Package ‘traveltimeR’

March 15, 2023

Title Interface to ‘Travel Time’ API
Version 1.1.4
Description ‘Travel Time’ API <https://docs.traveltime.com/api/overview/introduction> helps users find locations by journey time rather than using ‘as the crow flies’ distance.
Time-based searching gives users more opportunities for personalisation and delivers a more relevant search.
Maintainer TravelTime <frontend@traveltime.com>
License MIT + file LICENSE
Encoding UTF-8
RoxygenNote 7.2.3
Imports data.table, httr, jsonlite, RProtoBuf
URL https://github.com/traveltime-dev/traveltime-sdk-r
BugReports https://github.com/traveltime-dev/traveltime-sdk-r/issues
Depends R (>= 2.10)
NeedsCompilation no
Author TravelTime [aut, cre] (https://github.com/traveltime-dev)
Repository CRAN
Date/Publication 2023-03-15 08:20:02 UTC

R topics documented:

geocoding .................................................. 2
geocoding_reverse ...................................... 3
make_location ........................................... 4
make_search ............................................. 4
make_union_intersect ................................. 5
map_info .............................................. 6
routes .................................................. 7
supported_locations ................................. 8
**Geocoding (Search)**

**Description**

Match a query string to geographic coordinates.

**Usage**

```r
geocoding(
  query,
  within.country = NA,
  format.name = NA,
  format.exclude.country = NA,
  bounds = NA
)
```

**Arguments**

- `query` A query to geocode. Can be an address, a postcode or a venue.
- `within.country` Only return the results that are within the specified country. If no results are found it will return the country itself. Optional. Format: ISO 3166-1 alpha-2 or alpha-3
- `format.name` Format the name field of the response to a well formatted, human-readable address of the location. Experimental. Optional.
- `format.exclude.country` Exclude the country from the formatted name field (used only if format.name is equal true). Optional.
- `bounds` Used to limit the results to a bounding box. Expecting a character vector with four floats, marking a south-east and north-west corners of a rectangle: min-latitude,min-longitude,max-latitude,max-longitude. e.g. bounds for Scandinavia c(54.16243,4.04297,71.18316,31.81641). Optional.

**Details**

See [https://docs.traveltime.com/api/reference/geocoding-search/](https://docs.traveltime.com/api/reference/geocoding-search/) for details
geocoding_reverse

Value

API response parsed as list and as a raw json

Examples

```r
## Not run:
geocoding("Parliament square")
## End(Not run)
```

---

geocoding_reverse Reverse Geocoding

Description

Attempt to match a latitude, longitude pair to an address.

Usage

```r
geocoding_reverse(lat, lng)
```

Arguments

- `lat` Latitude of the point to reverse geocode.
- `lng` Longitude of the point to reverse geocode.

Details

See [https://docs.traveltime.com/api/reference/geocoding-reverse/](https://docs.traveltime.com/api/reference/geocoding-reverse/) for details

Value

API response parsed as list and as a raw json

Examples

```r
## Not run:
geocoding_reverse(lat=51.507281, lng=-0.132120)
## End(Not run)
```
make_location  

**Location objects constructor**

**Description**

Define your locations to use later in departure_searches or arrival_searches.

**Usage**

```r
make_location(id, coords)
```

**Arguments**

- `id`: You will have to reference this id in your searches. It will also be used in the response body. MUST be unique among all locations.
- `coords`: Location coordinates. Must use this format: list(lat = 0, lng = 0)

**Details**

See [https://docs.traveltime.com/api/reference/distance-matrix](https://docs.traveltime.com/api/reference/distance-matrix) for details

**Value**

A data.frame wrapped in a list. It is constructed in a way that allows jsonlite::toJSON to correctly transform it into a valid request body

**See Also**

See `time_filter` for usage examples

make_search  

**Search objects constructor**

**Description**

Searches based on departure or arrival times. Departure: Leave departure location at no earlier than given time. You can define a maximum of 10 searches Arrival: Arrive at destination location at no later than given time. You can define a maximum of 10 searches
make_union_intersect

Usage

make_search(
  id,
  travel_time = NA,
  coords = NA,
  departure_time = NA,
  arrival_time = NA,
  transportation = list(type = "driving"),
  ...
)

Arguments

id Used to identify this specific search in the results array. MUST be unique among
     all searches.
travel_time Travel time in seconds. Maximum value is 14400 (4 hours)
cords The coordinates of the location we should start the search from. Must use this
       format: list(lat = 0, lng = 0)
departure_time Date in extended ISO-8601 format
arrival_time Date in extended ISO-8601 format
transportation Transportation mode and related parameters.
... Any additional parameters to pass. Some functions require extra parameters to
      work. Check their API documentation for details.

Value

A data.frame wrapped in a list. It is constructed in a way that allows jsonlite::toJSON to correctly
transform it into a valid request body

See Also

See time_map for usage examples

make_union_intersect  Set objects constructor

Description

Allows you to define unions or intersections of shapes that are results of previously defined searches. You can define a maximum of 10 unions/intersections

Usage

make_union_intersect(id, search_ids)
Arguments

- id: Used to identify this specific search in the results array. MUST be unique among all searches.
- search_ids: An unnamed list of search ids which results will formulate this union.

Details

See [https://docs.traveltime.com/api/reference/isochrones](https://docs.traveltime.com/api/reference/isochrones) for details

Value

A data.frame wrapped in a list. It is constructed in a way that allows jsonlite::toJSON to correctly transform it into a valid request body.

See Also

See [time_map](https://docs.traveltime.com/api/reference/time-map) for usage examples

---

### map_info

#### Map Info

- **Description**
  Returns information about currently supported countries.

- **Usage**
  ```r
  map_info()
  ```

- **Details**
  See [https://docs.traveltime.com/api/reference/map-info/](https://docs.traveltime.com/api/reference/map-info/) for details

- **Value**
  API response parsed as list and as a raw json

- **Examples**
  ```r
  ## Not run:
  map_info()
  ```
  ```r
  ## End(Not run)
  ```
Description

Returns routing information between source and destinations.

Usage

routes(locations, departure_searches = NULL, arrival_searches = NULL)

Arguments

locations One or more objects created by make_location
departure_searches
arrival_searches One or more objects created by make_search

Details

See https://docs.traveltime.com/api/reference/routes/ for details

Value

API response parsed as a list and as a raw json

Examples

```r
## Not run:
locations <- c(
  make_location(
    id = "London center",
    coords = list(lat = 51.508930, lng = -0.131387)),
  make_location(
    id = "Hyde Park",
    coords = list(lat = 51.508824, lng = -0.167093)),
  make_location(
    id = "ZSL London Zoo",
    coords = list(lat = 51.536067, lng = -0.153596))
)

departure_search <-
  make_search(id = "departure search example",
              departure_location_id = "London center",
              arrival_location_ids = list("Hyde Park", "ZSL London Zoo"),
              departure_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
              transportation = list(type = "driving"),
              properties = list("travel_time", "distance", "route"))
```
arrival_search <-
make_search(id = "arrival search example",
    arrival_location_id = "London center",
    departure_location_ids = list("Hyde Park", "ZSL London Zoo"),
    arrival_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
    transportation = list(type = "public_transport"),
    properties = list("travel_time", "distance", "route", "fares"),
    range = list(enabled = TRUE, width = 1800, max_results = 1))

result <-
routes(
    departure_searches = departure_search,
    arrival_searches = arrival_search,
    locations = locations
)

## End(Not run)

**supported_locations**

**Supported Locations**

### Description
Find out what points are supported by the api. The returned map name for a point can be used to
determine what features are supported. See also the `map_info`.

### Usage

```r
supported_locations(locations)
```

### Arguments

- **locations**
  One or more objects created by `make_location`

### Details
See [https://docs.traveltime.com/api/reference/supported-locations/](https://docs.traveltime.com/api/reference/supported-locations/) for details

### Value
API response parsed as list and as a raw json

### Examples

```r
## Not run:
locationsDF <- data.frame(
    id = c("Kaunas", "London", "Bangkok", "Lisbon"),
    lat = c(54.900008, 51.506756, 13.761866, 38.721869),
    lng = c(23.957734, -0.128050, 100.544818, -9.138549)
)
time_filter

```r
locations <- apply(locationsDF, 1, function(x)
  make_location(id = x["id"], coords = list(lat = as.numeric(x["lat"]),
    lng = as.numeric(x["lng"]))))
supported_locations(unlist(locations, recursive = FALSE))

## End(Not run)
```

---

time_filter  

**Distance Matrix (Time Filter)**

**Description**

Given origin and destination points filter out points that cannot be reached within specified time limit. Find out travel times, distances and costs between an origin and up to 2,000 destination points.

**Usage**

```r
time_filter(locations, departure_searches = NULL, arrival_searches = NULL)
```

**Arguments**

- `locations` One or more objects created by `make_location`
- `departure_searches` One or more objects created by `make_search`
- `arrival_searches` One or more objects created by `make_search`

**Details**


**Value**

API response parsed as a list and as a raw json

**Examples**

```r
## Not run:
locationsDF <- data.frame(
  id = c("London center", "Hyde Park", "ZSL London Zoo"),
  lat = c(51.508930, 51.508824, 51.536067),
  lng = c(-0.131387, -0.167093, -0.153596)
)
locations <- apply(locationsDF, 1, function(x)
  make_location(id = x["id"], coords = list(lat = as.numeric(x["lat"]),
    lng = as.numeric(x["lng"]))))
```
locations <- unlist(locations, recursive = FALSE)

departure_search <-
  make_search(id = "forward search example",
    departure_location_id = "London center",
    arrival_location_ids = list("Hyde Park", "ZSL London Zoo"),
    departure_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
    travel_time = 1800,
    transportation = list(type = "bus"),
    properties = list(‘travel_time’),
    range = list(enabled = TRUE, width = 600, max_results = 3))

arrival_search <-
  make_search(id = "backward search example",
    arrival_location_id = "London center",
    departure_location_ids = list("Hyde Park", "ZSL London Zoo"),
    arrival_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
    travel_time = 1800,
    transportation = list(type = "public_transport"),
    properties = list(‘travel_time’, "distance", "distance_breakdown", "fares"),
    range = list(enabled = TRUE, width = 600, max_results = 3))

result <-
  time_filter(  
    departure_searches = departure_search,  
    arrival_searches = arrival_search,  
    locations = locations  
  )

## End(Not run)

---

**time_filter_fast**

**Time Filter (Fast)**

**Description**

A very fast version of `time_filter`. However, the request parameters are much more limited. Currently only supports UK and Ireland.

**Usage**

```r

# Example

time_filter_fast(  
  locations,  
  arrival_many_to_one = NULL,  
  arrival_one_to_many = NULL
)
```
time_filter_fast

Arguments

locations One or more objects created by `make_location`
arrival_many_to_one One or more objects created by `make_search`
arrival_one_to_many One or more objects created by `make_search`

Details

See [https://docs.traveltime.com/api/reference/time-filter-fast/](https://docs.traveltime.com/api/reference/time-filter-fast/) for details

Value

API response parsed as a list and as a raw json

Examples

```r
## Not run:

locations <- c(
  make_location(
    id = 'London center',
    coords = list(lat = 51.508930, lng = -0.131387)),
  make_location(
    id = 'Hyde Park',
    coords = list(lat = 51.508824, lng = -0.167093)),
  make_location(
    id = 'ZSL London Zoo',
    coords = list(lat = 51.536067, lng = -0.153596))
)
arrival_many_to_one <- make_search(id = "arrive-at many-to-one search example",
  arrival_location_id = "London center",
  departure_location_ids = list("Hyde Park", "ZSL London Zoo"),
  travel_time = 1900,
  transportation = list(type = "public_transport"),
  properties = list("travel_time", "fares"),
  arrival_time_period = "weekday_morning")

arrival_one_to_many <- make_search(id = "arrive-at one-to-many search example",
  departure_location_id = "London center",
  arrival_location_ids = list("Hyde Park", "ZSL London Zoo"),
  travel_time = 1900,
  transportation = list(type = "public_transport"),
  properties = list("travel_time", "fares"),
  arrival_time_period = "weekday_morning")

result <- time_filter_fast(locations, arrival_many_to_one, arrival_one_to_many)

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

The Travel Time Matrix (Fast) endpoint is available with even higher performance through a version using Protocol Buffers. The endpoint takes as inputs a single origin location, multiple destination locations, a mode of transport, and a maximum travel time. The endpoint returns the travel times to each destination location, so long as it is within the maximum travel time.

**Usage**

```r
time_filter_fast_proto(
  departureLat,
  departureLng,
  country = c("uk", "ireland"),
  travelTime,
  destinationCoordinates,
  transportation = names(protoTransport),
  useDistance = FALSE
)
```

**Arguments**

- `departureLat`: origin latitude
- `departureLng`: origin longitude
- `country`: Origin country. Only UK and Ireland are supported.
- `travelTime`: Maximum journey time (in seconds).
- `destinationCoordinates`: data.frame with pairs of coordinates. Coordinates columns must be named 'lat' and 'lng'
- `transportation`: One of supported transportation methods: 'pt', 'driving+ferry', 'cycling+ferry', 'walking+ferry'.
- `useDistance`: return distance information

**Details**

See [https://docs.traveltime.com/api/start/travel-time-distance-matrix-proto](https://docs.traveltime.com/api/start/travel-time-distance-matrix-proto) for details

**Value**

API response parsed as a list and as a raw json
time_filter_postcodes Time Filter (Postcodes)

Description
Find reachable postcodes from origin (or to destination) and get statistics about such postcodes. Currently only supports United Kingdom.

Usage

time_filter_postcodes(departure_searches = NULL, arrival_searches = NULL)

Arguments

departure_searches
One or more objects created by make_search

arrival_searches
One or more objects created by make_search

Details
See https://docs.traveltime.com/api/reference/postcode-search/ for details

Value
API response parsed as a list and as a raw json

Examples

## Not run:

```r
time_filter_fast_proto(
  departureLat = 51.508930,
  departureLng = -0.131387,
  destinationCoordinates = data.frame(
    lat = c(51.508824, 51.536067),
    lng = c(-0.167093, -0.153596)
  ),
  transportation = 'driving+ferry',
  travelTime = 7200,
  country = "uk",
  useDistance = FALSE
)
```

## End(Not run)
Examples

```r
## Not run:
departure_search <-
    make_search(id = "public transport from Trafalgar Square",
               departure_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
               travel_time = 1800,
               coords = list(lat = 51.507609, lng = -0.128315),
               transportation = list(type = "public_transport"),
               properties = list("travel_time", "distance"))

arrival_search <-
    make_search(id = "public transport to Trafalgar Square",
               arrival_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
               travel_time = 1800,
               coords = list(lat = 51.507609, lng = -0.128315),
               transportation = list(type = "public_transport"),
               properties = list("travel_time", "distance"))

result <-
    time_filter_postcodes(  
        departure_searches = departure_search,
        arrival_searches = arrival_search
    )

## End(Not run)
```

---

**time_filter_postcode_districts**

*Time Filter (Postcode Districts)*

**Description**

Find districts that have a certain coverage from origin (or to destination) and get statistics about postcodes within such districts. Currently only supports United Kingdom.

**Usage**

```r
time_filter_postcode_districts(  
    departure_searches = NULL,
    arrival_searches = NULL
)
```

**Arguments**

- `departure_searches`
  
  One or more objects created by `make_search`

- `arrival_searches`
  
  One or more objects created by `make_search`
Details

See https://docs.traveltime.com/api/reference/postcode-district-filter/ for details

Value

API response parsed as a list and as a raw json

Examples

```r
## Not run:
dealure_search <-
  make_search(id = "public transport from Trafalgar Square",
              departure_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
              travel_time = 1800,
              coords = list(lat = 51.507609, lng = -0.128315),
              transportation = list(type = "public_transport"),
              reachable_postcodes_threshold = 0.1,
              properties = list("coverage", "travel_time_reachable", "travel_time_all"))

arrival_search <-
  make_search(id = "public transport to Trafalgar Square",
              arrival_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
              travel_time = 1800,
              coords = list(lat = 51.507609, lng = -0.128315),
              transportation = list(type = "public_transport"),
              reachable_postcodes_threshold = 0.1,
              properties = list("coverage", "travel_time_reachable", "travel_time_all"))

result <-
  time_filter_postcode_districts(
    departure_searches = departure_search,
    arrival_searches = arrival_search
  )
## End(Not run)
```
time_filter_postcode_sectors

Usage

time_filter_postcode_sectors(
    departure_searches = NULL,
    arrival_searches = NULL
)

Arguments

departure_searches
    One or more objects created by make_search

arrival_searches
    One or more objects created by make_search

Details

See https://docs.traveltime.com/api/reference/postcode-sector-filter/ for details

Value

API response parsed as a list and as a raw json

Examples

## Not run:
departure_search <-
    make_search(id = "public transport from Trafalgar Square",
        departure_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
        travel_time = 1800,
        coords = list(lat = 51.507609, lng = -0.128315),
        transportation = list(type = "public_transport"),
        reachable_postcodes_threshold = 0.1,
        properties = list("coverage", "travel_time_reachable", "travel_time_all"))

arrival_search <-
    make_search(id = "public transport to Trafalgar Square",
        arrival_time = strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ"),
        travel_time = 1800,
        coords = list(lat = 51.507609, lng = -0.128315),
        transportation = list(type = "public_transport"),
        reachable_postcodes_threshold = 0.1,
        properties = list("coverage", "travel_time_reachable", "travel_time_all"))

result <-
    time_filter_postcode_sectors(
        departure_searches = departure_search,
        arrival_searches = arrival_search
    )

## End(Not run)
**time_map**

---

### Isochrones (Time Map)

**Description**

Given origin coordinates, find shapes of zones reachable within corresponding travel time. Find unions/intersections between different searches

**Usage**

```r
time_map(
  departure_searches = NULL,
  arrival_searches = NULL,
  unions = NULL,
  intersections = NULL,
  format = NULL
)
```

**Arguments**

- **departure_searches**
  - One or more objects created by `make_search`
- **arrival_searches**
  - One or more objects created by `make_search`
- **unions**
  - One or more objects created by `make_union_intersect`
- **intersections**
  - One or more objects created by `make_union_intersect`
- **format**

**Details**

See [https://docs.traveltime.com/api/reference/isochrones/](https://docs.traveltime.com/api/reference/isochrones/) for details

**Value**

API response parsed as a list and as a raw json

**Examples**

```r
## Not run:

dateTime <- strftime(as.POSIXlt(Sys.time(), "UTC"), "%Y-%m-%dT%H:%M:%SZ")

departure_search1 <- make_search(id = "public transport from Trafalgar Square",
  departure_time = dateTime,
  travel_time = 900,
```

coords = list(lat = 51.507609, lng = -0.128315),
transportation = list(type = "public_transport"),
properties = list("is_only_walking"))

dept_search2 <-
make_search(id = "driving from Trafalgar Square",
departure_time = dateTime,
travel_time = 900,
coords = list(lat = 51.507609, lng = -0.128315),
transportation = list(type = "driving"))

arrival_search <-
make_search(id = "public transport to Trafalgar Square",
arrival_time = dateTime,
travel_time = 900,
coords = list(lat = 51.507609, lng = -0.128315),
transportation = list(type = "public_transport"),
range = list(enabled = TRUE, width = 3600))

union <- make_union_intersect(id = "union of driving and public transport",
search_ids = list("driving from Trafalgar Square",
'public transport from Trafalgar Square'))

intersection <- make_union_intersect(id = "intersection of driving and public transport",
search_ids = list("driving from Trafalgar Square",
'public transport from Trafalgar Square'))

result <-
time_map(
  departure_searches = c(departure_search1, departure_search2),
  arrival_searches = arrival_search,
  unions = union,
  intersections = intersection
)

## End(Not run)

---

**time_map_fast**

**Isochrones (Time Map) Fast**

**Description**

A very fast version of Isochrone API. However, the request parameters are much more limited.

**Usage**

time_map_fast(
  arrival_many_to_one = NULL,
  arrival_one_to_many = NULL,
  format = NULL
)
Arguments

arrival_many_to_one
   One or more objects created by `make_search`

arrival_one_to_many
   One or more objects created by `make_search`

format
time-map response format. See https://docs.traveltime.com/api/reference/iscochrones-fast#Response-Body for details.

Details

See https://docs.traveltime.com/api/reference/iscochrones-fast/ for details

Value

API response parsed as a list and as a raw json

Examples

## Not run:

```r
arrival_search <-
   make_search(id = "public transport to Trafalgar Square",
              travel_time = 900,
              coords = list(lat = 51.507609, lng = -0.128315),
              arrival_time_period = "weekday_morning",
              transportation = list(type = "public_transport"))

result <-
   time_map_fast(  
                  arrival_many_to_one = arrival_search
   )

## End(Not run)
```
Index

gecoding, 2
geocoding_reverse, 3

make_location, 4, 7–9, 11
make_search, 4, 7, 9, 11, 13, 14, 16, 17, 19
make_union_intersect, 5, 17
map_info, 6, 8

routes, 7

supported_locations, 8

time_filter, 4, 9, 10
time_filter_fast, 10
time_filter_fast_proto, 12
time_filter_postcode_districts, 14
time_filter_postcode_sectors, 15
time_filter_postcodes, 13
time_map, 5, 6, 17
time_map_fast, 18