

# Package ‘censable’

May 8, 2026

**Title** Making Census Data More Usable

**Version** 0.0.8

**URL** <https://christophertkenny.com/censable/>,  
<https://github.com/christopherkenny/censable>

**BugReports** <https://github.com/christopherkenny/censable/issues>

**Description** Creates a common framework for organizing, naming, and gathering population, age, race, and ethnicity data from the Census Bureau. Accesses the API <<https://www.census.gov/data/developers/data-sets.html>>. Provides tools for adding information to existing data to line up with Census data.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.2

**Suggests** roxygen2, spelling, testthat (>= 3.0.0)

**Imports** dplyr (>= 1.0.4), rlang (>= 0.4.11), sf (>= 1.0.0), tibble, stringr, memoise, purrr, censusapi, methods, tinytiger

**Depends** R (>= 4.1.0)

**Config/testthat/edition** 3

**Language** en-US

**NeedsCompilation** no

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

**Maintainer** Christopher T. Kenny <ctkenny@proton.me>

**Repository** CRAN

**Date/Publication** 2025-09-01 19:50:01 UTC

## Contents

add_r_environ . . . . .	3
breakdown_geoid . . . . .	4
build_acs . . . . .	4
build_dec . . . . .	6
collapse4 . . . . .	9
collapse4_pop . . . . .	9
collapse4_vap . . . . .	10
collapse5 . . . . .	10
collapse5_pop . . . . .	11
collapse5_vap . . . . .	11
construct_geoid . . . . .	12
custom_geoid . . . . .	13
fips_2000 . . . . .	14
fips_2010 . . . . .	14
fips_2020 . . . . .	15
join_abb_ansi . . . . .	15
join_abb_fips . . . . .	16
join_abb_name . . . . .	16
join_ansi_abb . . . . .	17
join_ansi_fips . . . . .	17
join_ansi_name . . . . .	18
join_fips_abb . . . . .	18
join_fips_ansi . . . . .	19
join_fips_name . . . . .	20
join_name_abb . . . . .	20
join_name_ansi . . . . .	21
join_name_fips . . . . .	21
key . . . . .	22
match_abb . . . . .	22
match_ansi . . . . .	23
match_fips . . . . .	23
match_name . . . . .	24
mt_county . . . . .	24
recode_abb_ansi . . . . .	25
recode_abb_fips . . . . .	26
recode_abb_name . . . . .	26
recode_ansi_abb . . . . .	27
recode_ansi_fips . . . . .	28
recode_ansi_name . . . . .	28
recode_fips_abb . . . . .	29
recode_fips_ansi . . . . .	29
recode_fips_name . . . . .	30
recode_name_abb . . . . .	30
recode_name_ansi . . . . .	31
recode_name_fips . . . . .	32
stata . . . . .	32

---

add_r_environ	<i>Add Entry to Renviron</i>
---------------	------------------------------

---

### Description

Adds a value to the Renvironment of the form name=value. Designed for flexibly adding API keys for future sessions. Defaults are set up for entering a Census API key to work with tidycensus. By default this key will be configured to work with tidycensus. Package internally allows this key to work with censusapi when used through censable.

### Usage

```
add_r_environ(  
  value,  
  name = "CENSUS_API_KEY",  
  overwrite = FALSE,  
  install = FALSE  
)
```

### Arguments

value	Character. Value to add.
name	Defaults to CENSUS_API_KEY. Character. Name to give value.
overwrite	Defaults to FALSE. Boolean. Should existing item with name name in Renviron be overwritten?
install	Defaults to FALSE. Boolean. Should this be added '~/.Renviron' file?

### Value

value, invisibly

### Examples

```
## Not run:  
add_r_environ('1234', 'SECRET_API_KEY')  
  
## End(Not run)
```

---

breakdown_geoid	<i>Breakdown Census GEOID into Components</i>
-----------------	---

---

### Description

Breakdown Census GEOID into Components

### Usage

```
breakdown_geoid(.data, GEOID = "GEOID", area_type = "spine")
```

### Arguments

.data	dataframe, tibble, or sf tibble
GEOID	Column in .data with Census GEOID
area_type	String, default is 'spine' with type of GEOID. Options are 'spine' for states, counties, tracts, block groups, and blocks. 'shd' for lower state legislative districts, 'ssd' for upper state legislative districts, 'cd' for congressional districts, or 'zcta' for zip code tabulation areas.

### Value

.data with added identifying columns based on area\_type

### Examples

```
data(mt_county)
mt_county <- mt_county |> breakdown_geoid()
```

---

build_acs	<i>Build Data from the American Community Survey</i>
-----------	--

---

### Description

Creates a dataset, using the decennial census information, with the standard variables used for redistricting. Creates a stable base for getting data from censusapi for common calls in redistricting.

#' # Output columns are:

- GEOID: Geographic Identifier
- NAME: Name of County
- pop: total population
- pop\_white: total population, Non-Hispanic White
- pop\_black: total population, Non-Hispanic Black
- pop\_hisp: total population, Hispanic

- pop\_aian: total population, Non-Hispanic American Indian and Alaskan Native
- pop\_asian: total population, Non-Hispanic Asian
- pop\_nhpi: total population, Non-Hispanic Native Hawaiian and Pacific Islander
- pop\_other: total population, Non-Hispanic Other
- pop\_two: total population, Non-Hispanic Two Plus Races
- vap: voting age population
- vap\_white: voting age population, Non-Hispanic White
- vap\_black: voting age population, Non-Hispanic Black
- vap\_hisp: voting age population, Hispanic
- vap\_aian: voting age population, Non-Hispanic American Indian and Alaskan Native
- vap\_asian: voting age population, Non-Hispanic Asian
- vap\_nhpi: voting age population, Non-Hispanic Native Hawaiian and Pacific Islander
- vap\_other: voting age population, Non-Hispanic Other
- vap\_two: voting age population, Non-Hispanic Two Plus Races
- geometry: sf geometry

Arguments for geography are not checked, so will error if invalid. This is by design to avoid blocking usage that could become valid.

Currently valid options for geography:

- 'state'
- 'county'
- 'tract'
- 'block group'
- 'block'
- 'county subdivision'
- 'zcta'
- 'congressional district'
- 'state legislative district (upper chamber)'
- 'state legislative district (lower chamber)'
- 'school district (unified)'
- 'school district (elementary)'
- 'school district (secondary)'

## Usage

```
build_acs(  
  geography,  
  state = NULL,  
  county = NULL,  
  geometry = TRUE,
```

```

    year = 2020,
    survey = "acs5",
    groups = "all"
  )

  mem_build_acs(
    geography,
    state = NULL,
    county = NULL,
    geometry = TRUE,
    year = 2020,
    survey = "acs5",
    groups = "all"
  )

```

### Arguments

geography	Required. The geography level to use.
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 (after August 2021)
survey	whether the get estimates from the 5-year ('acs5'), 3-year ('acs3'), or 1-year ('acs1') survey. Default is 'acs5'.
groups	defaults to 'all', which gets pop and vap. If 'pop', only gets pop. If 'vap', only gets vap. Any other strings default to 'all'.

### Value

tibble with observations for each observation of the geography in the state or county. Data includes up to 3 sets of columns for each race or ethnicity category: population (pop), voting age population (vap), and citizen voting age population (cvap)

### Examples

```

# uses the Census API
tb <- build_acs(geography = 'tract', state = 'NY', county = 'Rockland', geometry = TRUE)

```

---

 build\_dec

*Build Data from the Decennial Census*


---

### Description

Creates a dataset, using the decennial census information, with the standard variables used for re-districting. Creates a stable base for getting data from censusapi for common calls in redistricting.

**Usage**

```
build_dec(
  geography,
  state = NULL,
  county = NULL,
  geometry = TRUE,
  year = 2020,
  groups = "all"
)
```

```
mem_build_dec(
  geography,
  state = NULL,
  county = NULL,
  geometry = TRUE,
  year = 2020,
  groups = "all"
)
```

**Arguments**

geography	Required. The geography level to use.
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 (after August 2021)
groups	defaults to 'all', which gets pop and vap. If 'pop', only gets pop. If 'vap', only gets vap. Allows for analogous seven category race with 'all7', 'pop7', and 'vap7'. For counts for any part by race, you can supply ap:race, where race is in c('black', 'white', 'aian', 'other', 'asian', 'nhpi'). Anything that can't be matched defaults to 'all', so you can pass '' to get 'all'.

**Value**

tibble with observations for each observation of the geography in the state or county. Data includes up to 2 sets of columns for each race or ethnicity category: population (pop) and voting age population (vap)

**Default output columns are:**

- GEOID: Geographic Identifier
- NAME: Name of County
- pop: total population
- pop\_white: total population, Non-Hispanic White
- pop\_black: total population, Non-Hispanic Black
- pop\_hisp: total population, Hispanic

- pop\_aian: total population, Non-Hispanic American Indian and Alaskan Native
- pop\_asian: total population, Non-Hispanic Asian
- pop\_nhpi: total population, Non-Hispanic Native Hawaiian and Pacific Islander
- pop\_other: total population, Non-Hispanic Other
- pop\_two: total population, Non-Hispanic Two Plus Races
- vap: voting age population
- vap\_white: voting age population, Non-Hispanic White
- vap\_black: voting age population, Non-Hispanic Black
- vap\_hisp: voting age population, Hispanic
- vap\_aian: voting age population, Non-Hispanic American Indian and Alaskan Native
- vap\_asian: voting age population, Non-Hispanic Asian
- vap\_nhpi: voting age population, Non-Hispanic Native Hawaiian and Pacific Islander
- vap\_other: voting age population, Non-Hispanic Other
- vap\_two: voting age population, Non-Hispanic Two Plus Races
- geometry: sf geometry

Arguments for geography are not checked, so will error if invalid. This is by design, to avoid blocking usage that could become valid.

Currently valid options for geography:

- 'state'
- 'county'
- 'tract'
- 'block group'
- 'block'
- 'county subdivision'
- 'zcta'
- 'congressional district'
- 'state legislative district (upper chamber)'
- 'state legislative district (lower chamber)'
- 'school district (unified)'
- 'school district (elementary)'
- 'school district (secondary)'
- 'voting district' may also work, though seems to be less reliable

### Examples

```
# uses the Census API
tb <- build_dec(geography = 'block', state = 'NY', county = 'Rockland', geometry = TRUE)
```

---

collapse4	<i>Collapse Full Race Categories into 4 Categories</i>
-----------	--

---

**Description**

Collapses Other, AIAN, Asian, NHPI, and Two+ into other, by prefix.

**Usage**

```
collapse4(.data, prefix)
```

**Arguments**

.data	tibble, data.frame, or sf tibble
prefix	The prefix(es) for the race categories. Must be a character vector.

**Value**

.data with columns collapsed

**Examples**

```
data(mt_county)
mt_county <- mt_county |> collapse4(prefix = c('pop_', 'vap_'))
```

---

collapse4_pop	<i>Collapse Population Race Categories into 4 Categories</i>
---------------	--

---

**Description**

Collapses Other, AIAN, Asian, NHPI, and Two+ into other.

**Usage**

```
collapse4_pop(.data, prefix = "pop_")
```

**Arguments**

.data	tibble, data.frame, or sf tibble
prefix	Default is pop_. The prefix for the race categories.

**Value**

.data with columns collapsed

**Examples**

```
data(mt_county)
mt_county <- mt_county |> collapse4_pop()
```

---

collapse4_vap	<i>Collapse Voting Age Population Race Categories into 4 Categories</i>
---------------	---

---

**Description**

Collapses Other, AIAN, Asian, NHPI, and Two+ into other.

**Usage**

```
collapse4_vap(.data, prefix = "vap_")
```

**Arguments**

.data	tibble, data.frame, or sf tibble
prefix	Default is vap_. The prefix for the race categories.

**Value**

.data with columns collapsed

**Examples**

```
data(mt_county)
mt_county <- mt_county |> collapse4_vap()
```

---

collapse5	<i>Collapse Full Race Categories into 5 Categories</i>
-----------	--

---

**Description**

Collapses Other, AIAN, NHPI, and Two+ into Other, by prefix.

**Usage**

```
collapse5(.data, prefix)
```

**Arguments**

.data	tibble, data.frame, or sf tibble
prefix	The prefix(es) for the race categories. Must be a character vector.

**Value**

.data with columns collapsed

**Examples**

```
data(mt_county)
mt_county <- mt_county |> collapse5(prefix = c('pop_', 'vap_'))
```

---

collapse5_pop	<i>Collapse Population Race Categories into 5 Categories</i>
---------------	--

---

**Description**

Collapses Other, AIAN, NHPI, and Two+ into other.

**Usage**

```
collapse5_pop(.data, prefix = "pop_")
```

**Arguments**

.data	tibble, data.frame, or sf tibble
prefix	Default is pop_. The prefix for the race categories.

**Value**

.data with columns collapsed

**Examples**

```
data(mt_county)
mt_county <- mt_county |> collapse5_pop()
```

---

collapse5_vap	<i>Collapse Voting Age Population Race Categories into 5 Categories</i>
---------------	---

---

**Description**

Collapses Other, AIAN, NHPI, and Two+ into other.

**Usage**

```
collapse5_vap(.data, prefix = "vap_")
```

**Arguments**

`.data` tibble, data.frame, or sf tibble  
`prefix` Default is `vap_`. The prefix for the race categories.

**Value**

`.data` with columns collapsed

**Examples**

```
data(mt_county)
mt_county <- mt_county |> collapse5_vap()
```

---

`construct_geoid` *Create GEOID from Default Columns*

---

**Description**

Create GEOID from Default Columns

**Usage**

```
construct_geoid(
  .data,
  area_type,
  state = "state",
  county = "county",
  tract = "tract",
  block_group = "block group",
  block = "block",
  cd = "cd",
  shd = "shd",
  ssd = "ssd",
  zcta = "zcta"
)
```

**Arguments**

`.data` dataframe, tibble, or sf tibble  
`area_type` Defaults to creating the smallest possible with 'spine' for states, counties, tracts, block groups, and blocks. You can also pass one of the on spine geographies to create that specific level. Other options are 'shd' for lower state legislative districts, 'ssd' for upper state legislative districts, 'cd' for congressional districts, or 'zcta' for zip code tabulation areas.  
`state` name of column with state component  
`county` name of column with county component

tract	name of column with tract component
block_group	name of column with block group component
block	name of column with block component
cd	name of column with cd component
shd	name of column with shd component
ssd	name of column with ssd component
zcta	name of column with zcta component

**Value**

.data with new column GEOID

**Examples**

```
data(mt_county)
mt_county <- mt_county |> breakdown_geoid()
mt_county <- mt_county |> dplyr::select(-dplyr::all_of('GEOID'))
mt_county <- mt_county |> construct_geoid()
```

---

custom_geoid	<i>Create a GEOID from Columns</i>
--------------	------------------------------------

---

**Description**

Create a GEOID from Columns

**Usage**

```
custom_geoid(.data, ...)
```

**Arguments**

.data	dataframe, tibble, or sf tibble
...	columns of .data in the order you want to make the GEOID

**Value**

.data with new column GEOID

**Examples**

```
data(mt_county)
mt_county <- mt_county |> custom_geoid(GEOID)
```

---

`fips_2000`*Counties FIPS 2000*

---

**Description**

Contains three columns:

- state: state FIPS
- county: county FIPS
- name: county name

**Usage**

```
data('fips_2000')
```

**Value**

tibble

**Examples**

```
data('fips_2000')
```

---

`fips_2010`*Counties FIPS 2010*

---

**Description**

Contains three columns:

- state: state FIPS
- county: county FIPS
- name: county name

**Usage**

```
data('fips_2010')
```

**Value**

tibble

**Examples**

```
data('fips_2010')
```

---

`fips_2020`*Counties FIPS 2020*

---

**Description**

Contains three columns:

- state: state FIPS
- county: county FIPS
- name: county name

**Usage**

```
data('fips_2020')
```

**Value**

tibble

**Examples**

```
data('fips_2020')
```

---

`join_abb_ansi`*Join Abb by ANSI*

---

**Description**

Adds a column with state abbreviation joining by a column with state ansi

**Usage**

```
join_abb_ansi(.data, .ansi)
```

**Arguments**

<code>.data</code>	data.frame or tibble
<code>.ansi</code>	column with state ansi

**Value**

`.data` with column `.ansi` replaced with state abbreviation

**Examples**

```
data('stata')
stata |> join_abb_ansi(ansi)
```

---

join_abb_fips	<i>Join Abb by FIPS</i>
---------------	-------------------------

---

**Description**

Adds a column with state abbreviation joining by a column with state fips

**Usage**

```
join_abb_fips(.data, .fips)
```

**Arguments**

.data	data.frame or tibble
.fips	column with state fips

**Value**

.data with column .fips replaced with state abb

**Examples**

```
data('stata')  
stata |> join_abb_fips(fips)
```

---

join_abb_name	<i>Join Abb by Name</i>
---------------	-------------------------

---

**Description**

Adds a column with state abbs joining by a column with state names

**Usage**

```
join_abb_name(.data, .name)
```

**Arguments**

.data	data.frame or tibble
.name	column with state name

**Value**

.data with column .name replaced with abbreviation

**Examples**

```
data('stata')
stata |> join_abb_name(name)
```

---

join_ansi_abb	<i>Join ANSI by Abb</i>
---------------	-------------------------

---

**Description**

Adds a column with state ansi joining by a column with state abbreviation

**Usage**

```
join_ansi_abb(.data, .abb)
```

**Arguments**

.data	data.frame or tibble
.abb	column with state abbreviation

**Value**

.data with column .abb replaced with state ansi

**Examples**

```
data('stata')
stata |> join_ansi_abb(abb)
```

---

join_ansi_fips	<i>Join ANSI by FIPS</i>
----------------	--------------------------

---

**Description**

Adds a column with state ansi joining by a column with state fips

**Usage**

```
join_ansi_fips(.data, .fips)
```

**Arguments**

.data	data.frame or tibble
.fips	column with state fips

**Value**

.data with column .fips replaced with state ansi

**Examples**

```
data('stata')
stata |> join_ansi_fips(fips)
```

---

join_ansi_name	<i>Join ANSI by Name</i>
----------------	--------------------------

---

**Description**

Adds a column with state ansi joining by a column with state name

**Usage**

```
join_ansi_name(.data, .name)
```

**Arguments**

.data	data.frame or tibble
.name	column with state name

**Value**

.data with column .name replaced with ansi

**Examples**

```
data('stata')
stata |> join_ansi_name(name)
```

---

join_fips_abb	<i>Join FIPS by Abb</i>
---------------	-------------------------

---

**Description**

Adds a column with state fips joining by a column with state abbreviation

**Usage**

```
join_fips_abb(.data, .abb)
```

**Arguments**

.data            data.frame or tibble  
.abb            column with state abbreviation

**Value**

.data with column .abb replaced with state name

**Examples**

```
data('stata')  
stata |> join_fips_abb(abb)
```

---

*join\_fips\_ansi*            *Join FIPS by ANSI*

---

**Description**

Adds a column with state fips joining by a column with state ansi

**Usage**

```
join_fips_ansi(.data, .ansi)
```

**Arguments**

.data            data.frame or tibble  
.ansi            column with state ansi

**Value**

.data with column .ansi replaced with state fips

**Examples**

```
data('stata')  
stata |> join_fips_ansi(ansi)
```

---

join_fips_name	<i>Join FIPS by Name</i>
----------------	--------------------------

---

**Description**

Adds a column with state fips joining by a column with state name

**Usage**

```
join_fips_name(.data, .name)
```

**Arguments**

.data	data.frame or tibble
.name	column with state name

**Value**

.data with column .name replaced with fips

**Examples**

```
data('stata')
stata |> join_fips_name(name)
```

---

join_name_abb	<i>Join Name by Abb</i>
---------------	-------------------------

---

**Description**

Adds a column with state name joining by a column with state abbreviation

**Usage**

```
join_name_abb(.data, .abb)
```

**Arguments**

.data	data.frame or tibble
.abb	column with state abbreviation

**Value**

.data with column .abb replaced with state name

**Examples**

```
data('stata')
stata |> join_name_abb(abb)
```

---

join_name_ansi	<i>Join Name by ANSI</i>
----------------	--------------------------

---

**Description**

Adds a column with state name joining by a column with state ansi

**Usage**

```
join_name_ansi(.data, .ansi)
```

**Arguments**

.data	data.frame or tibble
.ansi	column with state ansi

**Value**

.data with column .ansi replaced with state name

**Examples**

```
data('stata')
stata |> join_name_ansi(name)
```

---

join_name_fips	<i>Join Name by FIPS</i>
----------------	--------------------------

---

**Description**

Adds a column with state name joining by a column with state fips

**Usage**

```
join_name_fips(.data, .fips)
```

**Arguments**

.data	data.frame or tibble
.fips	column with state fips

**Value**

.data with column .fips replaced with state name

**Examples**

```
data('stata')
stata |> join_name_fips(fips)
```

---

key	<i>Check or Get Census API Key</i>
-----	------------------------------------

---

**Description**

Check or Get Census API Key

**Usage**

```
has_census_key()
get_census_key(key = "")
```

**Arguments**

key	Census API Key as a character
-----	-------------------------------

**Value**

logical if has, key if get

**Examples**

```
has_census_key()
```

---

match_abb	<i>Try to Match to State Abbreviation</i>
-----------	---

---

**Description**

Searches for an exact match and offers the best match if no exact match

**Usage**

```
match_abb(state)
```

**Arguments**

state	character with state FIPS, Abbreviation, Name, or ANSI
-------	--

**Value**

Abbreviation if a match is found or character(0) if no match is found

**Examples**

```
match_abb('NY')
match_abb('01')
```

---

match\_ansi

*Try to Match to State ANSI*

---

**Description**

Searches for an exact match and offers the best match if no exact match

**Usage**

```
match_ansi(state)
```

**Arguments**

state                    character with state FIPS, Abbreviation, Name, or ANSI

**Value**

ANSI if a match is found or character(0) if no match is found

**Examples**

```
match_ansi('NY')
match_ansi('01')
```

---

match\_fips

*Try to Match to State FIPS*

---

**Description**

Searches for an exact match and offers the best match if no exact match

**Usage**

```
match_fips(state)
```

**Arguments**

state                    character with state FIPS, Abbreviation, Name, or ANSI

**Value**

FIPS code if a match is found or character(0) if no match is found

**Examples**

```
match_fips('NY')
match_fips('01')
```

---

match_name	<i>Try to Match to State Name</i>
------------	-----------------------------------

---

**Description**

Searches for an exact match and offers the best match if no exact match

**Usage**

```
match_name(state)
```

**Arguments**

state                    character with state FIPS, Abbreviation, Name, or ANSI

**Value**

Name if a match is found or character(0) if no match is found

**Examples**

```
match_name('NY')
match_name('01')
```

---

mt_county	<i>Montana County Data</i>
-----------	----------------------------

---

**Description**

- GEOID: Geographic Identifier
- NAME: Name of County
- pop: total population
- pop\_white: total population, Non-Hispanic White
- pop\_black: total population, Non-Hispanic Black
- pop\_hisp: total population, Hispanic
- pop\_aian: total population, Non-Hispanic American Indian and Alaskan Native

- pop\_asian: total population, Non-Hispanic Asian
- pop\_nhpi: total population, Non-Hispanic Native Hawaiian and Pacific Islander
- pop\_other: total population, Non-Hispanic Other
- pop\_two: total population, Non-Hispanic Two Plus Races
- vap: voting age population
- vap\_white: voting age population, Non-Hispanic White
- vap\_black: voting age population, Non-Hispanic Black
- vap\_hisp: voting age population, Hispanic
- vap\_aian: voting age population, Non-Hispanic American Indian and Alaskan Native
- vap\_asian: voting age population, Non-Hispanic Asian
- vap\_nhpi: voting age population, Non-Hispanic Native Hawaiian and Pacific Islander
- vap\_other: voting age population, Non-Hispanic Other
- vap\_two: voting age population, Non-Hispanic Two Plus Races
- geometry: sf geometry

**Usage**

```
data('mt_county')
```

**Value**

sf tibble with one observation for each county in Montana

**Examples**

```
data('mt_county')
```

---

recode_abb_ansi	<i>Recode Abb by ANSI</i>
-----------------	---------------------------

---

**Description**

Replaces state ansi with state abbreviation

**Usage**

```
recode_abb_ansi(.data, .ansi)
```

**Arguments**

.data	data.frame or tibble
.ansi	column with state ansi

**Value**

.data with column .ansi replaced with state abbreviation

**Examples**

```
data('stata')
stata |> recode_abb_ansi(ansi)
```

---

recode_abb_fips	<i>Recode Abb by FIPS</i>
-----------------	---------------------------

---

**Description**

Replaces state fips with state abb

**Usage**

```
recode_abb_fips(.data, .fips)
```

**Arguments**

.data	data.frame or tibble
.fips	column with state fips

**Value**

.data with column .fips replaced with state abb

**Examples**

```
data('stata')
stata |> recode_abb_fips(fips)
```

---

recode_abb_name	<i>Recode Abb by Name</i>
-----------------	---------------------------

---

**Description**

Replaces state name with state abbreviation

**Usage**

```
recode_abb_name(.data, .name)
```

**Arguments**

.data            data.frame or tibble  
.name            column with state name

**Value**

.data with column .name replaced with abbreviation

**Examples**

```
data('stata')  
stata |> recode_abb_name(name)
```

---

recode_ansi_abb	<i>Recode ANSI by Abb</i>
-----------------	---------------------------

---

**Description**

Replaces state abbreviation with state ansi

**Usage**

```
recode_ansi_abb(.data, .abb)
```

**Arguments**

.data            data.frame or tibble  
.abb            column with state abbreviation

**Value**

.data with column .abb replaced with state ansi

**Examples**

```
data('stata')  
stata |> recode_ansi_abb(abb)
```

---

recode_ansi_fips	<i>Recode ANSI by FIPS</i>
------------------	----------------------------

---

**Description**

Replaces state fips with state ansi

**Usage**

```
recode_ansi_fips(.data, .fips)
```

**Arguments**

.data	data.frame or tibble
.fips	column with state fips

**Value**

.data with column .fips replaced with state ansi

**Examples**

```
data('stata')
stata |> recode_ansi_fips(fips)
```

---

recode_ansi_name	<i>Recode ANSI by Name</i>
------------------	----------------------------

---

**Description**

Replaces state name with state ansi

**Usage**

```
recode_ansi_name(.data, .name)
```

**Arguments**

.data	data.frame or tibble
.name	column with state name

**Value**

.data with column .name replaced with ansi

**Examples**

```
data('stata')
stata |> recode_ansi_name(name)
```

---

recode_fips_abb	<i>Recode FIPS by Abb</i>
-----------------	---------------------------

---

**Description**

Replaces state abbreviation with state fips

**Usage**

```
recode_fips_abb(.data, .abb)
```

**Arguments**

.data	data.frame or tibble
.abb	column with state abbreviation

**Value**

.data with column .abb replaced with state name

**Examples**

```
data('stata')
stata |> recode_fips_abb(abb)
```

---

recode_fips_ansi	<i>Recode FIPS by ANSI</i>
------------------	----------------------------

---

**Description**

Replaces state ansi with state fips

**Usage**

```
recode_fips_ansi(.data, .ansi)
```

**Arguments**

.data	data.frame or tibble
.ansi	column with state ansi

**Value**

.data with column .ansi replaced with state fips

**Examples**

```
data('stata')
stata |> recode_fips_ansi(ansi)
```

---

recode_fips_name	<i>Recode FIPS by Name</i>
------------------	----------------------------

---

**Description**

Replaces state name with state fips

**Usage**

```
recode_fips_name(.data, .name)
```

**Arguments**

.data	data.frame or tibble
.name	column with state name

**Value**

.data with column .name replaced with fips

**Examples**

```
data('stata')
stata |> recode_fips_name(name)
```

---

recode_name_abb	<i>Recode Name by Abb</i>
-----------------	---------------------------

---

**Description**

Replaces state abbreviation with state name

**Usage**

```
recode_name_abb(.data, .abb)
```

**Arguments**

.data            data.frame or tibble  
.abb            column with state abbreviation

**Value**

.data with column .abb replaced with state name

**Examples**

```
data('stata')  
stata |> recode_name_abb(abb)
```

---

recode_name_ansi	<i>Recode Name by ANSI</i>
------------------	----------------------------

---

**Description**

Replaces state ansi with state name

**Usage**

```
recode_name_ansi(.data, .ansi)
```

**Arguments**

.data            data.frame or tibble  
.ansi            column with state ansi

**Value**

.data with column .ansi replaced with state name

**Examples**

```
data('stata')  
stata |> recode_name_ansi(name)
```

---

recode_name_fips	<i>Recode Name by FIPS</i>
------------------	----------------------------

---

**Description**

Replaces state fips with state name

**Usage**

```
recode_name_fips(.data, .fips)
```

**Arguments**

.data	data.frame or tibble
.fips	column with state fips

**Value**

.data with column .fips replaced with state name

**Examples**

```
data('stata')
stata |> recode_name_fips(fips)
```

---

stata	<i>stata (State Data)</i>
-------	---------------------------

---

**Description**

tibble with columns:

- fips: Federal Information Processing Standards codes
- abb: two letter postal abbreviations
- name: title case state name
- ansi: American National Standards Institute codes
- region: Census Regions (for 50 states and D.C.)
- division: Census Divisions (for 50 states and D.C.)

**Usage**

```
data('stata')
```

**Value**

tibble with state identifying information

**Examples**

```
data('stata')
```

# Index

- \* **build**
  - build\_acs, 4
  - build\_dec, 6
- \* **collapse**
  - collapse4, 9
  - collapse4\_pop, 9
  - collapse4\_vap, 10
  - collapse5, 10
  - collapse5\_pop, 11
  - collapse5\_vap, 11
- \* **data**
  - fips\_2000, 14
  - fips\_2010, 14
  - fips\_2020, 15
  - mt\_county, 24
  - stata, 32
- \* **geoid**
  - breakdown\_geoid, 4
  - construct\_geoid, 12
  - custom\_geoid, 13
- \* **join**
  - join\_abb\_ansi, 15
  - join\_abb\_fips, 16
  - join\_abb\_name, 16
  - join\_ansi\_abb, 17
  - join\_ansi\_fips, 17
  - join\_ansi\_name, 18
  - join\_fips\_abb, 18
  - join\_fips\_ansi, 19
  - join\_fips\_name, 20
  - join\_name\_abb, 20
  - join\_name\_ansi, 21
  - join\_name\_fips, 21
- \* **match**
  - match\_abb, 22
  - match\_ansi, 23
  - match\_fips, 23
  - match\_name, 24
- \* **recode**
  - recode\_abb\_ansi, 25
  - recode\_abb\_fips, 26
  - recode\_abb\_name, 26
  - recode\_ansi\_abb, 27
  - recode\_ansi\_fips, 28
  - recode\_ansi\_name, 28
  - recode\_fips\_abb, 29
  - recode\_fips\_ansi, 29
  - recode\_fips\_name, 30
  - recode\_name\_abb, 30
  - recode\_name\_ansi, 31
  - recode\_name\_fips, 32
- add\_r\_envIRON, 3
- breakdown\_geoid, 4
- build\_acs, 4
- build\_dec, 6
- collapse4, 9
- collapse4\_pop, 9
- collapse4\_vap, 10
- collapse5, 10
- collapse5\_pop, 11
- collapse5\_vap, 11
- construct\_geoid, 12
- custom\_geoid, 13
- fips\_2000, 14
- fips\_2010, 14
- fips\_2020, 15
- get\_census\_key (key), 22
- has\_census\_key (key), 22
- join\_abb\_ansi, 15
- join\_abb\_fips, 16
- join\_abb\_name, 16
- join\_ansi\_abb, 17
- join\_ansi\_fips, 17

join\_ansi\_name, [18](#)  
join\_fips\_abb, [18](#)  
join\_fips\_ansi, [19](#)  
join\_fips\_name, [20](#)  
join\_name\_abb, [20](#)  
join\_name\_ansi, [21](#)  
join\_name\_fips, [21](#)

key, [22](#)

match\_abb, [22](#)  
match\_ansi, [23](#)  
match\_fips, [23](#)  
match\_name, [24](#)  
mem\_build\_acs (build\_acs), [4](#)  
mem\_build\_dec (build\_dec), [6](#)  
mt\_county, [24](#)

recode\_abb\_ansi, [25](#)  
recode\_abb\_fips, [26](#)  
recode\_abb\_name, [26](#)  
recode\_ansi\_abb, [27](#)  
recode\_ansi\_fips, [28](#)  
recode\_ansi\_name, [28](#)  
recode\_fips\_abb, [29](#)  
recode\_fips\_ansi, [29](#)  
recode\_fips\_name, [30](#)  
recode\_name\_abb, [30](#)  
recode\_name\_ansi, [31](#)  
recode\_name\_fips, [32](#)

stata, [32](#)