Package 'acs'

July 22, 2017

Type Package

Title Download, Manipulate, and Present American Community Survey and Decennial Data from the US Census

Version 2.1.1

Date 2017-07-20

Author Ezra Haber Glenn [aut, cre]

Maintainer Ezra Haber Glenn <eglenn@mit.edu>

URL http://dusp.mit.edu/faculty/ezra-glenn,
http://eglenn.scripts.mit.edu/citystate/,
http://mailman.mit.edu/mailman/listinfo/acs-r

Description Provides a general toolkit for downloading, managing, analyzing, and presenting data from the U.S. Census (<https://www.census.gov/data/developers/data-sets.html>), including SF1 (Decennial short-form), SF3 (Decennial long-form), and the American Community Survey (ACS). Confidence intervals provided with ACS data are converted to standard errors to be bundled with estimates in complex acs objects. Package provides new methods to conduct standard operations on acs objects and present/plot data in statistically appropriate ways.

Suggests

Imports graphics, stats, plyr, utils, httr

Depends R (>= 2.10), stringr, methods, XML

License GPL-3

LazyData yes

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2017-07-21 23:04:47 UTC

R topics documented:

`acs-package` *Download, Manipulate, and Present American Community Survey and Decennial Data from the US Census*

Description

Provides a general toolkit for downloading, managing, analyzing, and presenting data from the U.S. Census, including SF1 (Decennial "short-form"), SF3 (Decennial "long-form"), and the American Community Survey (ACS). Confidence intervals provided with ACS data are converted to standard errors to be bundled with estimates in complex acs objects. Package provides new methods to conduct standard operations on acs objects and present/plot data in statistically appropriate ways.

Details

Package:	acs
Type:	Package
Version:	2.1.1
Date:	2017-07-20
License:	GPL-3
Depends:	stringr, methods, XML

The package defines a new "acs" class (containing estimates, standard errors, geography, and metadata for tables from the U.S. Census American Community Survey), with methods to deal appropriately with common tasks, such as combining subgroups or geographies, mathematical operations on estimates, tests of significance, and computing (and plotting) confidence intervals.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

References

1. "A Compass for Understanding and Using American Community Survey Data: What State and Local Governments Need to Know." Washington, DC: U.S. Census Bureau. 2009. http://www.census.gov/library/publications/2009/acs/state-and-local.html.
2. "acs.R: An R Package for Neighborhood-Level Data from the U.S. Census." Ezra Haber Glenn, Department of Urban Studies and Planning, Massachusetts Institute of Technology. Presented at the Computers in Urban Planning and Urban Management Conference, July 6, 2011. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2171390.
3. "Working with acs.R (June 2013)", Ezra Haber Glenn. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2552524
4. CityState webpage: http://eglenn.scripts.mit.edu/citystate/
5. User Group Mailing List: http://mailman.mit.edu/mailman/listinfo/acs-r

---

acs-class *Class* "acs"

Description

The acs class provides a convenient wrapper for demographic data from the U.S. Census, especially the American Community Survey. Estimates and standard errors are kept together, along with geographic information and metadata necessary to manipulate and analyze data in this form.

Objects from the Class

acs objects can be created by calls of the form `new("acs", ...)`, or through helper functions provided by the package (currently `read.acs` and `acs.fetch`), or from the output of subsetting or other calls on existing acs objects. Once created, acs objects can be manipulated through new methods to deal appropriately with common analytical tasks such as combining subgroups or geographies, mathematical operations on estimates, and computing (and plotting) confidence intervals.

Slots

`endyear`: Object of class `"integer"` indicating the last year included in the dataset (e.g., 2012 for data from the 2008–2012 ACS)

`span`: Object of class `"integer"` representing the number of years the dataset spans (e.g., 3 for data from the 2011–2013 ACS); for decennial census datasets (SF1 and SF3), span = 0.

`geography`: Object of class `"data.frame"` containing columns extracted from the data's geographic header: typically includes geographic place names, census summary level values, and unique numeric identifiers, but can contain any geographic names or labels desired. When acs objects are created or modified, the first geography column will be used to label estimates and standard errors.

`acs.colnames`: Object of class `"character"` giving the variable names for each column

`modified`: Object of class `"logical"` to indicate whether the object has been modified since construction

`acs.units`: Object of class `"factor"` designating the type of units in each column (e.g., count or percentage or dollars); only used minimally, to check appropriateness of some operations; mostly reserved for future functionality

`currency.year`: Object of class `"integer"` indicating the year that all currency values have been adjusted to (by default the same as endyear, but able to be modified by the user for comparisons with other years; see `currency.convert`.)

`estimate`: Object of class `"matrix"` holding the reported ACS estimates

`standard.error`: Object of class `"matrix"` holding the calculated values of the standard errors for each estimate, derived from the reported 90% confidence intervals

Methods

acs.colnames `signature(object = "acs")`: Standard accessor function; returns character vector

acs.units `signature(object = "acs")`: Standard accessor function; returns factor vector

currency.year `signature(object = "acs")`: Standard accessor function; returns integer

endyear `signature(object = "acs")`: Standard accessor function; returns integer

estimate `signature(object = "acs")`: Standard accessor function; returns matrix

geography `signature(object = "acs")`: Standard accessor function; returns data.frame

modified `signature(object = "acs")`: Standard accessor function; return logical

span `signature(object = "acs")`: Standard accessor function; returns integer

standard.error `signature(object = "acs")`: Standard accessor function; returns matrix

sum `signature(object = "acs")`: Aggregates (adds) all estimates in the object, and adds the corresponding standard errors in a statistically appropriate way; returns new acs object

summary `signature(object = "acs")`: Prints standard summary data on both estimates and standard errors

confint `signature(object = "acs")`: Prints estimates with confidence intervals

[`signature(x = "acs")`: subsetting works for acs objects using standard `[i,j]` square bracket notation, similar to two-dimensional matrices; returns a new acs object with estimates, standard errors, and associated metadata for "i" rows (geographies) and "j" columns (variable columns); essentially, subsetting for this class is structured to mirror standard operations on matrix objects

[<- `signature(x = "acs")`: new values may be replaced/assigned to an existing acs object using standard `[i,j]` bracket notation. The assignment can accept a number of different forms: a valid acs object (including a subsetted one), a list of two matrices (ideally named "estimate" and "error" or "standard.error"), or a numeric object which may be coerced into a matrix (to be used as estimates, with zero-values assigned to corresponding standard errors).

In addition to these methods, new methods for basic arithmetic functions (+, -, *, /) have been provided to deal appropriately with combining estimates and standard errors.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

Examples

```
showClass("acs")
# load some data from the ACS
data(kansas09)
str(kansas09)

# access slots
endyear(kansas09)
span(kansas09)
estimate(kansas09)[1:5,1:5]
standard.error(kansas09[1:5,1:5])


# subset
kansas09[1:4,6:9]

# more complicated subsets
kansas09[c("Linn County, Kansas", "Wilson County, Kansas") ,
   grep(pattern="21.years", x=acs.colnames(kansas09))]

# addition on estimates and errors
kansas09[1:4,25]+kansas09[1:4,49]

# can even multiply and divide
# males per female, by county
kansas09[1:4,2]/kansas09[1:4,26]

# (males<5 plus females<5) * 12
(kansas09[7,3]+kansas09[7,27]) * 12

# some replacement: males<5 as a percentage of all males
kansas09[,3]=kansas09[,3]/kansas09[,2]
```

`acs.fetch` *Downloads demographic data (ACS, SF1, SF3) via the Census API and converts to a proper acs object with estimates, standard errors, and associated metadata.*

Description

When passed a valid geo.set object and either lookup terms (table.number, table.name, keyword) or a valid acs.lookup object, queries the Census API and downloads data to create a new acs-class object. Geographical information is preserved, metadata in bundled together with the acs object, and margins of errors are converted to standard errors to accompany estimates (see `help(acs)`).

Usage

```
acs.fetch(endyear, span = 5, geography, table.name,
      table.number, variable, keyword, dataset = "acs",
      key, col.names = "auto", ...)
```

Arguments

`endyear`	an integer indicating the latest year of the data in the survey (e.g., for data from the 2007-2011 5-year ACS data, endyear would be 2011)
`span`	an integer indicating the span (in years) of the desired ACS data (should be 1, 3, or 5 for ACS datasets, and 0 for decennial census SF1 and SF3 datasets); defaults to 5, but ignored and reset to 0 if dataset="sf1" or "sf3".
`geography`	a valid geo.set object specifying the census geography or geographies to be fetched; can be created "on the fly" with a call to `geo.make()`
`table.name`	a string giving the search term(s) to find in the name of the ACS census table (for example, "Sex" or "Age"); accepts multiple words, which must all be found in the returned table names; always case-sensitive. (Note: when set, this variable is passed to an internal call to acs.lookup—see `acs.lookup`).
`table.number`	a string (not a number) indicating the table from the Census to fetch; examples: "B01003" or "B23013"; always case-sensitive. Used to fetch all variables for a given table number; if "table.number" is provided, other lookup variables ("table.name" or "keyword") will be ignored.
`variable`	an object of acs.lookup class, or a string (not a number) or vector of strings indicating the exact variable number(s) the Census to fetch. See details for more. Non-acs.lookup examples include "B01003_001" or "B23013_003" or c("B01003_001", "B23013_003"). Used to fetch specific variables, as opposed to all variables for a given table number; if "variable" is provided, all other lookup variables ("table.name", "table.number", and "keyword") will be ignored.
`keyword`	a string or vector of strings giving the search term(s) to find in the name of the census variable (for example, "Male" or "Haiti"); accepts multiple words, which must all be found in the returned variable names; always case-sensitive. (Note: when set, this variable is passed to an internal call to acs.lookup—see `acs.lookup`).
`dataset`	either "acs" (the default), "sf1", or "sf3", indicating whether to fetch data from in the American Community Survey or the SF1/SF3 datasets. See details for more information about available data.
`key`	a string providing the Census API key to use for when fetching data. Typically saved once via `api.key.install` and passed automatically with each call; see `api.key.install`

`col.names` either "auto","pretty", or a vector of strings of the same length as the number of variables to be fetched. When "auto" (the default), census variable codes will be used as column names for the fetched data; when "pretty", descriptive variables names will be used; otherwise col.names will be used.

`...` Not used interactively (reserved for recursive calls).

Details

Assuming you have created some geography with geo.make and you have already installed an API key, the call is quite simple: for example, `acs.fetch(endyear=2014, geography=my.geo, table.number="B01003")`. (For more on API keys, see `api.key.install`; if you haven't installed one, you can always add a "key=YOUR.KEY.HERE" argument to `acs.fetch` each time.)

By default, acs.fetch will download 5-year ACS, but as of version 2.0 users must specify a specific "endyear". Users may also select 1- or 3-year ACS data using the "span=" option, as well as Decennial data using the "dataset" option. (When dataset="sf1" or "sf3", span will be reset to 0 regardless of any explict or default options.) At present, the API provides five-, three- and one-year data for a variety of different endyears, and Decennial data for 2010, 2000, and 1990; see the chart below and/or visit `http://www.census.gov/data/developers/data-sets.html` to learn more about what is available through the API. (Warning: support for 1990 is a bit unreliable as of the date of this version, due to non-standard variable lookup tables.)

- American Community Survey 5-Year Data (dataset="acs", span=5): 2005-2009 through 2010-2014
- American Community Survey 3 Year Data (dataset="acs", span=3): 2013, 2012
- American Community Survey 1 Year Data (dataset="acs", span=1): 2014, 2013, 2012, 2011
- SF1/Short-Form (dataset="sf1"): 1990, 2000, 2010
- SF3/Long-Form (dataset="sf3"): 1990, 2000

Downloading based on a table number is probably the most fool-proof way to get the data you want, but acs.fetch will also accept a number of other arguments instead of "table.number". Users can provide strings to search for in table names (e.g., table.name="Age by Sex" or table.name="First Ancestry Reported") or keywords to find in the names of variables (e.g., keyword="Male" or keyword="Haiti")—but be warned: given how many tables there are, you may get more matches than you expected and suffer from the "download overload" of fast-scrolling screens full of data. (But don't lose hope: see the `acs.lookup` tool, which can help with this problem.)

On the other hand, if you know you want a specific variable or two (not a whole table, just a few columns of it, such as variable="B05001_006" or variable=c("B16001_058", "B16001_059")), you can ask for that with `acs.fetch(variable="these.variable.codes", ...)`.

Note: when "combine=T" for the fetched geo.set, data will be aggregated in the resulting acs abject. Some users may therefore wish to specify "one.zero=T" as an additional argument to `acs.fetch`; see `sum-methods`.

Value

Returns a new acs-class object with estimates, standard errors (derived from the census 90% margins of error), and metadata of the fetched data from the Census API.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

References

1. http://www.census.gov/developers/
2. http://www.census.gov/data/developers/data-sets.html

See Also

acs.lookup.

`acs.lookup` *Search for relevant demographic variables and tables from the US Census.*

Description

The `acs.fetch` function is used to download data from the US Census American Community Survey. The `acs.lookup` function provides a convenience function to use in advance to locate tables and variables that may be of interest.

`acs.lookup` takes arguments similar to `acs.fetch` — in particular, "table.number", "table.name", and "keyword", as well as "endyear","span", and "dataset" — and searches for matches in the meta-data of the Census tables. When multiple search terms are passed to a given argument (e.g., `keyword=c("Female", "GED")`), the tool returns matches where ALL of the terms are found; similarly, when more than one lookup argument is used (e.g., `table.number="B01001", keyword="Female"`), the tool searches for matches that include all of the terms (i.e., terms are combined with a logical "AND", not a logical "OR").

Results from acs.lookup — which are acs.lookup class objects — can then be inspected, subsetted (with [square brackets]), and combined (with `c` or `+`) to create custom acs.lookup objects to store and later pass to `acs.fetch`.

Usage

```
acs.lookup(endyear, span = 5, dataset = "acs", keyword,
     table.name, table.number, case.sensitive = T)
```

Arguments

`endyear`	an integer indicating the latest year of the data in the survey (e.g., for data from the 2007-2011 5-year ACS data, endyear would be 2011; limited by acceptable values currently provided by the Census API)
`span`	an integer indicating the span (in years) of the desired ACS data (should be 1, 3, or 5); defaults to 5. Ignored and reset to 0 if dataset="sf1" or "sf3".

dataset	either "acs" (the default), "sf1", or "sf3", indicating whether to look for tables and variables in the American Community Survey, the SF1 dataset (decennial/"short-form"), or the SF3 dataset (decennial/"long-form").
keyword	a string or vector of strings giving the search term(s) to find in the name of the ACS census variable (for example, "Male" or "Haiti"); accepts multiple words, which must all be found in the returned variable names.
table.name	a string giving the search term(s) to find in the name of the ACS census table (for example, "Sex" or "Age" or "Age by Sex"); accepts multiple words, which must all be found in the returned table names.
table.number	a string (not a number) indicating the desired table from the Census to fetch; examples: "B01003" or "B23013"; always case-sensitive. Used to identify all variables for a given table number.
case.sensitive	a logical flag indicating whether searching is case-sensitive (the default) or not. Note that the Census is not entirely consistent in capitalization in table and variable names, so setting case.sensitive=F may be useful in finding all possible matches.

Details

In many cases, acs.lookup is called internally by acs.fetch, to determine the variable codes to use for a given table.name or table.number. Since each lookup involves a search of static XML tables (provided by the census for each endyear/span combination, and included by the acs package in /extdata), searches involving more recent years (e.g., for version 2.0, endyears > 2014) may fail. In such situations, users may wish to call acs.fetch with the "variable=" option, perhaps reusing variables from a saved acs.lookup search for a previous year.

For example, once the 2011-2015 5-year ACS data is available via the API, users can attempt the following to access Table B01003, even before the new version of the package is installed with the correct variable lookup tables: acs.fetch(endyear=2015, span=5,variable=acs.lookup(endyear=2014, span=5, table

Value

Returns an acs.lookup class object with the results of the query. acs.lookup objects can be subsetted and combined, and passed to the "variable" argument of acs.fetch.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

See Also

acs.lookup-class

Examples

```
## Not run: acs.lookup(endyear=2014, span=5, table.number="B01001")
acs.lookup(endyear=2012, span=1, table.number="B01001", keyword="Female")
acs.lookup(endyear=2012, span=1, keyword=c("Female", "GED"))
acs.lookup(endyear=2000, dataset="sf3", table.number="P56")
```

```
acs.lookup(endyear=1990, dataset="sf3", table.number="H058")
age.by.sex=acs.lookup(endyear=2014, span=5, table.name="Age by Sex")
age.by.sex
workers.age.by.sex=age.by.sex[4:6]
workers.age.by.sex

## End(Not run)
```

`acs.lookup-class` *Class* `"acs.lookup"`

Description

A new class to hold the results of calls to `acs.lookup`, typically for possible modification and then passing to calls to `acs.fetch` (using the "variable=" argument).

Objects from the Class

Objects can be created by calls of the form `new("acs.lookup", ...)`, but more typically will be created as output from calls to `acs.lookup`.

Slots

`endyear`**:** Object of class `"numeric"` indicating the year of the census dataset; e.g., for data from the 2005-2009 ACS, endyear=2009

`span`**:** Object of class `"numeric"` indicating the span in years of the census dataset; e.g., for data from the 2005-2009 ACS, span=5. For decennial census datasets (SF1 and SF3), span = 0.

`args`**:** Object of class `"list"` containing the search terms used in the call to `acs.lookup`, including some or all of: keyword, table.name, endyear, dataset, table.number, and case.sensitive.

`results`**:** Object of class `"data.frame"` containing character strings in four columns: variable.code, table.number, table.name, and variable.name.

Methods

+ `signature(e1 = "acs.lookup", e2 = "acs.lookup")`: used for combining two acs.lookup objects into one

c `signature(x = "acs.lookup")`: used for combining two acs.lookup objects into one

endyear `signature(object = "acs.lookup")`: returns endyear from acs.lookup object

[`signature(object = "acs.lookup")`: used for subsetting an acs.lookup object

results `signature(object = "acs.lookup")`: returns results (as dataframe) from acs.lookup object

span `signature(object = "acs.lookup")`: returns span from acs.lookup object

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

See Also

acs.lookup

Examples

```
showClass("acs.lookup")
```

`acs.tables.install` *Downloads and stores XML tables used to lookup variable codes, table names, and other metadata associated with acs package.*

Description

To obtain variable codes and other metadata needed to access the Census API, both `acs.fetch` and `acs.lookup` must consult various XML lookup files, which are provided by the Census with each data release. To keep the acs package-size small, as of version 2.0 these files are accessed online at run-time for each query. As an alternative, users may use `acs.tables.install` to download and archive all current tables (approximately 10MB, as of version 2.0 release).

Use of this function is completely optional and the package should work fine without it (assuming the computer is online and is able to access the lookup tables), but running it once may result in faster searches and quicker downloads for all subsequent sessions. (The results are saved and archived, so once a user has run the function, it is unnecessary to run again, unless the acs package is re-installed or updated.)

Usage

```
acs.tables.install()
```

Value

Downloads the files and saves them to the package's "extdata" directory; return an error if no files found.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

References

http://www.census.gov/data/developers/data-sets.html

See Also

acs.fetch acs.lookup

`api.key.install` *Installs an API key from the US Census to use with calls to* `acs.fetch`.

Description

The `acs.fetch` function requires an "API key" to use when downloading data from the US Census API. Rather than pass this rather long string to the function each time, users can save the key as part of the package installation, using the `api.key.install` function. Once installed, an api key is saved on the system and available for use in future sessions. (To replace a key, simply call the function again with the new key.)

Usage

```
api.key.install(key, file = "key.rda")
```

Arguments

`key`	The API key provided to the user upon registering with the US Census Developer's page. A string.
`file`	An alternate name to use when storing key; reserved for future use.

Details

The requirement for a key seems to be laxly enforced by the Census API, but is nonetheless coded into the acs package. Users without a key may find success by simply installing a blank key (i.e., key="") via `api.key.install(key="")`; similarly, calls to `acs.fetch` and `geo.make(..., check=T)` may succeed with a `key=""` argument. Note that while this may work today, it may fail in the future if the API decides to start enforcing the requirement.

Value

Saves the key and exits silently, unless an error is encountered.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

References

To request an API key, see http://www.census.gov/developers/

See Also

`acs.fetch`

`api.key.migrate` *After updating the acs package, installs an archived API key from a previous installation.*

Description

The `acs.fetch` function requires an "API key" to use when downloading data from the US Census API. Rather than pass this rather long string to the function each time, users can save the key as part of the package installation, using the `api.key.install` function. Once installed, an api key is saved on the system and available for use in future sessions. (To replace a key, simply call the function again with the new key.)

During the update process, this key may be lost or left in the wrong location. A call to `api.key.migrate()` can help restore an archived key, if found.

Usage

```
api.key.migrate()
```

Value

Migrates the key (if found) and exits silently; return an error if no archived key is found.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

References

To request an API key, see http://www.census.gov/developers/

See Also

`acs.fetch` `api.key.install`

`cbind.acs` *Combine acs Objects by Columns*

Description

Take a pair of acs objects and combine by columns.

Usage

```
          ## S3 method for class 'acs'
cbind(e1, e2, ...)
```

Arguments

e1, e2	two acs-class objects
...	provided for consistency with cbind S3 method

Details

When passed two acs-class objects, cbind will first check to confirm whether the objects contain compatible data: same endyear and span; same geography. If not, it will issue a warning, but will still proceed.

After this check, the function will return a new acs object that has resulted from combining the two arguments column-wise. The effect is essentially the same as cbind on the underlying estimate and standard.error matrices, with all the additional acs metadata tended to.

Value

Returns a single new acs object with all of the data contained in the two arguments.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

---

confint.acs *Return upper and lower bounds of given confidence intervals for acs objects.*

Description

When passed an acs object, confint will return a list of two-column dataframes (one dataframe for each variable specified in parm) including lower and upper bounds for given confidence intervals. Intervals can be one- or two-sided.

Usage

```
## S3 method for class 'acs'
confint(object, parm = "all", level = 0.95, alternative = "two.sided", ...)
```

Arguments

object	a acs object (or subset).
parm	which variables/columns to return confidence intervals for; defaults to "all", which computes confidence intervals for all estimates in the acs object.
level	the confidence level required – e.g., .95 = 95% confidence.
alternative	whether the interval should be one-sided (i.e., one-tailed – "greater" or "less" – extending to Inf (or -Inf) on one side) or "two-sided".
...	additional argument(s) for methods.

Value

Returns a list of dataframes (one for each variable specified in `parm`) of the lower and upper bounds of the confidence interval for each row of the data.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>.

Examples

```
# load ACS data
data(kansas09)

# confidence intervals for select columns
confint(kansas09[20:25,], parm=c(4,5,10))

# another way to accomplish this
confint(kansas09[20:25,c(4,5,10)])

# store data and extract at will
my.conf <- confint(kansas09)
str(my.conf)
my.conf[32]
my.conf$Universe...TOTAL.POPULATION.IN.THE.UNITED.STATES..U.S..citizen.by.naturalization

# try a different value for level
confint(kansas09[1:10,6], level=.75)

# ... or a one-sided confidence interval
confint(kansas09[1:10,6], level=.75, alternative="greater")
confint(kansas09[1:10,29], level=.75, alternative="less")
```

---

cpi *Consumer Price Index data (1913-2015).*

Description

Contains data on the Consumer Price Index for All Urban Consumers (CPI-U) for the years 1913-2015 from the U.S. Bureau of Labor Statistics. Used by the acs package for currency conversion functions. Scaled for base (100) to be 1982-84.

Usage

```
data(cpi)
```

Format

A named vector containing 103 observations, one for each year from 1913 through 2015.

Source

http://www.bls.gov/cpi/

See Also

currency.year

currency.convert

---

currency.convert *Convert dollar values of acs object to a new base year.*

Description

currency.convert provides a helper function to create a new copy of an acs-class object with a modified currency.year and converted dollar values without altering the original object.

Usage

```
currency.convert(object, rate="auto", newyear=NA_integer_, verbose=F)
```

Arguments

object	an acs object
rate	an optional rate to apply; "auto" (the default) will look up values from the cpi dataset.
newyear	an integer specifying the new value of currency.year to convert into
verbose	whether to print additional information about the conversion

Details

currency.convert provides a helper function to create a new copy of an acs-class object with a modified currency.year and converted dollar values without altering the original object. When rate="auto" (the default), currency.convert will look up values from the cpi database to use in conversion. When a numeric rate is provided through this option, actual cpi values are ignored. When verbose=T, currency.convert will provide additional information about the rates of conversion and the acs.colnames converted.

As of version 2.0 the package includes CPI data from 1913 through 2015, allowing conversion of dollar values for any years in this range.

Value

Returns a new acs object with updated dollar values and currency.year metadata.

Unlike currency.year<-, currency.convert does not alter the original object.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

See Also

currency.year

cpi

Examples

```
lawrence10                                  # median income data, endyear = 2010
currency.convert(lawrence10, newyear=2014)  # convert $$ to 2014 dollars
currency.convert(lawrence10, newyear=1929)  # convert $$ to 1929 dollars
```

---

currency.year — *Return (or change) currency.year value from the metadata of an acs object.*

Description

Standard accessor/replacement method for metadata contained within S4 acs-class objects.

Usage

```
currency.year(object)

currency.year(object)<-value
```

Arguments

object	an acs object
value	an integer value to be used in replacement

Details

currency.year will return the (integer) value of the dollar-year of object.

Assigning a new value to currency.year (through currency.year(object)<-value or currency.year(object)=value) will change the value of currency.year in the object's metadata and also modify all dollar values of the object (as determined by acs.units(object)=="dollars") to be in the dollars of the desired new year.

A related function, currency.convert provides a helper function to create a new copy of an acs-class object with a modified currency.year and converted dollar values without altering the original object. When rate="auto" (the default), currency.convert will look up values from the cpi database to use in conversion. When a numeric rate is provided through this option, actual cpi

values are ignored. When verbose=T, currency.convert will provide additional information about the rates of conversion and the acs.colnames converted.

As of version 2.0 the package includes CPI data from 1913 through 2015, allowing conversion of dollar values for any years in this range.

Value

Returns (or replaces) an integer value from the "currency.year" slot of an object.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

See Also

cpi

currency.convert

acs-class

---

divide.acs *Divide one acs object or variable by another.*

Description

The acs package provides a new S4 method for standard division operations using "/" notation. However, due to the nature of estimates and errors, there are actually two types of division, with slightly different meanings: depending on which variables are being divided, the process may be either a "proportion"-type division (in which the numerator is a subset of the denominator) or a "ratio"-type division (in which this is not the case). When dividing with standard "a/b" notation, the package will always use the more conservative ratio-type procedure.

When appropriate, "proportion"-type division may be desirable, as it results in lower standarard errors. To allow users to specify which type of division to use for acs objects, the package includes a new "divide.acs" function. (See details.)

Usage

```
divide.acs(numerator, denominator, method="ratio", verbose=T, output="result")
```

Arguments

numerator	an acs object to divide
denominator	an acs object to divide by
method	either "ratio" (the default) or "proportion", to indicate what kind of division is desired
verbose	whether to provide additional warnings or just shut up
output	either "result" (the default), "div.method", or "both"

Details

In certain cases, "proportion-style" division will fail, due to the creation of a negative number under a square root when calculating new standard errors. To address this problem and prevent unnecessary NaN values in the standard.errors, the package implements the recommended Census practice of simply using "ratio-style" division in those cases.

If method="proportion" (not the default) and verbose=T (the default), `division.acs` will provide a warning to indicate when "ratio-style" division has been used, including the number of standard error cells so affected. Users wishing to examine a detailed, cell-by-cell report may run `divide.acs` with the output="div.method" of output="both" to get additional diagnostic information.

See "A Compass for Understanding and Using American Community Survey Data" below for details on when this substitution is recommended.

Value

Returns a new acs object with the results of the division (the default), or (when result="div.method") a martix with diagnostic information, or (when result="both"), a list with both of these objects (the first name $result and the second $div.method).

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

References

1. "A Compass for Understanding and Using American Community Survey Data: What State and Local Governments Need to Know." Washington, DC: U.S. Census Bureau. 2009. http://www.census.gov/library/publications/2009/acs/state-and-local.html.

See Also

acs-class

---

endyear *Return or replace endyear value from the metadata of an acs object.*

Description

`endyear()` will return the (integer) value of the latest year of the object (for example, for the 2005-2009 ACS survey, `endyear` = 2009.) When used for assignment, `endyear<-` will change the value of the endyear slot in an acs object, warning the user that this is an unusual thing to do.

Usage

```
endyear(object)

endyear(object)<-value
```

Arguments

object	an acs object
value	an integer to use as the new endyear

Details

Normal operations on acs objects should not involve altering the value of endyear (although users may wish to change the value of currency.year for comparisons with other objects). Sometimes endyear may be set incorrectly when data is imported, in which case endyear<- may be necessary.

Value

Returns (or replaces) an integer value from the endyear slot of an object.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

See Also

currency.year, which is often what users will be intending to modify

acs-class

---

fips.state — *FIPS codes and geographic names for use in searching and creating geographies in the acs package.*

Description

FIPS codes and geographic names for use in searching and creating geographies in the acs package. (Used internally.)

Usage

```
data(fips.state)
```

Format

Each table is a dataframe containing FIPS codes and names from the US Census.

Source

State: http://www.census.gov/geo/reference/ansi_statetables.html

County: http://www.census.gov/geo/www/codes/county/download.html

County Subdivision: http://www.census.gov/geo/www/codes/cousub/download.html

Place: http://www.census.gov/geo/www/codes/place/download.html

School: http://www.census.gov/geo/www/codes/sd/

American Indian Area: http://www.census.gov/geo/www/codes/aia/

---

flatten.geo.set *Convenience function to "flatten" a nested geo.set object.*

Description

In the acs package, geo.set objects may contain nested levels of geo.set objects, which is often desired (to organize complex sets and subsets of geography). Sometimes, however, when combining these sets, users may prefer to remove the nesting levels. This convenience function will recursively prowl through a geo.set and return a single flat geo.set (one level deep) containing of the composite geographies.

Usage

```
flatten.geo.set(x)
```

Arguments

x	the geo.set to be flattened

Value

Returns a new geo.set object.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

See Also

geo.set-class

Examples

```
# a multiple-county geo.set
psrc=geo.make(state="WA", county=c(33,35,53,61))

# combine geo.sets
north.mercer.island=geo.make(state=53, county=33, tract=c(24300,24400))
optional.tract=geo.make(state=53, county=33, tract=24500)
# add in one more tract to create new, larger geo
north.mercer.island.plus=north.mercer.island + optional.tract

# created a nested geo.set
 my.nested.geo.set=c(north.mercer.island.plus, psrc)
str(my.nested.geo.set)
length(my.nested.geo.set)

# .. and flatten in out
#    note difference in structure and length
my.flat.geo.set=flatten.geo.set(my.nested.geo.set)
str(my.flat.geo.set)
length(my.flat.geo.set)
```

---

geo.lookup *Search Census geographies*

Description

When working with the acs package and the `acs.fetch` and `geo.make` functions, it can be difficult to find exactly the right geographic units: `geo.make` expects single matches to the groups of arguments it is given, which can be problematic when trying to find names for places or county subdivisions, which are unfamiliar to many users (and often seem very close or redundant: e.g., knowing whether to look for "Moses Lake city" vs. "Moses Lake CDP"). To help, the `geo.lookup` function will search on the same arguments as `geo.make`, but outputs all the matches for your inspection.

Usage

```
geo.lookup(state, county, county.subdivision, place,
  american.indian.area, school.district, school.district.elementary,
  school.district.secondary, school.district.unified)
```

Arguments

`state`	either the two-digit numeric FIPS code for the state, the two-letter postal abbreviation, or a character string to match in the state name
`county`	either the numeric FIPS code for the county or a character string to match in the county name

`county.subdivision`
: either the numeric FIPS code for the county subdivision or a character string to match in the county subdivision name

`place`
: either the numeric FIPS code for the place or a character string to match in the place name

`american.indian.area`
: either the numeric FIPS code for the American Indian Area/Alaska Native Area/Hawaiian Home Land, or a character string to match in the names of these Census areas

`school.district`
: either the numeric FIPS code for the state school district (any type), or a character string to search for in the names of the school districts.

`school.district.elementary`
: either the numeric FIPS code for the state school district (elementary), or a character string to search for in the names of these elementary school districts.

`school.district.secondary`
: either the numeric FIPS code for the state school district (secondary), or a character string to search for in the names of these secondary school districts.

`school.district.unified`
: either the numeric FIPS code for the state school district (unified), or a character string to search for in the names of these unified school districts.

Details

Unlike `geo.make`, `geo.lookup` searches for matches anywhere in geographic names (except when dealing with state names), and will output a dataframe showing candidates that match some or all of the arguments. (When multiple arguments are provided, the logic is a little complicated: basically, with the exception of American Indian Areas, to be included all geographies must match the given state name; when a county and a subdivision are both given, both must match; otherwise, geographies are included that match any — but not necessarily all — of the other arguments.)

Value

Returns a dataframe of the matching geographies, with one column for each of the given search terms.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

See Also

`geo.make`

Examples

```
geo.lookup(state="WA", county="Ska", county.subdivision="oo")
geo.lookup(state="WA", county="Kit", place="Ra")

# find all counties in WA or OR with capital M or B in name
```

```
geo.lookup(state=c("WA", "OR"), county=c("M","B"))

# find all unified school districts in Kansas with "Ma" in name
geo.lookup(state="KS", school.district.unified="Ma")

# find all american indian areas with "Hop" in name
geo.lookup(american.indian.area="Hop")
```

`geo.make` *Create a new geo.set object for use with the acs package.*

Description

The `geo.make` function is used to create new user-specified geographies for use with the `acs.fetch` function of the acs package. At the most basic level, a user specifies some combination of existing census levels (state, county, county subdivision, place, tract, block group, msa, csa, puma, and more – see arguments), and the function returns a new geo.set object holding this information.

When specifying state, county, county subdivision, place, american indian area, and/or any of the state school district arguments, `geo.make` will accept either FIPS codes or common geographic names, and will try to match on partial strings; there is also limited support for regular expressions, but by default the searches are case sensitive and matches are expected at the start of names. (For example, `geo.make(state="WA", county="Kits")` should find Kitsap County, and the more adventurous `yakima=geo.make(state="Washi",county=".*kima")` should work to create the a geo.set for Yakima county.)

Other geographies (including tract, block.group, csa, msa, region, division, urban.area, necta, puma, zip.code. and/or congressional.district) can only be specified by FIPS codes (or "*" for all).

Tracts should be specified as six digit numbers, although initial zeroes may be removed; note that trailing zeroes are often removed in common usage, so a tract that may be referred to as "tract 243" is technically FIPS code 24300; likewise "tract 3872.01" is FIPS code 387201 for the purposes of geo.make.

Usage

```
geo.make(us, region, division, state, county, county.subdivision,
  place, tract, block.group, msa, csa, necta, urban.area,
  congressional.district, state.legislative.district.upper,
  state.legislative.district.lower, puma, zip.code,
  american.indian.area, school.district.elementary,
  school.district.secondary, school.district.unified,
  combine = F, combine.term = "aggregate", check = FALSE, key = "auto")
```

Arguments

us	either the number 1, the character "*", or TRUE, indicating whether the geo.set should contain data for the entire U.S.; if selected, no other geography options may be specified; setting us corresponds to using census summary level 010.

`region` a numeric code (or wildcard "*" for all) corresponding to the desired FIPS region (e.g., region=1 for Census Northeast Region); if selected, no other geography options may be specified; setting region corresponds to using census summary level 020.

`division` a numeric code (or wildcard "*" for all) corresponding to the desired FIPS division (e.g., division=4 for Census West North Central Division); if selected, no other geography options may be specified; setting division corresponds to using census summary level 030.

`american.indian.area`
either the numeric code (or wildcard "*" for all) corresponding to the desired FIPS American Indian Area/Alaska Native Area/Hawaiian Home Land, or a character string to match in the names of these Census areas; if selected, no other geography options may be specified; setting american.indian.area corresponds to using census summary level 250.

`state` either the two-digit numeric FIPS code for the state, the two-letter postal abbreviation, or a character string to match in the state name (or wildcard "*" for all); setting state without other options corresponds to using census summary level 040, but it may be used in conjunction with other summary levels below.

`county` either the numeric FIPS code (or wildcard "*" for all) for the county or a character string to match in the county name; setting state and county without other options corresponds to using census summary level 050, but they may be used in conjunction with other summary levels below.

`county.subdivision`
either the numeric FIPS code (or wildcard "*" for all) for the county subdivision or a character string to match in the county subdivision name; setting state, county, and county.subdivision without other options corresponds to using census summary level 060.

`place` either the numeric FIPS code (or wildcard "*" for all) for the place or a character string to match in the place name; setting state and place without other options corresponds to using census summary level 160.

`tract` a six digit numeric FIPS code (or wildcard "*" for all) for the census tract, including trailing zeroes; remove decimal points; leading zeroes may be omitted; see description; tract may be used with state and county to create geo.sets for census summary levels 140, and with state, county, and block.group for summary level 150.

`block.group` the numeric FIPS code (or wildcard "*" for all) for the block.group; block.group may be used with state, county, and tract to create geo.sets for census summary levels 150.

`msa` a numeric code (or wildcard "*" for all) corresponding to the desired FIPS metropolitan statistical area/micropolitan statistical area (e.g., msa=10100 for Aberdeen, SD micropolitan statistical area); setting msa without other options corresponds to using census summary level 310, but it may be used in conjunction with state for summary level 320.

`csa` a numeric code (or wildcard "*" for all) corresponding to the desired FIPS combined statistical area (e.g., csa=104 for Census Albany-Schenectady-Amsterdam,

NY CSA); setting csa without other options corresponds to using census summary level 330, but it may be used in conjunction with state for summary level 340.

`necta` a numeric code (or wildcard "*" for all) corresponding to the desired FIPS New England City and Town Area (e.g., necta=77650 for Rutland, VT Micropolitan NECTA); if selected, no other geography options may be specified; setting necta corresponds to using census summary level 350.

`urban.area` a numeric code (or wildcard "*" for all) corresponding to the desired FIPS urban area (e.g., urban.area=3169 for Aromas, CA Urban Cluster); if selected, no other geography options may be specified; setting urban.area corresponds to using census summary level 400.

`congressional.district`
a numeric code (or wildcard "*" for all) corresponding to the desired FIPS congressional district (e.g., state="ME" and congressional.district=1 for Maine's first congressional district); setting state and congressional.district without other options corresponds to using census summary level 500, but they may be used in conjunction with county for summary level 510.

`state.legislative.district.upper`
a numeric or character code (or wildcard "*" for all) corresponding to the desired FIPS state legislative district (upper chamber); these codes vary from state to state, and are sometimes numbers (1, 2, 3, etc. in Massachusetts) and sometimes letters ("A", "B", "C", etc. in Alaska); setting state and state.legislative.district.upper without other options corresponds to using census summary level 610.

`state.legislative.district.lower`
a numeric or character code (or wildcard "*" for all) corresponding to the desired FIPS state legislative district (lower chamber); these codes vary from state to state, and are sometimes numbers (1, 2, 3, etc. in Massachusetts) and sometimes letters ("A", "B", "C", etc. in Alaska); setting state and state.legislative.district.lower without other options corresponds to using census summary level 620.

`puma` a numeric code (or wildcard "*" for all) corresponding to the desired FIPS public use microdata area (e.g., state=10 and puma=103 for PUMA 103 in Delaware); setting state and puma without other options corresponds to using census summary level 795.

`zip.code` a numeric code (or wildcard "*" for all) corresponding to the desired zip code tabulation area (e.g., zip.code=91303 for zip code 91303); if selected, no other geography options may be specified; setting zip.code corresponds to using census summary level 860.

`school.district.elementary`
a numeric code (or wildcard "*" for all) corresponding to the desired FIPS state school district (elementary), or a character string to search for in the names of these districts; setting state and school.district.elementary without other options corresponds to using census summary level 950.

`school.district.secondary`
a numeric code (or wildcard "*" for all) corresponding to the desired FIPS state school district (secondary), or a character string to search for in the names of these districts; setting state and school.district.secondary without other options corresponds to using census summary level 960.

`school.district.unified`
: a numeric code (or wildcard "*" for all) corresponding to the desired FIPS state school district (unified), or a character string to search for in the names of these districts; setting state and school.district.unified without other options corresponds to using census summary level 970.

`combine`
: a logical flag to indicate whether the component geographies of the geo.set are to be combined when data is downloaded; see details.

`combine.term`
: a character string to provide a label for aggregate geography, if data is combined

`check`
: logical flag indicating whether to run a check for valid geographies with Census API; defaults to FALSE; when TRUE, a current API key must be provided or installed

`key`
: when check=T and no API key has been previously installed through `api.key.install`, a string key may be provided here

Details

In addition to creating individual combinations of census geographies, users can pass vector arguments (with recycling) to geo.make to create sets of geographies. Important: each set of arguments must match with exactly one known Census geography: if, for example, the names of two places (or counties, or whatever) would both match, the `geo.make` function will return an error. (To the development team, this seemed preferable to simply including both matches, since all sorts of place names might match a string, and it is doubtful a user really wants them all.) The one exception to this "single match" rule is that for the smallest level of geography specified, a user can enter "*" to indicate that all geographies at that level should be selected.

When creating new geographies, note, too, that not all combinations are valid. In particular the package attempts to follow paths through the Census summary levels (such as summary level 140: "state-county-tract" or summary level 160: "state-place"). So when specifying, for example, state, county, and place, the county will be ignored.

Note: when a geo.set with "combine=T" is passed to `acs.fetch`, downloaded data will be aggregated in the resulting acs abject. Some users may therefore wish to specify "one.zero=T" as an additional argument to `acs.fetch`; see `sum-methods`.

The following table may be helpful in figuring out which options to set for which Census summary levels. For more information on which datasets and endyear/span combinations are available for each summary level, see http://www.census.gov/data/developers/data-sets.html (click each dataset and search for "Examples and Supported Geography").

SUMMARY LEVEL	ARGUMENTS REQUIRED
010	us
020	region
030	division
040	state
050	state, county
060	state, county, county.subdivision
140	state, county, tract
150	state, county, tract, block.group
160	state, place
250	american.indian.area

310	msa
320	state, msa
330	csa
340	state, csa
350	necta
400	urban.area
500	state, congressional.district
510	state, congressional.district, county
610	state, state.legislative.district.upper
620	state, state.legislative.district.lower
795	state, puma
860	zip.code
950	state, school.district.elementary
960	state, school.district.secondary
970	state, school.district.unified

All other arguments/combinations will either be ignored or result in a failure.

Value

Returns a geo.set class object.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

References

1. "acs.R: An R Package for Neighborhood-Level Data from the U.S. Census." Ezra Haber Glenn, Department of Urban Studies and Planning, Massachusetts Institute of Technology. Presented at the Computers in Urban Planning and Urban Management Conference, July 6, 2011. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2171390.
2. Census API Supported Geography: http://www.census.gov/data/developers/data-sets.html

See Also

geo.set-class

Examples

```
# some single-state geo.sets
washington=geo.make(state=53)
alabama=geo.make(state="Alab")

# a county match
yakima=geo.make(state="WA", county="Yakima")
yakima
```

```
# a multiple-county geo.set
psrc=geo.make(state="WA", county=c(33,35,53,61))
psrc

# combine geo.sets
north.mercer.island=geo.make(state=53, county=33, tract=c(24300,24400))
optional.tract=geo.make(state=53, county=33, tract=24500)
# add in one more tract to create new, larger geo
north.mercer.island.plus=north.mercer.island + optional.tract

# using wildcards

# all unified school districts in Kansas
geo.make(state="KS", school.district.unified="*")

# all state house districts in Alaska
geo.make(state="AK", state.legislative.district.lower="*")

# all tracts in Kings County, NY
geo.make(state="NY", county="King", tract="*")
```

`geo.set-class` *Class* "`geo.set`"

Description

The geo.set class provides a convenient wrapper for user-defined geographies, used for downloading data from the U.S. Census American Community Survey. A geo.set may hold the designation of a single geography (say, a census tract, a county, or a state), or may bundle together multiple geographies of various levels, which may or may not be "combined" when downloaded. Note that geo.sets may even contain nested geo.sets.

Note: even a single geographic unit — one specific tract or county — must be wrapped up as a geo.set. Technically, each individual element in the set is known as a "geo", but users will rarely (if ever) interact will individual elements such as this; wrapping all groups of geographies — even groups consisting of just one element — in geo.sets like this will help make them easier to deal with as the geographies get more complex.

geo.set objects may be combined with the simple addition operator (+). By default, this will always return "flat" geo.sets with all the geographies in a single list. The combination operator (c), on the other hand, will generally return nested hierarchies, embedding sets within sets. When working with nested sets like this, the "combine" flag can be set at each level to aggregate subsets within the structure (although be careful — if a higher level of set includes "combine=T" you'll never actually see the unaggregated subsets deeper down).

Using these different techniques, users are able to create whatever sort of new geographies they need — aggregating some geographies, keeping others distinct (but still bundled as a set for convenience), mixing and matching different levels of Census geography, and so on.

Objects from the Class

Objects can be created by calls of the form `new("geo.set", ...)`, or more frequently through the `geo.make()` helper function.

Slots

`geo.list`**:** Object of class `"list"` containing individual census geographies (as geo class object) and/or geo.sets.

`combine`**:** Object of class `"logical"` indicating whether or not data from the constituent geographies should be combined when downloaded. Set with `combine<-` or specified when using `geo.make`.

`combine.term`**:** Object of class `"character"` indicating a new label to use when data is combined; ignored when combine set to F. Set with `combine.term<-` or specified when using `geo.make`.

Methods

[`signature(x = "geo.set")`: subset geo.set, similar to single-bracket list subsetting in R

[[`signature(x = "geo.set")`: subset geo.set, similar to double-bracket list subsetting in R

+ `signature(e1 = "geo", e2 = "geo")`: combine two geo objects; returns a geo.set (generally reserved for internal use)

+ `signature(e1 = "geo", e2 = "geo.set")`: combine a geo object onto an existing geo.set; returns a geo.set (generally reserved for internal use)

+ `signature(e1 = "geo.set", e2 = "geo")`: combine an existing geo.set object with a geo object; returns a geo.set (generally reserved for internal use)

+ `signature(e1 = "geo.set", e2 = "geo.set")`: combine two geo.set objects; always flattens each set – no nesting

c `signature(x = "geo.set")`: combine two or more geo.set objects, preserving the structure of each – allows nesting

combine<- `signature(object = "geo.set")`: used to set or change value of combine

combine `signature(object = "geo.set")`: returns logical value of combine

combine.term<- `signature(object = "geo.set")`: used to set or change combine.term

combine.term `signature(object = "geo.set")`: returns combine.term

geo.list `signature(object = "geo.set")`: returns the geo.list of the geo.set (used internally)

length `signature(x = "geo.set")`: returns an integer indicating how many geographies it contains; non-recursive.

name `signature(object = "geo")`: returns the text name of an individual geo object.

sumlev `signature(object = "geo")`: returns the summary level of an individual geo object.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

References

http://eglenn.scripts.mit.edu/citystate/category/code/

See Also

geo.make

Examples

```
showClass("geo.set")
```

---

geography *Return or replace geography metadata of an acs object.*

Description

geography() will return the geography of an acs object, as a dataframe. Depending on the format of the data at import (and possibly the values of geocols=, if the object was created with read.acs), this may have multiple columns, but the number of geographic rows should be the same as the number of rows of the acs estimates and standard errors.

When used for assignment, geography<- will change the values contained in the metadata, replacing the existing dataframe with a new one. To replace a single value or a limited subset, call with subsetting (e.g., geography(object)[i,j]<-value or geography(object)[[i]]<-value; note that the brackets should occur *outside* the call – you are subsetting the dataframe, not the object).

To help with replacement operations, the package provides a new prompt method, which can be used to interactively set new values for geography (as well as other metadata); see prompt.acs.

Usage

```
geography(object)

geography(object)<-value
```

Arguments

object	an acs object
value	a dataframe containing geographic metadata; must contain the same number of rows as the object

Value

Returns (or replaces) a dataframe containing the geography slot of an object.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

See Also

prompt.acs, a helper function to interactively generate a new geography dataframe to be used for replacement.

acs-class

Examples

```
data(lawrence10)
geography(lawrence10)
str(geography(lawrence10))
```

---

kansas07 *County-level data from the 2007 American Community Survey for Kansas for use in examples of acs package.*

Description

County-level data from the 2007 American Community Survey for Kansas. Contains demographic data on sex, age, and citizenship. Used for examples in acs package. kansas07 and the corresponding five-year survey data in kansas09 provide acs objects to test and demonstrate various functions in the package.

Usage

```
data(kansas07)
```

Format

An acs-class object with 7 rows/geographies and 55 demographic variables, representing county-level ACS data for the state of Kansas. Also includes geographic and other metadata.

Note that in comparison to kansas09, kansas07 has far fewer rows, which illustrates the fact that the Census only provides ACS one-year data for the largest counties (over 65,000 population).

Source

U.S. Census American Community Survey, 2007; http://www.census.gov/

Examples

```
data(kansas07)
str(kansas07)
class(kansas07)

geography(kansas07)

# subsetting
kansas07[1:3,2:4]
```

```
# row-wise addition
kansas07[1,6]+kansas07[2,6]

# column-wise addition
kansas07[1:4,3]+kansas07[1:4,27]
```

---

kansas09 *County-level data from the 2005-2009 American Community Survey for Kansas for use in examples of acs package.*

Description

County-level data from the 2005-2009 American Community Survey for Kansas. Contains demographic data on sex, age, and citizenship. Used for examples in acs package. `kansas09`, and the corresponding one-year survey data in `kansas07`, provide acs objects to test and demonstrate various functions in the package.

Usage

```
data(kansas09)
```

Format

An acs-class object with 105 rows/geographies and 55 demographic variables, representing county-level ACS data for the state of Kansas. Also includes geographic and other metadata.

Source

U.S. Census American Community Survey, 2009; http://www.census.gov/

Examples

```
data(kansas09)
str(kansas09)
class(kansas09)

geography(kansas09)

# subsetting
kansas09[1:3,2:4]

# row-wise addition
kansas09[1,6]+kansas09[2,6]

# column-wise addition
kansas09[1:4,3]+kansas09[1:4,27]
```

lawrence10 *Tract-level data from the 2006-2010 American Community Survey for Lawrence, MA for use in examples of acs package.*

Description

Tract-level data from the 2006-2010 American Community Survey for Lawrence, MA. Contains median household income. Used for examples in acs package.

Usage

```
data(lawrence10)
```

Format

An acs-class object with 18 rows/geographies and 1 variable, representing tract-level ACS data for the city of Lawrence, MA from 2006-2010. Also includes geographic and other metadata.

Source

U.S. Census American Community Survey, 2010; http://www.census.gov/

Examples

```
data(lawrence10)
str(lawrence10)
class(lawrence10)

# subsetting
lawrence10[1:3,1]

# row-wise subtraction
lawrence10[1,1]+lawrence10[2,1]
```

plot-methods *acs Methods for Function* plot

Description

Plot acs objects, with both estimates and confidence intervals.

Usage

```
## S4 method for signature 'acs'
plot(x, conf.level=.95, err.col="red", err.lwd=1,
err.pch="-", err.cex=2, err.lty=2, x.res=300, labels="auto",
by="geography", true.min=T, ...)
```

Arguments

x	the acs object to be plotted
conf.level	the desired confidence interval to use for error bars; numeric between 0<1
err.col	the color to use for the error bars; analogous to graphic parameter col
err.lwd	the line weight to use for the error bars; analogous to graphic parameter lwd
err.pch	the point character to use for the error bars; analogous to graphic parameter pch
err.cex	the scaling factor to use for the error bars; analogous to graphic parameter cex
err.lty	the line type to use for the error bars; analogous to graphic parameter lty
x.res	when plot called with a single acs value (see below), x.res determines the resolution of the resulting density plot; integer (defaults to 300, i.e., the curve is drawn with 300 points)
labels	the labels to use for the x axis; defaults to either geography names or acs.colnames based on dimensions of object plotted; vector of proper length required
by	in cases where multiple rows and columns are plotted, whether to provide a different plot for each value of geography (the default) or acs.colnames; accepts either "geography" or "acs.colnames"
true.min	whether to limit the lower bound of a confidence interval to some value or now; TRUE (the default) allows for negative lower bounds; also accepts FALSE to limit lower bounds to 0, or any number, to use that as a minimum lower bound; see details.
...	provided to allow for passing of additional arguments to plot functions

Methods

signature(object = "acs") When passed an acs object (possibly involving subsetting), plot will call a plot showing both estimates and confidence intervals for the data contained in the object.

If the object contains only one row or only one column, plot will use this dimension as the y-axis and will plot each observation along the x-axis, as three points (an estimate bracketed by upper and lower confidence bounds). If the object contains multiple rows and columns, plot will return a 1-by-y "plot of plots": by default there will be one plot per row showing all the data for each geography, although this can be changed by specifying by="acs.colnames", to plot each variable as its own plot, with all of the geographies along the x-axis.

In the special case where the dimensions of the object are exactly c(1,1) (i.e., a single geography and column), plot will return a density plot of the estimate. In this case, conf.level, err.col, err.lty, and err.lwd will be used to determine the properties of the margins of

error lines. (For none, use `conf.level=F`. For these density plots, users may also wish to set `xlim` and `x.res`, which specify the horizontal extent and resolution of the plot.)

`plot` accepts many of the standard graphical arguments to `plot`, such as `main`, `sub`, `xlab`, `pch`, and `col`, as well new ones listed above.

In some cases, the lower bound of a confidence interval may extend below 0; in some cases this is desired, especially when a variable is actually stating the *difference* between two estimates. In other cases, this may seem confusing (for example, when reporting the estimated count in a particular category). Setting `true.min` to `FALSE` (or 0) will limit the lower boundary of any confidence intervals computed and plotted.

Examples

```
# load ACS data
data(kansas07)

# plot a single value
plot(kansas07[4,4])

# plot by geography
plot(kansas07[,10])

# plot by columns
plot(kansas07[4,3:10])

# a density plot for a single variable
plot(kansas07[7,10])

# same, using some graphical parameters
plot(kansas07[7,10], col="blue", err.col="purple", err.lty=3)

plot(kansas07[7,49], col="lightblue", type="h", x.res=3000,
err.col="purple", err.lty=3, err.lwd=4, conf.level=.99,
main=(paste("Distribution of Females>85 Years in ",
geography(kansas07)[7,1], sep="")),
sub="(99-percent margin of error shown in purple)")

# something more complicated...

plot(kansas07[c(1,3,4),3:25], err.col="purple",
pch=16, err.pch="x", err.cex=1, ylim=c(0,5000),
col=rainbow(23), conf.level=.99,
labels=paste("grp. ",1:23))
```

`prompt.acs` *Prompt for new values for metadata in an acs object.*

Description

Helper function to interactively set new values for row- and/or column-names in an acs object.

Usage

```
## S3 method for class 'acs'
prompt(object, filename=NA, name=NA, what="acs.colnames",
geocols="all", ...)
```

Arguments

object	an acs object
filename	not used; provided for S3 generic/method consistency
name	not used; provided for S3 generic/method consistency
what	which acs-class metadata slot to prompt for; either "acs.colnames" (the default), "acs.units", or "geography"
geocols	a vector, or "all", specifying which columns from the geography metadata to prompt for (optional; defaults to "all"; ignored when what="acs.colnames")
...	not used; provided for S3 generic/method consistency

Details

The acs package provides this S3 prompt method for acs-class objects, primarily as a "helper" function to use in calls to geography(object)<-, acs.units(object)<-, or acs.colnames(object)<-. prompt provides an interactive interface, prompting the user for new metadata values based on the existing ones.

When what="geography" and geocols is not "all", prompt will only prompt for replacements of the values of geocols, but will still return values for all geography columns, suitable for passing to geography(object)<-.

Anytime during the interactive prompt() session, a user may enter a blank line to terminate, returning only the changed values up to that point (along with the unchanged values for remaining entries.)

Value

Returns a value of the same class and dimensions as the current geography, acs.units, or acs.colnames of object, but with new names, suitable for passing to one of the replacement methods (acs.colnames<-, (acs.units<-, or geography<-).

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

See Also

geography<-

acs.colnames<-

acs.units<-

Examples

```
data(kansas07)

acs.colnames(kansas07)=prompt(kansas07, what="acs.colnames")

geography(kansas07)=prompt.acs(kansas07, what="geography")
```

---

rbind.acs *Combine acs Objects by Rows*

Description

Take a pair of acs objects and combine by rows.

Usage

```
          ## S3 method for class 'acs'
rbind(e1, e2, ...)
```

Arguments

e1, e2	two acs-class objects
...	provided for consistency with cbind S3 method

Details

When passed two acs-class objects, rbind will first check to confirm whether the objects contain compatible data: same endyear and span; same column names. If not, it will issue a warning, but will still proceed.

After this check, the function will return a new acs object that has resulted from combining the two arguments row-wise. The effect is essentially the same as rbind on the underlying estimate and standard.error matrices, with all the additional acs metadata tended to.

Value

Returns a single new acs object with all of the data contained in the two arguments.

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

`read.acs` *Reads a comma-delimited file from the American Community Survey and creates an acs object with estimates, standard errors, and associated metadata.*

Description

When passed a comma-delimited file from the U.S. Census American Community Survey (typically downloaded via the FactFinder website and unzipped), read.acs returns an acs object with estimates, standard errors, and associated metadata.

Most users will prefer to start with `acs.fetch` to import data; `read.acs` is maintained as a "legacy" function, primarily for use in situations where data is not available via the Census API.

Usage

```
read.acs(filename, endyear = "auto", span = "auto", col.names= "auto",
acs.units = "auto", geocols = "auto", skip = "auto")
```

Arguments

`filename`	the name of the `.csv`, `.zip`, or `.txt` file to be input
`endyear`	an integer (or "auto") indicating the latest year of the data in the survey (e.g., for data from the 2005-2009 5-year ACS data, endyear would be 2009)
`span`	an integer (should be 1, 3, or 5), or "auto" to have read.acs guess the span from the filename (e.g., for data from the 2005-2009 5-year ACS data, span would be 5)
`col.names`	a vector of column names to be used as `acs.colnames` for the object; defaults to "auto", which will result in auto-generated names from the headers lines of the input file
`acs.units`	a vector of factors indicating what sort of data is contained within each column of data ("count","dollars","proportion", "ratio", "other")
`geocols`	a vector of integers indicating which columns contain the geographic header information; defaults to "auto", which is the same as 3:1, which seems to be the standard for FactFinder-2 downloads
`skip`	an integer indicating how many rows to skip before processing the csv file; defaults to "auto", which will try to guess the proper value

Details

After executing a query on the U.S. Census American FactFinder site (http://factfinder2.census.gov), users can download their results as a zip file containing data in comma-delimited file format (for example, "ACS_10_5YR_B19013_with_ann.csv"). `read.acs` simplifies the creation of new acs objects from these files. The function uses some rudimentary algorithms to guess intelligently about values for metadata (such as `endyear` and `geography`), based on current file-format used by the Census "AmericanFactFinder 2" download site.

The specified `filename` can be an actual `.csv` file, or can be the name of a `.zip` file downloaded from the FactFinder site. If the latter, `read.acs` will extract the necessary data and leave the compressed zipfile in place.

As a default, `read.acs` assumes the first three columns will contain geographic header information, which seems to be the standard for the new Census American Factfinder download site. Users can also set different values for the `geocols=` to specify other columns for this geographic information. The function will use the first of these columns for geographic rownames to label estimates. (By default, then, this would be the third column of the actual file, since `geocols=3:1`. For files downloaded via the Census "legacy" version of FactFinder prior to 2012, users will probably want to specify `geocols=4:1`.

As for column names, by default `read.acs` will scan the file to determine how many of the initial rows contain "header" information, and will generate new `acs.colnames` by concatenating information found in these rows. Note that this can result in *very long* variable names, and users may want to modify the contents of `acs.colnames` after creation.

Alternatively, users can inspect downloaded csv files prior to import and specify the `skip=` option explicitly, as with `read.csv` and other `read.XXX` functions (i.e., the value of skip is equal to the number of rows prior to the last header row). Regardless of whether `skip=` is set or "auto", however, the column names will be created using all of the rows at the top of the file, *even the "skipped" ones*.

Finally, these new `acs.colnames` are used to guess intelligently about values for `acs.units`, but currently all this includes is a check for the word "dollars" in the names; if this is not found, the columns are assumed to be "counts".

When no other values are provided, `read.acs` will attempt to determine `endyear` and `span` from the filename.

Value

Returns a new acs-class object with estimates, standard errors (derived from the census 90% margins of error), and metadata associated with the survey,

Author(s)

Ezra Haber Glenn <eglenn@mit.edu>

sum-methods *acs Methods for Function* sum

Description

Returns the sum of all the estimates present in its arguments, along with proper treatment of standard errors.

Usage

```
## S4 method for signature 'acs'
sum(x, agg.term=c("aggregate", "aggregate"),
one.zero=FALSE, ..., na.rm=FALSE)
```

Arguments

x	the acs object to be summed
agg.term	a character vector (length 1 or 2) of labels to use for the geography or acs.colnames of the new object
one.zero	a logical flag indicating whether to include standard errors for only one zero-value estimates or all (the default); see details.
...	reserved for other arguments to pass
na.rm	whether to remove NAs from the values before summing; defaults to FALSE.

Details

Note: when aggregating ACS data, users may want to sum many fields with "0" values for estimates, especially when working with small geographies or detailed tables that split the population into many categories. In these cases, some analysts have suggested that the traditional summation procedure for standard errors (taking the square-root of the sum of the squares of the errors) may over-inflate the associated margins of error; instead, they recommend an alternative method, which ignores all but the single largest of the standard errors for any "zero-estimate" fields. Although this is somewhat unconventional, it is provided as an additional user-specified option here, through the "one.zero" argument.

Methods

`signature(object = "acs")` When passed an acs object (possibly involving subsetting), sum will return a new acs object created by aggregating (adding) all estimates in the object, and adding the corresponding standard errors in a statistically appropriate way. (Aggregate standard errors are computed by taking the square root of the sum of the squared standard errors of the terms to be aggregated.)

If the original object contains a single row, the geographic metadata and row name is preserved; if not, the geographic metadata is replaced with the term "aggregate" (or the contents of the first item of the (vector) option agg.term).

If the original object contains a single column, the column names and acs.units data are preserved; if not, the column names are replaced with the term "aggregate" or the contents of the second item of the (vector) option `agg.term`; note: if `agg.term` is only one item in length, it will be repeated here if needed.

All other acs-class metadata is preserved, except for the `modified` flag, which is set to TRUE.

Examples

```
# load ACS data
data(kansas09)

# aggregate the third column, all rows
sum(kansas09[,3])

# aggregate the fifth row, all column
sum(kansas09[5,])

# aggregate all rows, columns 3 through 25, rename rows "Kansas" and columns "Total Males"
sum(kansas09[, 3:25], agg.term=c("Kansas","Total Males"))
```

Index