambert-93.

Source

UMS RIATE

Examples

```
## Not run:
# Load data
data("com")

### osrmTable ###
# Inputs are data frames
# Travel time matrix
distCom <- osrmTable(loc = com[1:50, c("comm_id","lon","lat")])
# First 5 rows and columns
distCom$durations[1:5,1:5]

# Travel time matrix with different sets of origins and destinations
distCom2 <- osrmTable(src = com[1:10,c("comm_id","lon","lat")],
                      dst = com[11:20,c("comm_id","lon","lat")])
# First 5 rows and columns
distCom2$durations[1:5,1:5]

# Inputs are SpatialPointsDataFrames
distCom <- osrmTable(loc = src)
# First 5 rows and columns
distCom$durations[1:5,1:5]
```

```
# Travel time matrix with different sets of origins and destinations
distCom2 <- osrmTable(src = src, dst = dst)
# First 5 rows and columns
distCom2$durations[1:5,1:5]


### osrmRoute ###
# Travel path between points
route <- osrmRoute(src = com[1, c("comm_id", "lon","lat")],
                   dst = com[15, c("comm_id", "lon","lat")])
# Display the path
plot(com[c(1,15),3:4], asp =1, col = "red", pch = 20, cex = 1.5)
points(route[,1:2], type = "l", lty = 2)
text(com[c(1,15),3:4], labels = com[c(1,15),2], pos = 2)

# Travel path between points - output a SpatialLinesDataFrame
route2 <- osrmRoute(src=c("Bethune", 2.64781, 50.5199),
                    dst = c("Renescure", 2.369521, 50.72761),
                    sp = TRUE, overview = "full")

# Display the path
plot(com[c(1,4),3:4], asp =1, col = "red", pch = 20, cex = 1.5)
plot(route2, lty = 1,lwd = 4, add = TRUE)
plot(route2, lty = 1, lwd = 1, col = "white", add=TRUE)
text(com[c(1,4),3:4], labels = com[c(1,4),2], pos = 2)

# Input is SpatialPointsDataFrames
route3 <- osrmRoute(src = src[1,], dst = dst[1,], sp = TRUE)
route3@data



### osrmTrip ###
# Get a trip with a id lat lon data.frame
trips <- osrmTrip(loc = com[1:9, c(1,3,4)])

# Display the trip
plot(trips[[1]]$trip , col = 1:5)
points(com[1:10, 3:4], pch = 20, col = "red", cex = 0.5)

# Map
if(require("cartography")){
  osm <- getTiles(x = trips[[1]]$trip, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm)
  plot(trips[[1]]$trip, col = 1:5, add = TRUE)
  points(com[1:9, 3:4], pch = 20, col = "red", cex = 2)
}

# Get a trip with a SpatialPointsDataFrame
trips <- osrmTrip(loc = src)

# Map
```

```
if(require("cartography")){
  osm <- getTiles(x = trips[[1]]$trip, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm)
  plot(src, pch = 20, col = "red", cex = 2, add = TRUE)
  plot(trips[[1]]$trip, col = 1:5, add = TRUE, lwd=2)
}


### osrmIsochrone
# Get isochones with lon/lat coordinates, default breaks
iso <- osrmIsochrone(loc = c(6.026875, 48.93447))
plot(iso, col = paste0(rep("grey", nrow(iso)), c(seq(80,20,length.out = nrow(iso)))))

# Map
if(require("cartography")){
  osm <- getTiles(x = iso, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm)
  breaks <- sort(c(unique(iso$min), max(iso$max)))
  cartography::choroLayer(spdf = iso,
                          var = "center", breaks = breaks,
                          border = NA,
                          legend.pos = "topleft",legend.frame = TRUE,
                          legend.title.txt = "Isochrones\n(min)",
                          add = TRUE)
}

# Get isochones with a SpatialPointsDataFrame, custom breaks
iso2 <- osrmIsochrone(loc = src[1,], breaks = seq(from = 0, to = 40, by = 5))

# Map
if(require("cartography")){
  osm2 <- getTiles(x = iso2, crop = TRUE, type = "osmgrayscale")
  tilesLayer(x = osm2)
  breaks2 <- sort(c(unique(iso2$min), max(iso2$max)))
  cartography::choroLayer(spdf = iso2,
                          var = "center", breaks = breaks2,
                          border = NA,
                          legend.pos = "topleft",legend.frame = TRUE,
                          legend.title.txt = "Isochrones\n(min)",
                          add = TRUE)
}


## End(Not run)
```

Index