Package ‘geomander’

February 15, 2024

Type Package

Title Geographic Tools for Studying Gerrymandering

Version 2.3.0

Date 2024-02-15

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


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

RoxygenNote 7.2.3

LinkingTo Rcpp (>= 1.0.7)

Imports censable, cli, dataverse, dplyr, geos, ggplot2, magrittr, readr, rlang, Rcpp, sf, stringr, tibble, tidyr, tinytiger

Depends R (>= 3.0.2)

Suggests redist, knitr, rmarkdown, testthat (>= 3.0.0), RcppSimdJson, spelling

LazyData true

Encoding UTF-8

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

NeedsCompilation yes

Author Christopher T. Kenny [aut, cre] (<https://orcid.org/0000-0002-9386-6860>), Cory McCartan [ctb] (<https://orcid.org/0000-0002-6251-669X>)

Maintainer Christopher T. Kenny <christopherkenny@fas.harvard.edu>
Repository  CRAN
Date/Publication  2024-02-15 21:20:02 UTC

R topics documented:

geomander-package .................................................. 3
add_edge .......................................................... 5
adjacency ............................................................ 6
alarm_states .......................................................... 7
baf_to_vtd ............................................................ 7
block2prec .......................................................... 8
block2prec_by_county ................................................ 9
checkerboard .......................................................... 10
checkerboard_adj ..................................................... 10
test_contiguity ..................................................... 11
test_polygon_contiguity ............................................. 12
clean_vest .......................................................... 13
compare_adjacencies ................................................. 13
count_connections .................................................. 14
create_block_table .................................................. 14
create_tract_table ................................................... 15
dra2r ............................................................... 16
estimate_down ....................................................... 17
estimate_up .......................................................... 18
test_centerish ....................................................... 18
test_circle_center ................................................... 19
test_estimate_down .................................................. 20
test_estimate_up ..................................................... 21
test_filter .......................................................... 21
test_match ......................................................... 22
test_plot ........................................................... 24
test_plot_group ..................................................... 24
test_sort ............................................................. 25
test_trim ............................................................. 26
test_alarm ........................................................... 27
test_dra ............................................................. 27
test_heda ............................................................. 28
test_lewis ............................................................ 29
test_rpnearme ........................................................ 29
test_vest ............................................................. 30
global_gearys ....................................................... 31
global_moran ........................................................ 31
gstar ................................................................. 32
heda_states .......................................................... 33
local_gearys .......................................................... 33
local_moran .......................................................... 34
nrcsd ................................................................. 34
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
- adjacency
- alarm_states
- baf_to_vtd
- block2prec
- block2prec_by_county
- check_contiguity
- check_polygon_contiguity
- checkerboard
- checkerboard_adj

Add Edges to an Adjacency List
Build Adjacency List
List Available States from ALARM Data
Estimate Plans from a Block Assignment File to Voting Districts
Aggregate Block Table by Matches
Aggregate Block Table by Matches and County
Check Contiguity by Group
Check Polygon Contiguity
Checkerboard
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_dra  Get Dave's Redistricting App Dataset
get_heda  Get Harvard Election Data Archive ("HEDA") Dataset
get_lewis  Get historical United States Congressional District Shapefiles
get_rpvnearme  Get Racially Polarized Voting Dataset from RPV Near Me
get_vest  Get Voting and Election Science Team ("VEST") Dataset
global_gearys  Compute Global Geary's C
global_morans  Compute Global Moran's I
gstar_i  Compute Standardized Getis Ord G*i
heda_states  List Available States from HEDA Dataverse
local_gearys  Compute Local Geary’s C
local_morans  Compute Local Moran’s I
nrcsd  nrcsd
orange  orange
precincts  precincts
r2dra  R to DRA
regionalize  Estimate Regions by Geographic Features
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
**Description**

Add Edges to an Adjacency List

**Usage**

```r
add_edge(adj, v1, v2, ids = NULL, zero = TRUE)
```

**Arguments**

- **adj**
  - list of adjacent precincts
- **v1**
  - vector of vertex identifiers for the first vertex. Can be an integer index or a value to look up in `ids`, if that argument is provided. If more than one identifier is present, connects each to corresponding entry in `v2`.
- **v2**
  - vector of vertex identifiers for the second vertex. Can be an integer index or a value to look up in `ids`, if that argument is provided. If more than one identifier is present, connects each to corresponding entry in `v2`.
- **ids**
  - A vector of identifiers which is used to look up the row indices for the vertices. If provided, the entries in `v1` and `v2` must match exactly one entry in `ids`.
- **zero**
  - boolean, TRUE if the list is zero indexed. False if one indexed.
Value

adjacency list.

Examples

data(towns)
adj <- adjacency(towns)

add_edge(adj, 2, 3)
add_edge(adj, "West Haverstraw", "Stony Point", towns$MUNI)

adjacency

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

shp sf dataframe
epsg numeric 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**

```r
alarm_states()
```

**Value**

character abbreviations for states

**Examples**

```r
## 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**

```r
baf_to_vtd(baf, plan_name, GEOID = "GEOID", year = 2020)
```

**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'.
- `year` the decade to request, either 2010 or 2020. Default is 2020.

**Details**

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

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

```r
block2prec_by_county(block_table, precinct, precinct_county_fips, epsg = 3857)
```

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 <- quotesingle.Var 087 quotesingle.Var
block <- create_block_table('NY', 'Rockland')
block2prec_by_county(block, towns, 'fips')
## End(Not run)
```
<table>
<thead>
<tr>
<th>checkerboard</th>
<th>Checkerboard</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

```r
data("checkerboard")
```

**Format**

An sf dataframe with 64 observations

**Examples**

```r
data('checkerboard')
```

<table>
<thead>
<tr>
<th>checkerboard_adj</th>
<th>Checkerboard Adjacency</th>
</tr>
</thead>
</table>

**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')
```
**Description**

Identify contiguous sets of units and numbers each set. Can be extended to repeat the procedure within a subgeography.

**Usage**

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**

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.

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 contiguity indicators. Each row is the units of `adj`. Columns include

- `group`: Values of the inputted `group` argument. If `group` is not specified, then all values will be 1.
- `component`: A number for each contiguous set of units within a `group`. If all units within a `group` are contiguous, all values are 1. If there are two sets, each discontiguous with the other, the larger one will be numbered 1 and the smaller one will be numbered as 2.
check_polygon_contiguity

Check Polygon Contiguity

Description

Cast shp to component polygons, build the adjacency, and check the contiguity. Avoids issues where a precinct is actually a multipolygon.

Usage

check_polygon_contiguity(shp, group, epsg = 3857)

Arguments

- **shp**: An sf data frame
- **group**: unquoted name of group identifier in shp. Typically, this is district assignment. If you’re looking for dis-contiguous precincts, this should be a row number.
- **epsg**: numeric EPSG code to planarize to. Default is 3857.

Value

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

Examples

```r
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**
```r
data(va18sub)
ova <- 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.
create_block_table

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)

count_connections
Count Times Precincts are Connected

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
)

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.

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

## 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,
  geometry = TRUE,
  year = 2019,
mem = FALSE,
epsg = 3857
)

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.

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

DRA to R

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
## Not run:
# Needs Census Bureau API
# dra_utah_test is available at https://bit.ly/3c6UDKk
blocklevel <- dra2r('dra_utah_test.csv', state = 'UT')
## End(Not run)
```

```
estimate_down  Estimate Down Levels

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)

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**

```r
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**

```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_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**

```r
geos_centerish(shp, epsg = 3857)
```
**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>shp</strong></td>
<td>An sf dataframe</td>
</tr>
<tr>
<td><strong>epsg</strong></td>
<td>numeric EPSG code to planarize to. Default is 3857.</td>
</tr>
</tbody>
</table>

**Value**

A geos geometry list

**Examples**

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

---

**Description**

Returns the centroid of the largest inscribed circle for each shape

**Usage**

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>shp</strong></td>
<td>An sf dataframe</td>
</tr>
<tr>
<td><strong>tolerance</strong></td>
<td>positive numeric tolerance to simplify by. Default is 0.01.</td>
</tr>
<tr>
<td><strong>epsg</strong></td>
<td>numeric EPSG code to planarize to. Default is 3857.</td>
</tr>
</tbody>
</table>

**Value**

A geos geometry list

**Examples**

```r
data(towns)
geos_circle_center(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

```
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**

```r
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.

**Value**

numeric vector with each value aggregated by group

**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))
geo_estimate_up(from = checkerboard, to = counties, value = checkerboard$i)
```

**geo_filter**

*Filter to Intersecting Pieces*

**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

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

## End(Not run)

data(towns)
data(rockland)
sub <- geo_filter(rockland, towns)
Arguments

from smaller geographic level to match up from
to larger geographic level to be matched to
method string from 'center', 'centroid', 'point', 'circle', or 'area' for matching method
by A character vector to match by. One element if both from and to share the subsetting column name. One element with a name (for from) and one element (for to).
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.

Details

Methods are as follows:

- centroid: matches each element of from to the to entry that the geographic centroid intersects
- center: very similar to centroid, but it matches an arbitrary center point within from if the centroid of from is outside the bounds of from. (This happens for non-convex shapes only).
- point: matches each element of from to the to entry that the "point on surface" intersects.
- circle: matches each element of from to the to entry that the centroid of the maximum inscribed circle intersects
- area: matches each element of from to the to element which has the largest area overlap

Value

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

Examples

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
geo_plot(shp)

Arguments
shp An sf shapefile

Value
ggplot

Examples
data(checkerboard)
geo_plot(checkerboard)

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
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.
path Path to save, only used if save is TRUE. Defaults to working directory.
geo_sort

Value

list of ggplots

Examples

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

data(checkerboard)
geo_sort(checkerboard)
**geo_trim**  

*Trim Away Small Pieces*

**Description**

Trim Away Small Pieces

**Usage**

```r
go_trimm(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.

**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_trimm(block, towns, thresh = 0.05)

## End(Not run)

data(towns)
data(rockland)
sub <- geo_filter(rockland, towns)
rem <- geo_trimm(sub, towns, thresh = 0.05)
```
get_alarm

Get ALARM Dataset

Description

Gets 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, year = 2020, geometry = TRUE, epsg = 3857)

Arguments

- **state**: two letter state abbreviation
- **year**: year to get data for. Either 2020 or 2010
- **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

ak <- get_alarm('AK', geometry = FALSE)

get_dra

Get Dave’s Redistricting App Dataset

Description

Gets a dataset from Dave’s Redistricting App.

Usage

get_dra(state, year = 2020, geometry = TRUE, clean_names = TRUE, epsg = 3857)
get_heda

Arguments

state  two letter state abbreviation
year   year to get data for. Either 2020 or 2010
geometry Default is TRUE. Add geometry to the data?
clean_names Clean names. Default is TRUE. If FALSE, returns default names.
epsg numeric EPSG code to planarize to. Default is 3857.

Details

See the full available data at https://github.com/dra2020/vtd_data.

Value
tibble with election data and optional geometry

Examples

ak <- get_dra('AK', geometry = FALSE)

shp <- get_heda('ND')
**get_lewis**

*Get historical United States Congressional District Shapefiles*

**Description**

Data sourced from the United States Congressional District Shapefiles, primarily hosted at [https://cdmaps.polisci.ucla.edu/](https://cdmaps.polisci.ucla.edu/). Files are fetched through the GitHub repository at [https://github.com/JeffreyBLewis/congressional-district-boundaries](https://github.com/JeffreyBLewis/congressional-district-boundaries).

**Usage**

```r
get_lewis(state, congress)
```

**Arguments**

- `state`: two letter state abbreviation
- `congress`: congress number, from 1 to 114.

**Value**

a sf tibble of the congressional district boundaries

**References**


**Examples**

```r
get_lewis(state = 'NM', congress = 111)
```

---

**get_rpvnearme**

*Get Racially Polarized Voting Dataset from RPV Near Me*

**Description**

Get Racially Polarized Voting Dataset from RPV Near Me

**Usage**

```r
get_rpvnearme(state, version = c(1, 2))
```

**Arguments**

- `state`: the state postal code of the state
- `version`: the version of the data to use. 1 for the original, 2 for the extended.
get_vest

Value

a tibble of precinct-level estimates of votes (party) by race

Examples

get_rpvnearme('DE')

get_vest

Get Voting and Election Science Team ("VEST") Dataset

Description

Get Voting and Election Science Team ("VEST") Dataset

Usage

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

Arguments

state two letter state abbreviation
year year any in 2016-2021
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.
... additional arguments passed to sf::read_sf()

Value

sf tibble

Examples

## 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

Compute Standardized Getis Ord GI

Description

Returns the Getis Ord GI 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 GI scores

Examples

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

*List Available States from HEDA Dataverse*

**Description**

List Available States from HEDA Dataverse

**Usage**

```
heda_states()
```

**Value**

character abbreviations for states

**Examples**

```
heda_states()
```

---

**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(’checkerboard’)
checkerboard <- checkerboard %>% mutate(m = as.numeric((id + i) %% 2 == 0))
local_morans(shp = checkerboard, wts = checkerboard$m)
```

**nrcsd**
*The data contains the North Rockland Central School District.*

**Usage**
```
data(’nrcsd’)
```

**Format**
An sf dataframe with 1 observation

**Examples**
```
data(’nrcsd’)
```
**orange**

---

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

**Usage**
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**
data('orange')

---

**precincts**

---

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

**Usage**
data("precincts")

**Format**
An sf dataframe with 278 observations

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

**Examples**
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 <- tinytiger::tt_congressional_districts() %>% filter(STATEFP == '49')
cnty <- tinytiger::tt_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)
### regionalize

**Estimate Regions by Geographic Features**

**Description**

This offers a basic method for dividing a shape into separate pieces.

**Usage**

```r
regionalize(shp, lines, adj = adjacency(shp), epsg = 3857)
```

**Arguments**

- `shp`: sf tibble to estimate regions for.
- `lines`: sf tibble which divides `shp` into regions.
- `adj`: adjacency graph.
- `epsg`: numeric EPSG code to planarize to. Default is 3857.

**Value**

integer vector of regions with `nrow(shp)` entries.

**Examples**

```r
data(towns)
# make some weird roadlike feature passing through the towns
lines <- sf::st_sfc(sf::st_linestring(sf::st_coordinates(sf::st_centroid(towns))),
                     crs = sf::st_crs(towns))
regionalize(towns, lines)
```

---

### rockland

**rockland**

**Description**

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

**Usage**

```r
data("rockland")
```

**Format**

An sf dataframe with 4764 observations.
seam_adj

Details

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

Examples

```r
data('rockland')
```

seam adj  
*Filter Adjacency to Edges Along Border*

Description

Filter Adjacency to Edges Along Border

Usage

```r
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.

Value

subset of adj

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)

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**

```r
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**

```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)

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**

```r
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**

```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)

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

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)
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

\[
\text{split_precinct}(\text{lower}, \text{precinct}, \text{split}\_\text{by}, \text{lower}\_\text{wt}, \text{split}\_\text{by}\_\text{id}, \text{epsg} = 3857)
\]

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**

```r
st_centerish(shp, epsg = 3857)
```

**Arguments**

- `shp`: An sf dataframe
- `epsg`: 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)
```

---

**st_circle_center**

*Get the centroid of the maximum inscribed circle*

**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`: positive numeric tolerance to simplify by. Default is 0.01.
- `epsg`: 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_circle_center(towns)
```

---

**subtract_edge**

Subtract Edges from an Adjacency List

**Usage**
```r
subtract_edge(adj, v1, v2, ids = NULL, zero = TRUE)
```

**Arguments**
- `adj`: list of adjacent precincts
- `v1`: vector of vertex identifiers for the first vertex. Can be an integer index or a value to look up in `ids`, if that argument is provided. If more than one identifier is present, disconnects each to corresponding entry in `v2`, if an edge exists.
- `v2`: vector of vertex identifiers for the second vertex. Can be an integer index or a value to look up in `ids`, if that argument is provided. If more than one identifier is present, disconnects each to corresponding entry in `v2`, if an edge exists.
- `ids`: A vector of identifiers which is used to look up the row indices for the vertices. If provided, the entries in `v1` and `v2` must match exactly one entry in `ids`.
- `zero`: boolean, TRUE if `adj` is zero indexed. False if one indexed.

**Value**
adjacency list.

**Examples**
```r
data(towns)
adj <- adjacency(towns)
subtract_edge(adj, 2, 3)
subtract_edge(adj, "West Haverstraw", "Stony Point", towns$MUNI)
```
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)

Arguments

- **shp**: An sf data frame
- **adj**: adjacency list
- **group**: array of group identifiers. Typically district numbers or county names. Defaults to rep(1, length(adj)) if missing.
- **epsg**: numeric EPSG code to planarize to. Default is 3857.

Value

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

Examples

```r
library(dplyr)
data(checkerboard)
checkerboard <- checkerboard %>% filter(i != 1, j != 1)
adj <- adjacency(checkerboard)
suggest_component_connection(checkerboard, adj)
```

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
glimpse(towns)
```

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

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('va18sub')

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')
vest_states

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 <- tinytiger::tt_voting_districts(state = 'VA', county = '087', year = 2010) 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 year in 2016, 2018, or 2020

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, 43
  st_circle_center, 43

* datasets
  alarm_states, 7
  clean_vest, 13
  get_alarm, 27
  get_dra, 27
  get_heda, 28
  get_vest, 30
  heda_states, 33
  vest_states, 48

* datatable
  block2prec, 8
  block2prec_by_county, 9
  create_block_table, 14
  create_tract_table, 15
  geo_filter, 21
  geo_trim, 26

* data
  checkerboard, 10
  checkerboard_adj, 10
  nrcsd, 34
  orange, 35
  precincts, 35
  rockland, 37
  towns, 46
  va18sub, 47
  va_blocks, 47
  va_vtd, 48

* dra
  dra2r, 16
  r2dra, 36

* estimate
  estimate_down, 17
  estimate_up, 18
  geo_estimate_down, 20
  geo_estimate_up, 21
  geo_match, 22

* fix
  add_edge, 5
  adjacency, 6
  check_contiguity, 11
  check_polygon_contiguity, 12
  compare_adjacencies, 13
  geo_sort, 25
  split_precinct, 42
  subtract_edge, 44
  suggest_component_connection, 45
  suggest_neighbors, 45

* leftover
  count_connections, 14

* package
  geomander-package, 3

* plot
  geo_plot, 24
  geo_plot_group, 24

* seam
  seam_adj, 38
  seam_geom, 39
  seam_rip, 40
  seam_sew, 41

* spatcorr
  global_gearys, 31
  global_morans, 31
  gstar_i, 32
  local_gearys, 33
  local_morans, 34

  add_edge, 5
  adjacency, 6
  alarm_states, 7

  baf_to_vtd, 7
  block2prec, 8
  block2prec_by_county, 9
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccm (check_contiguity)</td>
<td>11</td>
</tr>
<tr>
<td>cct (check_contiguity)</td>
<td>11</td>
</tr>
<tr>
<td>check_contiguity</td>
<td>11</td>
</tr>
<tr>
<td>check_polygon_contiguity</td>
<td>12</td>
</tr>
<tr>
<td>checkerboard</td>
<td>10</td>
</tr>
<tr>
<td>checkerboard_adj</td>
<td>10</td>
</tr>
<tr>
<td>clean_vest</td>
<td>13</td>
</tr>
<tr>
<td>compare_adjacencies</td>
<td>13</td>
</tr>
<tr>
<td>count_connections</td>
<td>14</td>
</tr>
<tr>
<td>create_block_table</td>
<td>14</td>
</tr>
<tr>
<td>create_tract_table</td>
<td>15</td>
</tr>
<tr>
<td>dra2r</td>
<td>16</td>
</tr>
<tr>
<td>estimate_down</td>
<td>17</td>
</tr>
<tr>
<td>estimate_up</td>
<td>18</td>
</tr>
<tr>
<td>geo_estimate_down</td>
<td>20</td>
</tr>
<tr>
<td>geo_estimate_up</td>
<td>21</td>
</tr>
<tr>
<td>geo_filter</td>
<td>21</td>
</tr>
<tr>
<td>geo_match</td>
<td>22</td>
</tr>
<tr>
<td>geo_plot</td>
<td>24</td>
</tr>
<tr>
<td>geo_plot_group</td>
<td>24</td>
</tr>
<tr>
<td>geo_sort</td>
<td>25</td>
</tr>
<tr>
<td>geo_trim</td>
<td>26</td>
</tr>
<tr>
<td>geomander (geomander-package)</td>
<td>3</td>
</tr>
<tr>
<td>geos_centerish</td>
<td>18</td>
</tr>
<tr>
<td>geos_circle_center</td>
<td>19</td>
</tr>
<tr>
<td>get_alarm</td>
<td>27</td>
</tr>
<tr>
<td>get_dra</td>
<td>27</td>
</tr>
<tr>
<td>get_heda</td>
<td>28</td>
</tr>
<tr>
<td>get_lewis</td>
<td>29</td>
</tr>
<tr>
<td>get_rpvnearme</td>
<td>29</td>
</tr>
<tr>
<td>get_vest</td>
<td>30</td>
</tr>
<tr>
<td>global_gearys</td>
<td>31</td>
</tr>
<tr>
<td>global_morans</td>
<td>31</td>
</tr>
<tr>
<td>gstar_i</td>
<td>32</td>
</tr>
<tr>
<td>heda_states</td>
<td>33</td>
</tr>
<tr>
<td>local_gearys</td>
<td>33</td>
</tr>
<tr>
<td>local_morans</td>
<td>34</td>
</tr>
<tr>
<td>nrcsd</td>
<td>34</td>
</tr>
<tr>
<td>orange</td>
<td>35</td>
</tr>
<tr>
<td>precincts</td>
<td>35</td>
</tr>
<tr>
<td>r2dra</td>
<td>36</td>
</tr>
<tr>
<td>regionalize</td>
<td>37</td>
</tr>
<tr>
<td>rockland</td>
<td>37</td>
</tr>
<tr>
<td>seam_adj</td>
<td>38</td>
</tr>
<tr>
<td>seam_geom</td>
<td>39</td>
</tr>
<tr>
<td>seam_rip</td>
<td>40</td>
</tr>
<tr>
<td>seam_sew</td>
<td>41</td>
</tr>
<tr>
<td>sf::read_sf</td>
<td>28, 30</td>
</tr>
<tr>
<td>split_precinct</td>
<td>42</td>
</tr>
<tr>
<td>st_centerish</td>
<td>43</td>
</tr>
<tr>
<td>st_circle_center</td>
<td>43</td>
</tr>
<tr>
<td>subtract_edge</td>
<td>44</td>
</tr>
<tr>
<td>suggest_component_connection</td>
<td>45</td>
</tr>
<tr>
<td>suggest_neighbors</td>
<td>45</td>
</tr>
<tr>
<td>towns</td>
<td>46</td>
</tr>
<tr>
<td>va18sub</td>
<td>47</td>
</tr>
<tr>
<td>va_blocks</td>
<td>47</td>
</tr>
<tr>
<td>va_vtd</td>
<td>48</td>
</tr>
<tr>
<td>vest_states</td>
<td>48</td>
</tr>
</tbody>
</table>