Package ‘tidygeocoder’

January 18, 2021

Type Package
Title Geocoding Made Easy
Version 1.0.2
Description An intuitive interface for getting data from geocoder services.
BugReports https://github.com/jessecambon/tidygeocoder/issues
Depends R (>= 3.2.0)
License MIT + file LICENSE
Encoding UTF-8
LazyData true
Imports tibble, dplyr, httr, jsonlite
RoxygenNote 7.1.1
Suggests knitr, DT, markdown, ggplot2, ggrepel, maps, testthat (>= 2.1.0)
VignetteBuilder knitr
NeedsCompilation no
Author Jesse Cambon [aut, cre], Christopher Belanger [aut]
Maintainer Jesse Cambon <jesse.cambon@gmail.com>
Repository CRAN
Date/Publication 2021-01-18 20:20:02 UTC

R topics documented:

api_parameter_reference .......................................................... 2
extract_results .................................................................. 3
geo ................................................................................. 3
geocode .............................................................................. 6
### api_parameter_reference

<table>
<thead>
<tr>
<th>method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo_cascade</td>
<td>This dataset contains the mapping that allows this package to use a universal syntax to specify parameters for different geocoder services.</td>
</tr>
<tr>
<td>geo_census</td>
<td>The <code>generic_name</code> field is a universal field name while the <code>api_name</code> field shows the specific parameter name for the given geocoder service (<code>method</code>). When the <code>api_name</code> is missing this means that the parameter is not supported by the given geocoder service. When <code>generic_name</code> is missing this means the parameter is specific to that geocoding service.</td>
</tr>
<tr>
<td>get_api_query</td>
<td>Reference the documentation for <code>geo</code> for more information. Also reference <code>vignette(&quot;tidygeocoder&quot;)</code> for more details on constructing API queries.</td>
</tr>
<tr>
<td>louisville</td>
<td>Usage: <code>api_parameter_reference</code></td>
</tr>
<tr>
<td>query_api</td>
<td>Format: A tibble dataframe</td>
</tr>
<tr>
<td>sample_addresses</td>
<td>Source: Links to API documentation for each geocoder service are below.</td>
</tr>
</tbody>
</table>

- **Census**
- **Nominatim** ("osm")
- **Geocodio**
- **Location IQ** ("iq")
- **Google**
**extract_results**

See Also

get_api_query query_api geo geocode

---

**extract_results**

Extract geocoder results

---

**Description**

Parses the output of the query_api function. Latitude and longitude are extracted into the first two columns of the returned dataframe. This function is not used for batch geocoded results. Refer to query_api for example usage.

**Usage**

extract_results(method, response, full_results = TRUE, flatten = TRUE)

**Arguments**

method

method name

response

content from the geocoder service (returned by the query_api function)

full_results

if TRUE then the full results (not just latitude and longitude) will be returned.

flatten

if TRUE then flatten any nested dataframe content

**Value**

geocoder results in tibble format

See Also

get_api_query query_api geo

---

**geo**

Geocode addresses

---

**Description**

Geocodes addresses given as character values. The geocode function utilizes this function on addresses contained in dataframes. See example usage in vignette("tidygeocoder")

Note that not all geocoder services support certain address component parameters. For example, the Census geocoder only covers the United States and does not have a "country" parameter. Refer to api_parameter_reference for more details on geocoder service parameters and API usage.

This function uses the get_api_query, query_api, and extract_results functions to create, execute, and parse the geocoder API queries.
Usage

geo(
    address = NULL,
    street = NULL,
    city = NULL,
    county = NULL,
    state = NULL,
    postalcode = NULL,
    country = NULL,
    method = "census",
    cascade_order = c("census", "osm"),
    lat = lat,
    long = long,
    limit = 1,
    min_time = NULL,
    api_url = NULL,
    timeout = 20,
    mode = "",
    full_results = FALSE,
    unique_only = FALSE,
    return_addresses = TRUE,
    flatten = TRUE,
    batch_limit = 10000,
    verbose = FALSE,
    no_query = FALSE,
    custom_query = list(),
    return_type = "locations",
    iq_region = "us",
    geocodio_v = 1.6,
    param_error = TRUE
)

Arguments

address single line address (ie. ‘1600 Pennsylvania Ave NW, Washington, DC’). Do not combine with the address component arguments below (street, city, county, state, postalcode, country).

street street address (ie. ‘1600 Pennsylvania Ave NW’)

city city (ie. ‘Tokyo’)

county county (ie. ‘Jefferson’)

state state (ie. ‘Kentucky’)

postalcode postalcode (zip code if in the United States)

country country (ie. ‘Japan’)

method the geocoder service to be used. Refer to api_parameter_reference and the API documentation for each geocoder service for usage details and limitations.
- "census": US Census Geocoder. US street-level addresses only. Can perform batch geocoding.
- "osm": Nominatim (OSM). Worldwide coverage.
- "geocodio": Commercial geocoder. Covers US and Canada and has batch geocoding capabilities. Requires an API Key to be stored in the "GEOCODIO_API_KEY" environmental variable.
- "iq": Commercial Nominatim geocoder service. Requires an API Key to be stored in the "LOCATIONIQ_API_KEY" environmental variable.
- "google": Commercial Google geocoder service. Requires an API Key to be stored in the "GOOGLEGEOCODE_API_KEY" environmental variable.
- "cascade": Attempts to use one geocoder service and then uses a second geocoder service if the first service didn’t return results. The services and order is specified by the cascade_order argument. Note that this is not compatible with full_results = TRUE as geocoder services have different columns that they return.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cascade_order</td>
<td>a vector with two character values for the method argument in the order in which the geocoder services will be attempted for method = &quot;cascade&quot; (ie. c(census', geocodio'))</td>
</tr>
<tr>
<td>lat</td>
<td>latitude column name. Can be quoted or unquoted (ie. lat or 'lat').</td>
</tr>
<tr>
<td>long</td>
<td>longitude column name. Can be quoted or unquoted (ie. long or 'long').</td>
</tr>
<tr>
<td>limit</td>
<td>number of results to return per address. Note that not all methods support setting limit to a value other than 1. Also limit &gt; 1 is not compatible with batch geocoding if return_addresses = TRUE.</td>
</tr>
<tr>
<td>min_time</td>
<td>minimum amount of time for a query to take (in seconds) if using Location IQ or OSM. This parameter is used to abide by API usage limits. You can set it to a lower value (ie. 0) if using a local Nominatim server, for instance.</td>
</tr>
<tr>
<td>api_url</td>
<td>custom API URL. If specified, the default API URL will be overridden. This parameter can be used to specify a local Nominatim server.</td>
</tr>
<tr>
<td>timeout</td>
<td>query timeout (in minutes)</td>
</tr>
<tr>
<td>mode</td>
<td>set to 'batch' to force batch geocoding or 'single' to force single address geocoding (one address per query). If not specified then batch geocoding will be used if available (given method selected) when multiple addresses are provided, otherwise single address geocoding will be used.</td>
</tr>
<tr>
<td>full_results</td>
<td>returns all data from the geocoder service if TRUE. If FALSE then only longitude and latitude are returned from the geocoder service.</td>
</tr>
<tr>
<td>unique_only</td>
<td>only return results for unique addresses if TRUE</td>
</tr>
<tr>
<td>return_addresses</td>
<td>return input addresses with results if TRUE. Note that most services return the input addresses with full_results = TRUE and setting return_addresses to FALSE does not prevent this.</td>
</tr>
<tr>
<td>flatten</td>
<td>if TRUE then any nested dataframes in results are flattened if possible. Note that Geocodio batch geocoding results are flattened regardless.</td>
</tr>
<tr>
<td>batch_limit</td>
<td>limit to the number of addresses in a batch geocoding query. Both Geocodio and census batch geocoders have a 10,000 address limit so this is the default.</td>
</tr>
</tbody>
</table>
geocode

verbose if TRUE then detailed logs are output to the console

no_query if TRUE then no queries are sent to the geocoder and verbose is set to TRUE

custom_query API-specific parameters to be used, passed as a named list (ie. list(vintage = 'Current_Census2010')).

return_type only used when method = 'census'. Two possible values:
  • "locations" (default)
  • "geographies": returns additional geography columns. See the Census geocoder API documentation for more details.

iq_region 'us' (default) or 'eu'. Used for establishing API URL for the 'iq' method

geocodio_v version of geocodio api. 1.6 is default. Used for establishing API URL for the 'geocodio' method.

param_error if TRUE then an error will be thrown if certain parameters are invalid for the selected geocoder service (method). The parameters checked are limit, address, street, city, county, state, postalcode, and country. If method = 'cascade' then no errors will be thrown.

Value

parsed geocoding results in tibble format

See Also

geocoding results in tibble format

Examples

```r
geo(street = "600 Peachtree Street NE", city = "Atlanta", state = "Georgia", method = "census")

geo(address = c("Tokyo, Japan", "Lima, Peru", "Nairobi, Kenya"), method = 'osm')

geo(county = "Jefferson", state = "Kentucky", country = "US", method = 'osm')
```

geocode Geocode addresses in a dataframe
Description

Takes a dataframe containing addresses as an input and returns the dataframe results from a specified geocoder service by using the `geo` function. See example usage in `vignette("tidygeocoder")`.

This function passes all additional parameters (\ldots) to the `geo` function, so you can refer to its documentation for more details on possible arguments.

Note that the arguments used for specifying address columns (address, street, city, county, state, postalcode, and country) accept either quoted or unquoted column names (ie. "address_col" and `address_col` are both acceptable).

Usage

```r
geocode(
  .tbl,
  address = NULL,
  street = NULL,
  city = NULL,
  county = NULL,
  state = NULL,
  postalcode = NULL,
  country = NULL,
  lat = lat,
  long = long,
  return_addresses = FALSE,
  unique_only = FALSE,
  ...)
```

Arguments

- `.tbl` dataframe containing addresses
- `address` single line street address column name. Do not combine with address component arguments (street, city, county, state, postalcode, country)
- `street` street address column name
- `city` city column name
- `county` county column name
- `state` state column name
- `postalcode` postalcode column name (zip code if in the United States)
- `country` country column name
- `lat` latitude column name. Can be quoted or unquoted (ie. `lat` or 'lat').
- `long` longitude column name. Can be quoted or unquoted (ie. `long` or 'long').
- `return_addresses` if TRUE then addresses with standard names will be returned This is defaulted to FALSE because the address fields are already in the input dataset
geo_cascade

unique_only

if TRUE then only unique addresses and results will be returned. The input
dataframe’s format is not preserved. Addresses will also be returned if TRUE
(overrides return_addresses argument).

... arguments passed to the geo function

Value
input dataframe (.tbl) with geocoder results appended as columns

See Also
geo api_parameter_reference

Examples

library(dplyr)
sample_addresses[1:2,] %>% geocode(addr)
louisville[1:2,] %>% geocode(street = street, city = city, state = state, postalcode = zip)
sample_addresses[8:9,] %>% geocode(addr, method = 'osm', lat = 'lattes', long = 'longos')
sample_addresses[4:5,] %>% geocode(addr, method = 'cascade', lat = latitude, long = longitude)

geo_cascade

Convenience function for calling the geo function with method = 'cascade'

Description
Convenience function for calling the geo function with method = 'cascade'

Usage
geo_cascade(..., cascade_order = c("census", "osm"))

Arguments
... arguments passed from and to the geo function
cascade_order a vector with two character values for the method argument in the order in which
the geocoder services will be attempted (ie. c('census', 'geocodio'))
geo_census

Convenience functions for calling the geo function with a specified method

Description
Convenience functions for calling the geo function with a specified method

Usage
geo_census(...)
geo_osm(...)  
geo_geocodio(...)  
geo_iq(...)  
geo_google(...)  

Arguments
...
  arguments to be passed to the geo function

get_api_query

Construct a geocoder API query

Description
The geocoder API query is created using universal "generic" parameters and optional api-specific "custom" parameters. Generic parameters are converted into api parameters using the api_parameter_reference dataset.

The query_api function executes the queries created by this function.

Usage
get_api_query(method, generic_parameters = list(), custom_parameters = list())

Arguments
  method
    method name (ie. 'census')
  generic_parameters
    universal 'generic' parameters
  custom_parameters
    custom api-specific parameters
Value

API parameters as a named list

See Also

query_api geo api_parameter_reference

Examples

get_api_query("osm", list(address = "Hanoi, Vietnam"))

get_api_query("census", list(street = "11 Wall St", city = "NY", state = "NY"),
    list(benchmark = "Public_AR_Census2010"))

louisville

Louisville, Kentucky street addresses

Description

Louisville, Kentucky street addresses

Usage

louisville

Format

A tibble dataframe with component street addresses

street  Description of the address
city    Single line address
state   state
zip     zip code

Source

Downloaded from OpenAddresses.io on June 1st 2020
query_api

Execute a geocoder API query

Description

The `get_api_query` function can create queries for this function to execute.

Usage

```r
query_api(
  api_url,
  query_parameters,
  mode = "single",
  batch_file = NULL,
  address_list = NULL,
  content_encoding = "UTF-8",
  timeout = 20
)
```

Arguments

- **api_url**: Base URL of the API. Query parameters are appended to this
- **query_parameters**: API query parameters in the form of a named list
- **mode**: • "single": geocode a single address (all methods)
             • "list": batch geocode a list of addresses (geocodio)
             • "file": batch geocode a file of addresses (census)
- **batch_file**: A CSV file of addresses to upload (census)
- **address_list**: A list of addresses for batch geocoding (geocodio) should be ‘json’ for geocodio and ‘multipart’ for census
- **content_encoding**: Encoding to be used for parsing content
- **timeout**: Timeout in minutes

Value

- Raw results from the query

See Also

- `get_api_query`
- `extract_results`
- `geo`
sample_addresses

Examples

raw <- query_api("http://nominatim.openstreetmap.org/search", 
get_api_query("osm", list(address = 'Hanoi, Vietnam')))

extract_results('osm', jsonlite::fromJSON(raw))

<table>
<thead>
<tr>
<th>sample_addresses</th>
<th>Some sample addresses for testing</th>
</tr>
</thead>
</table>

Description

Some sample addresses for testing

Usage

sample_addresses

Format

A tibble dataframe with single line addresses

- **name**: Description of the address
- **addr**: Single line address
Index

* datasets
  api_parameter_reference, 2
  louisville, 10
  sample_addresses, 12

api_parameter_reference, 2, 3, 4, 6, 8–10

extract_results, 3, 3, 11

g eo, 2, 3, 3, 7–11
geo_cascade, 8
geo_census, 9
geo_geocodio (geo_census), 9
geo_google (geo_census), 9
geo_iq (geo_census), 9
geo_osm (geo_census), 9
geocode, 3, 6, 6
get_api_query, 3, 9, 11

louisville, 10

query_api, 3, 9, 10, 11

sample_addresses, 12