Package ‘geomander’

October 13, 2022

**Type** Package

**Title** Geographic Tools for Studying Gerrymandering

**Version** 2.1.0

**Date** 2022-06-21

**Description** A compilation of tools to complete common tasks for studying gerrymandering. This focuses on the geographic tool side of common problems, such as linking different levels of spatial units or estimating how to break up units. Functions exist for creating redistricting-focused data for the US.

**License** MIT + file LICENCE

**URL** https://www.christophertkenny.com/geomander/,
https://github.com/christopherkenny/geomander

**BugReports** https://github.com/christopherkenny/geomander/issues

**RoxygenNote** 7.2.0

**LinkingTo** Rcpp (>= 1.0.7)

**Imports** dplyr, magrittr, sf, stringr, tibble, tidyr, tigris (>= 1.5),
    ggplot2, readr, dataverse, censable, geos, cli, rlang, Rcpp

**Depends** R (>= 3.0.2)

**Suggests** redist, knitr, rmarkdown, testthat (>= 3.0.0)

**LazyData** true

**Encoding** UTF-8

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**NeedsCompilation** yes

**Author** Christopher T. Kenny [aut, cre]
    (<https://orcid.org/0000-0002-9386-6860>)

**Maintainer** Christopher T. Kenny <christopherkenny@fas.harvard.edu>

**Repository** CRAN

**Date/Publication** 2022-06-23 20:10:05 UTC
R topics documented:

- geomander-package .................................................. 3
- add_edge ................................................................. 5
- adjacency .............................................................. 5
- alarm_states ........................................................... 6
- baf_to_vtd .............................................................. 7
- block2prec ............................................................. 8
- block2prec_by_county .................................................. 8
- checkerboard ......................................................... 9
- checkerboard_adj .................................................... 10
- check_contiguity ..................................................... 10
- check_polygon_contiguity ............................................ 11
- clean_vest ............................................................. 12
- compare_adjacencies ................................................ 12
- count_connections .................................................. 13
- create_block_table .................................................. 14
- create_tract_table ................................................... 15
- dra2r ................................................................. 16
- estimate_down ....................................................... 16
- estimate_up .......................................................... 17
- geos_centerish ...................................................... 18
- geos_circle_center ................................................ 19
- geo_estimate_down ................................................... 19
- geo_estimate_up ..................................................... 20
- geo_filter ............................................................. 21
- geo_match ............................................................ 22
- geo_plot .............................................................. 22
- geo_plot_group ...................................................... 23
- geo_sort ............................................................. 24
- geo_trim ............................................................. 24
- get_alarm ............................................................. 25
- get_vest ............................................................ 26
- global_gearys ...................................................... 27
- global_morans ..................................................... 27
- gstar_i ............................................................... 28
- local_gearys ....................................................... 29
- local_morans ..................................................... 29
- nrcsd ............................................................... 30
- orange ............................................................. 31
- precincts ............................................................ 31
- r2dra ............................................................... 32
- rockland ............................................................ 33
- seam_adj ............................................................ 33
- seam_geom .......................................................... 34
- seam_rip ............................................................ 35
- seam_sew ............................................................. 36
- split_precinct ...................................................... 36
**Description**

A compilation of tools to complete common tasks for studying gerrymandering. This focuses on the geographic tool side of common problems, such as linking different levels of spatial units or estimating how to break up units. Functions exist for creating redistricting-focused data for the US.

**Package Content**

Index of help topics:

- `add_edge` Add Edges to an Adjacency List
- `adjacency` Build Adjacency List
- `alarm_states` List Available States from ALARM Data
- `baf_to_vtd` Estimate Plans from a Block Assignment File to Voting Districts
- `block2prec` Aggregate Block Table by Matches
- `block2prec_by_county` Aggregate Block Table by Matches and County
- `check_contiguity` Check Contiguity by Group
- `check_polygon_contiguity` Check Polygon Contiguity
- `checkerboard` Checkerboard
- `checkerboard_adj` Checkerboard Adjacency
- `clean_vest` Clean Vest Names
- `compare_adjacencies` Compare Adjacency Lists
- `count_connections` Count Times Precincts are Connected
- `create_block_table` Create Block Level Data
- `create_tract_table` Create Tract Level Data
- `dra2r` DRA to R
- `estimate_down` Estimate Down Levels
- `estimate_up` Estimate Up Levels
- `geo_estimate_down` Estimate Down Geography Levels
- `geo_estimate_up` Estimate Up Geography Levels
geo_filter Filter to Intersecting Pieces
geo_match Match Across Geographic Layers
geo_plot Plots a Shape with Row Numbers as Text
geo_plot_group Create Plots of Shapes by Group with Connected Components Colored
geo_sort Sort Precincts
geo_trim Trim Away Small Pieces
geomander-package Geographic Tools for Studying Gerrymandering
geos_centerish Get the kind of center of each shape
geos_circle_center Get the centroid of the maximum inscribed circle
get_alarm Get ALARM Dataset
get_vest Get VEST Dataset
global_gearys Compute Global Geary's C
global_morans Compute Global Moran's I
gstar_i Compute Standardized Getis Ord G*i
local_gearys Compute Local Geary's C
local_morans Compute Local Moran's I
nrcsd nrcsd
orange orange
precincts precincts
r2dra R to DRA
rockland rockland
seam_adj Filter Adjacency to Edges Along Border
seam_geom Filter Shape to Geographies Along Border
seam_rip Remove Edges along a Boundary
seam_sew Suggest Edges to Connect Two Sides of a Border
split_precinct Split a Precinct
st_centerish Get the kind of center of each shape
st_circle_center Get the centroid of the maximum inscribed circle
subtract_edge Subtract Edges from an Adjacency List
suggest_component_connection Suggest Connections for Disconnected Groups
suggest_neighbors Suggest Neighbors for Lonely Precincts
towns towns
va18sub va18sub
va_blocks va_blocks
va_vtd va_vtd
vest_states List Available States from VEST Dataverse

Maintainer

NA

Author(s)

NA
add_edge

Add Edges to an Adjacency List

Description

Add Edges to an Adjacency List

Usage

add_edge(adj, v1, v2, zero = TRUE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adj</td>
<td>list of adjacent precincts</td>
</tr>
<tr>
<td>v1</td>
<td>integer or integer array for first vertex to connect. If array, connects each to corresponding entry in v2.</td>
</tr>
<tr>
<td>v2</td>
<td>integer or integer array for second vertex to connect. If array, connects each to corresponding entry in v1.</td>
</tr>
<tr>
<td>zero</td>
<td>boolean, TRUE if the list is zero indexed. False if one indexed.</td>
</tr>
</tbody>
</table>

Value

adjacency list.

Examples

data(towns)
adj <- adjacency(towns)
add_edge(adj, 2, 3)

adjacency

Build Adjacency List

Description

This mimics redist’s redist.adjacency using GEOS to create the patterns, rather than sf. This is faster than that version, but forces projections.

Usage

adjacency(shp, epsg = 3857)
Arguments

shpsf dataframe
epsgnumeric EPSG code to planarize to. Default is 3857.

Value

list with nrow(shp) entries

Examples

data(precincts)
adj <- adjacency(precincts)

alarm_states

List Available States from ALARM Data

Description

List Available States from ALARM Data

Usage

alarm_states()

Value

character abbreviations for states

Examples

## Not run:
# relies on internet availability and interactivity on some systems
alarm_states()

## End(Not run)
**baf_to_vtd**

Estimate Plans from a Block Assignment File to Voting Districts

**Description**

District lines are often provided at the census block level, but analyses often occur at the voting district level. This provides a simple way to estimate the block level to the voting district level.

**Usage**

\[ \text{baf\_to\_vtd}(\text{baf, plan\_name, GEOID = "GEOID"}) \]

**Arguments**

- **baf**: a tibble representing a block assignment file.
- **plan\_name**: character. Name of column in `baf` which corresponds to the districts.
- **GEOID**: character. Name of column which corresponds to each block’s GEOID, sometimes called "BLOCKID". Default is "GEOID".

**Details**

If a voting district is split between blocks, this currently uses the most common district.

**Value**

a tibble with a vtd-level assignment file

**Examples**

```r
# Not guaranteed to reach download from redistrict2020.org
## Not run:
# download and read baf ----
tf <- tempfile('.zip')
utils::download.file(url, tf)
utils::unzip(tf, exdir = dirname(tf))
baf <- readr::read_csv(file = paste0(dirname(tf), '/DE_SLDU_bef.csv'),
  col_types = 'ci')
names(baf) <- c('GEOID', 'ssd_20')

# convert to vtd level ----
baf_to_vtd(baf = baf, plan\_name = 'ssd_20', 'GEOID')
```

## End(Not run)
block2prec

Aggregate Block Table by Matches

Description

Aggregates block table values up to a higher level, normally precincts, hence the name block2prec.

Usage

block2prec(block_table, matches, geometry = FALSE)

Arguments

- **block_table**: Required. Block table output from create_block_table
- **matches**: Required. Grouping variable to aggregate up by, typically made with geo_match
- **geometry**: Boolean. Whether to keep geometry or not.

Value

dataframe with length(unique(matches)) rows

Examples

```r
set.seed(1)
data(rockland)
rockland$id <- sample(1:2, nrow(rockland), TRUE)
block2prec(rockland, rockland$id)
```

block2prec_by_county

Aggregate Block Table by Matches and County

Description

Performs the same type of operation as block2prec, but subsets a precinct geometry based on a County fips column. This helps get around the problem that county geometries often have borders that follow rivers and lead to funny shaped blocks. This guarantees that every block is matched to a precinct which is in the same county.

Usage

block2prec_by_county(block_table, precinct, precinct_county_fips, epsg = 3857)
**checkerboard**

**Arguments**

- block_table: Required. Block table output from create_block_table
- precinct: sf dataframe of shapefiles to match to.
- precinct_county_fips: Column within precincts
- epsg: numeric EPSG code to planarize to. Default is 3857.

**Value**

dataframe with nrow(precinct) rows

**Examples**

```r
## Not run:
# Need Census API
data(towns)
towns$fips <- '087'
block <- create_block_table('NY', 'Rockland')
block2prec_by_county(block, towns, 'fips')
## End(Not run)
```

---

**checkerboard**  
**Checkerboard**

**Description**

This data set contains 64 squares in an 8x8 grid, like a checkerboard.

**Usage**

data("checkerboard")

**Format**

An sf dataframe with 64 observations

**Examples**

data('checkerboard')
**checkerboard_adj**

*Checkerboard Adjacency*

**Description**

This data contains a zero indexed adjacency list for the checkerboard dataset.

**Usage**

```r
data("checkerboard_adj")
```

**Format**

A list with 64 entries

**Examples**

```r
data('checkerboard_adj')
```

---

**check_contiguity**

*Check Contiguity by Group*

**Description**

Given a zero-indexed adjacency list and an array of group identifiers, this returns a tibble which identifies the connected components. The three columns are ‘group’ for the inputted group, ‘group_number’ which uniquely identifies each group as a positive integer, and ‘component’ which identifies the connected component number for each corresponding entry of adjacency and group. If everything is connected within the group, then each element of ‘component’ will be ‘1’. Otherwise, the largest component is given the value ‘1’, the next largest ‘2’, and so on.

**Usage**

```r
check_contiguity(adj, group)
cct(adj, group)
ccm(adj, group)
```

**Arguments**

- `adj` : adjacency list
- `group` : array of group identifiers. Typically district numbers or county names. Defaults to 1 if no input is provided, checking that the adjacency list itself is one connected component.
Details

If nothing is provided to group, it will default to a vector of ones, checking if the adjacency graph is connected.
‘cct()’ is shorthand for creating a table of the component values. If everything is connected within each group, it returns a value of 1. In general, it returns a frequency table of components.
‘ccm()’ is shorthand for getting the maximum component value. It returns the maximum number of components that a group is broken into. This returns 1 if each group is connected.

Value

tibble with a column for each of inputted group, created group number, and the identified connected component number

Examples

data(checkerboard)
adj <- adjacency(checkerboard)
# These each indicate the graph is connected.
check_contiguity(adj)
cct(adj)
ccm(adj)
Examples

data(checkerboard)
check_polygon_contiguity(checkerboard, i)

clean_vest  Clean Vest Names

Description

Clean Vest Names

Usage

clean_vest(data)

Arguments

data  sf tibble from VEST

Value

data with cleaned names

Examples

data(va18sub)
va <- clean_vest(va18sub)

compare_adjacencies  Compare Adjacency Lists

Description

Compare Adjacency Lists

Usage

compare_adjacencies(adj1, adj2, shp, zero = TRUE)

Arguments

adj1  Required. A first adjacency list.
adj2  Required. A second adjacency list.
shp  shapefile to compare intersection types.
zero  Boolean. Defaults to TRUE. Are adj1 and adj2 zero indexed?
Value

tibble with row indices to compare, and optionally columns which describe the DE-9IM relationship between differences.

Examples

data(towns)
rook <- adjacency(towns)
sf_rook <- lapply(sf::st_relate(towns, pattern = 'F****1****'), function(x) {
  x - 1L
})
compare_adjacencies(rook, sf_rook, zero = FALSE)

Description

Count Times Precincts are Connected

Usage

count_connections(dm, normalize = FALSE)

Arguments

dm
district membership matrix

normalize
Whether to normalize all values by the number of columns.

Value

matrix with the number of connections between precincts

Examples

set.seed(1)
dm <- matrix(sample(1:2, size = 100, TRUE), 10)
count_connections(dm)
create_block_table  

Create Block Level Data

Description

Creates a block level dataset, using the decennial census information, with the standard redistricting variables.

Usage

create_block_table(
  state,
  county = NULL,
  geometry = TRUE,
  year = 2020,
  mem = FALSE,
  epsg = 3857,
  geography
)

Arguments

- **state**  
  Required. Two letter state postal code.

- **county**  
  Optional. Name of county. If not provided, returns blocks for the entire state.

- **geometry**  
  Defaults to TRUE. Whether to return the geometry or not.

- **year**  
  year, must be 2000, 2010, or 2020

- **mem**  
  Default is FALSE. Set TRUE to use memoized backend.

- **epsg**  
  numeric EPSG code to planarize to. Default is 3857.

- **geography**  
  Deprecated. Use geometry.

Value

dataframe with data for each block in the selected region. Data includes 2 sets of columns for each race or ethnicity category: population (pop) and voting age population (vap)

Examples

```r
## Not run:
# uses the Census API
create_block_table(state = 'NY', county = 'Rockland', geometry = FALSE)
```

```
## End(Not run)
```
create_tract_table

Create Tract Level Data

Description

Create Tract Level Data

Usage

create_tract_table(
  state,
  county = NULL,
  geometry = TRUE,
  year = 2019,
  mem = FALSE,
  epsg = 3857,
  geography
)

Arguments

state  Required. Two letter state postal code.
county Optional. Name of county. If not provided, returns tracts for the entire state.
geometry Defaults to TRUE. Whether to return the geography or not.
year year, must be >= 2009 and <= 2019.
mem Default is FALSE. Set TRUE to use memoized backend.
epsg numeric EPSG code to planarize to. Default is 3857.
geography Deprecated. Use geometry.

Value

dataframe with data for each tract in the selected region. Data includes 3 sets of columns for each race or ethnicity category: population (pop), voting age population (vap), and citizen voting age population (cvap)

Examples

## Not run:
# Relies on Census Bureau API
tract <- create_tract_table('NY', 'Rockland', year = 2018)

## End(Not run)
dra2r  

**Description**

Creates a block or precinct level dataset from DRA csv output.

**Usage**

`dra2r(dra, state, precincts, epsg = 3857)`

**Arguments**

- **dra**: The path to an exported csv or a dataframe with columns GEOID20 and District, loaded from a DRA export.
- **state**: the state postal code of the state
- **precincts**: an sf dataframe of precinct shapes to link the output to
- **epsg**: numeric EPSG code to planarize to. Default is 3857.

**Value**

sf dataframe either at the block level or precinct level

**Examples**

```r
# Example
blocklevel <- dra2r('dra_utah_test.csv', state = 'UT')
```

estimate_down  

**Description**

Non-geographic partner function to geo_estimate_down. Allows users to estimate down without the costly matching operation if they’ve already matched.

**Usage**

`estimate_down(wts, value, group)`
**estimate_up**

**Arguments**

- `wts` numeric vector. Defaults to 1. Typically population or VAP, as a weight to give each precinct.
- `value` numeric vector. Defaults to 1. Typically electoral outcomes, as a value to estimate down into blocks.
- `group` matches of length(wts) that correspond to row indices of value. Often, this input is the output of geo_match.

**Value**

numeric vector with each value split by weight

**Examples**

```r
library(dplyr)
set.seed(1)
data(checkerboard)
counties <- checkerboard %>%
group_by(id <= 32) %>%
summarize(geometry = sf::st_union(geometry)) %>%
mutate(pop = c(100, 200))
matches <- geo_match(checkerboard, counties)
estimate_down(wts = rep(1, nrow(checkerboard)), value = counties$pop, group = matches)
```

---

**estimate_up**  

*Estimate Up Levels*

**Description**

Non-geographic partner function to geo_estimate_up. Allows users to aggregate up without the costly matching operation if they've already matched.

**Usage**

`estimate_up(value, group)`

**Arguments**

- `value` numeric vector. Defaults to 1. Typically population values.
- `group` matches of length(value) that correspond to row indices of value. Often, this input is the output of geo_match.

**Value**

numeric vector with each value aggregated by group
Examples

library(dplyr)
set.seed(1)
data(checkerboard)
counties <- checkerboard %>%
  group_by(id <= 32) %>%
  summarize(geometry = sf::st_union(geometry)) %>%
  mutate(pop = c(100, 200))
matches <- geo_match(checkerboard, counties)
estimate_up(value = checkerboard$i, group = matches)

-----

geos_centerish  Get the kind of center of each shape

Description

Returns points within the shape, near the center. Uses the centroid if that’s in the shape, or point on surface if not.

Usage

geos_centerish(shp, epsg = 3857)

Arguments

  shp An sf dataframe
  epsg numeric EPSG code to planarize to. Default is 3857.

Value

  A geos geometry list

Examples

data(towns)
geos_centerish(towns)
### geos_circle_center

Get the centroid of the maximum inscribed circle

**Description**

Returns the centroid of the largest inscribed circle for each shape

**Usage**

```r
geos_circle_center(shp, tolerance = 0.01, epsg = 3857)
```

**Arguments**

- `shp`: An sf dataframe
- `tolerance`: positive numeric tolerance to simplify by. Default is 0.01.
- `epsg`: numeric EPSG code to planarize to. Default is 3857.

**Value**

A geos geometry list

**Examples**

```r
data(towns)
geos_centerish(towns)
```

---

### geo_estimate_down

Estimate Down Geography Levels

**Description**

Simple method for estimating data down to a lower level. This is most often useful for getting election data down from a precinct level to a block level in the case that a state or other jurisdiction split precincts when creating districts. Geographic partner to estimate_down.

**Usage**

```r
geo_estimate_down(from, to, wts, value, method = "center", epsg = 3857)
```
Arguments

from    Larger geography level
to      smaller geography level
wts     numeric vector of length nrow(to). Defaults to 1. Typically population or VAP, as a weight to give each precinct.
value   numeric vector of length nrow(from). Defaults to 1. Typically electoral outcomes, as a value to estimate down into blocks.
method  string from center, centroid, point, or area for matching levels
epsg    numeric EPSG code to planarize to. Default is 3857.

Value

numeric vector with each value split by weight

Examples

library(dplyr)
set.seed(1)
data(checkerboard)
counties <- checkerboard %>%
group_by(id <= 32) %>%
  summarize(geometry = sf::st_union(geometry)) %>%
  mutate(pop = c(100, 200))
geo_estimate_down(from = counties, to = checkerboard, value = counties$pop)

geo_estimate_up    Estimate Up Geography Levels

Description

Simple method for aggregating data up to a higher level. This is most often useful for getting population data from a block level up to a precinct level. Geographic partner to estimate_up.

Usage

geo_estimate_up(from, to, value, method = "center", epsg = 3857)

Arguments

from    smaller geography level
to      larger geography level
value   numeric vector of length nrow(from). Defaults to 1.
method  string from center, centroid, point, or area for matching levels
epsg    numeric EPSG code to planarize to. Default is 3857.
**geo_filter**

Description

Filter to Intersecting Pieces

Usage

```r
geo_filter(from, to, bool = FALSE, epsg = 3857)
```

Arguments

- `from`: Required. sf dataframe. the geography to subset
- `to`: Required. sf dataframe. the geography that from must intersect
- `bool`: Optional, defaults to FALSE. Should this just return a logical vector?
- `epsg`: numeric EPSG code to planarize to. Default is 3857.

Value

sf data frame or logical vector if bool == TRUE

Examples

```r
## Not run:
# Needs Census Bureau API
data(towns)
block <- create_block_table('NY', 'Rockland')
geo_filter(block, towns)

## End(Not run)
```

```r
data(towns)
data(rockland)
sub <- geo_filter(rockland, towns)
```
geo_match

*Match Across Geographic Layers*

**Description**

Match Across Geographic Layers

**Usage**

```r
geo_match(from, to, method = "center", tiebreaker = TRUE, epsg = 3857)
```

**Arguments**

- `from` smaller geographic level to match up from
- `to` larger geographic level to be matched to
- `method` string from center, centroid, point, or area for matching method
- `tiebreaker` Should ties be broken? boolean. If FALSE, precincts with no matches get value -1 and precincts with multiple matches get value -2.
- `epsg` numeric EPSG code to planarize to. Default is 3857.

**Value**

Integer Vector of matches length(to) with values in 1:nrow(from)

**Examples**

```r
library(dplyr)
data(checkerboard)
counties <- sf::st_as_sf(as.data.frame(rbind(
    sf::st_union(checkerboard %>% filter(i < 4)),
    sf::st_union(checkerboard %>% filter(i >= 4))
))
geo_match(from = checkerboard, to = counties)
geo_match(from = checkerboard, to = counties, method = 'area')
```

geo_plot

*Plots a Shape with Row Numbers as Text*

**Description**

One liner to plot a shape with row numbers

**Usage**

```r
geo_plot(shp)
```
### geo_plot_group

Create Plots of Shapes by Group with Connected Components Colored

#### Description
Create Plots of Shapes by Group with Connected Components Colored

#### Usage

```r
geo_plot_group(shp, adj, group, save = FALSE, path = "")
```

#### Arguments

- **shp**: An sf shapefile
- **adj**: adjacency list
- **group**: array of group identifiers. Typically district numbers or county names.
- **save**: Boolean, whether to save or not. Defaults to working directory.
- **path**: Path to save, only used if save is TRUE.

#### Value

list of ggplots

#### Examples

```r
library(dplyr)
data('checkerboard')
data('checkerboard_adj')

checkerboard <- checkerboard %>% mutate(discont = as.integer(j == 5 | j == 6))

p <- geo_plot_group(checkerboard, checkerboard_adj, checkerboard$discont)

p[[1]]
p[[2]]
```
**geo_sort**  
*Sort Precincts*

**Description**  
Reorders precincts by distance from the NW corner of the bounding box.

**Usage**  
`geo_sort(shp, epsg = 3857)`

**Arguments**

- `shp` sf dataframe, required.
- `epsg` numeric EPSG code to planarize to. Default is 3857.

**Value**

sf dataframe

**Examples**

```r
data(checkerboard)
geo_sort(checkerboard)
```

---

**geo_trim**  
*Trim Away Small Pieces*

**Description**  
Trim Away Small Pieces

**Usage**  
`geo_trim(from, to, thresh = 0.01, bool = FALSE, epsg = 3857)`

**Arguments**

- `from` Required. sf dataframe. the geography to subset
- `to` Required. sf dataframe. the geography that from must intersect
- `thresh` Percent as decimal of an area to trim away. Default is .01, which is 1%.
- `bool` Optional, defaults to FALSE. Should this just return a logical vector?
- `epsg` numeric EPSG code to planarize to. Default is 3857.
get_alarm

**Value**

sf data frame or logical vector if bool=TRUE

**Examples**

```r
## Not run:
# Needs Census Bureau API
data(towns)
block <- create_block_table('NY', 'Rockland')
geo_trim(block, towns, thresh = 0.05)

## End(Not run)
data(towns)
data(rockland)
sub <- geo_filter(rockland, towns)
rem <- geo_trim(sub, towns, thresh = 0.05)
```

---

**get_alarm**

*Get ALARM Dataset*

**Description**

Get's a dataset from the Algorithm-Assisted Redistricting Methodology Project. The current supported data is the 2020 retabulations of the VEST data, which can be downloaded with `get_vest`.

**Usage**

```
get_alarm(state, geometry = TRUE, epsg = 3857)
```

**Arguments**

- **state**: two letter state abbreviation
- **geometry**: Default is TRUE. Add geometry to the data?
- **epsg**: numeric EPSG code to planarize to. Default is 3857.

**Details**

See the full available data at [https://github.com/alarm-redist/census-2020](https://github.com/alarm-redist/census-2020).

**Value**

tibble with election data and optional geometry
Examples

# Takes a few seconds to run
ak <- get_alarm('AK')

---

get_vest

*Get VEST Dataset*

Description

Get VEST Dataset

Usage

get_vest(state, year, path = tempdir(), clean_names = TRUE, epsg = 3857)

Arguments

- **state**: two letter state abbreviation
- **year**: year in 2016, 2018, or 2020
- **path**: folder to put shape in. Default is `tempdir()`
- **clean_names**: Clean names. Default is TRUE. If FALSE, returns default names.
- **epsg**: numeric EPSG code to planarize to. Default is 3857.

Value

sf tibble

Examples

```r
## Not run:
# Requires Dataverse API
shp <- get_vest('CO', 2020)

## End(Not run)
```
global_gearys

Compute Global Geary’s C

Description

Computes the Global Geary’s Contiguity statistic. Can produce spatial weights from an adjacency or sf data frame, in which case the spatial_mat is a contiguity matrix. Users can also provide a spatial_mat argument directly.

Usage

global_gearys(shp, adj, wts, spatial_mat, epsg = 3857)

Arguments

shp sf data frame. Optional if adj or spatial_mat provided.
adj zero indexed adjacency list. Optional if shp or spatial_mat provided.
wts Required. Numeric vector with weights to use for Moran’s I.
spatial_mat matrix of spatial weights. Optional if shp or adj provided.
epsg numeric EPSG code to planarize to. Default is 3857.

Value

double

Examples

library(dplyr)
data('checkerboard')
checkerboard <- checkerboard %>% mutate(m = as.numeric((id + 1) %% 2 == 0))
global_gearys(shp = checkerboard, wts = checkerboard$m)

global_morans

Compute Global Moran’s I

Description

Computes the Global Moran’s I statistic and expectation. Can produce spatial weights from an adjacency or sf data frame, in which case the spatial_mat is a contiguity matrix. Users can also provide a spatial_mat argument directly.

Usage

global_morans(shp, adj, wts, spatial_mat, epsg = 3857)
Arguments

- **shp**: sf data frame. Optional if adj or spatial_mat provided.
- **adj**: zero indexed adjacency list. Optional if shp or spatial_mat provided.
- **wts**: Required. Numeric vector with weights to use for Moran’s I.
- **spatial_mat**: matrix of spatial weights. Optional if shp or adj provided.
- **epsg**: numeric EPSG code to planarize to. Default is 3857.

Value

- list

Examples

```r
library(dplyr)
data('checkerboard')
checkerboard <- checkerboard %>% mutate(m = as.numeric((id + i) %% 2 == 0))
global_morans(shp = checkerboard, wts = checkerboard$m)

gstar_i(shp = checkerboard, wts = checkerboard$m)
```

Description

Returns the Getis Ord G*i in standardized form.

Usage

`gstar_i(shp, adj, wts, spatial_mat, epsg = 3857)`

Arguments

- **shp**: sf data frame. Optional if adj or spatial_mat provided.
- **adj**: zero indexed adjacency list. Optional if shp or spatial_mat provided.
- **wts**: Required. Numeric vector with weights to use for Moran’s I.
- **spatial_mat**: matrix of spatial weights. Optional if shp or adj provided.
- **epsg**: numeric EPSG code to planarize to. Default is 3857.

Value

- vector of G*i scores

Examples

```r
library(dplyr)
data('checkerboard')
checkerboard <- checkerboard %>% mutate(m = as.numeric((id + i) %% 2 == 0))
gstar_i(shp = checkerboard, wts = checkerboard$m)
```
local_gearys

Compute Local Geary’s C

Description

Compute Local Geary’s C

Usage

local_gearys(shp, adj, wts, spatial_mat, epsg = 3857)

Arguments

shp sf data frame. Optional if adj or spatial_mat provided.
adj zero indexed adjacency list. Optional if shp or spatial_mat provided.
wts Required. Numeric vector with weights to use for Moran’s I.
spatial_mat matrix of spatial weights. Not required if shp or adj provided.
epsg numeric EPSG code to planarize to. Default is 3857.

Value

numeric vector

Examples

library(dplyr)
data('checkerboard')
checkerboard <- checkerboard %>% mutate(m = as.numeric((id + i) %% 2 == 0))
local_gearys(shp = checkerboard, wts = checkerboard$m)

local_morans

Compute Local Moran’s I

Description

Compute Local Moran’s I

Usage

local_morans(shp, adj, wts, spatial_mat, epsg = 3857)
Arguments

shp sf data frame. Optional if adj or spatial_mat provided.
adj zero indexed adjacency list. Optional if shp or spatial_mat provided.
wts Required. Numeric vector with weights to use for Moran’s I.
spatial_mat matrix of spatial weights. Optional if shp or adj provided.
epsg numeric EPSG code to planarize to. Default is 3857.

Value
tibble

Examples

library(dplyr)
data('Varcheckerboard')
checkerboard <- checkerboard %>% mutate(m = as.numeric((id + i) %% 2 == 0))
local_morans(shp = checkerboard, wts = checkerboard$m)

Description

The data contains the North Rockland Central School District.

Usage
data('nrcsd')

Format

An sf dataframe with 1 observation

Examples
data('nrcsd')
**orange**

<table>
<thead>
<tr>
<th>orange</th>
<th>orange</th>
</tr>
</thead>
</table>

**Description**
This data contains the blocks for Orange County NY, with geographies simplified to allow for better examples.

**Usage**
```r
data("orange")
```

**Format**
An sf dataframe with 10034 observations

**Details**
It can be recreated with:
```r
orange <- create_block_table('NY', 'Orange')
orange <- rmapshaper::ms_simplify(orange, keep_shapes = TRUE)
```

**Examples**
```r
data('orange')
```

<table>
<thead>
<tr>
<th>precincts</th>
<th>precincts</th>
</tr>
</thead>
</table>

**precincts**

**Description**
This data contains the election districts (or precincts) for Rockland County NY, with geographies simplified to allow for better examples.

**Usage**
```r
data("precincts")
```

**Format**
An sf dataframe with 278 observations

**References**
https://www.rocklandgis.com/portal/apps/sites/#/data/datasets/2d91f9db816c48318848ad66eb1a18e9

**Examples**
```r
data('precincts')
```
Description

Project a plan at the precinct level down to blocks into a format that can be used with DRA. Projecting down to blocks can take a lot of time for larger states.

Usage

```r
r2dra(precincts, plan, state, path, epsg = 3857)
```

Arguments

- **precincts**: Required. an sf dataframe of precinct shapes
- **plan**: Required. Either a vector of district assignments or the name of a column in precincts with district assignments.
- **state**: Required. the state postal code of the state
- **path**: Optional. A path to try to save to. Warns if saving failed.
- **epsg**: numeric EPSG code to planarize to. Default is 3857.

Value

tibble with columns Id, as used by DRA, identical to GEOID in census terms and District.

Examples

```r
## Not run:
# Needs Census Bureau API
cd <- tigris::congressional_districts() %>% filter(STATEFP == '49')
cnty <- tigris::counties(state = 49)
matchedcty <- geo_match(from = cnty, to = cd)
# use counties as precincts and let the plan be their center match:
r2dra(cnty, matchedcty, 'UT', 'r2dra_ex.csv')
## End(Not run)
```
Description
This data contains the blocks for Rockland County NY, with geographies simplified to allow for better examples.

Usage
data("rockland")

Format
An sf dataframe with 4764 observations

Details
It can be recreated with: rockland <- create_block_table('NY', 'Rockland') rockland <- rmappro::ms_simplify(rockland, keep_shapes = TRUE)

Examples
data('rockland')

seam_adj Filter Adjacency to Edges Along Border

Description
Filter Adjacency to Edges Along Border

Usage
seam_adj(adj, shp, admin, seam, epsg = 3857)

Arguments
adj  zero indexed adjacency graph
shp  tibble to subset and where admin column is found
admin quoted name of administrative unit column
seam administrative units to filter by
epsg numeric EPSG code to planarize to. Default is 3857.
seam_geom

Value

subset of adj

Examples

data("rockland")
data("orange")
data("nrcsd")

o_and_r <- rbind(orange, rockland)
o_and_r <- o_and_r %>% geo_filter(nrcsd) %>% geo_trim(nrcsd)
adj <- adjacency(o_and_r)

seam_adj(adj, shp = o_and_r, admin = 'county', seam = c('071', '087'))

seam_geom  Filter Shape to Geographies Along Border

Description

Filter Shape to Geographies Along Border

Usage

seam_geom(adj, shp, admin, seam, epsg = 3857)

Arguments

adj  zero indexed adjacency graph
shp  tibble to subset and where admin column is found
admin quoted name of administrative unit column
seam administrative units to filter by
epsg numeric EPSG code to planarize to. Default is 3857.

Value

subset of shp

Examples

data("rockland")
data("orange")
data("nrcsd")

o_and_r <- rbind(orange, rockland)
o_and_r <- o_and_r %>% geo_filter(nrcsd) %>% geo_trim(nrcsd)
seam_rip

adj <- adjacency(o_and_r)

seam_geom(adj, shp = o_and_r, admin = 'county', seam = c('071', '087'))

seam_rip

Remove Edges along a Boundary

Description

Remove Edges along a Boundary

Usage

seam_rip(adj, shp, admin, seam, epsg = 3857)

Arguments

adj  zero indexed adjacency graph
shp  tibble where admin column is found
admin quoted name of administrative unit column
seam units to rip the seam between by removing adjacency connections
epsg numeric EPSG code to planarize to. Default is 3857.

Value

adjacency list

Examples

data("rockland")
data("orange")
data("nrcsd")

o_and_r <- rbind(orange, rockland)
o_and_r <- o_and_r %>% geo_filter(nrcsd) %>% geo_trim(nrcsd)
adj <- adjacency(o_and_r)

seam_rip(adj, o_and_r, 'county', c('071', '087'))
seam_sew  

*Suggest Edges to Connect Two Sides of a Border*

**Description**

Suggest Edges to Connect Two Sides of a Border

**Usage**

`seam_sew(shp, admin, seam, epsg = 3857)`

**Arguments**

- `shp`: sf tibble where admin column is found
- `admin`: quoted name of administrative unit column
- `seam`: administrative units to filter by
- `epsg`: numeric EPSG code to planarize to. Default is 3857.

**Value**

tibble of edges connecting sides of a border

**Examples**

```r
data("rockland")
data("orange")
data("nrcsd")

o_and_r <- rbind(orange, rockland)
o_and_r <- o_and_r %>% geo_filter(nrcsd) %>% geo_trim(nrcsd)
adj <- adjacency(o_and_r)

adds <- seam_sew(o_and_r, 'county', c('071', '087'))
adj <- adj %>% add_edge(adds$v1, adds$v2)
```

split_precinct  

*Split a Precinct*

**Description**

States often split a precinct when they create districts but rarely provide the geography for the split precinct. This allows you to split a precinct using a lower geography, typically blocks.
**Usage**

`split_precinct(lower, precinct, split_by, lower_wt, split_by_id, epsg = 3857)`

**Arguments**

- **lower**  
  The lower geography that makes up the precinct, this is often a block level geography.
- **precinct**  
  The single precinct that you would like to split.
- **split_by**  
  The upper geography that you want to split precinct by
- **lower_wt**  
  Optional. Numeric weights to give to each precinct, typically VAP or population.
- **split_by_id**  
  Optional. A string that names a column in split_by that identifies each observation in split_by
- **epsg**  
  numeric EPSG code to planarize to. Default is 3857.

**Value**

sf data frame with precinct split

**Examples**

```r
library(sf)
data(checkerboard)
low <- checkerboard %>% dplyr::slice(1:3, 9:11)
prec <- checkerboard %>%
  dplyr::slice(1:3) %>%
  dplyr::summarize(geometry = sf::st_union(geometry))
dists <- checkerboard %>%
  dplyr::slice(1:3, 9:11) %>%
  dplyr::mutate(dist = c(1, 2, 2, 1, 3, 3)) %>%
  dplyr::group_by(dist) %>%
  dplyr::summarize(geometry = sf::st_union(geometry))

split_precinct(low, prec, dists, split_by_id = 'dist')
```

---

**st_centerish**  
*Get the kind of center of each shape*

**Description**

Returns points within the shape, near the center. Uses the centroid if that’s in the shape, or point on surface if not.

**Usage**

`st_centerish(shp, epsg = 3857)`
**Arguments**

- **shp**: An sf dataframe
- **epsg**: A numeric EPSG code to planarize to. Default is 3857.

**Value**

An sf dataframe where geometry is the center(ish) of each shape in shp

**Examples**

```r
data(towns)
st_centerish(towns)
```

---

**Description**

Returns the centroid of the largest inscribed circle for each shape

**Usage**

```r
st_circle_center(shp, tolerance = 0.01, epsg = 3857)
```

**Arguments**

- **shp**: An sf dataframe
- **tolerance**: A positive numeric tolerance to simplify by. Default is 0.01.
- **epsg**: A numeric EPSG code to planarize to. Default is 3857.

**Value**

An sf dataframe where geometry is the circle center of each shape in shp

**Examples**

```r
data(towns)
st_centerish(towns)
```
**subtract_edge**

*Subtract Edges from an Adjacency List*

**Description**

Subtract Edges from an Adjacency List

**Usage**

subtract_edge(adj, v1, v2, zero = TRUE)

**Arguments**

- `adj` : list of adjacent precincts
- `v1` : integer or integer array for first vertex to connect. If array, connects each to corresponding entry in v2.
- `v2` : integer or integer array for second vertex to connect. If array, connects each to corresponding entry in v1.
- `zero` : boolean, TRUE if adj is zero indexed. False if one indexed.

**Value**

adjacency list.

**Examples**

data(towns)
adj <- adjacency(towns)
subtract_edge(adj, 2, 3)

---

**suggest_component_connection**

*Suggest Connections for Disconnected Groups*

**Description**

Suggests nearest neighbors for connecting a disconnected group.

**Usage**

suggest_component_connection(shp, adj, group, epsg = 3857)
suggest_neighbors

Suggest Neighbors for Lonely Precincts

Description
For precincts which have no adjacent precincts, this suggests the nearest precinct as a friend to add. This is useful for when a small number of precincts are disconnected from the remainder of the geography, such as an island.

Usage
suggest_neighbors(shp, adj, idx, neighbors = 1)

Arguments
- shp: an sf shapefile
- adj: an adjacency list
- idx: Optional. Which indices to suggest neighbors for. If blank, suggests for those with no neighbors.
- neighbors: number of neighbors to suggest

Value
tibble with two columns of suggested rows of shp to connect in adj
Examples

```r
library(dplyr)
data(va18sub)
va18sub <- va18sub %>% filter(!VTDST %in% c('000516', '000510', '000505', '000518'))
adj <- adjacency(va18sub)
suggests <- suggest_neighbors(va18sub, adj)
adj <- adj %>% add_edge(v1 = suggests$x, v2 = suggests$y)
```

Description

This data contains 7 town boundaries for the towns which overlap North Rockland School District in NY.

Usage

data("towns")

Format

An sf dataframe with 7 observations

References

https://www.rocklandgis.com/portal/apps/sites/#/data/items/746ec7870a0b4f46b168e07369e79a27

Examples

```r
data('towns')
```

Description

This data contains a 90 precinct subset of Virginia from the 2018 Senate race. Contains results for Henrico County

Usage

data("va18sub")
Format

An sf dataframe with 90 observations

References


Examples

data('va_sub')

---

va_blocks

va_blocks

Description

This data contains the blocks Henrico County, VA with geographies simplified to allow for better examples.

Usage

data('va_blocks')

Format

An sf dataframe with 6354 observations

Details

blocks87 <- create_block_table(state = 'VA', county = '087') va_blocks <- rmapshaper::ms_simplify(va_blocks, keep_shapes = TRUE)

Examples

data('va_blocks')
va_vtd

---

**Description**

This data contains the blocks for Henrico County, VA with geographies simplified to allow for better examples.

**Usage**

data("va_blocks")

**Format**

An sf dataframe with 93 observations

**Details**

va_vtd <- tigris::voting_districts(state = 'VA') va_vtd <- rmapshaper::ms_simplify(va_vtd, keep_shapes = TRUE)

**Examples**

data('va_blocks')

---

vest_states

---

**Description**

List Available States from VEST Dataverse

**Usage**

vest_states(year)

**Arguments**

year  

**Value**

character abbreviations for states
Examples

## Not run:

# Requires Dataverse API
vest_states(2020)

## End(Not run)
# Index

* **center**
  - geos_centerish, 18
  - geos_circle_center, 19
  - st_centerish, 37
  - st_circle_center, 38

* **datasets**
  - alarm_states, 6
  - clean_vest, 12
  - get_alarm, 25
  - get_vest, 26
  - vest_states, 43

* **datatable**
  - block2prec, 8
  - block2prec_by_county, 8
  - create_block_table, 14
  - create_tract_table, 15
  - geo_filter, 21
  - geo_trim, 24

* **data**
  - checkerboard, 9
  - checkerboard_adj, 10
  - nrcsd, 30
  - orange, 31
  - precincts, 31
  - rockland, 33
  - towns, 41
  - va18sub, 41
  - va_blocks, 42
  - va_vtd, 43

* **dra**
  - dra2r, 16
  - r2dra, 32

* **estimate**
  - estimate_down, 16
  - estimate_up, 17
  - geo_estimate_down, 19
  - geo_estimate_up, 20
  - geo_match, 22

* **fix**
  - add_edge, 5
  - adjacency, 5
  - check_contiguity, 10
  - check_polygon_contiguity, 11
  - compare_adjacencies, 12
  - geo_sort, 24
  - split_precinct, 36
  - subtract_edge, 39
  - suggest_component_connection, 39
  - suggest_neighbors, 40

* **leftover**
  - count_connections, 13

* **package**
  - geomander-package, 3

* **plot**
  - geo_plot, 22
  - geo_plot_group, 23

* **seam**
  - seam_adj, 33
  - seam_geom, 34
  - seam_rip, 35
  - seam_sew, 36

* **spatcorr**
  - global_gearys, 27
  - global_morans, 27
  - gstar_i, 28
  - local_gearys, 29
  - local_morans, 29

    - add_edge, 5
    - adjacency, 5
    - alarm_states, 6
    - baf_to_vtd, 7
    - block2prec, 8
    - block2prec_by_county, 8
    - ccm (check_contiguity), 10
    - cct (check_contiguity), 10
    - check_contiguity, 10
INDEX

check_polygon_contiguity, 11
clean_vest, 12
compare_adjacencies, 12
count_connections, 13
create_block_table, 14
create_tract_table, 15
dra2r, 16
estimate_down, 16
estimate_up, 17
global_gearys, 27
global_morans, 27
geos_centerish, 18
greens, 18
get_alarm, 25
get_vest, 26
geo_estimate_down, 19
geo_estimate_up, 20
gp_filter, 21
gp_match, 22
gp_plot, 22
gp_plot_group, 23
gp_sort, 24
gp_trim, 24
geomander (geomander-package), 3
geos_centerish, 18
gp_circle_center, 19
get_alarm, 25
gp_vest, 26
glocal_gearys, 27
glocal_morans, 27
get_vest, 26
gp_circle_center, 19
get_alarm, 25
gp_vest, 26
glocal_gearys, 27
glocal_morans, 27
gstar_i, 28
local_gearys, 29
local_morans, 29
nrcsd, 30
orange, 31
precincts, 31
r2dra, 32
rockland, 33
st_centerish, 37
subtract_edge, 39
suggest_component_connection, 39
suggest_neighbors, 40
towns, 41
va18sub, 41
va_blocks, 42
va_vtd, 43
vest_states, 43