Package ‘tidygeocoder’

April 20, 2021

Type Package
Title Geocoding Made Easy
Version 1.0.3
Description An intuitive interface for getting data from geocoder services.
License MIT + file LICENSE


BugReports https://github.com/jessecambon/tidygeocoder/issues
Depends R (>= 3.3.0)
Imports tibble, dplyr, httr, jsonlite, lifecycle
Suggests knitr, rmarkdown, ggplot2, ggrepel, maps, testthat (>= 3.0.2)

VignetteBuilder knitr
Encoding UTF-8
LazyData true

RoxygenNote 7.1.1

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],
Daniel Possenriede [aut] (<https://orcid.org/0000-0002-6738-9845>)

Maintainer Jesse Cambon <jesse.cambon@gmail.com>

Repository CRAN

Date/Publication 2021-04-19 22:00:02 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** Geocoder service name
- **method_display_name** Geocoder service display name
- **site_url** Link to the main site of the geocoder service
- **api_documentation_url** Link to API documentation
- **api_usage_policy_url** Link to the usage policy
## api_key_reference

**API key environmental variables**

### 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 geocoder service.

### Usage

```
api_key_reference
```

### Format

A tibble dataframe

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

### See Also

- `geo`
- `reverse_geo`

## api_parameter_reference

**Geocoder service API parameter reference**

### Description

This dataset contains the mapping that allows this package to use a universal syntax to specify parameters for different geocoder 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 geocoder service specified by the `method` column. When `generic_name` is missing this means the parameter is specific to that geocoder 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 geocoder 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**: Geocoder service name
- **generic_name**: Universal parameter name
- **api_name**: Name of the parameter for the specified geocoder service
- **default_value**: Default value of the parameter
- **required**: Is the parameter required by the specified geocoder 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

See Also

geo reverse_geo get_api_query query_api min_time_reference batch_limit_reference
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` Geocoder 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 (i.e., 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 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
- `limit`: only used for "census" and "google" methods. Limits number of results per address.

**Value**

geocoder results in tibble format

**See Also**

`get_api_query`, `query_api`, `geo`

---

**extract_reverse_results**

*Extract reverse geocoding results*

**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 geocoder 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 geocoder 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**

geocoder results in tibble format

**See Also**

get_api_query query_api reverse_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, min_time_reference, and batch_limit_reference for more details on geocoder 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,
  min_time = NULL,
  api_url = NULL,
  timeout = 20,
  mode = "",
  full_results = FALSE,
  unique_only = FALSE,
  return_addresses = TRUE,
  flatten = TRUE,
  batch_limit = NULL,
  batch_limit_error = TRUE,
)```
geo

verbose = FALSE,
no_query = FALSE,
custom_query = list(),
return_type = "locations",
iq_region = "us",
geocodio_v = 1.6,
param_error = TRUE,
mapboxPermanent = 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 (zip code if in the United States)

country country (ie. 'Japan')

method the geocoder 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 "GEOCODIO_API_KEY". Batch geocoding is supported.
• "iq": Location IQ. An API key must be stored in "LOCATIONIQ_API_KEY".
• "google": Google. An API key must be stored in "GOOGLEGEOCODE_API_KEY".
• "opencage": OpenCage. An API key must be stored in "OPENCAGE_KEY".
• "mapbox": Mapbox. An API key must be stored in "MAPBOX_API_KEY".
• "here": HERE. An API key must be stored in "HERE_API_KEY". Batch geocoding is supported, but must be explicitly called with mode = "batch".
• "tomtom": TomTom. An API key must be stored in "TOMTOM_API_KEY". Batch geocoding is supported.
• "mapquest": MapQuest. An API key must be stored in "MAPQUEST_API_KEY". Batch geocoding is supported.
• "bing": Bing. An API key must be stored in "BINGMAPS_API_KEY". Batch geocoding is supported, but must be explicitly called with mode = "batch".
• "cascade": First uses 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.

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

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").

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

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.

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.

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'.

full_results returns all data from the geocoder service if TRUE. If FALSE then only longitude and latitude are returned from the geocoder service.

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.

flatten if TRUE 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 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

no_query if TRUE then no queries are sent to the geocoder service and verbose is set to TRUE. Used for testing.
custom_query: API-specific parameters to be used, passed as a named list (e.g., list(extratags = 1)).

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 the API URL for the "iq" method.

geocodio_v: version of geocodio API. Used for establishing the API URL for the "geocodio" method.

param_error: 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: if TRUE then the mapbox.places-permanent endpoint would be used. 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.

here_request_id: 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 return_addresses needs to be FALSE.

mapquest_open: if TRUE then MapQuest would use the Open Geocoding endpoint, that relies solely on data contributed to OpenStreetMap.

Value:

tibble (dataframe)

See Also:

gencode api_parameter_reference min_time_reference batch_limit_reference

Examples:

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 addresses in a dataframe

Description

Takes a dataframe containing addresses as an input and returns the results from a specified geocoder 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
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_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 geocoder services the maximum value of the limit parameter is 100. Pass limit = NULL to use the default limit value of the selected geocoder 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
go

Examples

library(dplyr)
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)
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
```
**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"))
```

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Minimum time required per query</th>
</tr>
</thead>
</table>

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

### Usage

min_time_reference

### Format

A tibble dataframe

- **method**: Geocoder 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 geocoder service are below:

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

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 = ""
)
```
query_api

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

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 geocoder service. The `reverse_geocode` 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 geocoder 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 = NULL,
  limit = 1,
  min_time = NULL,
  api_url = NULL,
  timeout = 20,
  mode = "",
  full_results = FALSE,
  unique_only = FALSE,
  return_coords = TRUE,
  flatten = TRUE,
  batch_limit = NULL,
  verbose = FALSE,
  no_query = FALSE,
  custom_query = 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 geocoder 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.
- "arcgis": ArcGIS.
- "geocodio": Geocodio. Geographic coverage is limited to the United States and Canada. An API key must be stored in "GEOCODIO_API_KEY". Batch geocoding is supported.
- "iq": LocationIQ. An API key must be stored in "LOCATIONIQ_API_KEY".
- "google": Google. An API key must be stored in "GOOGLEGEOCODE_API_KEY".
- "openencage": OpenCage. An API key must be stored in "OPENCAGE_API_KEY".
- "mapbox": Mapbox. An API key must be stored in "MAPBOX_API_KEY".
- "here": HERE. An API key must be stored in "HERE_API_KEY". Batch geocoding is supported, but must be explicitly called with `mode = "batch"`.
- "tomtom": TomTom. An API key must be stored in "TOMTOM_API_KEY". Batch geocoding is supported.
- "mapquest": MapQuest. An API key must be stored in "MAPQUEST_API_KEY". Batch geocoding is supported.
- "bing": Bing. An API key must be stored in "BINGMAPS_API_KEY". Batch geocoding is supported, but must be explicitly called with `mode = "batch"`.

address
name of the address column (in the output data)

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

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`.

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)

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"`.

full_results
returns all data from the geocoder service if TRUE. If FALSE then only a single address column will be returned from the geocoder service.

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.
reverse_geocode

- **flatten**: if TRUE 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
- **no_query**: if TRUE then no queries are sent to the geocoder service and verbose is set to TRUE. Used for testing.
- **custom_query**: API-specific parameters to be used, passed as a named list (ie. `list(extratags = 1)`).
- **iq_region**: "us" (default) or "eu". Used for establishing the API URL for the "iq" method.
- **geocodio_v**: version of geocodio API. Used for establishing the API URL for the "geocodio" method.
- **mapboxPermanent**: if TRUE then the `mapbox.places-permanent` endpoint would be used. 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.
- **here_request_id**: This parameter would return a previous HERE batch job, identified by its RequestID. The RequestID of a batch job is displayed when verbose = TRUE. Note that this option would ignore the current lat, long parameters on the request, so return_coords needs to be FALSE.
- **mapquestOpen**: if TRUE then MapQuest would use the Open Geocoding endpoint, that relies solely on data contributed to OpenStreetMap.

**Value**

tibble (dataframe)

**See Also**

- `reverse_geocode`
- `api_parameter_reference`
- `min_time_reference`
- `batch_limit_reference`

**Examples**

```r
reverse_geo(lat = 38.895865, long = -77.0307713, method = 'osm', verbose = TRUE)

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

Reverse geocode coordinates in a dataframe

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.

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 geocoder services the maximum value of the limit parameter is 100. Pass limit = NULL to use the default limit value of the selected geocoder 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

library(tibble)
library(dplyr)

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', verbose = TRUE,
    limit = 2, return_input = FALSE)

---

sample_addresses Sample addresses for testing

Description
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_info_reference, 2
  api_key_reference, 3
  api_parameter_reference, 3
  batch_limit_reference, 5
  louisville, 15
  min_time_reference, 15
  sample_addresses, 22

  api_info_reference, 2
  api_key_reference, 3
  api_parameter_reference, 3, 7, 9, 10, 12,
  14, 18–21

  batch_limit_reference, 4, 5, 7, 9, 10, 18, 20

  extract_results, 5, 7, 17
  extract_reverse_results, 6, 17, 18

  geo, 3–6, 7, 11–17
  geo_arccgis (geo_census), 13
  geo_bing (geo_census), 13
  geo_cascade (geo_census), 13
  geo_census, 13
  geo_geocodio (geo_census), 13
  geo_google (geo_census), 13
  geo_here (geo_census), 13
  geo_iq (geo_census), 13
  geo_mapbox (geo_census), 13
  geo_mapquest (geo_census), 13
  geo_opencage (geo_census), 13
  geo_osm (geo_census), 13
  geo_tomtom (geo_census), 13
  geocode, 7, 10, 11
  get_api_query, 4, 6, 7, 14, 16–18

  louisville, 15

  min_time_reference, 4, 7, 9, 10, 15, 18–20

  query_api, 4–7, 14, 16, 18

reverse_geo, 3–5, 7, 14–17, 18, 21, 22
reverse_geocode, 18, 20, 21
sample_addresses, 22