Package ‘tidygeocoder’

November 2, 2021

Type Package
Title Geocoding Made Easy
Version 1.0.5
Description An intuitive interface for getting data from geocoding services.
License MIT + file LICENSE
BugReports https://github.com/jessecambon/tidygeocoder/issues
Depends R (>= 3.3.0)
Imports tibble, dplyr, httr, jsonlite, lifecycle, progress
Suggests knitr, rmarkdown, ggplot2, ggrepel, maps, testthat (>= 3.0.2)
VignetteBuilder knitr
Encoding UTF-8
LazyData true
RoxygenNote 7.1.2
NeedsCompilation no
Author Jesse Cambon [aut, cre] (<https://orcid.org/0000-0001-6854-1514>), Diego Hernangómez [aut] (<https://orcid.org/0000-0001-8457-4658>), Christopher Belanger [aut] (<https://orcid.org/0000-0003-2070-5721>), Daniel Possenriede [aut] (<https://orcid.org/0000-0002-6738-9845>)
Maintainer Jesse Cambon <jesse.cambon@gmail.com>
Repository CRAN
Date/Publication 2021-11-02 22:20:05 UTC

R topics documented:

api_info_reference .................................................. 2
api_key_reference .................................................... 3
api_parameter_reference ............................................ 3
Description

This dataset is used for generating package documentation.

Usage

api_info_reference

Format

A tibble dataframe

**method**  Geocoding service name

**method_display_name**  Geocoding service display name

**site_url**  Link to the main site of the geocoding service

**api_documentation_url**  Link to API documentation

**api_usage_policy_url**  Link to the usage policy
Description

API keys are obtained from environmental variables. The `geo` and `reverse_geo` functions use this dataset to know which environmental variable to use for each geocoding service.

Usage

api_key_reference

Format

A tibble dataframe

- **method**: Geocoding service name
- **env_var**: Environmental variable name

See Also

- `geo` `reverse_geo`

---

api_parameter_reference

**Geocoding service API parameter reference**

Description

This dataset contains the mapping that allows this package to use a universal syntax to specify parameters for different geocoding services. Note that latitude and longitude input parameters for reverse geocoding are not in this dataset and are instead handled directly by the `reverse_geo` function.

The `generic_name` column is a universal parameter name that is shared between services. The `api_name` column is the parameter name for the given geocoding service specified by the `method` column. When `generic_name` is missing this means the parameter is specific to that geocoding service.

While the "census" and "google" services do not have a `limit` argument in their APIs, tidygeocoder provides a passthrough so you can still use the `limit` argument in `geo` and `reverse_geo` to limit the number of results per input.

Note that some geocoding services only use the `limit` argument for forward geocoding. Refer to API documentation of each service for more information.

Reference the documentation for `geo` and `reverse_geo` for more information. Also reference `vignette("tidygeocoder")` for more details on constructing API queries.
Usage

api_parameter_reference

Format

A tibble dataframe

- **method**: Geocoding service name
- **generic_name**: Universal parameter name
- **api_name**: Name of the parameter for the specified geocoding service
- **default_value**: Default value of the parameter
- **required**: Is the parameter required by the specified geocoding service?

Details

The API documentation for each service is linked to below:

- Nominatim
- US Census
- ArcGIS
- Geocodio
- Location IQ
- Google
- OpenCage
- Mapbox
- HERE
- TomTom
- MapQuest
- Bing
- Geoapify

See Also

geo reverse_geo get_api_query query_api min_time_reference batch_limit_reference
batch_limit_reference  Geocoding batch size limits

Description

The `geo` and `reverse_geo` functions use this dataset to set the maximum batch query size for each service.

Usage

batch_limit_reference

Format

A tibble dataframe

- **method**  Geocoding service name
- **batch_limit**  The maximum number of addresses or coordinates allowed per batch

See Also

geo, reverse_geo

extract_results  Extract forward geocoding results

Description

Parses the output of the `query_api` function for single address geocoding (ie. not batch geocoding). Latitude and longitude are extracted into the first two columns of the returned dataframe. Refer to `query_api` for example usage.

Usage

```
extract_results(
    method,
    response,
    full_results = TRUE,
    flatten = TRUE,
    limit = 1
)
```
extract_reverse_results

Arguments

- `method`: method name
- `response`: content from the geocoding 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
- `limit`: only used for "census" and "google" methods. Limits number of results per address.

Value

geocoding results in tibble format

See Also

- `get_api_query`
- `query_api`
- `geo`

Description

Parses the output of the `query_api` function for reverse geocoding. The address is extracted into the first column of the returned dataframe. This function is not used for batch geocoded results. Refer to `query_api` for example usage.

Usage

```r
extract_reverse_results(
  method,  # method name
  response,  # content from the geocoding service (returned by the `query_api` function)
  full_results = TRUE,  # if TRUE then the full results (not just an address column) will be returned.
  flatten = TRUE,  # if TRUE then flatten any nested dataframe content
  limit = 1  # only used for "census" and "google" method(s). Limits number of results per coordinate.
)
```

Arguments

- `method`: method name
- `response`: content from the geocoding service (returned by the `query_api` function)
- `full_results`: if TRUE then the full results (not just an address column) will be returned.
- `flatten`: if TRUE then flatten any nested dataframe content
- `limit`: only used for the "google" method(s). Limits number of results per coordinate.
Value

geocoding results in tibble format

See Also

get_api_query query_api reverse_geo

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 geocoding 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, min_time_reference, and batch_limit_reference for more details on geocoding service parameters and usage.

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

Usage

```r
geo(
  address = NULL,
  street = NULL,
  city = NULL,
  county = NULL,
  state = NULL,
  postalcode = NULL,
  country = NULL,
  method = "osm",
  cascade_order = c("census", "osm"),
  lat = "lat",
  long = "long",
  limit = 1,
  full_results = FALSE,
  mode = "",
  unique_only = FALSE,
  return_addresses = TRUE,
  min_time = NULL,
  progress_bar = show_progress_bar(),
  quiet = getOption("tidygeocoder.quiet", FALSE),
  api_url = NULL,
  timeout = 20,
  flatten = TRUE,
)```
batch_limit = NULL,
batch_limit_error = TRUE,
verbose = getOption("tidygeocoder.verbose", FALSE),
no_query = FALSE,
custom_query = list(),
api_options = list(),
return_type = "locations",
iq_region = "us",
geocodio_v = 1.6,
param_error = TRUE,
mapbox_permanent = FALSE,
here_request_id = NULL,
mapquest_open = FALSE)
)

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 (ie. zip code if in the United States)

country  country (ie. 'Japan')

method  the geocoding service to be used. API keys are loaded from environmental variables. Run usethis::edit_r_environ() to open your .Renviron file and add an API key as an environmental variable. For example, add the line GEOCODIO_API_KEY="YourAPIKeyHere"

- "osm": Nominatim.
- "census": US Census. Geographic coverage is limited to the United States. Batch geocoding is supported.
- "arcgis": ArcGIS.
- "geocodio": Geocodio. Geographic coverage is limited to the United States and Canada. An API key must be stored in the environmental variable "GEOCODIO_API_KEY”. Batch geocoding is supported.
- "iq": Location IQ. An API key must be stored in the environmental variable "LOCATIONIQ_API_KEY".
- "google": Google. An API key must be stored in the environmental variable "GOOGLEGEOCODE_API_KEY".
- "opencage": OpenCage. An API key must be stored in the environmental variable "OPENCAGE_KEY".
- "mapbox": Mapbox. An API key must be stored in the environmental variable "MAPBOX_API_KEY".
• "here": HERE. An API key must be stored in the environmental variable "HERE_API_KEY". Batch geocoding is supported, but must be explicitly called with mode = "batch".

• "tomtom": TomTom. An API key must be stored in the environmental variable "TOMTOM_API_KEY". Batch geocoding is supported.

• "mapquest": MapQuest. An API key must be stored in the environmental variable "MAPQUEST_API_KEY". Batch geocoding is supported.

• "bing": Bing. An API key must be stored in the environmental variable "BINGMAPS_API_KEY". Batch geocoding is supported, but must be explicitly called with mode = "batch".

• "geopy": Geoapy. An API key must be stored in the environmental variable "GEOAPIFY_KEY".

• "cascade" [Deprecated] use geocode_combine or geo_combine instead. The "cascade" method first uses one geocoding service and then uses a second geocoding 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 geocoding services have different columns that they return.

cascade_order [Deprecated] a vector with two character values for the method argument in the order in which the geocoding services will be attempted for method = "cascade" (ie. c("census", "geocodio"))

lat latitude column name. Can be quoted or unquoted (ie. 1at or "lat").

long longitude column name. Can be quoted or unquoted (ie. long or "long").

limit maximum number of results to return per input address. For many geocoding services the maximum value of the limit parameter is 100. Pass limit = NULL to use the default limit value of the selected geocoding service. For batch geocoding, limit must be set to 1 (default) if return_addresses = TRUE. Refer to api_parameter_reference for more details.

full_results returns all available data from the geocoding service if TRUE. If FALSE (default) then only latitude and longitude columns are returned from the geocoding service.

mode 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. For the "here" and "bing" methods the batch mode should be explicitly specified with mode = 'batch'.

unique_only only return results for unique inputs if TRUE

return_addresses 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.

min_time minimum amount of time for a query to take (in seconds). If NULL then min_time will be set to the default value specified in min_time_reference.

progress_bar if TRUE then a progress bar will be displayed for single input geocoding (1 input per query). By default the progress bar will not be shown for code executed when knitting R Markdown files or code within an RStudio notebook.
chunk. Can be set permanently with options(tidygeocoder.progress_bar = FALSE).

quiet if TRUE then console messages that are displayed by default regarding queries will be suppressed. FALSE is default. Can be set permanently with options(tidygeocoder.quiet = TRUE).

api_url custom API URL. If specified, the default API URL will be overridden. This parameter can be used to specify a local Nominatim server, for instance.

timeout query timeout (in minutes)

flatten if TRUE (default) then any nested dataframes in results are flattened if possible. Note that in some cases results are flattened regardless such as for Geocodio batch geocoding.

batch_limit limit to the number of addresses in a batch geocoding query. Defaults to the value in batch_limit_reference if not specified.

batch_limit_error [Deprecated] if TRUE then an error is thrown if the number of addresses exceeds the batch limit. (if executing a batch query). This is reverted to FALSE when using the cascade method.

verbose if TRUE then detailed logs are output to the console. FALSE is default. Can be set permanently with options(tidygeocoder.verbose = TRUE).

no_query if TRUE then no queries are sent to the geocoding service and verbose is set to TRUE. Used for testing.

custom_query API-specific parameters to be used, passed as a named list (ex. list(extratags = 1)).

api_options a named list of parameters specific to individual services. (ex. list(geocodio_v = 1.6, geocodio_hipaa = TRUE)). Each parameter begins with the name of the method (service) it applies to. The possible parameters are shown below with their default values.

• census_return_type (default: "locations"): set to "geographies" to return additional geography columns. Make sure to use full_results = TRUE if using the "geographies" setting.
• iq_region (default: "us"): set to "eu" to use the European Union API endpoint
• geocodio_v (default: 1.6): the version number of the Geocodio API to be used
• geocodio_hipaa (default: FALSE): set to TRUE to use the HIPAA compliant Geocodio API endpoint
• mapbox_permanent (default: FALSE): set to TRUE to use the mapbox.places-permanent endpoint. Note that this option should be used only if you have applied for a permanent account. Unsuccessful requests made by an account that does not have access to the endpoint may be billable.
• mapbox_open (default: FALSE): set to TRUE to use the Open Geocoding endpoint which relies solely on OpenStreetMap data
• here_request_id (default: NULL): this parameter would return a previous HERE batch job, identified by its RequestID. The RequestID of a batch job
is displayed when verbose is TRUE. Note that this option would ignore the current address parameter on the request, so the return_addresses or return_coords parameters need to be FALSE.

return_type [Deprecated] use the api_options parameter instead
iq_region [Deprecated] use the api_options parameter instead
geocodio_v [Deprecated] use the api_options parameter instead
param_error [Deprecated] if TRUE then an error will be thrown if any address parameters are used that are invalid for the selected service (method). If method = "cascade" then no errors will be thrown.
mapbox_permanent [Deprecated] use the api_options parameter instead
here_request_id [Deprecated] use the api_options parameter instead
mapquest_open [Deprecated] use the api_options parameter instead

Value
tibble (dataframe)

See Also
geocode api_parameter_reference min_time_reference batch_limit_reference

Examples

options(tidygeocoder.progress_bar = FALSE)

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

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

geo("100 Main St New York, NY", full_results = TRUE, method = "census", api_options = list(census_return_type = 'geographies'))

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 results from a specified geocoding service in a dataframe format using the geo function. See example usage in vignette("tidygeocoder").

This function passes all additional parameters (...) 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

geocode(
  .tbl,
  address = NULL,
  street = NULL,
  city = NULL,
  county = NULL,
  state = NULL,
  postalcode = NULL,
  country = NULL,
  lat = "lat",
  long = "long",
  return_input = TRUE,
  limit = 1,
  return_addresses = NULL,
  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
geocode

lat

long

return_input if TRUE then the input dataset will be combined with the geocoder query results and returned. If FALSE only the geocoder results will be returned.

limit maximum number of results to return per input address. For many geocoding services the maximum value of the limit parameter is 100. Pass limit = NULL to use the default limit value of the selected geocoding service. For batch geocoding, limit must be set to 1 (default) if return_addresses = TRUE. To use limit > 1 or limit = NULL set return_input to FALSE. Refer to api_parameter_reference for more details.

return_addresses if TRUE return input addresses. Defaults to TRUE if return_input is FALSE and FALSE if return_input is TRUE. This argument is passed to the geo() function.

unique_only if TRUE then only unique results will be returned and return_input will be set to FALSE.

... arguments passed to the geo function

Value
tibble (dataframe)

See Also
geo

Examples

library(dplyr, warn.conflicts = FALSE)
sample_addresses %>% slice(1:2) %>%
  geocode(addr, method = 'arcgis')

louisville %>% head(2) %>%
  geocode(street = street, city = city, state = state, postalcode = zip, method = 'census', full_results = TRUE)

sample_addresses %>% slice(8:9) %>%
  geocode(addr, method = 'osm', limit = 2, return_input = FALSE, full_results = TRUE)

sample_addresses %>% slice(4:5) %>%
  geocode(addr, method = 'arcgis', lat = latitude, long = longitude, full_results = TRUE)
geocode_combine  Combine multiple geocoding queries

Description

Executes multiple geocoding queries on a dataframe input and combines the results. To use a
character vector input instead, see the geo_combine function. Queries are executed by the geocode
function. See example usage in vignette("tidygeocoder").

Query results are by default labelled to show which query produced each result. Labels are either
placed in a query column (if return_list = FALSE) or used as the names of the returned list (if
return_list = TRUE). By default the method parameter value of each query is used as a query
label. If the same method is used in multiple queries then a number is added according to the order
of the queries (ie. osm1, osm2, ...). To provide your own custom query labels use the query_names
parameter.

Usage

geocode_combine(
  .tbl,
  queries,
  global_params = list(),
  return_list = FALSE,
  cascade = TRUE,
  query_names = NULL,
  lat = "lat",
  long = "long"
)

Arguments

.tbl dataframe containing addresses
queries a list of queries, each provided as a list of parameters. The queries are exe-
cuted by the geocode function in the order provided. (ex. list(list(method =
'osm'),list(method = 'census'),...))
global_params a list of parameters to be used for all queries (ex. list(address = 'address',full_results
 = TRUE))
return_list if TRUE then results from each service will be returned as separate dataframes.
If FALSE (default) then all results will be combined into a single dataframe.
cascade if TRUE (default) then only addresses that are not found by a geocoding service
will be attempted by subsequent queries. If FALSE then all queries will attempt
to geocode all addresses.
query_names optional vector with one label for each query provided (ex. c('geocodio batch','geocodio
single')).
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').
Value
tibble (dataframe)

See Also
geo_combine geo geocode

Examples

library(dplyr, warn.conflicts = FALSE)

sample_addresses %>%
  geocode_combine(
    queries = list(list(method = 'census'), list(method = 'osm')),
    global_params = list(address = 'addr'), cascade = TRUE)

more_addresses <- tibble::tribble(
  ~street_address, ~city, ~state, ~zip_cd,
  "624 W DAVIS ST #1D", "BURLINGTON", "NC", 27215,
  "201 E CENTER ST #268", "MEBANE", "NC", 27302,
  "100 Wall Street", "New York", "NY", 10005,
  "Bucharest", NA, NA, NA)

more_addresses %>%
  geocode_combine(
    queries = list(
      list(method = 'census', mode = 'batch'),
      list(method = 'census', mode = 'single'),
      list(method = 'osm')
    ),
    global_params = list(street = 'street_address',
                         city = 'city', state = 'state', postalcode = 'zip_cd'),
    query_names = c('census batch', 'census single', 'osm')
  )

more_addresses %>%
  geocode_combine(
    queries = list(
      list(method = 'census', mode = 'batch', street = 'street_address',
               city = 'city', state = 'state', postalcode = 'zip_cd'),
      list(method = 'arcgis', address = 'street_address')
    ),
    cascade = FALSE,
    return_list = TRUE
  )
geo_census

Convenience functions for calling geo()

Description

The method for geo() is specified in the function name.

[Deprecated]

Use the geo function directly instead.

Usage

geo_census(...)
geo_osm(...)
geo_geocodio(...)
geo_iq(...)
geo_google(...)
geo_opencage(...)
geo_mapbox(...)
geo_here(...)
geo_tomtom(...)
geo_mapquest(...)
geo_bing(...)
geo_arcgis(...)
geo_cascade(...)  

Arguments

...  arguments to be passed to the geo function
**geo_combine**

*Combine multiple geocoding queries*

**Description**

Passes address inputs in character vector form to the `geocode_combine` function for geocoding.

Note that address inputs must be specified for queries either with the `queries` parameter (for each query) or the `global_params` parameter (for all queries). For example `global_params = list(address = 'address')` passes addresses provided in the address parameter to all queries.

**Usage**

```r
geo_combine(
  queries,
  global_params = list(),
  address = NULL,
  street = NULL,
  city = NULL,
  county = NULL,
  state = NULL,
  postalcode = NULL,
  country = NULL,
  lat = lat,
  long = long,
  ...
)
```

**Arguments**

- `queries` - a list of queries, each provided as a list of parameters. The queries are executed by the `geocode` function in the order provided. (ex. `list(list(method = 'osm'),list(method = 'census'),...)`)
- `global_params` - a list of parameters to be used for all queries (ex. `list(address = 'address',full_results = TRUE)`)  
- `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 (ie. zip code if in the United States)
- `country` - country (ie. 'Japan')
- `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").

arguments passed to the `geocode_combine` function

Value
tibble (dataframe)

See Also
`geocode_combine` `geo` `geocode`

Examples

```r
options(tidygeocoder.progress_bar = FALSE)
example_addresses <- c("100 Main St New York, NY", "Paris", "Not a Real Address")

geo_combine(
  queries = list(  
    list(method = 'census'),
    list(method = 'osm')
  ),
  address = example_addresses,
  global_params = list(address = 'address')
)

geo_combine(
  queries = list(  
    list(method = 'arcgis'),
    list(method = 'census', mode = 'single'),
    list(method = 'census', mode = 'batch')
  ),
  global_params = list(address = 'address'),
  address = example_addresses,
  cascade = FALSE,
  return_list = TRUE
)

geo_combine(
  queries = list(  
    list(method = 'arcgis', address = 'city'),
    list(method = 'osm', city = 'city', country = 'country')
  ),
  city = c('Tokyo', 'New York'),
  country = c('Japan', 'United States'),
  cascade = FALSE
)
```
**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**

```r
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` `api_parameter_reference` `geo` `reverse_geo`

**Examples**

```r
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

min_time_reference  Minimum time required per query

Description
The geo and reverse_geo functions use this dataset to set the maximum query rate for each geocoding service. This rate is based on the usage restriction policies for each geocoding service.

Usage
min_time_reference

Format
A tibble dataframe

method  Geocoding service name
min_time  The minimum number of seconds required per query to comply with usage restrictions
description  A description of the usage rate restriction
Details

Links to the usage policies of each geocoding service are below:

- Nominatim
- US Census
- ArcGIS
- Geocodio
- Location IQ
- Google
- OpenCage
- Mapbox
- HERE
- TomTom
- MapQuest
- Bing
- Geoapify

See Also

geo reverse_geo

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,
  input_list = NULL,
  content_encoding = "UTF-8",
  timeout = 20,
  method = ""
)
```
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**: Determines the type of query to execute.
  - "single": Geocode a single input (all methods).
  - "list": Batch geocode a list of inputs (ex. geocodio).
  - "file": Batch geocode a file of inputs (ex. census).
- **batch_file**: A CSV file of input data to upload (for mode = 'file').
- **input_list**: A list of input data (for mode = 'list').
- **content_encoding**: Encoding to be used for parsing content.
- **timeout**: Timeout in minutes.
- **method**: If 'mapquest' or 'arcgis' then the query status code is changed appropriately.

Value

A named list containing the response content (`content`) and the HTTP request status (`status`).

See Also

- `get_api_query`
- `extract_results`
- `extract_reverse_results`
- `geo`
- `reverse_geo`

Examples

```r
raw1 <- query_api("http://nominatim.openstreetmap.org/search",
                  get_api_query("osm", list(address = 'Hanoi, Vietnam')))

raw1$status

extract_results('osm', jsonlite::fromJSON(raw1$content))

raw2 <- query_api("http://nominatim.openstreetmap.org/reverse",
                  get_api_query("osm", custom_parameters = list(lat = 38.895865, lon = -77.0307713)))

extract_reverse_results('osm', jsonlite::fromJSON(raw2$content))
```
### Description

Reverse geocodes geographic coordinates (latitude and longitude) given as numeric values. Latitude
and longitude inputs are limited to possible values. Latitudes must be between -90 and 90 and
longitudes must be between -180 and 180. Invalid values will not be sent to the geocoding service.
The `reverse_geo` function utilizes this function on coordinates contained in dataframes. See
example usage in vignette("tidygeocoder").

Refer to `api_parameter_reference`, `min_time_reference`, and `batch_limit_reference` for more details
on geocoding service parameters and usage.

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

### Usage

```r
reverse Geo(  
  lat,  
  long,  
  method = "osm",  
  address = "address",  
  limit = 1,  
  full_results = FALSE,  
  mode = "",  
  unique_only = FALSE,  
  return_coords = TRUE,  
  min_time = NULL,  
  progress_bar = show_progress_bar(),  
  quiet = getOption("tidygeocoder.quiet", FALSE),  
  api_url = NULL,  
  timeout = 20,  
  flatten = TRUE,  
  batch_limit = NULL,  
  verbose = getOption("tidygeocoder.verbose", FALSE),  
  no_query = FALSE,  
  custom_query = list(),  
  api_options = list(),  
  iq_region = "us",  
  geocodio_v = 1.6,  
  mapbox_permanent = FALSE,  
  here_request_id = NULL,  
  mapquest_open = FALSE  
)  
```
Arguments

lat       latitude values (input data)
long      longitude values (input data)
method    the geocoding service to be used. API keys are loaded from environmental variables. Run useR::edit_r_environ() to open your .Renviron file and add an API key as an environmental variable. For example, add the line GEOCODIO_API_KEY="YourAPIKeyHere".

- "osm": Nominatim.
- "arcgis": ArcGIS.
- "geocodio": Geocodio. Geographic coverage is limited to the United States and Canada. An API key must be stored in the environmental variable "GEOCODIO_API_KEY". Batch geocoding is supported.
- "iq": Location IQ. An API key must be stored in the environmental variable "LOCATIONIQ_API_KEY".
- "google": Google. An API key must be stored in the environmental variable "GOOGLEGEOCODE_API_KEY".
- "opencage": OpenCage. An API key must be stored in the environmental variable "OPENCAGE_KEY".
- "mapbox": Mapbox. An API key must be stored in the environmental variable "MAPBOX_API_KEY".
- "here": HERE. An API key must be stored in the environmental variable "HERE_API_KEY". Batch geocoding is supported, but must be explicitly called with mode = "batch".
- "tomtom": TomTom. An API key must be stored in the environmental variable "TOMTOM_API_KEY". Batch geocoding is supported.
- "mapquest": MapQuest. An API key must be stored in the environmental variable "MAPQUEST_API_KEY". Batch geocoding is supported.
- "bing": Bing. An API key must be stored in the environmental variable "BINGMAPS_API_KEY". Batch geocoding is supported, but must be explicitly called with mode = "batch".
- "geoapify": Geoapify. An API key must be stored in the environmental variable "GEOAPIFY_KEY".

address   name of the address column (in the output data)
limit     maximum number of results to return per input coordinate. For many geocoding services the maximum value of the limit parameter is 100. Pass limit = NULL to use the default limit value of the selected geocoding service. For batch geocoding, limit must be set to 1 (default) if return_coords = TRUE. Refer to api_parameter_reference for more details.

full_results      returns all available data from the geocoding service if TRUE. If FALSE (default) then only a single address column is returned from the geocoding service.

mode     set to 'batch' to force batch geocoding or 'single' to force single coordinate geocoding (one coordinate per query). If not specified then batch geocoding will be used if available (given method selected) when multiple coordinates are provided; otherwise single address geocoding will be used. For the "here" and "bing" methods the batch mode should be explicitly specified with mode = 'batch'.


reverse_geo

unique_only  only return results for unique inputs if TRUE
return_coords return input coordinates with results if TRUE. Note that most services return the input coordinates with full_results = TRUE and setting return_coords to FALSE does not prevent this.
min_time  minimum amount of time for a query to take (in seconds). If NULL then min_time will be set to the default value specified in min_time_reference.
progress_bar  if TRUE then a progress bar will be displayed for single input geocoding (1 input per query). By default the progress bar will not be shown for code executed when knitting R Markdown files or code within an RStudio notebook chunk. Can be set permanently with options(tidygeocoder.progress_bar = FALSE).
quiet  if TRUE then console messages that are displayed by default regarding queries will be suppressed. FALSE is default. Can be set permanently with options(tidygeocoder.quiet = TRUE).
api_url  custom API URL. If specified, the default API URL will be overridden. This parameter can be used to specify a local Nominatim server, for instance.
timeout  query timeout (in minutes)
flatten  if TRUE (default) then any nested dataframes in results are flattened if possible. Note that in some cases results are flattened regardless such as for Geocodio batch geocoding.
batch_limit  limit to the number of coordinates in a batch geocoding query. Defaults to the value in batch_limit_reference if not specified.
verbose  if TRUE then detailed logs are output to the console. FALSE is default. Can be set permanently with options(tidygeocoder.verbose = TRUE).
no_query  if TRUE then no queries are sent to the geocoding service and verbose is set to TRUE. Used for testing.
custom_query  API-specific parameters to be used, passed as a named list (ex. list(extratags = 1)).
api_options  a named list of parameters specific to individual services. (ex. list(geocodio_v = 1.6, geocodio_hipaa = TRUE)). Each parameter begins with the name of the method (service) it applies to. The possible parameters are shown below with their default values.
  - census_return_type (default: "locations"): set to "geographies" to return additional geography columns. Make sure to use full_results = TRUE if using the "geographies" setting.
  - iq_region (default: "us"): set to "eu" to use the European Union API endpoint
  - geocodio_v (default: 1.6): the version number of the Geocodio API to be used
  - geocodio_hipaa (default: FALSE): set to TRUE to use the HIPAA compliant Geocodio API endpoint
  -.mapboxPermanent (default: FALSE): set to TRUE to use the mapbox.places-permanent endpoint. Note that this option should be used only if you have applied for a permanent account. Unsuccessful requests made by an account that does not have access to the endpoint may be billable.
reverse_geocode

- `mapbox_open` (default: FALSE): set to TRUE to use the Open Geocoding endpoint which relies solely on OpenStreetMap data
- `here_request_id` (default: NULL): this parameter would return a previous HERE batch job, identified by its RequestID. The RequestID of a batch job is displayed when verbose is TRUE. Note that this option would ignore the current address parameter on the request, so the `return_addresses` or `return.coords` parameters need to be FALSE.

`iq_region` [Deprecated] use the `api_options` parameter instead
`geocodio_v` [Deprecated] use the `api_options` parameter instead
`mapbox_permanent` [Deprecated] use the `api_options` parameter instead
`here_request_id` [Deprecated] use the `api_options` parameter instead
`mapquest_open` [Deprecated] use the `api_options` parameter instead

Value
tibble (dataframe)

See Also
- `reverse_geocode`
- `api_parameter_reference`
- `min_time_reference`
- `batch_limit_reference`

Examples

```r
options(tidygeocoder.progress_bar = FALSE)

reverse_geo(lat = 38.895865, long = -77.0307713, method = 'osm')

reverse_geo(
  lat = c(38.895865, 43.6534817, 300),
  long = c(-77.0307713, -79.3839347, 600),
  method = 'osm', full_results = TRUE
)
```

Description

Takes a dataframe containing coordinates (latitude and longitude) and returns the reverse geocoding query results from a specified service by using the `reverse_geo` function. See example usage in vignette("tidygeocoder").

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

Usage

reverse_geocode(
  .tbl, lat, long,
  address = "address",
  return_input = TRUE,
  limit = 1,
  return_coords = NULL,
  unique_only = FALSE,
  ...
)

Arguments

.tbl dataframe containing coordinates
lat latitude column name (input data). Can be quoted or unquoted (ie. lat or 'lat').
long longitude column name (input data). Can be quoted or unquoted (ie. long or 'long').
address address column name (output data). Can be quoted or unquoted (ie. addr or 'addr').
return_input if TRUE then the input dataset will be combined with the geocoder query results and returned. If FALSE only the geocoder results will be returned.
limit maximum number of results to return per input coordinate. For many geocoding services the maximum value of the limit parameter is 100. Pass limit = NULL to use the default limit value of the selected geocoding service. For batch geocoding, limit must be set to 1 (default) if return_coords = TRUE. To use limit > 1 or limit = NULL set return_input to FALSE. Refer to api_parameter_reference for more details.
return_coords if TRUE return input coordinates. Defaults to TRUE if return_input is FALSE and FALSE if return_input is TRUE. This argument is passed to the reverse_geo() function.
unique_only if TRUE then only unique results will be returned and return_input will be set to FALSE.
... arguments passed to the reverse_geo function

Value
tibble (dataframe)

See Also

reverse_geo
Examples

```r
library(tibble)
library(dplyr, warn.conflicts = FALSE)

tibble(
    latitude = c(38.895865, 43.6534817),
    longitude = c(-77.0307713, -79.3839347)
) %>%
  reverse_geocode(
    lat = latitude,
    long = longitude,
    method = 'osm',
    full_results = TRUE
  )

louisville %>% head(3) %>%
  reverse_geocode(lat = latitude, long = longitude,
  method = 'arcgis')

louisville %>% head(2) %>%
  reverse_geocode(lat = latitude, long = longitude,
  method = 'osm',
  limit = 2, return_input = FALSE)
```

Sample addresses for testing

Description

Sample addresses for testing

Usage

```r
sample_addresses
```

Format

A tibble dataframe with single line addresses

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

* datasets
  api_info_reference, 2
  api_key_reference, 3
  api_parameter_reference, 3
  batch_limit_reference, 5
  louisville, 20
  min_time_reference, 20
  sample_addresses, 28

api_info_reference, 2
api_key_reference, 3
api_parameter_reference, 3, 7, 9, 11, 13, 19, 23, 24, 26, 27

batch_limit_reference, 4, 5, 7, 10, 11, 23, 25, 26

extract_results, 5, 7, 22
extract_reverse_results, 6, 22, 23

go, 3–6, 7, 12, 13, 15, 16, 18–22
geo_arcsGIS (geo_census), 16
geo_bing (geo_census), 16
geo_cascade (geo_census), 16
geo_census, 16
geo_combine, 9, 14, 15, 17
geo_geocodio (geo_census), 16
geo_google (geo_census), 16
geo_here (geo_census), 16
geo_iq (geo_census), 16
geo.mapbox (geo_census), 16
google_mapquest (geo_census), 16
go_opengeocode (geo_census), 16
geo_osm (geo_census), 16
go_tomtom (geo_census), 16
go_coding, 7, 11, 12, 14, 15, 17, 18
go_geocoding, 9, 14, 15, 17, 18
google_api_query, 4, 6, 7, 19, 21–23

louisville, 20