

Package: geomander (via r-universe)

September 21, 2026

Type Package

Title Geographic Tools for Studying Gerrymandering

Version 2.5.3

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://christophertkenny.com/geomander/>,
<https://github.com/christopherkenny/geomander>

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

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

LinkingTo Rcpp (>= 1.0.7)

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

Depends R (>= 4.1.0)

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

LazyData true

Encoding UTF-8

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website christopherkenny/ctktemplate

Language en-US

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev

Repository <https://christopherkenny.r-universe.dev>

Date/Publication 2026-07-19 19:49:34 UTC

RemoteUrl <https://github.com/christopherkenny/geomander>

RemoteRef HEAD

RemoteSha 514e362ebbb3416c7bb224969e17b8b581986b4b

Contents

geomander-package	3
add_edge	4
adjacency	5
alarm_states	5
baf_to_vtd	6
block2prec	7
block2prec_by_county	8
check_contiguity	8
check_polygon_contiguity	10
checkerboard	10
checkerboard_adj	11
clean_vest	11
compare_adjacencies	12
count_connections	12
create_block_table	13
create_tract_table	14
dra2r	15
estimate_down	16
estimate_up	16
geo_estimate_down	17
geo_estimate_up	18
geo_filter	19
geo_match	20
geo_plot	21
geo_plot_group	22
geo_sort	22
geo_trim	23
geos_centerish	24
geos_circle_center	24
get_alarm	25
get_dra	26
get_heda	26
get_lewis	27
get_rpvnearme	28
get_vest	29
global_gearys	29
global_morans	30
gstar_i	31
heda_states	32
local_gearys	32

local_morans	33
nrcsd	34
orange	34
precincts	35
r2dra	35
regionalize	36
rockland	37
seam_adj	37
seam_geom	38
seam_rip	39
seam_sew	40
split_precinct	41
st_centerish	42
st_circle_center	42
subtract_edge	43
suggest_component_connection	44
suggest_neighbors	44
towns	45
va_blocks	46
va_vtd	46
va18sub	47
vest_states	47
Index	49

geomander-package	<i>Geographic Tools for Studying Gerrymandering</i>
-------------------	---

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: This package was not yet installed at build time.

Maintainer

Christopher T. Kenny <ctkenny@proton.me>

Author(s)

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

add_edge	<i>Add Edges to an Adjacency List</i>
----------	---------------------------------------

Description

Add one or more undirected edges to an adjacency list in place and return the modified list.

Usage

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

Arguments

adj	List of neighbors, one entry per vertex.
v1	vector of vertex identifiers for the first vertex. Can be an integer index or a value to look up in <code>ids</code> , if that argument is provided. If more than one identifier is present, connects each to corresponding entry in <code>v2</code> .
v2	vector of vertex identifiers for the second vertex. Can be an integer index or a value to look up in <code>ids</code> , if that argument is provided. If more than one identifier is present, connects each to corresponding entry in <code>v2</code> .
ids	Optional identifier vector used to look up row indices. If provided, each entry in <code>v1</code> and <code>v2</code> must match exactly one entry.
zero	Logical. TRUE when the adjacency list stores zero-based neighbor indices and FALSE when it stores one-based indices.

Value

Adjacency list with the requested edges added symmetrically.

Examples

```
data(towns)
adj <- adjacency(towns)

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

adjacency	<i>Build an Adjacency List</i>
-----------	--------------------------------

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

<code>shp</code>	<code>sf</code> dataframe.
<code>epsg</code>	numeric EPSG code to planarize to. Default is 3857.

Value

Zero-indexed adjacency list with `nrow(shp)` entries.

Examples

```
data(precincts)
adj <- adjacency(precincts)
```

alarm_states	<i>List Available States from ALARM Data</i>
--------------	--

Description

List Available States from ALARM Data

Usage

```
alarm_states()
```

Value

character vector of state abbreviations inferred from the ALARM repository contents.

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

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

Arguments

baf	dataframe representing a block assignment file.
plan_name	Name of the column in baf containing district assignments.
GEOID	Name of the column containing block GEOIDs, sometimes called "BLOCKID". Defaults to "GEOID".
year	Decennial vintage for the crosswalk, either 2010 or 2020.

Details

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

When a voting district contains blocks assigned to more than one district, the output uses the modal district assignment.

Value

tibble with one row per voting district and the requested plan column

Examples

```
# Not guaranteed to reach download from redistrict2020.org
## Not run:
# download and read baf ----
url <- paste0('https://github.com/PlanScore/Redistrict2020/',
              'raw/main/files/DE-2021-01/DE_SLDU_bef.zip')
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 a Block Table by Matches

Description

Aggregate block-level attributes up to a larger geography, usually precincts, using a vector of group assignments such as the output of [geo_match\(\)](#).

Usage

```
block2prec(block_table, matches, geometry = FALSE)
```

Arguments

block_table	Block table, usually from create_block_table() .
matches	Integer grouping variable, typically from geo_match() .
geometry	Logical. If TRUE, union geometry within each matched group.

Details

If matches carries a "matching_max" attribute, the output preserves all target groups up to that size, even when some groups receive no matched rows. Missing nonnegative numeric summaries are filled with 0.

Value

dataframe with one row per matched group. When empty target groups are preserved, the number of rows may be larger than `length(unique(matches))`.

Examples

```

set.seed(1)
data(rockland)
rockland$id <- sample(c(1:2, 4), nrow(rockland), TRUE)
block2prec(rockland, rockland$id)

```

block2prec_by_county *Aggregate a Block Table by Matches Within County*

Description

Variant of `block2prec()` that matches blocks to precincts separately within each county. This helps avoid cross-county mismatches when county boundaries and census blocks do not line up cleanly.

Usage

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

Arguments

`block_table` Block table, usually from `create_block_table()`.
`precinct` sf object of target precinct shapes.
`precinct_county_fips` Name of the county identifier column in precinct.
`epsg` numeric EPSG code to planarize to. Default is 3857.

Value

dataframe with one row per row of precinct, ordered to align with the input precinct object.

Examples

```
## 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)
```

check_contiguity *Check Contiguity by Group*

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	Zero-indexed adjacency list.
group	Optional vector of group identifiers, typically district numbers or county names. If omitted, all rows are treated as belonging to one group.

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.
- group_number Integer encoding of group.
- 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 discontinuous with the other, the larger one will be numbered 1 and the smaller one will be numbered 2.

Examples

```
data(checkerboard)
adj <- adjacency(checkerboard)
# These each indicate the graph is connected.
check_contiguity(adj) # all contiguous
# If there are two discontinuous groups, there will be 2 values of `component`
cct(adj)
ccm(adj)
```

```
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 dataframe.
group	Unquoted column name in shp giving the grouping variable. Use a row identifier when checking whether individual rows are multipart.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

tibble in the same format as [check_contiguity\(\)](#).

Examples

```
data(checkerboard)
check_polygon_contiguity(checkerboard, i)
```

checkerboard	<i>Checkerboard</i>
--------------	---------------------

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	<i>Checkerboard Adjacency</i>
------------------	-------------------------------

Description

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

Usage

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

Format

A list with 64 entries

Examples

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

clean_vest	<i>Clean VEST Column Names</i>
------------	--------------------------------

Description

Rename VEST election-result columns into a more uniform office_year_party_candidate style.

Usage

```
clean_vest(data)
```

Arguments

data sf tibble from `get_vest()`.

Value

input object with renamed columns

Examples

```
data(va18sub)
va <- clean_vest(va18sub)
```

compare_adjacencies	<i>Compare Adjacency Lists</i>
---------------------	--------------------------------

Description

Compare two adjacency lists and report edges that differ between them.

Usage

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

Arguments

adj1	First adjacency list.
adj2	Second adjacency list.
shp	Optional sf object used to compute DE-9IM relations for differing pairs.
zero	Logical. TRUE when the adjacency lists are zero-indexed.

Value

tibble of differing edge pairs. When shp is supplied, includes DE-9IM relation and simple intersection class diagnostics.

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	<i>Count How Often Pairs of Units Share a District</i>
-------------------	--

Description

Summarize a district-membership matrix into pairwise co-assignment counts.

Usage

```
count_connections(dm, normalize = FALSE)
```

Arguments

dm	District membership matrix, typically with one row per unit and one column per plan or draw.
normalize	Logical. If TRUE, divide counts by the number of columns in dm.

Value

tibble in long form with columns x, y, and fill, where fill stores the count or proportion of shared assignments.

Examples

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

create_block_table	<i>Create a Census Block-Level Table</i>
--------------------	--

Description

Download block-level decennial census data with the standard redistricting variables used throughout the package.

Usage

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

Arguments

state	Two-letter state postal code.
county	Optional county name or code. If omitted, returns blocks for the entire state.
geometry	Logical. If TRUE, include geometry.
year	Census year. Intended for decennial years, especially 2010 and 2020.
mem	Logical. If TRUE, use the memoized censable backend.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

dataframe or sf object with one row per block. Includes population and voting-age-population summaries by race and ethnicity.

Examples

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

## End(Not run)
```

create_tract_table	<i>Create an ACS Tract-Level Table</i>
--------------------	--

Description

Download tract-level ACS data with the standard redistricting variables used throughout the package.

Usage

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

Arguments

state	Two-letter state postal code.
county	Optional county name or code. If omitted, returns tracts for the entire state.
geometry	Logical. If TRUE, include geometry.
year	ACS year, currently intended for 2009 through 2019.
mem	Logical. If TRUE, use the memoized censable backend.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

dataframe or sf object with one row per tract. Includes population, voting-age population, and citizen voting-age population summaries by race and ethnicity.

Examples

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

## End(Not run)
```

dra2r

*Convert DRA Export Data to an R Spatial Object***Description**

Read a DRA block assignment export and attach it to 2020 census blocks, optionally collapsing the assignments to a precinct layer.

Usage

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

Arguments

dra	Path to a DRA CSV export, or a dataframe containing columns GEOID20 and District.
state	State postal abbreviation used to fetch 2020 blocks.
precincts	Optional sf object of precinct shapes. If supplied, the function assigns each precinct the most common block-level district among the blocks matched to it.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

sf object at the block level when precincts is omitted, or at the precinct level when precincts is supplied.

Examples

```
## 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	<i>Estimate Values Down Using Precomputed Matches</i>
---------------	---

Description

Non-geographic companion to [geo_estimate_down\(\)](#). Use this when you already have a vector of matches and want to avoid recomputing them.

Usage

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

Arguments

wts	Numeric vector of weights. Defaults to 1.
value	Numeric vector of values on the larger geography. Defaults to 1.
group	Integer vector of matches of length <code>length(wts)</code> . Each entry should give the row of value that the corresponding lower-level unit maps to, often from geo_match() .

Value

numeric vector of length `length(group)` with values 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))
matches <- geo_match(checkerboard, counties)
estimate_down(wts = rep(1, nrow(checkerboard)), value = counties$pop, group = matches)
```

estimate_up	<i>Aggregate Values Up Using Precomputed Matches</i>
-------------	--

Description

Non-geographic companion to [geo_estimate_up\(\)](#). Use this when you already have a vector of group assignments.

Usage

```
estimate_up(value, group)
```

Arguments

value	Numeric vector to aggregate. Defaults to 1.
group	Integer vector of length length(value) giving the destination row for each value, often from geo_match() .

Value

numeric vector of length max(group) with values 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)
```

geo_estimate_down	<i>Estimate Values Down to a Finer Geography</i>
-------------------	--

Description

Split values observed on a larger geography across a smaller geography after first matching each row of to to a row of from. This is commonly used to distribute precinct-level election totals to blocks.

Usage

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

Arguments

from	Larger geography level containing the observed values.
to	Smaller geography level to estimate onto.
wts	Numeric vector of length nrow(to). Used to allocate each matched value across rows of to. Defaults to 1, which splits evenly within each matched group.
value	Numeric vector of length nrow(from) containing the values to split downward. Defaults to 1.
method	Matching method passed to geo_match() .
epsg	numeric EPSG code to planarize to. Default is 3857.

Details

If all weights for a matched group are zero, the function falls back to equal allocation within that group. Rows in to that do not match any row of from receive 0.

Value

numeric vector of length nrow(to) with estimated values

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

Aggregate Values Up to a Larger Geography

Description

Aggregate values from a smaller geography to a larger geography after matching each row of from to a row of to. This is commonly used to roll block-level counts up to precincts or districts.

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) to aggregate. Defaults to 1.
method	Matching method passed to geo_match() .
epsg	numeric EPSG code to planarize to. Default is 3857.

Details

Groups in to with no matched rows are included in the output and receive 0.

Value

numeric vector of length nrow(to) with values 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))
geo_estimate_up(from = checkerboard, to = counties, value = checkerboard$i)
```

geo_filter	<i>Filter to Intersecting Pieces</i>
------------	--------------------------------------

Description

Keep only the rows of `from` whose geometry intersects any geometry in `to`.

Usage

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

Arguments

<code>from</code>	<code>sf</code> object to subset.
<code>to</code>	<code>sf</code> object used as the target geography.
<code>bool</code>	Logical. If TRUE, return only the logical keep vector.
<code>epsg</code>	numeric EPSG code to planarize to. Default is 3857.

Value

subset of `from`, or a logical vector when `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)
```

 geo_match

Match Features Across Geographic Layers

Description

Match each row of `from` to one row of `to`, typically when `from` is a finer geography nested inside `to`. The result is an integer vector of row indices in `to` and can be reused by downstream helpers such as `estimate_up()`, `estimate_down()`, and `block2prec()`.

Usage

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

Arguments

<code>from</code>	An sf object to match from, usually the smaller geography.
<code>to</code>	An sf object to match to, usually the larger geography.
<code>method</code>	Matching method. One of "center", "centroid", "point", "circle", or "area".
<code>by</code>	Optional character scalar restricting matching within shared groups. Use a single value when <code>from</code> and <code>to</code> use the same column name, or a named scalar such as <code>c(county_from = "county_to")</code> when the column names differ.
<code>tiebreaker</code>	Logical. If TRUE, ambiguous or unmatched features are assigned to the nearest feature in <code>to</code> . If FALSE, unmatched rows receive -1 and rows with multiple matches receive -2.
<code>epsg</code>	numeric EPSG code to planarize to. Default is 3857.

Details

Methods are as follows:

- "centroid": match using the geometric centroid of each row in `from`.
- "center": use the centroid when it lies inside the polygon and otherwise fall back to a point-on-surface.
- "point": use point on surface for each row in `from`.
- "circle": use the centroid of the maximum inscribed circle.
- "area": match to the row in `to` with the largest area overlap.

When the output does not reference every row of `to`, an attribute `"matching_max"` is attached and records `nrow(to)`. Functions such as `block2prec()` use that attribute to preserve empty target groups.

Value

integer vector of length `nrow(from)`. Values are row indices in `to`, or `-1 / -2` when `tiebreaker = FALSE`. Empty geometries are returned as `NA`.

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

Plot an sf Object with Row Numbers

Description

Quick diagnostic plot that labels each row with its row number.

Usage

```
geo_plot(shp)
```

Arguments

`shp` An sf object.

Value

ggplot object

Examples

```
data(checkerboard)
geo_plot(checkerboard)
```

geo_plot_group	<i>Plot Groups with Connected Components Colored</i>
----------------	--

Description

Plot Groups with Connected Components Colored

Usage

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

Arguments

shp	An sf object.
adj	Adjacency list.
group	Optional vector of group identifiers, typically district numbers or county names.
save	Logical. If TRUE, save each plot to disk.
path	Directory prefix used when save = TRUE.

Value

list of ggplot objects, one per unique group

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	<i>Sort Features by Northwest-to-Southeast Order</i>
----------	--

Description

Reorder rows by the centroid distance from the northwest corner of the overall bounding box.

Usage

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

Arguments

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

Value

reordered sf dataframe

Examples

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

geo_trim	<i>Trim Away Small Pieces</i>
----------	-------------------------------

Description

Keep only rows of from whose intersection with to covers more than a given share of the row's area.

Usage

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

Arguments

from sf object to subset.
 to sf object used as the target geography.
 thresh Minimum retained overlap as a proportion of each row's area.
 bool Logical. If TRUE, return only the logical keep vector.
 epsg numeric EPSG code to planarize to. Default is 3857.

Value

subset of from, or a logical vector when bool = TRUE

Examples

```
## 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)
```

geos_centerish	<i>Get the kind of center of each shape</i>
----------------	---

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

geos geometry vector of points.

Examples

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

geos_circle_center	<i>Get the centroid of the maximum inscribed circle</i>
--------------------	---

Description

Returns the centroid of the largest inscribed circle for each shape

Usage

```
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

geos geometry vector of points.

Examples

```
data(towns)
geos_circle_center(towns)
```

get_alarm	<i>Get an ALARM Dataset</i>
-----------	-----------------------------

Description

Download a state dataset from the Algorithm-Assisted Redistricting Methodology Project. Depending on state and year, the data are returned at either the voting-district level or the census-block level. The current supported data is the 2020 and 2010 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 retrieve, either 2020 or 2010.
geometry	Logical. If TRUE, join the corresponding geometry.
epsg	numeric EPSG code to planarize to. Default is 3857.

Details

See the full available data at <https://github.com/alarm-redist/census-2020>.

Value

tibble or sf object with ALARM election data and, optionally, geometry.

Examples

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

get_dra	<i>Get a Dave's Redistricting App Dataset</i>
---------	---

Description

Download a state election dataset from the DRA public repository. Depending on state and year, the geometry may be joined at the VTD or block-group level.

Usage

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

Arguments

state	Two-letter state abbreviation.
year	Year to retrieve, either 2020 or 2010.
geometry	Logical. If TRUE, join the corresponding geometry.
clean_names	Logical. If TRUE, rename election columns into a more consistent scheme.
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 or sf object with DRA election data and, optionally, geometry.

Examples

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

get_heda	<i>Get a Harvard Election Data Archive ("HEDA") Dataset</i>
----------	---

Description

Download a state file from the Harvard Election Data Archive and return it as an sf object. Some states require special handling because the source files differ in format or level of geography.

Usage

```
get_heda(state, path = tempdir(), epsg = 3857, ...)
```

Arguments

state	Two-letter state abbreviation.
path	Directory used for extracted files. Defaults to <code>tempdir()</code> .
epsg	numeric EPSG code to planarize to. Default is 3857.
...	Additional arguments passed to <code>sf::read_sf()</code> .

Details

Most states are distributed as zipped shapefiles. California is stored differently and is aggregated here to 2010 tracts. Some states are shipped as loose shapefile components rather than zip archives. If the source file has no CRS metadata, the function assumes EPSG 4140 before optionally reprojecting.

The function currently returns the raw source column names because internal name cleaning is disabled in the implementation.

Value

sf tibble containing the requested HEDA dataset

Examples

```
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/>. Files are fetched through the GitHub repository at <https://github.com/JeffreyBLewis/congressional-district-boundaries>.

Usage

```
get_lewis(state, congress, path_only = FALSE)
```

Arguments

state	State name, FIPS, or two-letter abbreviation understood by censable.
congress	Congress number to retrieve.
path_only	Logical. If TRUE, return only the selected download URL.

Value

sf tibble of congressional district boundaries, or a character URL when `path_only = TRUE`.

References

Jeffrey B. Lewis, Brandon DeVine, Lincoln Pitcher, and Kenneth C. Martis. (2013) Digital Boundary Definitions of United States Congressional Districts, 1789-2012. [Data file and code book]. Retrieved from <https://cdmaps.polisci.ucla.edu> on [date of download].

Examples

```
path <- get_lewis(state = 'NM', congress = 111, path_only = TRUE)
if (attr(curlGetHeaders(path), 'status') == 200) {
  get_lewis(state = 'NM', congress = 111)
}
```

get_rpvnearme	<i>Get the RPV Near Me Dataset</i>
---------------	------------------------------------

Description

Download precinct-level racially polarized voting estimates from the RPV Near Me repository.

Usage

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

Arguments

state	State postal code.
version	Dataset version. Use 1 for the original release or 2 for the extended release.

Value

tibble of precinct-level estimates of party vote by race

Examples

```
get_rpvnearme('DE')
```

get_vest

*Get a Voting and Election Science Team ("VEST") Dataset***Description**

Download a state-year VEST precinct file from Harvard Dataverse.

Usage

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

Arguments

state	Two-letter state abbreviation.
year	Election year, currently one of 2016 through 2021.
path	Directory used for extracted files. Defaults to tempdir().
clean_names	Logical. If TRUE, rename election columns into a more consistent scheme.
epsg	numeric EPSG code to planarize to. Default is 3857.
...	Additional arguments passed to sf::read_sf() .

Value

sf tibble containing the requested VEST dataset

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 dataframe, 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 dataframe. Optional if adj or spatial_mat is supplied.
adj	Zero-indexed adjacency list. Optional if shp or spatial_mat is supplied.
wt	Numeric vector of observed values.
spatial_mat	Square spatial weights matrix. Optional if shp or adj is supplied.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

numeric scalar containing the global Geary's C statistic

Examples

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

global_morans	<i>Compute Global Moran's I</i>
---------------	---------------------------------

Description

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

Usage

```
global_morans(shp, adj, wt, spatial_mat, epsg = 3857)
```

Arguments

shp	sf dataframe. Optional if adj or spatial_mat is supplied.
adj	Zero-indexed adjacency list. Optional if shp or spatial_mat is supplied.
wt	Numeric vector of observed values.
spatial_mat	Square spatial weights matrix. Optional if shp or adj is supplied.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

named list returned by the underlying C++ implementation, including the global statistic and its moments.

Examples

```
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 G*i***Description**

Returns standardized local Getis-Ord G*i scores using either an sf object, a zero-indexed adjacency list, or a spatial weights matrix.

Usage

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

Arguments

shp	sf dataframe. Optional if adj or spatial_mat is supplied.
adj	Zero-indexed adjacency list. Optional if shp or spatial_mat is supplied.
wts	Numeric vector of observed values.
spatial_mat	Square spatial weights matrix. Optional if shp or adj is supplied.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

numeric vector of standardized G*i scores, one per observation

Examples

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

heda_states	<i>List Available HEDA States</i>
-------------	-----------------------------------

Description

List Available HEDA States

Usage

```
heda_states()
```

Value

character vector of lowercase state abbreviations with packaged HEDA support.

Examples

```
heda_states()
```

local_gearys	<i>Compute Local Geary's C</i>
--------------	--------------------------------

Description

Compute Local Geary's C for each observation using either an sf object, a zero-indexed adjacency list, or a spatial weights matrix.

Usage

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

Arguments

shp	sf dataframe. Optional if adj or spatial_mat is supplied.
adj	Zero-indexed adjacency list. Optional if shp or spatial_mat is supplied.
wts	Numeric vector of observed values.
spatial_mat	Square spatial weights matrix. Optional if shp or adj is supplied.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

tibble with one column, geary, and one row per observation

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 for each observation using either an sf object, a zero-indexed adjacency list, or a spatial weights matrix.

Usage

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

Arguments

shp	sf dataframe. Optional if adj or spatial_mat is supplied.
adj	Zero-indexed adjacency list. Optional if shp or spatial_mat is supplied.
wts	Numeric vector of observed values.
spatial_mat	Square spatial weights matrix. Optional if shp or adj is supplied.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

tibble with columns moran, expectation, and variance

Examples

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

nrcsd	<i>nrcsd</i>
-------	--------------

Description

The data contains the North Rockland Central School District.

Usage

```
data('nrcsd')
```

Format

An sf dataframe with 1 observation

Examples

```
data('nrcsd')
```

orange	<i>orange</i>
--------	---------------

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: `orange <- create_block_table('NY', 'Orange')` `orange <- rmapshaper::ms_simplify(orange, keep_shapes = TRUE)`

Examples

```
data('orange')
```

```
precincts
```

```
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')
```

```
r2dra
```

```
Convert an R Plan to DRA Block Assignment Format
```

Description

Project a plan defined on precincts down to 2020 census blocks and return the two-column format expected by DRA exports.

Usage

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

Arguments

<code>precincts</code>	sf object of precinct geometries.
<code>plan</code>	Either a vector of district assignments aligned with <code>precincts</code> , or the name of a column in <code>precincts</code> containing those assignments.
<code>state</code>	State postal abbreviation used to fetch 2020 blocks.
<code>path</code>	Optional output path. If supplied, the result is written as CSV and also returned.
<code>epsg</code>	numeric EPSG code to planarize to. Default is 3857.

Value

tibble with columns GEOID20 and District, suitable for writing to a DRA-style CSV.

Examples

```
## 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 Separated by Geographic Features

Description

Divide an adjacency graph into regions by removing edges whose centerlines intersect a set of line features, such as roads or rivers.

Usage

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

Arguments

shp	sf object to regionalize.
lines	sf line object representing separators.
adj	Optional zero-indexed adjacency graph. Defaults to adjacency(shp).
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

integer vector of region labels with nrow(shp) entries

Examples

```
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	<i>rockland</i>
----------	-----------------

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 <- rmapshaper::ms_simplify(rockland, keep_shapes = TRUE)`

Examples

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

seam_adj	<i>Filter Adjacency to the Edges Along a Seam</i>
----------	---

Description

Keep only the adjacency edges that connect the two sides of a selected seam.

Usage

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

Arguments

adj	Zero-indexed adjacency graph.
shp	sf object containing the administrative identifier.
admin	Name of the administrative-unit column in shp.
seam	Length-2 vector of administrative-unit values defining the seam.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

adjacency list containing only cross-seam edges

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	<i>Filter Shapes to Units Along a Seam</i>
-----------	--

Description

Keep only rows in shp that lie on the border between two administrative units and have at least one adjacency connection across that border.

Usage

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

Arguments

- adj Zero-indexed adjacency graph.
- shp sf object containing the administrative identifier.
- admin Name of the administrative-unit column in shp.
- seam Length-2 vector of administrative-unit values defining the seam.
- epsg numeric EPSG code to planarize to. Default is 3857.

Value

subset of shp containing only rows on the selected seam

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

```

seam_rip

*Remove Edges Across an Administrative Boundary***Description**

Remove adjacency connections between two or more administrative units, such as counties on opposite sides of a seam.

Usage

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

Arguments

adj	Zero-indexed adjacency graph.
shp	sf object containing the administrative identifier.
admin	Name of the administrative-unit column in shp.
seam	Vector of administrative-unit values to separate.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

adjacency list with all cross-seam edges removed

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 Sew a Seam

Description

Build a set of candidate cross-seam edges by constructing Voronoi cells from representative points and identifying neighboring cells across the seam.

Usage

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

Arguments

shp	sf object containing the administrative identifier.
admin	Name of the administrative-unit column in shp.
seam	Length-2 vector of administrative-unit values defining the seam.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

tibble with columns v1 and v2, suitable for use with [add_edge\(\)](#)

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)

```

split_precinct	<i>Split a Precinct</i>
----------------	-------------------------

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	Lower geography that makes up the precinct, often blocks.
precinct	Single-row sf object giving the precinct to split.
split_by	Geography that defines the pieces to split the precinct into.
lower_wt	Optional numeric vector of length nrow(lower) to aggregate within the split pieces, such as population or VAP.
split_by_id	Optional column name in split_by to copy onto the output.
epsg	numeric EPSG code to planarize to. Default is 3857.

Details

The function first filters lower and split_by to the selected precinct, matches lower-level units to split_by, and unions matched pieces. When lower_wt is supplied, a wt column is added with summed weights.

Value

sf dataframe with one row per observed split piece

Examples

```
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	<i>Get the kind of center of each shape</i>
--------------	---

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	numeric EPSG code to planarize to. Default is 3857.

Value

sf dataframe with the same rows and attributes as shp, but point geometries.

Examples

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

st_circle_center	<i>Get the Center of the Maximum Inscribed Circle</i>
------------------	---

Description

Compute the centroid of the largest inscribed circle for each feature.

Usage

```
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

sf dataframe with the same rows and attributes as shp, but point geometries.

Examples

```
data(towns)
st_circle_center(towns)
```

subtract_edge	<i>Remove Edges from an Adjacency List</i>
---------------	--

Description

Remove one or more undirected edges from an adjacency list and return the modified list.

Usage

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

Arguments

adj	List of neighbors, one entry per vertex.
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	Optional identifier vector used to look up row indices. If provided, each entry in v1 and v2 must match exactly one entry.
zero	Logical. TRUE when the adjacency list stores zero-based neighbor indices and FALSE when it stores one-based indices.

Value

Adjacency list with the requested edges removed symmetrically.

Examples

```
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

Suggest nearest cross-component links that could reconnect disconnected groups.

Usage

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

Arguments

shp	An sf dataframe.
adj	Adjacency list.
group	Optional vector of group identifiers. If omitted, all rows are treated as belonging to one group.
epsg	numeric EPSG code to planarize to. Default is 3857.

Value

tibble with columns x and y, giving candidate row pairs to connect with [add_edge\(\)](#).

Examples

```
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	sf object used to compute representative points.
adj	Adjacency list.
idx	Optional integer vector of row indices to repair. If omitted, the function uses rows with no neighbors.
neighbors	Number of candidate neighbors to return for each row in idx.

Value

tibble with columns x and y, giving pairs of row indices that could be connected with [add_edge\(\)](#).

Examples

```
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)
```

towns	<i>towns</i>
-------	--------------

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')
```

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	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 <- tinytiger::tt_voting_districts(state = 'VA', county = '087', year = 2010) va_vtd <- rmapshaper::ms_simplify(va_vtd, keep_shapes = TRUE)
```

Examples

```
data('va_blocks')
```

`va18sub`*va18sub*

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

Voting and Election Science Team, 2019, "va_2018.zip", 2 018 Precinct-Level Election Results, <https://doi.org/10.7910/DVN/UBKYRU/FQDLOO>, Harvard Dataverse, V4

Examples

```
data('va18sub')
```

`vest_states`*List Available VEST States*

Description

List Available VEST States

Usage

```
vest_states(year)
```

Arguments

`year` Year to inspect in the VEST Dataverse collection.

Value

character vector of state abbreviations available for that year

Examples

```
## Not run:  
# Requires Dataverse API  
vest_states(2020)  
  
## End(Not run)
```

Index

- * **center**
 - geos_centerish, [24](#)
 - geos_circle_center, [24](#)
 - st_centerish, [42](#)
 - st_circle_center, [42](#)
- * **datasets**
 - alarm_states, [5](#)
 - clean_vest, [11](#)
 - get_alarm, [25](#)
 - get_dra, [26](#)
 - get_heda, [26](#)
 - get_vest, [29](#)
 - heda_states, [32](#)
 - vest_states, [47](#)
- * **datatable**
 - block2prec, [7](#)
 - block2prec_by_county, [8](#)
 - create_block_table, [13](#)
 - create_tract_table, [14](#)
 - geo_filter, [19](#)
 - geo_trim, [23](#)
- * **data**
 - checkerboard, [10](#)
 - checkerboard_adj, [11](#)
 - nrcsd, [34](#)
 - orange, [34](#)
 - precincts, [35](#)
 - rockland, [37](#)
 - towns, [45](#)
 - va18sub, [47](#)
 - va_blocks, [46](#)
 - va_vtd, [46](#)
- * **dra**
 - dra2r, [15](#)
 - r2dra, [35](#)
- * **estimate**
 - estimate_down, [16](#)
 - estimate_up, [16](#)
 - geo_estimate_down, [17](#)
 - geo_estimate_up, [18](#)
 - geo_match, [20](#)
- * **fix**
 - add_edge, [4](#)
 - adjacency, [5](#)
 - check_contiguity, [8](#)
 - check_polygon_contiguity, [10](#)
 - compare_adjacencies, [12](#)
 - geo_sort, [22](#)
 - split_precinct, [41](#)
 - subtract_edge, [43](#)
 - suggest_component_connection, [44](#)
 - suggest_neighbors, [44](#)
- * **leftover**
 - count_connections, [12](#)
- * **package**
 - geomander-package, [3](#)
- * **plot**
 - geo_plot, [21](#)
 - geo_plot_group, [22](#)
- * **seam**
 - seam_adj, [37](#)
 - seam_geom, [38](#)
 - seam_rip, [39](#)
 - seam_sew, [40](#)
- * **spatcorr**
 - global_gearys, [29](#)
 - global_morans, [30](#)
 - gstar_i, [31](#)
 - local_gearys, [32](#)
 - local_morans, [33](#)
- add_edge, [4](#)
- add_edge(), [40](#), [44](#), [45](#)
- adjacency, [5](#)
- alarm_states, [5](#)
- baf_to_vtd, [6](#)
- block2prec, [7](#)
- block2prec(), [8](#), [20](#), [21](#)

block2prec_by_county, 8

ccm(check_contiguity), 8
cct(check_contiguity), 8
check_contiguity, 8
check_contiguity(), 10
check_polygon_contiguity, 10
checkerboard, 10
checkerboard_adj, 11
clean_vest, 11
compare_adjacencies, 12
count_connections, 12
create_block_table, 13
create_block_table(), 7, 8
create_tract_table, 14

dra2r, 15

estimate_down, 16
estimate_down(), 20
estimate_up, 16
estimate_up(), 20

geo_estimate_down, 17
geo_estimate_down(), 16
geo_estimate_up, 18
geo_estimate_up(), 16
geo_filter, 19
geo_match, 20
geo_match(), 7, 16–18
geo_plot, 21
geo_plot_group, 22
geo_sort, 22
geo_trim, 23
geomander (geomander-package), 3
geomander-package, 3
geos_centerish, 24
geos_circle_center, 24
get_alarm, 25
get_dra, 26
get_heda, 26
get_lewis, 27
get_rpvnearme, 28
get_vest, 29
get_vest(), 11, 25
global_gearys, 29
global_morans, 30
gstar_i, 31

heda_states, 32

local_gearys, 32
local_morans, 33

nrscsd, 34

orange, 34

precincts, 35

r2dra, 35
regionalize, 36
rockland, 37

seam_adj, 37
seam_geom, 38
seam_rip, 39
seam_sew, 40
sf::read_sf(), 27, 29
split_precinct, 41
st_centerish, 42
st_circle_center, 42
subtract_edge, 43
suggest_component_connection, 44
suggest_neighbors, 44

towns, 45

va18sub, 47
va_blocks, 46
va_vtd, 46
vest_states, 47