

Package ‘sociome’

October 21, 2021

Type Package

Title Operationalizing Social Determinants of Health Data for Researchers

Version 2.1.0

Maintainer Nik Krieger <nk@case.edu>

Description Accesses raw data via API and calculates social determinants of health measures for user-specified locations in the US, returning them in tidyverse- and sf-compatible data frames.

License MIT + file LICENSE

BugReports <https://github.com/NikKrieger/sociome/issues>

Depends R (>= 3.3.0)

Imports censusapi, dplyr (>= 1.0.1), magrittr (>= 1.5), mice (>= 3.10.0.1), psych, purrr (>= 0.3.4), rlang (>= 0.4.7), stringr (>= 1.4.0), tidycensus (>= 1.0), tidyr (>= 1.1.0)

Suggests USpopcenters, cluster, geosphere, ggplot2 (>= 3.3.2), sf (>= 0.9-7), testthat (>= 2.3.2), tibble (>= 3.0.3), tigris (>= 1.0), units

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

NeedsCompilation no

Author Nik Krieger [aut, cre],
Jarrod Dalton [aut],
Cindy Wang [aut],
Adam Perzynski [aut],
National Institutes of Health/National Institute on Aging [fnd] (The development of this software package was supported by a research grant from the National Institutes of Health/National Institute on Aging, (Principal Investigators: Jarrod E. Dalton, PhD and Adam T. Perzynski, PhD; Grant Number: 5R01AG055480-02). All of its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NIH.)

Repository CRAN

Date/Publication 2021-10-21 09:10:01 UTC

R topics documented:

acs_vars	2
append_dissimilarities	3
calculate_adi	5
census_api_key	6
decennial_vars	6
get_adi	7
get_areas_near_coordinates	12
get_geoids	15
lon_lat_from_area	16

Index **18**

acs_vars	<i>ACS variable names for ADI and ADI-3 calculation</i>
----------	---

Description

A dataset of the ACS variable names used to calculate the Area Deprivation Index (ADI) and Berg Indices (ADI-3).

Usage

```
acs_vars
```

Format

A [`'tibble'`][`'tibble::tibble'`] with 139 rows and 10 variables:

variable ACS variable name

description Brief description of the data the variable contains

set1 Logical, indicating the variables to be used when calculating ADI and ADI-3 using the 1- or 3-year estimates from 2011 and later or when using the 5-year estimates from 2012 or later

set2 Logical, indicating the variables to be used when calculating ADI and ADI-3 at the block group level using the 2015 or 2016 estimates

set3 Logical, indicating the variables to be used when calculating ADI using the 2011 5-year estimates

set4 Logical, indicating the variables to be used when calculating ADI and ADI-3 using the 2010 1- or 3-year estimates

set5 Logical, indicating the variables to be used when calculating ADI and ADI-3 using the 2010 5-year estimates

set6 Logical, indicating the variables to be used when calculating ADI and ADI-3 using the 2008 or 2009 1-year estimates

set7 Logical, indicating the variables to be used when calculating ACS estimates not previously mentioned, including the 2009 5-year estimates

dec2010 Logical, indicating the variables to use in conjunction with the few actual 2010 decennial census variables when running `get_adi(year = 2010, dataset = "decennial")`

Note that not all year/estimate combinations are currently supported by the census API and/or tidycensus, and some may never be supported.

See Also

[decennial_vars](#)

append_dissimilarities

Append list columns of Gower's distances and sampling weights to a data frame

Description

Runs `cluster::daisy()` on a data frame, breaks up the columns of the resulting dissimilarity into a list, and adds this list to the data frame as a list column. In addition or instead, it adds a transformed version of the dissimilarity list, which can be used as sampling weights.

Usage

```
append_dissimilarities(
  data,
  cols = dplyr::everything(),
  dissimilarity_measure_name = "dissimilarities",
  sampling_weight_name = "sampling_weights",
  metric = "gower",
  ...
)
```

Arguments

<code>data</code>	A data frame that has at least one row and at least one column.
<code>cols</code>	<code><tidy-select></code> Columns of data on which to calculate dissimilarities.
<code>dissimilarity_measure_name</code> , <code>sampling_weight_name</code>	The names of the list columns that will be added to data. Cannot match the names of the existing columns. Make one of them NULL if you don't want it added, but they can't both be NULL.
<code>metric</code> , ...	Passed to <code>cluster::daisy()</code> . Use ... at your own risk.

Details

All columns are fed to `cluster::daisy()` by default, but the user can select which ones using the `cols` argument.

Once the full dissimilarity matrix is obtained, the columns are separated into a list via `asplit()` and appended to data. Each element of the list is therefore a `double` vector with `nrow(data)` values. For any given row, its dissimilarity vector represents the row's dissimilarity to every row.

The optional/alternative "sampling weight" column is a transformed version of the dissimilarity list: 1. All dissimilarity measures of 0 are replaced with the next smallest dissimilarity value in the vector. In effect, this means that a row's dissimilarity to itself (and any rows identical to it) is replaced with the dissimilarity value of its next most similar row. (Exception: if all elements are 0, all of them are replaced with 1). 2. Then the reciprocal of each element is taken so that larger values represent greater similarity. 3. Each element is divided by the sum of the vector, which standardizes the elements to add to 1.

Requires the package `cluster` to be installed.

Value

A data frame, specifically the data argument with one or two more columns added to the end.

Examples

```
# Running this on all mtcars columns
mtdissim <- append_dissimilarities(mtcars)

# Therefore, these numbers represent the dissimilarity of each row to the
# fifth row:
mtdissim$dissimilarities[[5]]

# And these are the dissimilarities' corresponding sampling weights:
mtdissim$sampling_weights[[5]]

# Now we run it on mtcars without the wt and qsec columns so that we purposely
# end up with some duplicate rows (the first and second).
mtdissim_dup <- append_dissimilarities(mtcars, cols = !c(wt, qsec))

# These represent each row's dissimilarity to its first row.
# Since we specifically told it not to take wt and qsec into account, the
# first two rows are identical. Therefore, both values are zero.
mtdissim_dup$dissimilarities[[1]]

# Here are the corresponding sampling weights. Notice that the first two
# rows' sampling weights are the same as the sampling weight of row 30, which
# is the next most similar row.
mtdissim_dup$sampling_weights[[1]]
```

calculate_adi	<i>Calculate ADI and ADI-3 from census data.</i>
---------------	--

Description

Calculate the Area Deprivation Index and Berg Indices (ADI-3) using decennial US census or American Community Survey (ACS) variables.

Usage

```
calculate_adi(data_raw, keep_indicators = FALSE, seed = NA)
```

Arguments

data_raw	A data frame, tibble , or sf object ultimately obtained via tidycensus::get_acs() or tidycensus::get_decennial() , having the data necessary to compute the indicators of the ADI and ADI-3. The columns of his data frame must be named according to the elements of the variable column in acs_vars and/or decennial_vars . The easiest way to obtain data like this is to run get_adi(raw_data_only = TRUE) .
keep_indicators	Logical indicating whether or not to keep the component indicators of the ADI and ADI-3 as well as the original census variables used to calculate them. Defaults to FALSE. See acs_vars and decennial_vars for basic descriptions of the raw census variables.
seed	Passed to the seed argument of mice::mice() when imputation is needed.

Details

The function [get_adi\(\)](#) calls this function by default as its final step, but some users may want to calculate ADI and ADI-3 values for different combinations of areas in a given data set. [get_adi\(raw_data_only = TRUE\)](#) returns the raw census data used to calculate ADI and ADI-3. Users may select subsets of such a data set and pipe them into [calculate_adi\(\)](#).

This function discerns what kind of census data that data contains (ACS, or one of the decennial censuses) by checking for the existence of key variables unique to each kind of data set.

Areas listed as having zero households are excluded from ADI and ADI-3 calculation. Their resulting ADIs and ADI-3s will be NA.

If calling this function directly (i.e., not via [get_adi\(\)](#)) on a data set that contains median household income (B19013_001) and does not contain median family income (B19113_001), median household income will be used in place of median family income, with a warning(). See the "Missingness and imputation" section of [get_adi\(\)](#).

Value

A [tibble](#) (or [sf](#)) with the same number of rows as data. Columns include GEOID, NAME, ADI, Financial Strength, Economic_Hardship_and_Inequality, and Educational_Attainment. Further columns containing the indicators and raw values will also be present if `keep_indicators = TRUE`.

See Also

For more information, see [get_adi\(\)](#), especially the sections titled **ADI and ADI-3 factor loadings** and **Missingness and imputation**.

Examples

```
## Not run:
# Wrapped in \dontrun{} because these examples require a Census API key.

raw_census <- get_adi("state", year = 2017, raw_data_only = TRUE)

calculate_adi(raw_census)

calculate_adi(raw_census, keep_indicators = TRUE)

## End(Not run)
```

census_api_key	<i>Census API Key installer</i>
----------------	---------------------------------

Description

See [tidycensus::census_api_key\(\)](#).

decennial_vars	<i>Decennial census variable names for ADI calculation</i>
----------------	--

Description

A dataset of the decennial census variable names used to calculate the Area Deprivation Index (ADI) and the Berg Indices (ADI-3).

Usage

```
decennial_vars
```

Format

A [tibble](#) with 137 rows and 4 variables:

variable Decennial census variable name

sumfile The summary tape file of the decennial census variable

year The year of the decennial census variable

description Brief description of the data the variable contains

See Also

[acs_vars](#)

get_adi

Get Area Deprivation Index (ADI) and Berg Indices (ADI-3)

Description

Returns the ADI and ADI-3 of user-specified areas.

Usage

```
get_adi(
  geography,
  state = NULL,
  county = NULL,
  geoid = NULL,
  zcta = NULL,
  year,
  dataset = c("acs5", "acs3", "acs1", "decennial"),
  geometry = FALSE,
  shift_geo = FALSE,
  keep_indicators = FALSE,
  raw_data_only = FALSE,
  cache_tables = TRUE,
  key = NULL,
  seed = NA,
  ...
)
```

Arguments

geography A character string denoting the level of census geography whose ADIs and ADI-3s you'd like to obtain. Must be one of `c("state", "county", "tract", "block group", "zcta")`. Required.

state	A character string specifying states whose ADI and ADI-3 data is desired. Defaults to NULL. Can contain full state names, two-letter state abbreviations, or a two-digit FIPS code/GEOID (must be a vector of strings, so use quotation marks and leading zeros if necessary). Must be left as NULL blank if using the geoid or zcta parameter.
county	A vector of character strings specifying the counties whose ADI and ADI-3 data you're requesting. Defaults to NULL. If not NULL, the state parameter must have a length of 1. County names and three-digit FIPS codes are accepted (must contain strings, so use quotation marks and leading zeros if necessary). Must be blank if using the geoid parameter.
geoid	A character vector of GEOIDs (use quotation marks and leading zeros). Defaults to NULL. Must be blank if state, county, or zcta is used. Can contain different levels of geography (see details).
zcta	A character vector of ZCTAs or the leading digit(s) of ZCTAs (use quotation marks and leading zeros). Defaults to NULL. Must be blank if state, county, or geoid is used. Strings under 5 digits long will yield all ZCTAs that begin with those digits. Requires that geography = "zcta". If geography = "zcta" and zcta = NULL, all ZCTAs in the US will be used.
year	Single integer specifying the year of US Census data to use.
dataset	The data set used to calculate ADIs and ADI-3s. Must be one of c("acs5", "acs3", "acs1", "decennial") denoting the 5-, 3-, and 1-year ACS along with the decennial census. Defaults to "acs5". When dataset = "decennial", year must be in c(1990, 2000, 2010). The 2010 decennial census did not include the long-form questionnaire used in the 1990 and 2000 censuses, so this function uses the 5-year estimates from the 2010 ACS to supply the data not included in the 2010 decennial census. In fact, the only 2010 decennial variables used are H003002, H014002, P020002, and P020008. Important: data are not always available depending on the level of geography and data set chosen. See https://www.census.gov/programs-surveys/acs/guidance/estimates.html .
geometry	Logical value indicating whether or not shapefile data should be included in the result, making the result an sf object instead of a plain tibble. Defaults to FALSE. The shapefile data that is returned is somewhat customizable: see the shift_geo and ... arguments.
shift_geo	Logical value. See the shift_geo argument of tidy census::get_acs() or tidy census::get_decennial() for details. See ... below for other ways to customize the shapefile data returned.
keep_indicators	Logical value indicating whether or not the resulting tibble or sf object will contain the socioeconomic measures used to calculate the ADI and ADI-3 values. Defaults to FALSE. See acs_vars and decennial_vars for basic descriptions of the raw census variables.

raw_data_only	Logical, indicating whether or not to skip calculation of the ADI and ADI-3 and only return the census variables. Defaults to FALSE.
cache_tables	The plural version of the cache_table argument in <code>tidycensus::get_acs()</code> or <code>tidycensus::get_decennial()</code> . (<code>get_adi()</code> calls the necessary <code>tidycensus</code> function many times in order to return ADIs and ADI-3s, so many tables are cached if TRUE). Defaults to TRUE.
key	Your Census API key as a character string. Obtain one at http://api.census.gov/data/key_signup.html . Defaults to NULL. Not necessary if you have already loaded your key with <code>census_api_key()</code> .
seed	Passed to <code>calculate_adi()</code> .
...	Additional arguments to be passed onto <code>tidycensus::get_acs()</code> or <code>tidycensus::get_decennial()</code> . Currently, none of these functions' formal arguments can be meaningfully customized (doing so will either throw an error or have no effect). However, when setting <code>geometry = TRUE</code> , the <code>tidycensus</code> functions do pass meaningful arguments onto the appropriate <code>tigris</code> function (namely, one of <code>tigris::states()</code> , <code>tigris::counties()</code> , <code>tigris::tracts()</code> , <code>tigris::block_groups()</code> , or <code>tigris::zctas()</code> , according to the the value of <code>geography</code>). This enables the user to somewhat customize the shapefile data obtained.

Details

Returns a `tibble` or `sf` object of the Area Deprivation Indices (ADIs) and Berg Indices (ADI-3s) of user-specified locations in the United States, utilizing US Census data. Locations that are listed as having zero households are excluded from ADI and ADI-3 calculation: their ADI and ADI-3 values will be NA.

Value

If `geometry = FALSE`, (the default) a `tibble`. If `geometry = TRUE` is specified, an `sf`.

Reference area

The concept of "reference area" is important to understand when using this function. The algorithm that produced the original ADIs employs factor analysis. As a result, the ADI is a relative measure; the ADI of a particular location is dynamic, varying depending on which other locations were supplied to the algorithm. In other words, **ADI will vary depending on the reference area you specify.**

For example, the ADI of Orange County, California is x when calculated alongside all other counties in California, but it is y when calculated alongside all counties in the US. The `get_adi()` function enables the user to define a **reference area** by feeding a vector of GEOIDs to its `geo_id` parameter (or alternatively for convenience, states and/or counties to state and county). The function then gathers data from those specified locations and performs calculations using their data alone.

The Berg Indices (ADI-3) were developed with this principle of relativity in mind, and as such there is no set of seminal ADI-3 values. Thus, the terms "Berg Indices" and "ADI-3" refer more nearly to any values generated using the algorithm employed in this package.

Areas listed as having zero households are excluded from the reference area, and their ADI and ADI-3 values will be NA.

The geoid parameter

Elements of `geoid` can represent different levels of geography, but they all must be either 2 digits (for states), 5 digits (for counties), 11 digits (for tracts), or 12 digits (for block groups). It must contain character strings, so use quotation marks as well as leading zeros where applicable.

ADI and ADI-3 factor loadings

The returned `tibble` or `sf` is of class `adi`, and it contains an attribute called `loadings`, which contains a tibble of the PCA loadings of each factor. This is accessible through `attr(name_of_tibble, "loadings")`.

Missingness and imputation

While this function allows flexibility in specifying reference areas (see the **Reference area** section above), data from the US Census are masked for sparsely populated places, resulting in many missing values.

Imputation is attempted via `mice::mice(m = 1, maxit = 50, method = "pmm", seed = seed)`. If imputation is unsuccessful, an error is thrown, but the dataset of indicators on which imputation was unsuccessful is available via `rlang::last_error()$adi_indicators` and the raw census data are available via `rlang::last_error()$adi_raw_data`. The former excludes areas with zero households, but the latter includes them.

One of the indicators of both ADI and the Financial Strength component of ADI-3 is median family income, but methodological issues with the 2015 and 2016 ACS have rendered this variable unavailable at the block group level for those years. When requested, this function will use median household income in its place, with a `warning()`. See <https://www.census.gov/programs-surveys/acs/technical-documentation/user-notes/2016-01.html>.

API-related error handling

Depending on user input, this function may call its underlying functions (`tidycensus::get_acs()` or `tidycensus::get_decennial()`) many times in order to accommodate their behavior. When these calls are broken up by state or by state and county, a message is printed indicating the state or state and county whose data is being pulled. These calls are wrapped in `purrr::insistently(purrr::rate_delay(), quiet = TRUE)`, meaning that they are attempted over and over until success, and `tidycensus` error messages are printed as they occur.

Warnings and disclaimers

Please note that this function calls data from US Census servers, so execution may take a long time depending on the user's internet connection and the amount of data requested.

For advanced users, if changing the `dataset` argument, be sure to know the advantages and limitations of the 1-year and 3-year ACS estimates. See <https://www.census.gov/programs-surveys/acs/guidance/estimates.html> for details.

Examples

```
## Not run:
# Wrapped in \dontrun{} because all these examples take >5 seconds
```

```

# and require a Census API key.

# ADI of all census tracts in Cuyahoga County, Ohio
get_adi(geography = "tract", year = 2017, state = "OH", county = "Cuyahoga")

# ADI and ADI-3 of all counties in Connecticut, using the 2014 ACS1 survey.
# Returns a warning because there are only 8 counties.
# A minimum of 30 locations is recommended.
get_adi(geography = "county", state = "CT", year = 2014, dataset = "acs1")

# Areas with zero households will have an ADI and ADI-3 of NA:
queens <-
  get_adi(
    "tract",
    year = 2017,
    state = "NY",
    county = "Queens",
    keep_indicators = TRUE,
    geometry = TRUE
  )
queens %>%
  dplyr::as_tibble() %>%
  dplyr::select(GEOID, NAME, ADI, households = B11005_001) %>%
  dplyr::filter(is.na(ADI) | households == 0) %>%
  print(n = Inf)

# geoid argument allows for highly customized reference populations.
# ADI of all census tracts in the GEOIDs stored in "delmarva" below:
# Notice the mixing of state- ("10") and county-level GEOIDs (the others).
delmarva_geoids <- c("10", "51001", "51131", "24015", "24029", "24035",
  "24011", "24041", "24019", "24045", "24039", "24047")
delmarva <-
  get_adi(
    geography = "tract",
    geoid = delmarva_geoids,
    dataset = "acs5",
    year = 2009,
    geometry = TRUE
  )

# Demonstration of geom_sf() integration:
require(ggplot2)

# The na.value argument changes the fill of NA ADI areas.
delmarva %>% ggplot() + geom_sf(aes(fill = ADI), lwd = 0)

# Setting direction = -1 makes the less deprived areas the lighter ones
# The argument na.value changes the color of zero-household areas
queens %>%
  ggplot() +
  geom_sf(aes(fill = ADI), lwd = 0) +
  scale_fill_viridis_c(na.value = "red", direction = -1)

```

```
# Obtain factor loadings:
attr(queens, "loadings")

## End(Not run)
```

```
get_areas_near_coordinates
```

Make a tibble of census areas closest to a user-specified center

Description

Returns a [tibble](#) containing the census areas whose centers of population are closest to some user-specified center. To specify the center, the user can manually enter longitude/latitude coordinates or use the helper function [lon_lat_from_area\(\)](#) to automatically grab the longitude/latitude coordinates of the center of population of an area. The cutoff point for how many areas will be return depends on the function used.

Usage

```
areas_in_radius(
  geography = c("state", "county", "tract", "block group"),
  center = lon_lat_from_area(state = "DC"),
  radius = 5,
  units = "miles",
  measure_from = "center of population",
  year = 2010,
  distance_fun = geosphere::distVincentyEllipsoid,
  batch_size = 50L
)

closest_n_areas(
  geography = c("state", "county", "tract", "block group"),
  center = lon_lat_from_area(state = "DC"),
  n = 50,
  measure_from = "center of population",
  year = 2010,
  distance_fun = geosphere::distVincentyEllipsoid,
  units = NULL,
  batch_size = 50L
)

closest_population(
  geography = c("state", "county", "tract", "block group"),
  center = lon_lat_from_area(state = "DC"),
  population = 1e+06,
  measure_from = "center of population",
  year = 2010,
```

```

    distance_fun = geosphere::distVincentyEllipsoid,
    units = NULL,
    batch_size = 50L
  )

```

Arguments

geography	The type of census areas that the resulting table will contain. One of <code>c("state", "county", "tract", "block_group")</code> .
center	The longitude/latitude coordinates of the center of the circle. A double vector of length 2 whose elements are finite numbers. Passed to the <code>y</code> argument of <code>geosphere::distm()</code> . The first element is the longitude coordinate (positive for west, negative for east). The second element is the latitude coordinate (positive for north, negative for south). The convenience function <code>lon_lat_from_area()</code> can be used to obtain the longitude/latitude coordinates of the center of population of a user-specified census area. Defaults to the center of population of the District of Columbia according to the 2010 decennial census.
radius	A single, non-negative number specifying the radius of the circle. Defaults to 5.
units	A single string specifying the units of the resulting distance column. If <code>NULL</code> , the <code>units</code> package does not need to be installed, and units will be meters. Otherwise, this will be passed to the <code>value</code> argument of <code>units::set_units(mode = "standard")</code> . For <code>areas_in_radius()</code> , this also used for the units of radius.
measure_from	Currently can only be "center of population", the default.
year	Must be either 2000 or the default of 2010.
distance_fun	Passed to the <code>fun</code> argument of <code>geosphere::distm()</code> . Defaults to <code>geosphere::distVincentyEllipsoid</code> which results in the most accurate measurement but is also the slowest.
batch_size	The number of distances calculated in each iterative call to <code>geosphere::distm()</code> . When the request is satisfied, these functions stop calculating distances in order to prevent potentially hundreds of thousands of unnecessary calculations. Defaults to 50.
n	A single positive integer specifying how many of the areas closest to center should be gathered. Defaults to 50.
population	A single positive integer specifying the target total population of the areas returned. See Details .

Details

`areas_in_radius()` returns all areas whose centers of population are within the user-specified radius around center.

`closest_n_areas()` returns the top `n` areas whose centers of population are closest areas to center.

Conceptually, `closest_population()` sequentially gathers the next closest area to center until the total population of the areas meets or exceeds population.

Distances are determined with `geosphere::distm()`.

Requires the packages `USpopcenters` and `geosphere` to be installed. Requires the units to be installed unless `units = NULL`.

Centers of population are based on the 2010 decennial census. See <https://www.census.gov/geographies/reference-files/time-series/geo/centers-population.2010.html> Only states, counties, tracts, and block groups are currently supported.

Value

A `tibble` with each of the columns found in the corresponding `USpopcenters` table, with two columns appended:

`geoid` - all FIPS code columns combined with `paste0()`.

`distance` - the number of units the area's LONGITUDE/LATITUDE center of population is away from the coordinates given in `center`.

See Also

[lon_lat_from_area\(\)](#)

Examples

```
# All states whose centers of population are within 300 kilometers of the
# center of population of New York County, New York (i.e, Manhattan):
areas_in_radius(
  geography = "state",
  center = lon_lat_from_area(state = "NY", county = "New York"),
  radius = 300,
  units = "km"
)

# The four census tracts whose centers of population are closest to the
# Four Corners (distance column is in meters due to setting units = NULL):
closest_n_areas("tract", center = c(-109.0452, 36.9991), n = 4, units = NULL)

# The counties closest to center of population of Kauai County, Hawaii whose
# total population reaches 3 million people:
closest_population(
  geography = "county",
  center = lon_lat_from_area("15007"),
  population = 3e6,
  units = "barleycorns"
)
```

get_geoids	<i>Obtain GEOIDs of areas</i>
------------	-------------------------------

Description

Returns a [tibble](#) or [sf](#) of GEOIDs, names, and decennial census population of user-specified locations.

Usage

```
get_geoids(  
  geography,  
  state = NULL,  
  county = NULL,  
  geoid = NULL,  
  year = 2010,  
  geometry = FALSE,  
  cache_tables = TRUE,  
  key = NULL,  
  ...  
)
```

Arguments

geography	A character string denoting the level of census geography whose GEOIDs you'd like to obtain. Must be one of c("state", "county", "tract", "block group", "block"). Note that block-level data cannot be obtained from 1990 and 2000 decennial census data due to limitations in tidycensus::get_decennial() . Whereas block-level 2010 decennial census data are available, block-level ADI and ADI-3 cannot be calculated due to the removal of the long-form questionnaire from the 2010 decennial census.
state, county, geoid, geometry, cache_tables, key	See the descriptions of the arguments in get_adi() .
year	Single integer specifying the year of US Census data to use. Defaults to 2010. Based on this year, data from the most recent decennial census will be returned (specifically, <code>year <- floor(year / 10) * 10</code> is run).
...	Additional arguments to be passed to tidycensus::get_decennial() . Use at your own risk.

Details

This allows users to quickly obtain all GEOIDs in a specified location at a specific level of geography without having to manually look them up somewhere else.

This facilitates calls to [get_adi\(\)](#) that involve somewhat complicated reference areas.

Examples

```
## Not run:
# Wrapped in \dontrun{} because it requires a Census API key.

# Get all tract GEOIDs for Manhattan
tracts <- get_geoids(geography = "tract", state = "New York", county = "New York")
tracts

# Get all block GEOIDs for the fifth tract on that list
get_geoids(geography = "block", geoid = tracts$GEOID[5])

## End(Not run)
```

lon_lat_from_area *Grab the longitude/latitude of the center of population of a census area*

Description

The user specifies a census area, and the function returns the longitude/latitude coordinates of the area's center of population according to the 2010 census.

Usage

```
lon_lat_from_area(geoid = NULL, state = NULL, county = NULL, year = 2010)
```

Arguments

geoid	A single string specifying the geoid of a census area. Must be 2, 5, 11, or 12 digits. Must be NULL if state is not NULL.
state	A single string containing the FIPS code, two-letter abbreviation, or full state name of a US state or the District of Columbia or Puerto Rico. Not case sensitive. Must be NULL if geoid is not NULL.
county	A single string specifying the name of a county in state or the three- or five-digit GEOID of a county in state. Not case sensitive. If entering a county name, it must match the beginning of only one of the county names in state. If entering a five-digit GEOID, it will throw an error if its first two digits do not match the GEOID of state. Must be NULL if state is NULL.
year	Either 2000 or the default of 2010.

Details

Centers of population are based on the 2010 decennial census. See <https://www.census.gov/geographies/reference-files/time-series/geo/centers-population.2010.html> Only states, counties, tracts, and block groups are currently supported.

Requires the data package USpopcenters to be installed.

Value

A **double** vector of length 2. The first element is LONGITUDE (positive for east, negative for west). The second element is LATITUDE (positive for north, negative for south).

See Also

[areas_in_radius\(\)](#)

Examples

```
# The center of population of Alaska
lon_lat_from_area(state = "alASKA")

# The center of population of Cook County, Illinois.
lon_lat_from_area(state = "IL", county = "Cook")

# The center of population of some tract in Manhattan
lon_lat_from_area(geoid = "36061021600")
```

Index

- * **datasets**
 - acs_vars, [2](#)
 - decennial_vars, [6](#)

- acs_vars, [2, 5, 7, 8](#)
- append_dissimilarities, [3](#)
- areas_in_radius
 - (get_areas_near_coordinates), [12](#)
- areas_in_radius(), [17](#)
- asplit(), [4](#)
- attr, [10](#)

- calculate_adi, [5](#)
- calculate_adi(), [9](#)
- census_api_key, [6](#)
- census_api_key(), [9](#)
- closest_n_areas
 - (get_areas_near_coordinates), [12](#)
- closest_population
 - (get_areas_near_coordinates), [12](#)
- cluster::daisy(), [3, 4](#)

- decennial_vars, [3, 5, 6, 8](#)
- double, [4, 17](#)

- floor, [15](#)

- geosphere::dism(), [13, 14](#)
- geosphere::distVincentyEllipsoid, [13](#)
- get_adi, [3, 5, 7](#)
- get_adi(), [5, 6, 15](#)
- get_areas_near_coordinates, [12](#)
- get_geoids, [15](#)

- lon_lat_from_area, [16](#)
- lon_lat_from_area(), [12–14](#)

- mice::mice, [10](#)

- mice::mice(), [5](#)

- nrow, [4](#)

- paste0(), [14](#)
- purrr::insistently, [10](#)
- purrr::rate_delay(), [10](#)

- rlang::last_error(), [10](#)

- sf, [5, 6, 8–10, 15](#)

- tibble, [5–10, 12, 14, 15](#)
- tidycensus::census_api_key(), [6](#)
- tidycensus::get_acs(), [5, 8–10](#)
- tidycensus::get_decennial(), [5, 8–10, 15](#)
- tigris::block_groups(), [9](#)
- tigris::counties(), [9](#)
- tigris::states(), [9](#)
- tigris::tracts(), [9](#)
- tigris::zctas(), [9](#)

- units::set_units, [13](#)