Package 'spew'

November 4, 2017

Type Package

Depends R (>= 2.10)

Title SPEW Framework for Generating Synthetic Ecosystems

Version 1.3.0

Date 2017-11-03

Author Lee F. Richardson, Shannon Gallagher, Samuel L. Ventura, William F. Eddy

Maintainer Lee F. Richardson <leerichardson2013@gmail.com>

Description Tools for generating synthetic synthetic using the SPEW (Synthetic Populations and Ecosystems of the World) framework.
We provide functions for the 'synthesis' step of the SPEW, which converts harmonized data into a synthetic ecosystem. We also provide functions for visualizing and summarizing synthetic ecosystems generated by SPEW.
For details see Gallagher, S., Richardson, L.F., Ventura, S.L., Eddy W.F. (2017) <arXiv:1701.02383>.

License GPL-3

URL http://github.com/leerichardson/spew

BugReports http://github.com/leerichardson/spew/issues

RoxygenNote 6.0.1

VignetteBuilder knitr

Suggests testthat, knitr, rmarkdown, doParallel, foreach, stringdist, ggmap, rgeos, Rmpi, mipfp, quadprog, data.table, rgdal, ggplot2, reshape2, devtools, parallel

Imports sp, maptools, plyr, methods

NeedsCompilation no

Repository CRAN

Date/Publication 2017-11-03 23:28:38 UTC

R topics documented:

`spew-package` *spew: an R package for generating synthetic ecosystems*

Description

spew: an R package for generating synthetic ecosystems

add_characteristic *Add a characteristic to an existing population*

Description

Add a characteristic to an existing population

Usage

```
add_characteristic(synth_pop_path, args, pop_type = "people",
  method = "demo_matching", doPar = FALSE)
```

Arguments

synth_pop_path	path to folder where the synth pop(s) is/are
args	list of additional arguments
pop_type	either "household" or "people" population or "both" for household characteristics joined to people
method	"demo_matching"
doPar	logical, default is FALSE

Value

a written synth pop with added characteristics

add_char_demo *Add characteristic by matching on demographics*

Description

Add characteristic by matching on demographics

Usage

```
add_char_demo(synth_pop_fn, output_path, pop_type, args)
```

Arguments

synth_pop_fn	full name of the synth pop
output_path	the path to the folder of where the new output is to go
pop_type	"b" for both; "household", or "people" populations
args	list of additional arguments including the suffix

Value

logical, writes out new synth pop)

align_pums *Match the pums variables with marginal totals*

Description

Match the pums variables with marginal totals

Usage

```
align_pums(pums, marginals, suffix = "_marg")
```

Arguments

pums	dataframe subsetted to only the marginal vars
marginals	list containing all of the marginal totals
suffix	what we add to the variable name to create the new variable name. Default is "_marg"

Value

pums dataframe with the marginal columns binded on

allocate_count *Re-allocate excess counts to other locations*

Description

Re-allocate excess counts to other locations

Usage

```
allocate_count(counts, count_id)
```

Arguments

counts	numeric vector of current counts
count_id	numeric index indicating which count is excess

Value

new_counts a new numeric count vector with the

assign_place *Assign a place to a person*

Description

Assign a place to a person

Usage

```
assign_place(people, data_list)
```

Arguments

people	data frame of synthetic people produced by SPEW
data_list	of the data and identifying name

Value

column corresponding to the people of the place assignment

assign_place_coords *Assign a place with long/lat coords to a synthetic population*

Description

Assign a place with long/lat coords to a synthetic population

Usage

```
assign_place_coords(pop, places, place_name = "place", method = "uniform",
  dist_fxn = euclidean_dist, cap_fxn = cap_default)
```

Arguments

pop	data frame with "longitude" and "latitude" columns
places	data frame with "longitude" and "latitude" and an "ID" column, perhaps a "capacity" column
place_name	string that will become the column name of the place
method	c("uniform", "capacity") The method on how we assign places. The "uniform" argument means that we do not consider capacity in assignments, only distance. Conversely, "capacity" means we do consider capacity when assigning agents to places. The default is "uniform".
dist_fxn	a function with args x1, y1, x2, y2 that returns a single number. The default is Euclidean Distance $d((x1, y1), (x2,y2) = sqrt(x1-x2)^2 + (y1 - y2)^2). The distance should satisfy the requirements of a metric.
cap_fxn	a function with one argument, a single capacity. This should be a monotone increasing function. The default is cap_default.

Value

data frame with column of place_name with the place_ids, e.g. assignments of the places to the agents.

`assign_schools` *Assign schools to a synthetic population.*

Description

Assign schools to a synthetic population.

Usage

```
assign_schools(people, schools, weightSchools = weight_dists,
  distFun = haversine)
```

Arguments

`people`	data frame of synthetic people
`schools`	list with public and private school data frames
`weightSchools`	Function to weight the schools
`distFun`	Function to compute distance between schools and agents

Value

a column of school assignments by school ID

References

See PUMS CODEBOOK: http://www2.census.gov/programs-surveys/acs/tech_docs/pums/data_dict/PUMS_Data_Dictionar
2014.pdf

`assign_schools_inner` *Function which assigns schools*

Description

Function which assigns schools

Usage

```
assign_schools_inner(df, schools, weightSchools, distFun)
```

Arguments

df	subset of people split so all age, grade, SCH, and county should be the same in the df
schools	list of schools, one data frame of private and one of public
weightSchools	Function to weight the schools
distFun	Function to compute distance between schools and agents

Value

column of school ID assignments

---

assign_weights *Assign weights for ipf-based sampling*

Description

Assign weights for ipf-based sampling

Usage

```
assign_weights(pums, table, alpha, k, marginals)
```

Arguments

pums	dataframe with marginal columns
table	dataframe with marginal combinations and frequencies
alpha	number between 0 and 1, weight of categorical variables
k	number between 0 and 1, weight of orginal variables
marginals	list with marginal data

Value

weight matrix, rows = rows in pums, columns = rows in table

assign_workplaces *Assign an ESRI workplace to synthetic population*

Description

We look at the ESR variable from the synthetic people, which is the employment status recode. If values 1, 2, 4, and 5 are considered employed. We then randomly assign the person inside the county, with weights based on numbers of workers at the place. We return the ESRI ID of the workplace Note that in this way, workers do not cross borders.

Usage

```
assign_workplaces(people, workplaces)
```

Arguments

people	data frame of synthetic people produced by SPEW
workplaces	data frame of workplaces from ESRI

Value

column of workplace IDs for synthetic people

assign_workplaces_inner *Function which assigns workplaces*

Description

Function which assigns workplaces

Usage

```
assign_workplaces_inner(df, workplaces)
```

Arguments

df	subset of people with emp either 0 or 1 and the county number
workplaces	dataframe or ESRI schools

Value

ID of ESRI workplace or NA

base_map_theme *The base map theme for SPEW*

Description

The base map theme for SPEW

Usage

```
base_map_theme()
```

Value

base map theme

calc_dists *Calculate distance b/w cont table row and pums*

Description

Calculate distance b/w cont table row and pums

Usage

```
calc_dists(pums, table_row, alpha, k, marginals)
```

Arguments

pums	dataframe with marginal columns
table_row	row of table to calculate distance
alpha	number between 0 and 1, weight of categorical variables
k	number between 0 and 1, weight of orginal variables
marginals	list with marginal data

`call_spew` *Wrapper for reading, formatting, and writing SPEW ecosystems*

Description

Generates SPEW synthetic ecosystems on the Olympus Computing Cluster

Usage

```
call_spew(input_dir, output_dir, folders = NULL, data_group = "US",
  run_type = "SEQ", sampling_method = "uniform",
  locations_method = "uniform", output_type = "write",
  convert_count = FALSE, vars = list(household = NA, person = NA),
  road_noise = 2e-04, outfile_loc = "", timer = TRUE, verbose = TRUE)
```

Arguments

`input_dir`	character specifying input directory
`output_dir`	charcater specifying output directory
`folders`	list specifying sub-directories for each data-source
`data_group`	character, either "US", "ipums", or "none"
`run_type`	Whether to run sequentially in parallel. Default is "SEQ", for a sequential run. If parallel, back-end is either "MPI", "SOCK", or "MC"
`sampling_method`	character indicating the type of sampling method to use, defaults to "uniform". Can also be "ipf" with appropriate marginal data.
`locations_method`	character indicating the type of location sampling to use, defaults to "uniform", can also be "roads".
`output_type`	Default is "console" if we want to resulting population as an R variable. Alternative is "write", which is used on Olympus for writing out .csv files of the population
`convert_count`	logical meant to indicate if we are going to convert population totals from people to household counts. Default: FALSE, assumes the population is the total number of households
`vars`	list with two elements: household and person. This specifies which variables to include in the corresponding household and person PUMS data-set
`road_noise`	Noise added to households during road-based sampling
`outfile_loc`	Defaults to "", so we print out the parallel run information. Only set to "/dev/null" for internal testing putposes.
`timer`	logical indicating we want to time the run
`verbose`	logical indicating we want to print output during the run. Default is FALSE.#'

Value

logical indicating whether or not this run of spew ended successfully

---

cap_default *How to weight the capacities of of school.*

Description

How to weight the capacities of of school.

Usage

```
cap_default(cap)
```

Arguments

cap	a single number

Details

this default version returns the ceiling(capacity / 10).

Value

the weighted capacity, a single number.

---

ccount *Adjust number of households*

Description

Only changes things when population total represents the number of people (instead of households)

Usage

```
ccount(convert_count = FALSE, persons, np, nh, n_house)
```

Arguments

convert_count	logical, TRUE means make the switch
persons	vector of number of persons in each household
np	rows of person pums
nh	rows of household pums
n_house	original count of number of households

Value

updated n_house

checkDF *Check if df is in the right format*

Description

Check if df is in the right format

Usage

```
checkDF(df, type = "coords")
```

Arguments

df	data frame
type	("coords"). For type "coords", we check to make sure "longitude" and "latitude" are column names

Value

logical

check_logfile *Check to see if a SPEW log-file is complete*

Description

Check to see if a SPEW log-file is complete

Usage

```
check_logfile(logfile)
```

Arguments

logfile	character vecor of a SPEW Log-file

Value

character string indicating the status of the SPEW log-file

check_path *Check the path to output to run diags*

Description

Check the path to output to run diags

Usage

```
check_path(output_dir)
```

Arguments

output_dir	directory to TOP level region

Value

logical

check_place_ids *Check the Place ID's match*

Description

Check the Place ID's match

Usage

```
check_place_ids(id1, id2)
```

Arguments

id1	character vector of place_ids
id2	character vector of place_ids

Value

Either "Place ids match!" or an error detailing why

check_pop_table *Check the pop_table has all the necessary components*

Description

Check the pop_table has all the necessary components

Usage

```
check_pop_table(pop_table)
```

Arguments

pop_table	dataframe where rows correspond to places where populations should be generated. Other requird columns are "n_house" and "puma_id"

Value

Either a character "Pop table is ready!", or an error message detailing what went wrong

check_puma_ids *Check the puma id's match*

Description

Check the puma id's match

Usage

```
check_puma_ids(pop_table_ids, pums_ids)
```

Arguments

pop_table_ids	character vector of ids
pums_ids	character vector of ids

Value

Either "Puma ids match!" or an error detailing why

check_pums *Check that the pums has all the required components*

Description

Check that the pums has all the required components

Usage

```
check_pums(pums)
```

Arguments

pums	list with the household and person level pums

Value

Either a character "Pums is ready!" or an error message detailing what went wrong

check_shapefile *Check the shapefile has the necessary components*

Description

Check the shapefile has the necessary components

Usage

```
check_shapefile(shapefile)
```

Arguments

shapefile	sp class object used for assigning households to particular locations

Value

Either a character"Shapefile is ready!" or an error message detailing what went wrong

check_var_names *Check to see if variable names are in SPEW outputs*

Description

Check to see if variable names are in SPEW outputs

Usage

```
check_var_names(header_h, header_p, vars_to_sum_h, vars_to_sum_p,
  vars_to_sum_env = NULL)
```

Arguments

header_h	character vector of household variables in household SPEW output
header_p	character vector of people variables in household SPEW output
vars_to_sum_h	character vector of household variables we want to summarize
vars_to_sum_p	character vector of people variables we want to summarize
vars_to_sum_env	character vector of variables from the person data frame which correspond to environment assignments. Default is NULL

Value

logical

clean_names *Remove whitespace, capitals, and non ASCII*

Description

Remove whitespace, capitals, and non ASCII

Usage

```
clean_names(names)
```

Arguments

names	character vector of names to clean

Value

names character vector of all lowercase, non-capital and ASCII names

combine_counts *Combine two rows of a pop_table into one*

Description

Combine two rows of a pop_table into one

Usage

```
combine_counts(pop_table, place1, place2)
```

Arguments

pop_table	dataframe to update
place1	character indicating the name of the place to add counts to
place2	character indicating the name of the place to remove

Value

pop_table a data-frame with the place2 counts added to place1, and place2 removed from the pop_table

create_column *Parse a SPEW Log-file to into an appropriate column*

Description

Parse a SPEW Log-file to into an appropriate column

Usage

```
create_column(logfile, name, type = "numeric")
```

Arguments

logfile	character vector of a SPEW log-file
name	character with the name of the column to extract from the SPEW log-file
type	default is "numeric". Determines whether or not the final lgogfile should be converted to a numeric.

delaware *Input data for Keny County, Delaware*

Description

Input data for Keny County, Delaware

Usage

```
delaware
```

Format

An object of class `list` of length 8.

demo_sample *Sample extra characteristics from char pums and add them to the pop df*

Description

Sample extra characteristics from char pums and add them to the pop df

Usage

```
demo_sample(pop_df, char_pums, var_names, args = NULL)
```

Arguments

`pop_df`	a df that has been subsetted
`char_pums`	data frame of PUMS characteristics to add
`var_names`	variables we want to match on
`args`	list of additional arguments

`euclidean_dist` *Get the euclidean distance between two points (x1, y1) and (x2, y2)*

Description

Get the euclidean distance between two points (x1, y1) and (x2, y2)

Usage

```
euclidean_dist(x1, y1, x2, y2)
```

Arguments

`x1`	longitude of object 1 (vector)
`y1`	latitude of object 1 (vector)
`x2`	longitude of object 2 (vector)
`y2`	latitude of object 2 (vector)

Value

numeric

`extract_st_co_tr` *Extract the state, county, and tract ID from a string*

Description

Extract the state, county, and tract ID from a string

Usage

```
extract_st_co_tr(files)
```

Arguments

`files`	character vector of the files

Value

a data frame with the state, county, and tract IDs, along with the aggregated ID

`extrapolate_probs_to_pums` *Take unique probabilites for table and spread them to rest of PUMS*

Description

Take unique probabilites for table and spread them to rest of PUMS

Usage

```
extrapolate_probs_to_pums(p, n, pums, var_name)
```

Arguments

p	probabilities from solve.QP of length(unique(pums[, var_name]))
n	vector with sorted unique categories. p and n correspond to one another
pums	PUMS data frame
var_name	varaible name we are matching on

Value

probabilities for whole data frame

`extrapolate_probs_to_pums_joint` *Take unique probabilites for table and spread them to rest of PUMS*

Description

Take unique probabilites for table and spread them to rest of PUMS

Usage

```
extrapolate_probs_to_pums_joint(p, n, pums, var_names, tab)
```

Arguments

p	probabilities from solve.QP of length(unique(pums[, var_name]))
n	matrix with unique categories p and n correspond to one another
pums	PUMS data frame
var_names	varaible name we are matching on
tab	data frame with categories and frequency

Value

probabilities for whole data frame

fill_cont_table *Fill marginal contingency with ipf*

Description

Fill marginal contingency with ipf

Usage

```
fill_cont_table(pums, marginals, place_id, n_house)
```

Arguments

pums	dataframe of aligned pums data
marginals	list with marginal totals
place_id	ID saying which column to extract the marginals from
n_house	number of households to sample

Value

table a data-frame containing all of the marginal combinations and their frequencies

fips_to_name *Translate FIPS number to place name*

Description

Translate FIPS number to place name

Usage

```
fips_to_name(fips, level, df)
```

Arguments

fips	string – US FIPS code length 2 for state or length 5 for county
level	either "state" or "county"
df	a data frame table to translate FIPS to placename. It must have column names STATEFP, STATE_NAME, COUNTYFP, and County.

Value

the placenames corresponding to each FIPS #

format_data *Format data before entering make*

Description

Format data before entering make

Usage

```
format_data(data_list, data_group, verbose = TRUE)
```

Arguments

data_list	list which contains all of the data from the read_data function
data_group	character vector indiciating which group the data is located in
verbose	whether to print out the timings

Value

data_list list with an updated pop_table element which indicates the places in which we will generate synthetic ecosystems. The table should include three columns: the place_id, number of households to sample, and the puma id. Note the the place_id should correspond to the place_id from the shapefile

get_base_map *Get the base map for plotting*

Description

Get the base map for plotting

Usage

```
get_base_map(coords_df, get_world_map = TRUE, f = 0.1)
```

Arguments

coords_df	data frame with "longitude", "latitude", and "region_id" columns
get_world_map	logical. Get an underlying map from ggmap. Default is TRUE
f	border size of map

Value

a ggplotting object

get_centers *Get the center longitude and latitude for each region*

Description

Get the center longitude and latitude for each region

Usage

```
get_centers(coords_df)
```

Arguments

coords_df	data frame with "longitude", "latitude", and "region_id" columns

Value

data frame with mean lon/lat and region

get_coords_scaled *Getting a plotting data frame*

Description

Getting a plotting data frame

Usage

```
get_coords_scaled(sum_list, samp_size, coords, pop_totals)
```

Arguments

sum_list	output for people from summarize_spew
samp_size	how many records to keep for plotting
coords	logical
pop_totals	number of people to sample from each region

Value

data frame for plotting, longitude and latitude coordinates along with region

get_data_group *Extract data-group from location name*

Description

Extract data-group from location name

Usage

```
get_data_group(location_name)
```

Arguments

location_name	name character vector

get_dfs *Get the dataframes from SPEW summary output*

Description

Get the dataframes from SPEW summary output

Usage

```
get_dfs(sum_list, vars_to_sum, has_marg = FALSE)
```

Arguments

sum_list	from summarize_spew
vars_to_sum	variables to make data frames of
has_marg	logical. Does the synthetic ecosystem have marginals to look at? Default is FALSE

Details

the output is very gg-verse friendly

Value

list of data frame for each variable

get_dists *Get the distances between the schools and the people.*

Description

Get the distances between the schools and the people.

Usage

```
get_dists(df, schools, dist)
```

Arguments

df	data frame of people
schools	dataframe of subsetted schools
dist	"haversine"

Value

distance between two points on the globe

get_dist_mat *Get the distance matrix*

Description

Get the distance matrix between two data frames that have "longitude" and "latitude" columns

Usage

```
get_dist_mat(pop, places, dist_fxn = haversine_dist)
```

Arguments

pop	data frame of the population with m rows
places	data frame of places to assign with n rows
dist_fxn	currently "haversine" with args for x1, y1, x2, y2, returning a scalar value between 0 and 1

Value

m x n matrix with scaled distance between 0 and 1. Eg. Entry ij means that the scaled distance between row i from pop and row j from places is entry ij.

get_envs *Gather the unique assignments for the region*

Description

Gather the unique assignments for the region

Usage

```
get_envs(p_sum_list, env_vars = NULL)
```

Arguments

p_sum_list	output from summarize_spew
env_vars	the environment variables we want to summarize. Default is NULL

Value

a list of unique assignments for each of the env_variables

get_filenames *Get the filenames of the SPEW output, separated by the level*

Description

Get the filenames of the SPEW output, separated by the level

Usage

```
get_filenames(output_dir, summary_level = 2, agent_type = "household",
  pop_type = "US")
```

Arguments

output_dir	path to top level directory of SPEW folders. Ex. "./10" for Delaware
summary_level	For the US, 1-state, 2-county, 3-tract. For IPUMS, 1 -country, 2-province. For "custom," these are defined by the user's input data.
agent_type	either "household" or "people"
pop_type	"US" for a US population, "IPUMS" for IPUMS population, or "custom" for a custom population. This effects what the summary levels represent

Value

list of lists where the first list is a list of the different filenames for each summary level and the second list contains a dataframe of the different file paths along with the name of the larger, aggregated region

get_header *Extract the header from a population*

Description

Extract the header from a population

Usage

```
get_header(output_dir, type = "household")
```

Arguments

output_dir	directory to TOP level region
type	either "household" or "people"

Value

a character vector of the header

get_level *Obtain the correct level for ipums data*

Description

Obtain the correct level for ipums data

Usage

```
get_level(shapefile_names, pop_table)
```

Arguments

shapefile_names	character vector of the names which come from the given shapefile
pop_table	data frame in which we want to obtain the levels

Value

level character of the level in which we will use

get_pop_totals *Get the population totals from a summarized SPEW region*

Description

Get the population totals from a summarized SPEW region

Usage

```
get_pop_totals(hh_sum_list, p_sum_list)
```

Arguments

hh_sum_list	output from summarize_spew for households
p_sum_list	output from summarize_spew for people

Value

list of a data frame of population totals, one record for each region, number of houses, and number of people

get_rows *Extract rows with a certain character*

Description

Extract rows with a certain character

Usage

```
get_rows(dat, char)
```

Arguments

dat	character vector
char	character which specifies the sequence we are looking for

Value

output which is the subsetted rows of the initial character vector (logfile)

get_shapefile_indices *Obtain the shapefile indices corresponding to the pop table*

Description

Obtain the shapefile indices corresponding to the pop table

Usage

```
get_shapefile_indices(shapefile_names, count_names)
```

Arguments

shapefile_names	character vector with the shapefile names
count_names	character vector with the count names

Value

numeric vector indicating the appropriate indices for shapefiles which correspond to the count_names

get_targets *Obtain the target marginals for IPF*

Description

Obtain the target marginals for IPF

Usage

```
get_targets(marg_cols, marginals, place_id, n_house)
```

Arguments

marg_cols	dataframe of marginal columns
marginals	list of marginal data
place_id	id corresponding to the rows of marginals
n_house	number of households in this sample

Value

list with target_data and target list for this particular place_id

get_total_time *Extract the total run-time from a SPEW log-file*

Description

Extract the total run-time from a SPEW log-file

Usage

```
get_total_time(logfile)
```

Arguments

logfile	spew log-file created on olympus

get_weight_dists *Weight place assignment probabilities*

Description

Weight place assignment probabilities

Usage

```
get_weight_dists(dist_mat, places, method = "uniform",
  cap_fxn = cap_default)
```

Arguments

dist_mat	a m x n matrix where m is the number of people and n is the number of schools
places	data frame of places with an ID column
method	("uniform", "capacity")
cap_fxn	a function with one argument, a single capacity. This should be a monotone increasing function. The default is cap_default.

Value

m x n matrix of probabilities. Each row should sum to 1

`haversine` *Get the haversine distance between two points (x1, y1) and (x2, y2) scaled between 0 and 1.*

Description

Get the haversine distance between two points (x1, y1) and (x2, y2) scaled between 0 and 1.

Usage

```
haversine(x1, y1, x2, y2)
```

Arguments

x1	longitude of object 1 (vector)
y1	latitude of object 1 (vector)
x2	longitude of object 2 (vector)
y2	latitude of object 2 (vector)

Value

numeric

References

http://andrew.hedges.name/experiments/haversine/

`haversine_dist` *Get the haversine distance between two points (x1, y1) and (x2, y2) scaled between 0 and 1.*

Description

Get the haversine distance between two points (x1, y1) and (x2, y2) scaled between 0 and 1.

Usage

```
haversine_dist(x1, y1, x2, y2)
```

Arguments

x1	longitude of object 1 (vector)
y1	latitude of object 1 (vector)
x2	longitude of object 2 (vector)
y2	latitude of object 2 (vector)

Value

numeric

References

http://andrew.hedges.name/experiments/haversine/

`impute_missing_vals` *Impute Missing Values in a data frame*

Description

Impute Missing Values in a data frame

Usage

```
impute_missing_vals(df, method = "mean")
```

Arguments

`df`	where ever column may be imputed (so each column is numeric)
`method`	of imputation, "mean" imputes the mean of the other values into the NA values. "bootstrap" resamples from non-NA vals.

Value

df of same dimension, now with no missing vals

`make_ipf_obj` *Set up for creating a set of marginal information for IPF sampling*

Description

Set up for creating a set of marginal information for IPF sampling

Usage

```
make_ipf_obj(var_name, type = "ord", bounds, category_name,
  output_file = NULL, df = NULL)
```

Arguments

var_name	name of the variable matching the PUMS/microdata
type	"ord" for ordinal or "cat" for categorical, the type of variable to be converted to
bounds	dataframe of upper and lower bounds (inclusive) for variables (numeric)
category_name	short name of the category, will be visible to person. Either length one and the bounds will be pasted to it or length of the number of rows of the bounds with names of your choice.
output_file	if not NULL then we save the file as a RDS object to output_file
df	data frame with place_id and category counts

Value

a list in the format for use in IPF

---

make_mm_obj *Make moment matching object*

Description

Make moment matching object

Usage

```
make_mm_obj(moments_list, assumption = "independence", nMom = 1,
  type = "cont", region = NULL, path = NULL)
```

Arguments

moments_list	a list with each entry as a data frame. The first df is the first moments, the second the second moments, etc. Each df has the following format: place_id | puma_id | var1 moment | var 2 moment |. The dfs are named mom1, mom2, ...
assumption	is either "independent" or "joint". "independent" assumes that the distributions of the characteristics are independent of one another. "joint" means we use the empirical distribution of the microdata when finding weights in moment matching.
nMom	number of moments. Currently, we only support the first moment, e.g. the average of a variable. Default is 1.
type	either "cont" for continuous variable or "ord" for ordered variable
region	identifier for region
path	if not NULL we will save this object to the specified path as a RDS object.

Value

list of moment object

merge_reduce *Wrapper function for merge*

Description

Wrapper function for merge

Usage

```
merge_reduce(x, y)
```

Arguments

x	x component of merge
y	y component of merge

Value

a function

Note

This sets the merge default to all = TRUE. To use in the Reduce function

organize_summaries *Organize the summaries into a more palatable format*

Description

Organize the summaries into a more palatable format

Usage

```
organize_summaries(hh_sum_list, p_sum_list, header_h, header_p, vars_to_sum_h,
  vars_to_sum_p, env_vars, samp_size, top_region_id, coords = TRUE,
  has_marg = FALSE)
```

Arguments

hh_sum_list	output of summarize_spew for regions for households
p_sum_list	output of summarize_spew for regions for people
header_h	character vector of all variables in SPEW households
header_p	character vector of all variables in SPEW people
vars_to_sum_h	character vector of household variables we want to summarize

vars_to_sum_p	character vector of people variables we want to summarize
env_vars	variables to summarize
samp_size	total number sampled in each region for plotting
top_region_id	main region's ID or name
coords	logical of whether to extract the longitude/latitude coordinates and store. Default is TRUE
has_marg	logical. Does the synthetic ecosystem have marginals to look at? Default is FALSE

Value

list including the top region name, headers for households and people, a coordinate plotting data frame scaled to the density of the population with max samp_size records, and data frames of tables for each region for each of the characteristics

---

`people_to_households` *Convert a population count to household count*

Description

Convert a population count to household count

Usage

```
people_to_households(hh_sizes, n_people)
```

Arguments

hh_sizes	numeric vector with the household sizes for this particular sampling region
n_people	numeric indicating the number of people in this specific region

Value

number of households

plot_agents *Plot the agents of synthetic ecosystem*

Description

Plot the agents of synthetic ecosystem

Usage

```
plot_agents(syneco, input_data, g = NULL, color_list = list(bds = "white",
  interior = "gray60", roads = "gray10", agents = "darkorchid3", envs =
  c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
  "#CC79A7")))
```

Arguments

syneco	the outputted synthetic ecosystem data from spew
input_data	a list of the essential data and possibly supplementary data, shapefile must be one of the names
g	a ggplot. Default is NULL.
color_list	optional list of colors to provide to the synthetic ecosystem. This must be a list with the following components: "bds", "interior", "roads", "agents", "envs" where each entry in the list is a color or vector of colors

Value

a ggplot of the region

plot_bds *Plot the boundaries of the synthetic ecosystem*

Description

Plot the boundaries of the synthetic ecosystem

Usage

```
plot_bds(shapefile, g = NULL, color_list = list(bds = "white", interior =
  "gray60", roads = "gray10", agents = "darkorchid3", envs = c("#E69F00",
  "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")))
```

Arguments

shapefile	a fortified shapefile object, ready for plotting in ggplot.
g	a ggplot. Default is NULL.
color_list	optional list of colors to provide to the synthetic ecosystem. This must be a list with the following components, "bds", "interior", "roads", "agents", "envs" where each entry in the list is a color or vector of colors

Value

a ggplot of the region

---

plot_characteristic_proportions

Plot characteristic summary output from summarize_top_spew_region

---

Description

Plot characteristic summary output from summarize_top_spew_region

Usage

```
plot_characteristic_proportions(feature_name = "Feature",
  legend_name = "Types", feature_df, category_names = NULL,
  text_size = 10, region_colors = c("#999999", "#E69F00", "#56B4E9",
  "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7"))
```

Arguments

feature_name	string which will be in the title
legend_name	name of the legend title
feature_df	features summary, output from summarize_top_region()
category_names	optional labels to display as the category types. Default is whatever is contained in the feature_df.
text_size	axis text size. Default is 10
region_colors	a string of colors to color the map. Default is from the colorblind friendly palette.

Value

a gg map object

plot_env *Plot the environments of the synthetic ecosystem*

Description

Plot the environments of the synthetic ecosystem

Usage

```
plot_env(input_data, g = NULL, color_list = list(bds = "white", interior =
  "gray60", roads = "gray10", agents = "darkorchid3", envs = c("#E69F00",
  "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")))
```

Arguments

input_data	a list of the essential data and possibly supplementary data, shapefile must be one of the names
g	a ggplot. Default is NULL.
color_list	optional list of colors to provide to the synthetic ecosystem. This must be a list with the following components, "bds", "interior", "roads", "agents", "envs" where each entry in the list is a color or vector of colors

Value

a ggplot of the region

plot_interior *Plot the interior of the synthetic ecosystem*

Description

Plot the interior of the synthetic ecosystem

Usage

```
plot_interior(shapefile, g = NULL, color_list = list(bds = "white", interior
  = "gray60", roads = "gray10", agents = "darkorchid3", envs = c("#E69F00",
  "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")))
```

Arguments

shapefile	a fortified shapefile object, ready for plotting in ggplot.
g	a ggplot. Default is NULL.
color_list	optional list of colors to provide to the synthetic ecosystem. This must be a list with the following components, "bds", "interior", "roads", "agents", "envs" where each entry in the list is a color or vector of colors

Value

a ggplot of the region

`plot_labs` *Add the labels and the theme to the plot*

Description

Add the labels and the theme to the plot

Usage

```
plot_labs(region_name, g = NULL)
```

Arguments

`region_name`	string, will become the title of the plot
`g`	a ggplot. Default is NULL.

`plot_pop_totals` *Plot characteristic summary output from summarize_top_spew_region as totals*

Description

Plot characteristic summary output from summarize_top_spew_region as totals

Usage

```
plot_pop_totals(feature_df, type = "n_people", text_size = 10,
  region_colors = c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442",
  "#0072B2", "#D55E00", "#CC79A7"))
```

Arguments

`feature_df`	features summary, output from summarize_top_region()
`type`	either "n_house" or "n_people". Default is n_people
`text_size`	axis text size. Default is 10
`region_colors`	a string of colors to color the map. Default is from the colorblind friendly palette.

Value

a gg map object

Examples

```
data(tartanville)

tartanville_syneco <- spew(tartanville$pop_table, tartanville$shapefile,
                           tartanville$pums_h, tartanville$pums_p)
plot_syneco(tartanville, tartanville_syneco, region_name = "Tartanville")
```

---

plot_region *Plot SPEW region*

Description

Plot SPEW region

Usage

```
plot_region(coords_df, get_world_map = FALSE, f = 0.1,
  region_colors = c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442",
  "#0072B2", "#D55E00", "#CC79A7"))
```

Arguments

coords_df	data frame with "longitude", "latitude", and "region_id" columns
get_world_map	logical. Get an underlying map from ggmap. Default is TRUE
f	border size between 0 and 1. Default is .1.
region_colors	a string of colors to color the map. Default is from the colorblind friendly palette.

Value

a ggmap object

---

plot_roads *Plot the roads of the synthetic ecosystem*

Description

Plot the roads of the synthetic ecosystem

Usage

```
plot_roads(roads, g = NULL, color_list = list(bds = "white", interior =
  "gray60", roads = "gray10", agents = "darkorchid3", envs = c("#E69F00",
  "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")))
```

Arguments

roads	a fortified shapefile object of the roads, ready for plotting in ggplot.
g	a ggplot. Default is NULL.
color_list	optional list of colors to provide to the synthetic ecosystem. This must be a list with the following components, "bds", "interior", "roads", "agents", "envs" where each entry in the list is a color or vector of colors

Value

a ggplot of the region

---

plot_syneco *Plot Synthetic Ecosystem*

Description

Plot Synthetic Ecosystem

Usage

```
plot_syneco(input_data, syneco, region_name = NULL, color_list = list(bds =
  "white", interior = "gray60", roads = "gray10", agents = "darkorchid3", envs =
  c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
  "#CC79A7")))
```

Arguments

input_data	a list of the essential data and possibly supplementary data, shapefile must be one of the names
syneco	the outputted synthetic ecosystem data from spew
region_name	string, will become the title of the plot
color_list	optional list of colors to provide to the synthetic ecosystem. This must be a list with the following components, "bds", "interior", "roads", "agents", "envs" where each entry in the list is a color or vector of colors

Value

plot of the synthetic ecosystem

Examples

```
data(tartanville)

tartanville_syneco <- spew(tartanville$pop_table, tartanville$shapefile,
                           tartanville$pums_h, tartanville$pums_p)

plot_syneco(tartanville, tartanville_syneco, region_name = "Tartanville")
```

print_region_list *Write out information on each region*

Description

Write out information on each region

Usage

```
print_region_list(region_list)
```

Arguments

region_list	a list containing all of the summary and diagnostic information for each

read_data *Read SPEW input data from files*

Description

Based on directory structure for input data on the Olympus computing cluster

Usage

```
read_data(input_dir, folders = list(pop_table = NULL, pums = NULL, shapefiles
  = NULL, roads = NULL, schools = NULL, lookup = NULL, workplaces = NULL),
  data_group = "US", vars = list(household = NA, person = NA),
  verbose = FALSE)
```

Arguments

input_dir	character vector specifying the ecosystem directory
folders	list containing the subdirectory file-path containing where each type of data is located
data_group	character either "US", "ipums" or "none": Gives the format that read_data expects.
vars	list with household and person variables specifying which PUMS variables to use
verbose	logical determining whether to write output to console

Value

list: Each elements contains a standardized data-source

read_marginals *Read in the marginals population characteristic totals*

Description

Read in the marginals population characteristic totals

Usage

```
read_marginals(input_dir, folders, data_group, sampling_method = "ipf")
```

Arguments

input_dir	character vector specifying the directory containing all of the input data
folders	list which contains the path of each sub-directory with the specific data
data_group	character either "US", "ipums" or "none" which tells read_data if the input data follows a particular format. Used mainly for the pre-formatted data-types we have on our Olympus
sampling_method	should be "ipf"

Value

data frame with counts

read_moments *Read in the R data object for moment matching*

Description

Read in the R data object for moment matching

Usage

```
read_moments(input_dir, folders, data_group)
```

Arguments

input_dir	character vector specifying the directory containing all of the input data
folders	list which contains the path of each sub-directory with the specific data
data_group	character either "US", "ipums" or "none" which tells read_data if the input data follows a particular format. Used mainly for the pre-formatted data-types we have on our Olympus

Value

moment object

read_pop_table *Read in the population counts*

Description

Read in the population counts

Usage

```
read_pop_table(input_dir, folders, data_group)
```

Arguments

input_dir	character vector specifying the directory containing all of the input data
folders	list which contains the path of each sub-directory with the specific datadelaware_marginals
data_group	character either "US", "ipums" or "none" which tells read_data if the input data follows a particular format. Used mainly for the pre-formatted data-types we have on our Olympus

Value

data frame with counts

read_roads *Read in road lines shapefiles*

Description

Read in road lines shapefiles

Usage

```
read_roads(path_to_roads, road_id)
```

Arguments

path_to_roads	full path to the directory of where the roads are stored, the directory should have zip files that have been unzipped for each county
road_id	ID of the tract the roads are in

Value

an appended SpatialDataFrame object with all the roads in the state

read_shapespatial_to_ogr *Read in shapefile using readOGR*

Description

Read in shapefile using readOGR

Usage

```
read_shapespatial_to_ogr(full_path)
```

Arguments

full_path	entire path to shapefile ending in .shp, i.e. what you would pass to readShapeSpatial

Value

the shapefile, class SpatialPolygonsDataFrame

Note

This is to keep up-to-date and use rgdal:readOGR instead of the now unsupported maptools::readShapeSpatial

remove_count *Remove a row from the pop_table*

Description

Remove a row from the pop_table

Usage

```
remove_count(pop_table, place)
```

Arguments

pop_table	dataframe to update
place	character of place_ids to remove

Value

pop_table updated dataframe with the desired place removed

remove_excess *Remove comma's, accents, etc. from name*

Description

Remove comma's, accents, etc. from name

Usage

```
remove_excess(name)
```

Arguments

name	character

Value

name with all of the excess baggage removed

remove_holes *Remove holes from an object of class Polygon*

Description

Remove holes from an object of class Polygon

Usage

```
remove_holes(polygon)
```

Arguments

polygon	object of class Polygon or Polygons

Value

polygon without and Polygons with holes

Note

Borrowed the idea from the wild1 package, which I wasn't able to load for R 3.2.2, so I found the source code here: https://github.com/cran/wild1/blob/master/R/remove.holes.r

remove_words *Remove excess words*

Description

Remove excess words

Usage

```
remove_words(word, names)
```

Arguments

word	character of the word you want to replace
names	character vector of the words we can potentially replace

Value

names character vector with the excess word removed

replace_word *Replace an existing word*

Description

Replace an existing word

Usage

```
replace_word(word, replace, names)
```

Arguments

word	character of the word you want to replace
replace	character of what you want to replace the word
names	character vector of words which we can replace

Value

names character vector of the replaced word

`sample_households` *Sample appropriate indices from household PUMS*

Description

Sample appropriate indices from household PUMS

Usage

```
sample_households(method, n_house, pums_h, pums_p = NULL, puma_id = NULL,
  place_id = NULL, marginals = NULL)
```

Arguments

`method`	character indicating the method for sampling
`n_house`	numeric indicating the number of households to sample
`pums_h`	dataframe of the households we are sampling from
`pums_p`	optionally include a dataframe of agents
`puma_id`	vector indicating which specific puma in PUMS we are sampling from, if any
`place_id`	unique code identifying the place
`marginals`	optionally used for IPF or MM sampling

Value

numeric with the indicies of the household PUMS to sample

Examples

```
data("tartanville")
example_place_id <- tartanville$pop_table$place_id[1]
example_puma_id <- tartanville$pop_table$puma_id[1]
example_n_house <- tartanville$pop_table$n_house[1]
sample_households(method = "uniform",
                  pums_h = tartanville$pums_h,
                  pums_p = tartanville$pums_p,
                  n_house = example_n_house,
                  place_id = example_place_id,
                  puma_id = example_puma_id)
```

sample_ipf *Sample households PUMS accoording to IPF*

Description

Sample households PUMS accoording to IPF

Usage

```
sample_ipf(n_house, pums_h, pums_p, marginals, alpha = 0, k = 0.001,
  puma_id = NULL, place_id = NULL)
```

Arguments

n_house	number of households to sample
pums_h	household pums
pums_p	people pums, for appending on certain variables
marginals	list with characteristics of marginals
alpha	number between 0 and 1, weight of categorical variables
k	number between 0 and 1, weight of orginal variables
puma_id	id indicating the current puma
place_id	id indicating the current region

Value

households data frame with re-sampled pums_h households

sample_locations *Generic sampling locations function*

Description

Generic sampling locations function

Usage

```
sample_locations(method, place_id, n_house, shapefile, noise = 0.001,
  shapefile_id = NULL)
```

Arguments

method	character vector either "uniform" or "roads" determining how we are sampling locations
place_id	numeric specifiying the ID of the region we are subsampling
n_house	numeric indicating the number of households
shapefile	sp class with all of the locations for each place id. Note that this is a list with two shapefiles if me
noise	the standard deviation of how much we jitter the road locations in each direction (only if method is "roads")
shapefile_id	optionally include a shapefile id for subsetting shapefile

Value

SpatialPoints object with coordinates for the n households

Examples

```
data("tartanville")
example_place_id <- tartanville$pop_table$place_id[1]
example_puma_id <- tartanville$pop_table$puma_id[1]
example_n_house <- tartanville$pop_table$n_house[1]
sample_locations(method = "uniform",
                 shapefile = tartanville$shapefile,
                 n_house = example_n_house,
                 place_id = example_place_id)
```

---

sample_locations_roads

Sample coordinates from roads

Description

Sample coordinates from roads

Usage

```
sample_locations_roads(place_id, n_house, shapefile, noise = 1e-04,
  shapefile_id)
```

Arguments

place_id	numeric specifiying the ID of the region we are subsampling
n_house	numeric indicating the number of households
shapefile	sp class with all of the locations for each place id. In addition, we must have road shapefiles so shapefile is a list with both the tracts and the roads, tracts is the first object and roads the second.
noise	the standard deviation of how much we jitter the road locations in each direction
shapefile_id	optionally include a shapefile id for subsetting shapefile

---

sample_locations_uniform

Sample from a particular polygon shapefile

Description

Sample from a particular polygon shapefile

Usage

```
sample_locations_uniform(place_id, n_house, shapefile, noise = 0.001,
  shapefile_id = NULL)
```

Arguments

place_id	numeric specifiying the ID of the region we are subsampling
n_house	numeric indicating the number of households
shapefile	sp class with all of the locations for each place id
noise	numeric indicating how must noise to add to sampled points This is only used if the number of points to sample exceeds 100,000, in which case the spsample function takes too long. Instead, we sample 100,000 points, sample n_house from these 100,000, and then add random noise.
shapefile_id	optionally include a shapefile id for subsetting shapefile

Value

SpatialPoints object with coordinates for the n households

sample_mm *Sample households PUMS according to MM*

Description

Sample households PUMS according to MM

Usage

```
sample_mm(n_house, pums_h, pums_p, mm_obj, puma_id = NULL, place_id = NULL)
```

Arguments

n_house	number of households to sample
pums_h	household pums
pums_p	people pums, for appending on certain variables
mm_obj	list with moment matching data
puma_id	id indicating the current puma
place_id	id indicating the current region

sample_people *Sample from the individual person PUMS data frame*

Description

Sample from the individual person PUMS data frame

Usage

```
sample_people(method, household_pums, pums_p, puma_id = NULL,
  place_id = NULL)
```

Arguments

method	character indicating the method for sampling
household_pums	dataframe with the sampled houehold PUMS
pums_p	dataframe of the individual microdata
puma_id	ID of microdata sampling region
place_id	ID of place

Value

people numeric vector indicating the indices of people to sample

sample_uniform *Sample households uniformly*

Description

Sample households uniformly

Usage

```
sample_uniform(n_house, pums_h, puma_id = NULL, place_id = NULL)
```

Arguments

n_house	number of households to sample
pums_h	the household pums
puma_id	if specifying the subset of PUMS to sample s
place_id	id of the current region

sample_with_cont *Sample from pums*

Description

Sample from pums

Usage

```
sample_with_cont(pums, table, alpha, k, marginals)
```

Arguments

pums	dataframe with marginal columns
table	dataframe with marginal combinations and frequencies
alpha	number between 0 and 1, weight of categorical variables
k	number between 0 and 1, weight of orginal variables
marginals	list with marginal data

Value

indices of sampled households

samp_roads *Sample the locations from a SpatialLines object*

Description

Sample the locations from a SpatialLines object

Usage

```
samp_roads(n_house, new_shp, noise)
```

Arguments

n_house	number of households
new_shp	SpatialLines or Spatial Points object
noise	std deviation of Gaussian noise added to coordinates, default is .001

Value

SpatialPoints object with coordinates for the n_house households

Note

When the class is Spatial Points, the following sampling method takes place: "When x is of a class deriving from Spatial-class for which no spsample-methods exists, sampling is done in the bounding box of the object, using spsample.Spatial" s This was added because the Puerto Rico intersection gave SpatialPoints

solve_mm_for_joint *Do the Moment Matching solving for joint distribution*

Description

Do the Moment Matching solving for joint distribution

Usage

```
solve_mm_for_joint(place_id, mm_row, pums, assumption, meq = 2)
```

Arguments

place_id	unique code identifying the place
mm_row	dataframe of 1 row, includes place_id, puma_id, and averages, with names matching PUMS
pums	the subsetted PUMS from which the sample will be drawn
assumption	"joint"
meq	(number of moments + 1) to match (2 is default and corresponds to matching the average)

Value

x vector of length of the number of records in the PUMS. These are probabilities for each of the records in the PUMS.

---

solve_mm_for_var *Do the MM solving for an individual variable*

Description

Do the MM solving for an individual variable

Usage

```
solve_mm_for_var(var_ind, place_id, mm_row, pums, assumption, meq = 2)
```

Arguments

var_ind	variable index of mm_row, colname should match one in PUMS.
place_id	unique code identifying the place
mm_row	dataframe of 1 row, includes place_id, puma_id, and averages, with names matching PUMS
pums	the subsetted PUMS from which the sample will be drawn
assumption	"independence"
meq	(number of moments + 1) to match (2 is default and corresponds to matching the average)

Value

x vector of length of the number of records in the PUMS. These are probabilities for each of the records in the PUMS.

solve_mm_weights — *Weight the records of the PUMS so the averages in mm_df will be obtained*

Description

Weight the records of the PUMS so the averages in mm_df will be obtained

Usage

```
solve_mm_weights(place_id, mm_row, pums, assumption = "independence",
  meq = 2)
```

Arguments

place_id	unique code identifying the place
mm_row	dataframe of 1 row, includes place_id, puma_id, and averages, with names matching PUMS
pums	the subsetted PUMS from which the sample will be drawn
assumption	"independence"
meq	(number of moments + 1) to match (2 is default and corresponds to matching the average)

Details

the function solve.QP from the "quadprog" package is used.

Value

x vector of length of the number of records in the PUMS. These are probabilities for each of the records.

spew — *SPEW algorithm to generate synthetic ecosystems*

Description

SPEW algorithm to generate synthetic ecosystems

Usage

```
spew(pop_table, shapefile, pums_h, pums_p, schools = NULL,
  workplaces = NULL, marginals = NULL, output_type = "console",
  output_dir = NULL, convert_count = FALSE, run_type = "SEQ",
  sampling_method = "uniform", locations_method = "uniform",
  outfile_loc = "", road_noise = 2e-04, timer = FALSE, verbose = FALSE)
```

Arguments

pop_table	dataframe where rows correspond to places where populations should be generated. Other requird columns are "n_house" and "puma_id"
shapefile	sp class object used for assigning households to particular locations
pums_h	dataframe with microdata corresponding to housegolds
pums_p	dataframe with microdata corresponding to people
schools	list with names "public" and "private" with a dataframe of schools corresponding to public or private, respectively
workplaces	dataframe of workplaces with a workplace_id column, employees column, and stcotr column
marginals	list of marginal population totals. Each element of the list contains the marginal totals of a separate variable
output_type	Default is "console" if we want to resulting population as an R variable. Alternative is "write", which is used on Olympus for writing out .csv files of the population
output_dir	character specifying ecosystem directory write to. Only used if output_type = "write"
convert_count	logical meant to indicate if we are going to convert population totals from people to household counts. Default: FALSE, assumes the population is the total number of households
run_type	Whether to run sequentially in parallel. Default is "SEQ", for a sequential run. If parallel, back-end is either "MPI", "SOCK", or "MC"
sampling_method	character indicating the type of sampling method to use, defaults to "uniform". Can also be "ipf" with appropriate marginal data.
locations_method	character indicating the type of location sampling to use, defaults to "uniform", can also be "roads".
outfile_loc	Defaults to "", so we print out the parallel run information. Only set to "/dev/null" for internal testing putposes.
road_noise	Noise added to households during road-based sampling
timer	logical indicating we want to time the run
verbose	logical indicating we want to print output during the run. Default is FALSE.

Value

logical indicating whether or not this run of spew ended successfully

Examples

```
# Call spew with default data from tartanville ---
data(tartanville)
tartanville_syneco <- spew(pop_table = tartanville$pop_table,
                           shapefile = tartanville$shapefile,
```

```
                          pums_h = tartanville$pums_h,
                          pums_p = tartanville$pums_p)

# Call spew with road-based location sampling ---
roads_shapefile <- list(boundaries = tartanville$shapefile,
                        roads = tartanville$roads)
tartanville_syneco_roads <- spew(tartanville$pop_table, roads_shapefile,
                                 tartanville$pums_h, tartanville$pums_p,
                                 locations_method = "roads", road_noise = .05)

# Call spew with ipf agent-sampling ---

# Household income marginal
var_name <- "HHINC"
type <- "ord"
bounds <- data.frame(lower = c(0, 50), upper = c(49, Inf))
category_name <- c("HHINC_0-49", "HHINC_50-Inf")
df <- data.frame(place_id = paste0("T", 1:9),  v1 = c(30, 0, 5, 10, 13, 9, 2, 1, 5))
df$v2 <- tartanville$pop_table$n_house - df$v1
ipf_obj_hhinc<- make_ipf_obj(var_name, type, bounds, category_name, df = df)
# Head of Household Race Marginal
var_name <- c("RAC1P")
type <- "cat"
bounds <- data.frame(lower = c(1, 2), upper = c(1, 2))
category_name <- c("Tartan", "Argyle")
df2 <- data.frame(place_id = paste0("T", 1:9),  v1 = c(28, 0, 4, 1, 5, 8, 2, 1, 3))
df2$v2 <- tartanville$pop_table$n_house - df2$v1
ipf_obj_rac1p <- make_ipf_obj(var_name, type, bounds, category_name, df = df2)
ipf_obj <- list(HHINC = ipf_obj_hhinc[[1]], RAC1P = ipf_obj_rac1p[[1]])
supplementary_data <- list(moments = ipf_obj)

tartanville_syneco_ipf <- spew(tartanville$pop_table, tartanville$shapefile,
                               tartanville$pums_h, tartanville$pums_p,
                                marginals = supplementary_data$moments,
                               sampling_method = "ipf")

# Call spew with moment-matching agent-sampling
NP_avg <- c(3.2, 0, 6.0, 2.0, 3.2, 3.1, 4.0, 4.8, 3.9)
supplementary_data <- list(moments = make_mm_obj(moments_list =
                           list(mom1 = data.frame(place_id = paste0("T", 1:9),
                                puma_id = "T", NP = NP_avg)),
                           assumption = "independence", nMom = 1, type = "cont"))
tartanville_syneco_mm <- spew(pop_table = tartanville$pop_table,
                              shapefile = tartanville$shapefile,
                              pums_h = tartanville$pums_h,
                              pums_p = tartanville$pums_p,
                              marginals = supplementary_data$moments,
                              sampling_method = "mm")
```

spewlog_to_df *Convert a SPEW Logfile into a data-frame*

Description

Convert a SPEW Logfile into a data-frame

Usage

```
spewlog_to_df(logfile, columns, path = NULL)
```

Arguments

logfile	character with the file-name
columns	character vector indicating the
path	path to the logfile (used for debugging)
	names of the columns we are extracting from the log-file

spew_mc *Run SPEW in Parallel with a Multicore backend*

Description

Internal function, only called by the main spew function

Usage

```
spew_mc(num_places, pop_table, shapefile, pums_h, pums_p, schools, workplaces,
  marginals, output_type, output_dir, convert_count, sampling_method,
  locations_method, outfile_loc, export_objects, road_noise, timer, verbose)
```

Arguments

num_places	The number of regions (place_ids) to be generated for a given location
pop_table	dataframe where rows correspond to places where populations should be generated. Other requird columns are "n_house" and "puma_id"
shapefile	sp class object used for assigning households to particular locations
pums_h	dataframe with microdata corresponding to housegolds
pums_p	dataframe with microdata corresponding to people
schools	list with names "public" and "private" with a dataframe of schools corresponding to public or private, respectively
workplaces	dataframe of workplaces with a workplace_id column, employees column, and stcotr column

marginals	list of marginal population totals. Each element of the list contains the marginal totals of a separate variable
output_type	Default is "console" if we want to resulting population as an R variable. Alternative is "write", which is used on Olympus for writing out .csv files of the population
output_dir	directory to write output if output_type = "write", NULL otherwise
convert_count	logical meant to indicate if we are going to convert population totals from people to household counts. Default: FALSE, assumes the population is the total number of households
sampling_method	character indicating the type of sampling method to use, defaults to "uniform". Can also be "ipf" with appropriate marginal data.
locations_method	character indicating the type of location sampling to use, defaults to "uniform", can also be "roads".
outfile_loc	Defaults to "", so we print out the parallel run information. Only set to "/dev/null" for internal testing putposes.
export_objects	objects for exporting to parallel backends
road_noise	Noise added to households during road-based sampling
timer	logical indicating we want to time the run
verbose	logical indicating we want to print output during the run. Default is FALSE.#'

Value

region_list with output_type for all num_place locations

`spew_mpi` *Run SPEW in Parallel with an MPI backend*

Description

Internal function, only called by the main spew function

Usage

```
spew_mpi(num_places, pop_table, shapefile, pums_h, pums_p, schools, workplaces,
  marginals, output_type, output_dir, convert_count, sampling_method,
  locations_method, outfile_loc, export_objects, road_noise, timer, verbose)
```

Arguments

num_places	The number of regions (place_ids) to be generated for a given location
pop_table	dataframe where rows correspond to places where populations should be generated. Other requird columns are "n_house" and "puma_id"
shapefile	sp class object used for assigning households to particular locations
pums_h	dataframe with microdata corresponding to housegolds
pums_p	dataframe with microdata corresponding to people
schools	list with names "public" and "private" with a dataframe of schools corresponding to public or private, respectively
workplaces	dataframe of workplaces with a workplace_id column, employees column, and stcotr column
marginals	list of marginal population totals. Each element of the list contains the marginal totals of a separate variable
output_type	Default is "console" if we want to resulting population as an R variable. Alternative is "write", which is used on Olympus for writing out .csv files of the population
output_dir	directory to write output if output_type = "write", NULL otherwise
convert_count	logical meant to indicate if we are going to convert population totals from people to household counts. Default: FALSE, assumes the population is the total number of households
sampling_method	character indicating the type of sampling method to use, defaults to "uniform". Can also be "ipf" with appropriate marginal data.
locations_method	character indicating the type of location sampling to use, defaults to "uniform", can also be "roads".
outfile_loc	Defaults to "", so we print out the parallel run information. Only set to "/dev/null" for internal testing putposes.
export_objects	objects for exporting to parallel backends
road_noise	Noise added to households during road-based sampling
timer	logical indicating we want to time the run
verbose	logical indicating we want to print output during the run. Default is FALSE.#'

Value

region_list with output_type for all num_place locations

spew_place *Generate synthetic ecosystem for single place*

Description

Used by 'spew' to split the generated of synthetic ecosytstems into independent regions.

Usage

```
spew_place(index, pop_table, shapefile, pums_h, pums_p, schools = NULL,
  workplaces = NULL, marginals = NULL, output_type = "console",
  sampling_method = "uniform", locations_method = "uniform",
  convert_count = FALSE, output_dir = NULL, road_noise = 2e-04,
  timer = FALSE, verbose = FALSE)
```

Arguments

index	specfic row of pop-table to generate
pop_table	dataframe where rows correspond to places where populations should be generated. Other requird columns are "n_house" and "puma_id"
shapefile	sp class object used for assigning households to particular locations
pums_h	dataframe with microdata corresponding to housegolds
pums_p	dataframe with microdata corresponding to people
schools	list with names "public" and "private" with a dataframe of schools corresponding to public or private, respectively
workplaces	dataframe of workplaces with a workplace_id column, employees column, and stcotr column
marginals	list of marginal population totals. Each element of the list contains the marginal totals of a separate variable
output_type	Default is "console" if we want to resulting population as an R variable. Alternative is "write", which is used on Olympus for writing out .csv files of the population
sampling_method	character indicating the type of sampling method to use, defaults to "uniform". Can also be "ipf" with appropriate marginal data.
locations_method	character indicating the type of location sampling to use, defaults to "uniform", can also be "roads".
convert_count	logical meant to indicate if we are going to convert population totals from people to household counts. Default: FALSE, assumes the population is the total number of households
output_dir	directory to write output if output_type = "write", NULL otherwise
road_noise	Noise added to households during road-based sampling
timer	logical indicating we want to time the run
verbose	logical indicating we want to print output during the run. Default is FALSE.#'

Value

List of a synthetic ecosystem

Examples

```
data(tartanville)
tartanville_T1 <- spew_place(index = 1,
                             pop_table = tartanville$pop_table,
                             shapefile = tartanville$shapefile,
                             pums_h = tartanville$pums_h,
                             pums_p = tartanville$pums_p)
```

---

spew_seq *Run SPEW Sequentially*

Description

Internal function, only called by the main spew function

Usage

```
spew_seq(num_places, pop_table, shapefile, pums_h, pums_p, schools, workplaces,
  marginals, output_type, output_dir, convert_count, sampling_method,
  locations_method, outfile_loc, export_objects, road_noise, timer, verbose)
```

Arguments

num_places	The number of regions (place_ids) to be generated for a given location
pop_table	dataframe where rows correspond to places where populations should be generated. Other requird columns are "n_house" and "puma_id"
shapefile	sp class object used for assigning households to particular locations
pums_h	dataframe with microdata corresponding to housegolds
pums_p	dataframe with microdata corresponding to people
schools	list with names "public" and "private" with a dataframe of schools corresponding to public or private, respectively
workplaces	dataframe of workplaces with a workplace_id column, employees column, and stcotr column
marginals	list of marginal population totals. Each element of the list contains the marginal totals of a separate variable
output_type	Default is "console" if we want to resulting population as an R variable. Alternative is "write", which is used on Olympus for writing out .csv files of the population
output_dir	directory to write output if output_type = "write", NULL otherwise

convert_count	logical meant to indicate if we are going to convert population totals from people to household counts. Default: FALSE, assumes the population is the total number of households
sampling_method	character indicating the type of sampling method to use, defaults to "uniform". Can also be "ipf" with appropriate marginal data.
locations_method	character indicating the type of location sampling to use, defaults to "uniform", can also be "roads".
outfile_loc	Defaults to "", so we print out the parallel run information. Only set to "/dev/null" for internal testing putposes.
export_objects	objects for exporting to parallel backends
road_noise	Noise added to households during road-based sampling
timer	logical indicating we want to time the run
verbose	logical indicating we want to print output during the run. Default is FALSE.#'

Value

region_list with output_type for all num_place locations

`spew_sock` *Run SPEW in Parallel with a SOCK backend*

Description

Internal function, only called by the main spew function

Usage

```
spew_sock(num_places, pop_table, shapefile, pums_h, pums_p, schools, workplaces,
  marginals, output_type, output_dir, convert_count, sampling_method,
  locations_method, outfile_loc, export_objects, road_noise, timer, verbose)
```

Arguments

num_places	The number of regions (place_ids) to be generated for a given location
pop_table	dataframe where rows correspond to places where populations should be generated. Other requird columns are "n_house" and "puma_id"
shapefile	sp class object used for assigning households to particular locations
pums_h	dataframe with microdata corresponding to housegolds
pums_p	dataframe with microdata corresponding to people
schools	list with names "public" and "private" with a dataframe of schools corresponding to public or private, respectively

workplaces	dataframe of workplaces with a workplace_id column, employees column, and stcotr column
marginals	list of marginal population totals. Each element of the list contains the marginal totals of a separate variable
output_type	Default is "console" if we want to resulting population as an R variable. Alternative is "write", which is used on Olympus for writing out .csv files of the population
output_dir	directory to write output if output_type = "write", NULL otherwise
convert_count	logical meant to indicate if we are going to convert population totals from people to household counts. Default: FALSE, assumes the population is the total number of households
sampling_method	character indicating the type of sampling method to use, defaults to "uniform". Can also be "ipf" with appropriate marginal data.
locations_method	character indicating the type of location sampling to use, defaults to "uniform", can also be "roads".
outfile_loc	Defaults to "", so we print out the parallel run information. Only set to "/dev/null" for internal testing putposes.
export_objects	objects for exporting to parallel backends
road_noise	Noise added to households during road-based sampling
timer	logical indicating we want to time the run
verbose	logical indicating we want to print output during the run. Default is FALSE.#'

Value

region_list with output_type for all num_place locations

`standardize_pop_table` *Make sure pop_table has the appropriate columns*

Description

Make sure pop_table has the appropriate columns

Usage

```
standardize_pop_table(pop_table, data_group)
```

Arguments

pop_table	Non-standardized population
data_group	character either "US", "ipums" or "none": Gives the format that read_data expects.

subset_pums *Align pums with marginal totals*

Description

Align pums with marginal totals

Usage

```
subset_pums(pums_h, pums_p, marginals, puma_id)
```

Arguments

pums_h	dataframe of household pums
pums_p	dataframe of people pums
marginals	list of marginals totals
puma_id	id for subsetting the pums

Value

pums with only relevant

subset_schools *Subset the schools to that of the county*

Description

Subset the schools to that of the county

Usage

```
subset_schools(df, schools)
```

Arguments

df	subset of people split so all age, grade, SCH, and county should be the same in the df
schools	list of schools, one data frame of private and one of public

Value

dataframe of subsetted schools to the county and public or private

subset_shapes_roads *Subset the shapefile and road lines to proper roads within specified tract*

Description

Subset the shapefile and road lines to proper roads within specified tract

Usage

```
subset_shapes_roads(place_id, shapefile)
```

Arguments

place_id	numeric specifying the ID of the region we are sub-sampling
shapefile	sp class with all of the locations for each place id. In addition, we must have road shapefiles so shapefile is a list with both the tracts and the roads, tracts is the first object and roads the second.

Value

new_shp - roads within the tract, a SpatialLines object

summarize_environment *Return the unique environment assignments in a region*

Description

Return the unique environment assignments in a region

Usage

```
summarize_environment(df, env_vars)
```

Arguments

df	the dataframe for the region
env_vars	the environmental variables we want to summarize

Value

list of unique environment assignments for each environment

summarize_features *Summarize individual features of a region*

Description

Summarize individual features of a region

Usage

```
summarize_features(df, marginals, vars_to_sum)
```

Arguments

df	data frame of individuals in region subsetted to vars to sum and marginal equivalents
marginals	marginal list or NULL
vars_to_sum	name of variables to summarize

Value

list of table of each feature

summarize_spew *Summarize a SPEW region*

Description

Summarize a SPEW region

Usage

```
summarize_spew(filenames, marginals = NULL, vars_to_sum, env_vars = NULL,
  coords = TRUE, samp_size = 10^4, read = FALSE, pops = NULL,
  type = NULL)
```

Arguments

filenames	list of full path to files to summarize and the id of the region
marginals	list containing all of the marginal totals. See ?make_ipf_marg for more details. Default is NULL
vars_to_sum	variables we wish to summarize, should correspond to marginals object
env_vars	environment variables to summarize. Default is NULL
coords	logical of whether to extract the longitude/latitude coordinates and store
samp_size	number of lon/lat coordinates to sample. Default is 10^4

read	logical of whether we need to read in the populations. Default is FALSE
pops	list of the populations produced by SPEW for a household OR people
type	either NULL, "households", or "people".

Value

a list of length of 1 for the region ID , 1 for population size, and optionally one for a dataframe of stored lat/long coords, the number of variables to summarize , and the environmental variables

---

`summarize_spew_out` *Summarize spew output*

Description

Summarize spew output

Usage

```
summarize_spew_out(syneco = NULL, vars_to_sum_h, vars_to_sum_p,
  vars_to_sum_env = NULL, samp_size = 10^4, type = "US",
  summary_level = 2, marginals = NULL, output_dir = NULL,
  top_region_id = NULL, has_marg = FALSE)
```

Arguments

syneco	output from the 'spew' function
vars_to_sum_h	character vector of variables from the household data frame output to summarize
vars_to_sum_p	character vector of variables from the person data frame output to summarize
vars_to_sum_env	character vector of variables from the person data frame which correspond to environment assignments. Default is NULL.
samp_size	number of agents to retain from each lower-level region, for plotting purposes only. Default is 10^4.
type	Only used when output_dir is specified."US" for a US population, "IPUMS" for IPUMS population, or "custom" for a custom population. This effects what the summary levels are.
summary_level	Only used when output_dir is specified. IFor the US, 1-state, 2-county, 3-tract. For IPUMS, 1 -country, 2-province. For "custom," these are defined by the user's input data.
marginals	list containing all of the marginal totals. See ?make_ipf_marg for more details.
output_dir	path to top level directory of SPEW folders. Ex. "./10" for Delaware. Default is NULL. In the case it is NULL, we do not need to read in data.
top_region_id	name of the region. Default is NULL. It is only used in the case where we directly summarize the syneco object.
has_marg	Does the region of marginals to refer to? Logical. Default is FALSE.

Value

list with the household summary list, people summary list, header for households, and header for people, and a data frame of plotting coordinates by summary region

Note

This function is only guaranteed to work when you provide marginals describing how a category is "cut." If a certain category is not represented, then the final totals in each category may be off.

Examples

```
data(tartanville)

tartanville_syneco <- spew(tartanville$pop_table, tartanville$shapefile,
                           tartanville$pums_h, tartanville$pums_p)

out <-  summarize_spew_out(tartanville_syneco,
                           vars_to_sum_h = c("puma_id"),
                           vars_to_sum_p = c("SEX"),
                           vars_to_sum_env = NULL,
                           top_region_id = "Tartanville")
print(out)
```

---

summarize_spew_region *Summarize a singular region from spew output*

Description

Summarize a singular region from spew output

Usage

```
summarize_spew_region(spew_region, type = "households", marginals,
  vars_to_sum, vars_to_sum_env = NULL)
```

Arguments

spew_region	a single region (entry in the list) from spew
type	either "households or "people". Default is "households". The type to summarize
marginals	marginals object
vars_to_sum	names of the variables to summarize
vars_to_sum_env	environment variables to summarize. Default is NULL

Value

a list of length of 1 for the region ID , 1 for population size, and optionally one for a dataframe of stored lat/long coords, the number of variables to summarize , and the environmental variables

summarize_syneco *Summarize synthetic ecosystem for SPEW console output*

Description

Summarize synthetic ecosystem for SPEW console output

Usage

```
summarize_syneco(syneco, vars_to_sum_h, vars_to_sum_p, vars_to_sum_env = NULL,
  samp_size = 10^4, marginals = NULL, top_region_id = "Ecosystem",
  has_marg = FALSE)
```

Arguments

syneco	output from the 'spew' function
vars_to_sum_h	character vector of variables from the household data frame output to summarize
vars_to_sum_p	character vector of variables from the person data frame output to summarize
vars_to_sum_env	character vector of variables from the person data frame which correspond to environment assignments. Default is NULL.
samp_size	number of agents to retain from each lower-level region, for plotting purposes only. Default is 10^4.
marginals	list containing all of the marginal totals. See ?make_ipf_marg for more details.
top_region_id	Name supplied of region.
has_marg	Does the region of marginals to refer to? Logical. Default is FALSE.

Value

list with the household summary list, people summary list, header for households, and header for people, and a data frame of plotting coordinates by summary region

summarize_top_region *Summarize the region in a more human-readable format*

Description

Summarize the region in a more human-readable format

Usage

```
summarize_top_region(output_dir, type = "US", vars_to_sum_h, vars_to_sum_p,
  vars_to_sum_env = NULL, samp_size = 10^4, summary_level = 2,
  marginals = NULL)
```

Arguments

output_dir	path to top level directory of SPEW folders. Ex. "./10" for Delaware
type	"US" for a US population, "IPUMS" for IPUMS population, or "custom" for a custom population. This effects what the summary levels are.
vars_to_sum_h	character vector of variables from the household data frame output to summarize
vars_to_sum_p	character vector of variables from the person data frame output to summarize
vars_to_sum_env	character vector of variables from the person data frame which correspond to environment assignments. Default is NULL.
samp_size	number of agents to retain from each lower-level region, for plotting purposes only. Default is 10^4.
summary_level	For the US, 1-state, 2-county, 3-tract. For IPUMS, 1 -country, 2-province. For "custom," these are defined by the user's input data.
marginals	list containing all of the marginal totals. See ?make_ipf_marg for more details.

Value

list with the household summary list, people summary list, header for households, and header for people, and a data frame of plotting coordinates by summary region

Note

This function is only guaranteed to work when you provide marginals describing how a category is "cut." If a certain category is not represented, then the final totals in each category may be off.

---

tartanville *Input data for Tartanville*

Description

Input data for Tartanville

Usage

```
tartanville
```

Format

An object of class `list` of length 6.

update_freqs *Update frequencies to match # of households*

Description

Update frequencies to match # of households

Usage

```
update_freqs(freqs, n_house)
```

Arguments

freqs	current number of households
n_house	number of households

uruguay *Input data for Uruguay*

Description

Input data for Uruguay

Usage

```
uruguay
```

Format

An object of class `list` of length 8.

us *Table of US states and counties*

Description

Table of US states and counties

Usage

```
us
```

Format

An object of class `data.frame` with 3235 rows and 8 columns.

us_pums_sf *An example marginal distribution table*

Description

An example marginal distribution table

Usage

```
us_pums_sf
```

Format

An object of class `list` of length 4.

verify_column *Verify the column is the correct size*

Description

Verify the column is the correct size

Usage

```
verify_column(column, df_size)
```

Arguments

`column`	to be verified
`df_size`	size

weight_dists *Weight school assignment probabilities*

Description

Weight school assignment probabilities

Usage

```
weight_dists(dist_mat, schools)
```

Arguments

dist_mat	a m x n matrix where m is the number of people and n is the number of schools
schools	data frame of schools

Value

m x n matrix of probabilities. Each row should sum to 1

---

weight_dists2 *Weight school assignment probabilities*

Description

Weight school assignment probabilities

Usage

```
weight_dists2(dist_mat, schools)
```

Arguments

dist_mat	a m x n matrix where m is the number of people and n is the number of schools
schools	data frame of schools

Value

m x n matrix of probabilities. Each row should sum to 1

---

weight_dists_C *Weight school assignment probabilities by capacity only*

Description

Weight school assignment probabilities by capacity only

Usage

```
weight_dists_C(dist_mat, schools)
```

Arguments

dist_mat	a m x n matrix where m is the number of people and n is the number of schools
schools	data frame of schools

Value

m x n matrix of probabilities. Each row should sum to 1

weight_dists_D *Weight school assignment probabilities, distance only*

Description

Weight school assignment probabilities, distance only

Usage

```
weight_dists_D(dist_mat, schools)
```

Arguments

dist_mat	a m x n matrix where m is the number of people and n is the number of schools
schools	data frame of schools

Value

m x n matrix of probabilities. Each row should sum to 1

write_data *Output our final synthetic populations as csv's*

Description

Output our final synthetic populations as csv's

Usage

```
write_data(df, place_id, puma_id, type, output_dir, append = FALSE)
```

Arguments

df	dataframe with the final synthetic population
place_id	numeric indicating the name of the particular region samples
puma_id	numeric indicating the puma this synthetic population belongs to
type	character vector with the type, either "household" or "people"
output_dir	character containing the directory in which we want to write the final csv's
append	logical determining if we want to append to the current file

Value

data indicating the indices of people to sample

write_pop_table *Write out the final, formatted table*

Description

Write out the final, formatted table

Usage

```
write_pop_table(pop_table, output_dir)
```

Arguments

pop_table	the population table df
output_dir	character vector of the directory to write the population table

Value

logical TRUE if completed. As well as a written pop_table to the given output directory

write_schools *Write school environment*

Description

Write school environment

Usage

```
write_schools(schools, env_dir)
```

Arguments

schools	school data-object
env_dir	filepath to write out the environment

Value

logical TRUE if completed successfully

`write_workplaces` *Write workplaces environment*

Description

Write workplaces environment

Usage

```
write_workplaces(workplaces, env_dir)
```

Arguments

`workplaces`	school data-object
`env_dir`	filepath to write out the environment

Value

logical TRUE if completed successfully

Index